



Hooks 用户指南

AWS CloudFormation



AWS CloudFormation: Hooks 用户指南

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

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

Table of Contents

什么是 AWS CloudFormation Hooks ?	1
创建和管理 Hook	2
概念	3
Hook	3
故障模式	4
钩住目标	4
目标操作	5
钩子处理器	5
防护挂钩	5
AWS CLI 用于使用防护挂钩的命令	6
为 Hook 编写防护规则	6
准备创建防护挂钩	19
激活警戒钩	21
查看 Guard Hooks 日志	26
删除防护挂钩	26
Lambda Hooks	27
AWS CLI 用于使用 Lambda 挂钩的命令	28
为挂钩创建 Lambda 函数	28
准备创建 Lambda 挂钩	47
激活 Lambda 挂钩	48
查看 Lambda 挂钩的日志	52
删除 Lambda 挂钩	52
自定义挂钩	53
先决条件	55
启动 Hooks 项目	56
建模挂钩	59
注册挂钩	126
测试挂钩	130
更新挂钩	139
取消注册挂钩	139
发布挂钩	140
架构语法	147
禁用挂钩	156
禁用和启用挂钩 (控制台)	156

禁用并启用挂钩 (AWS CLI)	157
配置架构	158
挂钩配置架构属性	158
挂钩配置示例	159
堆栈级别过滤器	160
FilteringCriteria	161
StackNames	161
StackRoles	162
Include 和 Exclude	162
堆栈级过滤器示例	163
目标过滤器	167
目标过滤器示例	168
使用通配符	170
使用 CloudFormation 模板创建 Hook	173
文档历史记录	175
.....	clxxvii

什么是 AWS CloudFormation Hooks ?

AWS CloudFormation Hooks 是一项功能，您可以使用它来确保您的 CloudFormation 资源、堆栈、变更集符合组织的安全、运营和成本优化最佳实践。CloudFormation Hook 还可以确保您的 AWS Cloud Control API 资源达到同样的合规性。借助 CloudFormation Hook，您可以提供在配置之前主动检查 AWS 资源配置的代码。如果发现不合规的资源，则 AWS CloudFormation 要么操作失败并阻止资源置备，要么发出警告并允许配置操作继续进行。

您可以使用 Hook 来强制执行各种要求和指导方针。例如，与安全相关的挂钩可以验证安全组是否符合您的 Amazon Virtual Private Cloud (Amazon VPC) 的相应入站和出站流量规则。与成本相关的挂钩可以将开发环境限制为仅使用较小的 [亚马逊弹性计算云 \(Amazon EC2\)](#) 实例类型。专为数据可用性而设计的挂钩可以强制对 [亚马逊关系数据库服务 \(Amazon RDS\)](#) 进行自动备份。

CloudFormation Hooks 是 [AWS CloudFormation 注册表](#) 中支持的扩展类型。通过注册表，可以轻松地在公开和私下分发和激活 Hooks。您可以使用预先构建的 Hook，也可以使用构建自己的 Hook。 [CloudFormation CLI](#)

本指南概述了 AWS CloudFormation Hook 的结构，以及开发、注册、测试、管理和发布自己的 Hook 的指南。

创建和管理 AWS CloudFormation Hook

AWS CloudFormation Hook 提供了一种在允许创建、修改或删除堆栈之前评估 CloudFormation 资源的机制。此功能可帮助您确保您的 CloudFormation 资源符合组织的安全、运营和成本优化最佳实践。

要创建 Hook，有三个选项。

- Guard Hook — 使用 AWS CloudFormation Guard 规则评估资源。
- Lambda 挂钩 — 将资源评估请求转发给函数。AWS Lambda
- Custom Hook — 使用您手动开发的自定义 Hook 处理程序。

Guard Hook

要创建 Guard Hook，请执行以下主要步骤：

1. 使用 Guard 域特定语言 (DSL) 将您的资源评估逻辑写成 Guard 策略规则。
2. 将防护策略规则存储在 Amazon S3 存储桶中。
3. 导航到 CloudFormation 控制台并开始创建 Guard Hook。
4. 提供指向您的防护规则的 Amazon S3 路径。
5. 选择 Hook 将评估的特定目标。
6. 选择将调用您的 Hook 的部署操作 (创建、更新、删除)。
7. 选择 Hook 在评估失败时的响应方式。
8. 配置完成后，激活 Hook 以开始强制执行。

Lambda Hook

要创建 Lambda 挂钩，请执行以下主要步骤：

1. 将您的资源评估逻辑写成 Lambda 函数。
2. 导航到 CloudFormation 控制台并开始创建 Lambda 挂钩。
3. 为您的 Lambda 函数提供亚马逊资源名称 (ARN)。
4. 选择 Hook 将评估的特定目标。
5. 选择将调用您的 Hook 的部署操作 (创建、更新、删除)。
6. 选择 Hook 在评估失败时的响应方式。
7. 配置完成后，激活 Hook 以开始强制执行。

Custom Hook

自定义挂钩是使用 CloudFormation 命令行界面 (CFN-CLI) 在 CloudFormation 注册表中注册的扩展。

要创建自定义 Hook，请执行以下主要步骤：

1. 启动项目 — 生成开发自定义 Hook 所需的文件。
2. 为挂钩建模 — 编写一个架构来定义 Hook 和指定可以调用 Hook 的操作的处理程序。
3. 注册并激活挂钩 — 创建挂钩后，需要在要使用挂钩的账户和区域中注册挂钩，这样即可激活挂钩。

以下主题提供了有关创建和管理 Hook 的更多信息。

主题

- [AWS CloudFormation 挂钩概念](#)
- [防护挂钩](#)
- [Lambda Hooks](#)
- [使用开发自定义 Hook CloudFormation CLI](#)

AWS CloudFormation 挂钩概念

以下术语和概念对你理解和使用 AWS CloudFormation Hooks 至关重要：

- [Hook](#)
- [钩住目标](#)
- [目标操作](#)
- [钩子处理器](#)

Hook

Hook 包含在 CloudFormation 创建、更新或删除堆栈或特定资源之前立即调用的代码。它也可以在创建更改集操作期间调用。Hooks 可以检查即将配置的 CloudFormation 模板、资源或变更集。此外，可以在[云控件API](#)创建、更新或删除特定资源之前立即调用 Hook。

如果 Hook 发现任何不符合 Hook 逻辑中定义的组织准则的配置，则您可以选择向WARN用户或FAIL CloudFormation 阻止配置资源。

挂钩具有以下特征：

- 主动验证 — 通过在创建、更新或删除不合规资源之前识别出这些资源，从而降低风险、运营开销和成本。
- 自动强制执行 — 在您的中提供强制执行 AWS 账户，以防止由配置不合规的资源。
CloudFormation

故障模式

你的 Hook 逻辑可以返回成功或失败。成功响应将允许操作继续。不合规的资源出现故障可能会导致以下结果：

- FAIL— 停止配置操作。
- WARN— 允许继续配置并显示警告消息。

在WARN模式下创建 Hook 是在不影响堆栈操作的情况下监控 Hook 行为的有效方法。首先，在WARN模式下激活 Hook 以了解哪些操作会受到影响。评估了潜在影响后，可以将挂钩切换到FAIL模式以开始防止不合规的操作。

钩住目标

挂钩目标指定挂钩将评估的操作。这些操作可以是：

- CloudFormation (Resources) 支持的资源
- 堆栈模板 (STACK)
- 零钱套装 (CHANGE_SET)
- 云控制支持的资源 API (CLOUD_CONTROL)

您可以定义一个或多个目标，用于指定 Hook 将评估的最广泛的操作。例如，您可以创作一个 Hook 定位RESOURCES以定位所有 AWS 资源并STACK定位所有堆栈模板。您还可以应用过滤器将 Hook 的范围缩小到要评估的确切操作。有关更多信息，请参阅[AWS CloudFormation Hooks 堆栈级别过滤器](#)和[AWS CloudFormation 挂钩目标过滤器](#)。

目标操作

目标操作是触发 Hook 的操作类型。操作可以是CREATEUPDATE、或DELETE。

Note

使用RESOURCESTACK、和 CLOUD_CONTROL Hook 目标时，所有目标操作都适用。使用 CHANGE_SET Hook 目标时，只有CREATE操作才适用。

钩子处理器

对于自定义 Hook，这是处理评估的代码。它与目标调用点和目标操作相关联，后者标记 Hook 运行的确切点。你编写处理程序来托管这些特定点的逻辑。例如，带有PRE目标操作的目标调用CREATE点会生成一个 preCreate Hook 处理程序。当匹配的目标调用点和服务正在执行关联的目标操作时，Hook 处理程序中的代码就会运行。

有效值：(preCreate| preUpdate |preDelete)

Important

导致状态为的堆栈操作UpdateCleanup不会调用 Hook。例如，在以下两个场景中，不会调用 Hook 的preDelete处理程序：

- 从模板中移除一个资源后，堆栈即会更新。
- 更新类型为[替换](#)的资源被删除。

防护挂钩

要在您的账户中使用 AWS CloudFormation Guard 挂钩，您必须为要使用该挂钩的账户和区域激活挂钩。激活 Hook 后，它可以在激活该挂钩的账户和区域的堆栈操作中使用。

激活 Guard Hook 时，CloudFormation 会在账户的注册表中为已激活的 Hook 创建一个条目作为私有 Hook。这允许您设置 Hook 包含的任何配置属性。配置属性定义了如何为给定 AWS 账户 和区域配置挂钩。

主题

- [AWS CLI 用于使用防护挂钩的命令](#)
- [编写 Guard 规则来评估防护挂钩的资源](#)
- [准备创建防护挂钩](#)
- [在你的账户中激活 Guard Hook](#)
- [查看您账户中防御挂钩的日志](#)
- [在你的账户中删除防护挂钩](#)

AWS CLI 用于使用防护挂钩的命令

使用 Guard Hook 的 AWS CLI 命令包括：

- [activate-type](#)开始激活 Guard Hook 的过程。
- [set-type-configuration](#)来指定您账户中的 Hook 的配置数据。
- [list-types](#)列出你账户中的 Hook。
- [describe-type](#)返回有关特定 Hook 或特定 Hook 版本的详细信息，包括当前配置数据。
- [deactivate-type](#)从您的账户中移除之前激活的 Hook。

编写 Guard 规则来评估防护挂钩的资源

AWS CloudFormation Guard 是一种开源的、通用领域专用语言 (DSL)，你可以用来创作 policy-as-code。本主题介绍如何使用 Guard 编写示例规则，这些规则可以在 Guard Hook 中运行以自动评估 CloudFormation 和 AWS Cloud Control API 操作。它还将重点关注你的守卫规则可用的不同类型的输入，具体取决于你的 Guard Hook 的运行时间。可以将 Guard Hook 配置为在以下类型的操作期间运行：

- 资源操作
- 堆栈操作
- 更改集合操作

有关编写 Guard 规则的更多信息，请参阅[编写 AWS CloudFormation Guard 规则](#)

主题

- [资源操作守卫规则](#)
- [堆栈操作防护规则](#)

- [更改集合操作守卫规则](#)

资源操作守卫规则

每当您创建、更新或删除资源时，都被视为资源操作。例如，如果您运行更新 CloudFormation 堆栈以创建新资源，则表示您已完成资源操作。当您使用 Cloud Control 创建API、更新或删除资源时，这也被视为资源操作。您可以将 Guard Hook 配置为目标，RESOURCE并在挂钩的TargetOperations配置中配置CLOUD_CONTROL操作。当您的 Guard Hook 评估资源操作时，Guard 引擎会评估资源输入。

主题

- [守卫资源输入语法](#)
- [示例 Guard 资源操作输入](#)
- [资源变更的守护规则](#)

守卫资源输入语法

Guard 资源输入是可供您的 Guard 规则评估的数据。

以下是资源输入的示例形状：

```
HookContext:
  AWSAccountID: String
  StackId: String
  HookTypeName: String
  HookTypeVersion: String
  InvocationPoint: [CREATE_PRE_PROVISION, UPDATE_PRE_PROVISION, DELETE_PRE_PROVISION]
  TargetName: String
  TargetType: RESOURCE
  TargetLogicalId: String
  ChangeSetId: String
Resources:
  {ResourceLogicalID}:
    ResourceType: {ResourceType}
    ResourceProperties:
      {ResourceProperties}
Previous:
  ResourceLogicalID:
    ResourceType: {ResourceType}
    ResourceProperties:
      {PreviousResourceProperties}
```

HookContext

AWSAccountID

AWS 账户 包含正在评估的资源的 ID。

StackId

作为资源操作一部分的 CloudFormation 堆栈的堆栈 ID。如果来电者是云控制，则此字段为空 API。

HookTypeName

正在运行的 Hook 的名称。

HookTypeVersion

正在运行的 Hook 的版本。

InvocationPoint

配置逻辑中挂钩运行的确切位置。

有效值：(CREATE_PRE_PROVISION| UPDATE_PRE_PROVISION |DELETE_PRE_PROVISION)

TargetName

正在评估的资源的名称。

TargetType

正在评估的资源的资源类型 (例如：AWS::S3::Bucket)。

TargetLogicalId

正在评估TargetLogicalId的资源的。如果 Hook 的来源是 CloudFormation，则这将是资源的逻辑 ID (也称为逻辑名称)。如果 Hook 的起源是 Cloud ControlAPI，则这将是一个构造值。

ChangeSetId

为导致 Hook 调用而执行的更改集 ID。如果资源变更是由 Cloud Control 或API、或delete-stack操作启动的 create-stackupdate-stack，则此值为空。

Resources

ResourceLogicalID

当操作由启动时 CloudFormation，ResourceLogicalID就是 CloudFormation 模板中资源的逻辑 ID。

当操作由 Cloud Control 启动时API , ResourceLogicalID是资源类型、名称、操作 ID 和请求 ID 的组合。

ResourceType

资源的类型名称 (例如 : AWS::S3::Bucket) 。

ResourceProperties

正在修改的资源的建议属性。当 Guard Hook 针对 CloudFormation 资源变化运行时，任何函数、参数和转换都将得到完全解析。如果要删除资源，则此值将为空。

Previous

ResourceLogicalID

当操作由启动时 CloudFormation , ResourceLogicalID就是 CloudFormation 模板中资源的逻辑 ID。

当操作由 Cloud Control 启动时API , ResourceLogicalID是资源类型、名称、操作 ID 和请求 ID 的组合。

ResourceType

资源的类型名称 (例如 : AWS::S3::Bucket) 。

ResourceProperties

与正在修改的资源相关的当前属性。如果要删除资源，则此值将为空。

示例 Guard 资源操作输入

以下示例输入显示了一个 Guard Hook，它将接收正在更改的AWS::S3::Bucket资源的定义。这是 Guard 可用于评估的数据。

```
HookContext:
  AwsAccountId: "00000000"
  StackId: ""
  HookTypeName: org::s3policy::hook
  HookTypeVersion: "00001"
  InvocationPoint: UPDATE_PRE_PROVISION
  TargetName: AWS::S3::Bucket
  TargetType: RESOURCE
  TargetLogicalId: MyS3Bucket
```

```

ChangeSetId: ""
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: amzn-s3-demo-bucket
      ObjectLockEnabled: true
Previous:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: amzn-s3-demo-bucket
      ObjectLockEnabled: false

```

要查看该资源类型的所有可用属性，请参阅[AWS::S3::Bucket](#)。

资源变更的守护规则

当 Guard Hook 评估资源变化时，它首先下载使用该挂钩配置的所有规则。然后根据资源输入对这些规则进行评估。如果有任何规则的评估失败，Hook 就会失败。如果没有失败，Hook 就会通过。

以下示例是一条防护规则，用于评估该ObjectLockEnabled属性是否true适用于任何AWS::S3::Bucket资源类型。

```

let s3_buckets_default_lock_enabled = Resources.*[ Type == 'AWS::S3::Bucket']

rule S3_BUCKET_DEFAULT_LOCK_ENABLED when %s3_buckets_default_lock_enabled != empty {
  %s3_buckets_default_lock_enabled.Properties.ObjectLockEnabled exists
  %s3_buckets_default_lock_enabled.Properties.ObjectLockEnabled == true
  <<
    Violation: S3 Bucket ObjectLockEnabled must be set to true.
    Fix: Set the S3 property ObjectLockEnabled parameter to true.
  >>
}

```

当此规则针对以下输入运行时，它将失败，因为该ObjectLockEnabled属性未设置为true。

```

Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: amzn-s3-demo-bucket

```

```
ObjectLockEnabled: false
```

当此规则针对以下输入运行时，它将通过，因为设置ObjectLockEnabled为true。

```
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: amzn-s3-demo-bucket
      ObjectLockEnabled: true
```

当 Hook 失败时，失败的规则将传播回 CloudFormation 或 Cloud Control API。如果已为 Guard Hook 配置了日志存储桶，则将在那里提供其他规则反馈。此额外反馈包括Violation和Fix信息。

堆栈操作防护规则

创建、更新或删除 CloudFormation 堆栈时，您可以将 Guard Hook 配置为从评估新模板开始，并可能阻止堆栈操作继续进行。您可以将 Guard Hook TargetOperations 配置为挂钩配置中的目标STACK操作。

主题

- [防护堆栈输入语法](#)
- [示例 Guard 堆栈操作输入](#)
- [堆栈变更的保护规则](#)

防护堆栈输入语法

Guard 堆栈操作的输入为您的 Guard 规则提供了要评估的完整 CloudFormation模板。

以下是堆栈输入的示例形状：

```
HookContext:
  AWSAccountID: String
  StackId: String
  HookTypeName: String
  HookTypeVersion: String
  InvocationPoint: [CREATE_PRE_PROVISION, UPDATE_PRE_PROVISION, DELETE_PRE_PROVISION]
  TargetName: String
  TargetType: STACK
```

```
ChangeSetId: String
{Proposed CloudFormation Template}
Previous:
  {CloudFormation Template}
```

HookContext

AWSAccountID

AWS 账户 包含资源的 ID。

StackId

作为堆栈操作一部分的 CloudFormation 堆栈的堆栈 ID。

HookTypeName

正在运行的 Hook 的名称。

HookTypeVersion

正在运行的 Hook 的版本。

InvocationPoint

配置逻辑中挂钩运行的确切位置。

有效值 : (CREATE_PRE_PROVISION| UPDATE_PRE_PROVISION |DELETE_PRE_PROVISION)

TargetName

正在评估的堆栈的名称。

TargetType

当作为堆栈级 Hook 运行STACK时，此值将为。

ChangeSetId

为导致 Hook 调用而执行的更改集 ID。如果堆栈操作由、或delete-stack操作启动 create-stackupdate-stack，则此值为空。

Proposed CloudFormation Template

传递给 CloudFormationcreate-stack或update-stack操作的完整 CloudFormation 模板值。这包括诸如ResourcesOutputs、和之类的内容Properties。它可以是JSON或YAML字符串，具体取决于提供的内容 CloudFormation。

在delete-stack操作中，此值将为空。

Previous

上次成功部署的 CloudFormation 模板。如果正在创建或删除堆栈，则此值为空。

在delete-stack操作中，此值将为空。

Note

提供的模板是传递给堆栈update操作create的模板。删除堆栈时，不提供模板值。

示例 Guard 堆栈操作输入

以下示例输入显示了将接收完整模板和先前部署的模板的 Guard Hook。本示例中的模板使用的JSON格式为。

```
HookContext:
  AwsAccountId: 123456789012
  StackId: arn:aws:cloudformation:us-east-1:123456789012:stack/myteststack
  HookTypeName: org::templatechecker::hook
  HookTypeVersion: "00001"
  InvocationPoint: UPDATE_PRE_PROVISION
  TargetName: my-example-stack
  TargetType: CHANGE_SET
  TargetLogicalId: arn:aws:cloudformation:us-east-1:123456789012:changeSet/
SampleChangeSet/12a3b456-0e10-4ce0-9052-5d484a8c4e5b
  ChangeSetId: arn:aws:cloudformation:us-east-1:123456789012:changeSet/
SampleChangeSet/12a3b456-0e10-4ce0-9052-5d484a8c4e5b
Resources: {
  "S3Bucket": {
    "Type": "AWS::S3::Bucket",
    "Properties": {
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
          {"ServerSideEncryptionByDefault":
            {"SSEAlgorithm": "aws:kms",
             "KMSMasterKeyID": "KMS-KEY-ARN" }},
          {"BucketKeyEnabled": true }
        ]
      }
    }
  }
}
```

```

    }
  }
  Previous: {
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
      "S3Bucket": {
        "Type": "AWS::S3::Bucket",
        "Properties": {}
      }
    }
  }
}

```

堆栈变更的保护规则

当 Guard Hook 评估堆栈更改时，它首先下载使用该挂钩配置的所有规则。然后根据资源输入对这些规则进行评估。如果有任何规则的评估失败，Hook 就会失败。如果没有失败，Hook 就会通过。

以下示例是一条 Guard 规则，用于评估是否有任何 `AWS::S3::Bucket` 资源类型包含名为的属性 `BucketEncryption`，且该属性 `SSEAlgorithm` 设置为 `aws:kms` 或 `AES256`。

```

let s3_buckets_s3_default_encryption = Resources.*[ Type == 'AWS::S3::Bucket']

rule S3_DEFAULT_ENCRYPTION_KMS when %s3_buckets_s3_default_encryption !empty {
  %s3_buckets_s3_default_encryption.Properties.BucketEncryption exists

  %s3_buckets_s3_default_encryption.Properties.BucketEncryption.ServerSideEncryptionConfiguration
  in ["aws:kms","AES256"]
  <<
    Violation: S3 Bucket default encryption must be set.
    Fix: Set the S3 Bucket property
    BucketEncryption.ServerSideEncryptionConfiguration.ServerSideEncryptionByDefault.SSEAlgorithm
  to either "aws:kms" or "AES256"
  >>
}

```

当规则针对以下模板运行时，它会运行的 `fail`。

```

AWSTemplateFormatVersion: 2010-09-09
Description: S3 bucket without default encryption
Resources:
  EncryptedS3Bucket:
    Type: 'AWS::S3::Bucket'

```

```
Properties:
  BucketName: !Sub 'encryptedbucket-${AWS::Region}-${AWS::AccountId}'
```

当规则针对以下模板运行时，它会运行的pass。

```
AWSTemplateFormatVersion: 2010-09-09
Description: S3 bucket with default encryption using SSE-KMS with an S3 Bucket Key
Resources:
  EncryptedS3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: !Sub 'encryptedbucket-${AWS::Region}-${AWS::AccountId}'
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: 'aws:kms'
              KMSEncryptionKeyId: KMS-KEY-ARN
              BucketKeyEnabled: true
```

更改集合操作守卫规则

创建 CloudFormation 变更集后，您可以将 Guard Hook 配置为评估模板和变更集中提出的更改，以阻止变更集的执行。

主题

- [防护变更集输入语法](#)
- [示例：防护变更集操作输入](#)
- [变更集操作的保护规则](#)

防护变更集输入语法

Guard 更改集输入是可供您的 Guard 规则评估的数据。

以下是更改集输入的示例形状：

```
HookContext:
  AWSAccountID: String
  StackId: String
  HookTypeName: String
  HookTypeVersion: String
```

```

InvocationPoint: [CREATE_PRE_PROVISION, UPDATE_PRE_PROVISION, DELETE_PRE_PROVISION]
TargetName: CHANGE_SET
TargetType: CHANGE_SET
TargetLogicalId: ChangeSet ID
ChangeSetId: String
{Proposed CloudFormation Template}
Previous:
  {CloudFormation Template}
Changes: [{ResourceChange}]

```

ResourceChange模型语法为：

```

logicalResourceId: String
resourceType: String
action: CREATE, UPDATE, DELETE
lineNumber: Number
beforeContext: JSON String
afterContext: JSON String

```

HookContext

AWSAccountID

AWS 账户 包含资源的 ID。

StackId

作为堆栈操作一部分的 CloudFormation 堆栈的堆栈 ID。

HookTypeName

正在运行的 Hook 的名称。

HookTypeVersion

正在运行的 Hook 的版本。

InvocationPoint

配置逻辑中挂钩运行的确切位置。

有效值：(CREATE_PRE_PROVISION| UPDATE_PRE_PROVISION |DELETE_PRE_PROVISION)

TargetName

正在评估的堆栈的名称。

TargetType

当作为更改集级别的 Hook 运行CHANGE_SET时，将使用此值。

TargetLogicalId

该值将是变更集ARN的值。

ChangeSetId

为导致 Hook 调用而执行的更改集 ID。如果堆栈操作由、或delete-stack操作启动 create-stackupdate-stack，则此值为空。

Proposed CloudFormation Template

为create-change-set操作提供的完整 CloudFormation 模板。它可以是JSON或YAML字符串，具体取决于提供的内容 CloudFormation。

Previous

上次成功部署的 CloudFormation 模板。如果正在创建或删除堆栈，则此值为空。

Changes

Changes模型。这列出了资源更改。

更改

logicalResourceId

已更改资源的逻辑资源名称。

resourceType

将要更改的资源类型。

action

对资源执行的操作类型。

有效值：(CREATE| UPDATE |DELETE)

lineNumber

模板中与变更相关的行号。

beforeContext

变更前资源的属性JSON字符串：

```
{"properties": {"property1": "value"}}
```

afterContext

更改后的资源属性JSON字符串：

```
{"properties": {"property1": "new value"}}
```

示例：防护变更集操作输入

以下示例输入显示了将接收完整模板的 Guard Hook、之前部署的模板和资源更改列表。本示例中的模板使用的JSON格式为。

```
HookContext:
  AwsAccountId: "000000000"
  StackId: my-example-stack
  HookTypeName: org::templatechecker::hook
  HookTypeVersion: "00001"
  InvocationPoint: UPDATE_PRE_PROVISION
  TargetName: my-example-stack
  TargetType: STACK
  TargetLogicalId: arn...:changeSet/change-set
  ChangeSetId: ""
Resources: {
  "S3Bucket": {
    "Type": "AWS::S3::Bucket",
    "Properties": {
      "BucketName": "amzn-s3-demo-bucket",
      "VersioningConfiguration": {
        "Status": "Enabled"
      }
    }
  }
}
Previous: {
  "AWSTemplateFormatVersion": "2010-09-09",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
```

```

        "Properties": {
            "BucketName": "amzn-s3-demo-bucket",
            "VersioningConfiguration": {
                "Status": "Suspended"
            }
        }
    }
}
Changes: [
    {
        "logicalResourceId": "S3Bucket",
        "resourceType": "AWS::S3::Bucket",
        "action": "UPDATE",
        "lineNumber": 5,
        "beforeContext": "{\"Properties\":{\"VersioningConfiguration\":{\"Status\": \"Suspended\"}}}",
        "afterContext": "{\"Properties\":{\"VersioningConfiguration\":{\"Status\": \"Enabled\"}}}"
    }
]

```

变更集操作的保护规则

以下示例是一条防护规则，用于评估对 Amazon S3 存储桶的更改，并确保该规则未VersionConfiguration被禁用。

```

let s3_buckets_changing = Changes[resourceType == 'AWS::S3::Bucket']

rule S3_VERSIONING_STAY_ENABLED when %s3_buckets_changing !empty {
    let afterContext = json_parse(%s3_buckets_changing.afterContext)
    when %afterContext.Properties.VersioningConfiguration.Status !empty {
        %afterContext.Properties.VersioningConfiguration.Status == 'Enabled'
    }
}

```

准备创建防护挂钩

在创建 Guard Hook 之前，必须满足以下先决条件：

- 您必须已经创建了守卫规则。有关更多信息，请参阅 [为 Hook 编写防护规则](#)。

- 创建 Hook 的用户或角色必须具有足够的权限才能激活 Hook。
- 要使用 AWS CLI 或创建 Guard Hook，必须手动创建具有 IAM 权限的执行角色和允许 CloudFormation 调用 Guard Hook 的信任策略。SDK

为 Guard Hook 创建执行角色

Hook 使用执行角色来获得在你中调用该 Hook 所需的权限 AWS 账户。

如果您从中创建 Guard Hook，则可以自动创建此角色 AWS Management Console；否则，您必须自己创建此角色。

以下部分向您展示如何设置权限以创建 Guard Hook。

所需的权限

按照 IAM 用户指南中的 [使用自定义信任策略创建角色](#) 中的指导，使用自定义信任策略创建角色。

然后，完成以下步骤来设置您的权限：

1. 将以下最低权限策略附加到要用于创建 Guard Hook 的 IAM 角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3::my-guard-output-bucket/*",
        "arn:aws:s3::my-guard-rules-bucket"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3::my-guard-output-bucket/*"
      ]
    }
  ]
}
```



```
    ]
  }
]
}
```

2. 通过向角色添加信任策略，授予您的 Hook 代入该角色的权限。以下显示了您可以使用的信任策略示例。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "hooks.cloudformation.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

在你的账户中激活 Guard Hook

以下主题向您展示了如何在您的账户中激活 Guard Hook，这样它就可以在激活该挂钩的账户和区域中使用。

主题

- [激活 Guard Hook \(主机版\)](#)
- [激活警戒钩 \(AWS CLI\)](#)
- [相关资源](#)


激活 Guard Hook (主机版)

激活 Guard Hook 以在你的账户中使用

1. 登录 AWS Management Console 并在 <https://console.aws.amazon.com/cloudformation> 上打开 AWS CloudFormation 控制台。

2. 在屏幕顶部的导航栏上，选择要创建 Hook in AWS 区域 的位置。
3. 如果您尚未创建任何防护规则，请创建您的防护规则，将其存储在 Amazon S3 中，然后返回此过程。请参考中的示例规则[编写 Guard 规则来评估防护挂钩的资源](#)开始使用。

如果您已经创建了防护规则并将其存储在 S3 中，请继续执行下一步。

 Note

存储在 S3 中的对象必须具有以下文件扩展名之一：`.guard.zip`、或 `.tar.gz`。

4. 对于 Guard Hook 来源，请将您的防护规则存储在 S3 中，请执行以下操作：
 - 对于 S3 URI，请指定规则文件的 S3 路径，或者使用浏览 S3 按钮打开对话框来浏览和选择 S3 对象。
 - (可选) 对于对象版本，如果您的 S3 存储桶启用了版本控制，则可以选择 S3 对象的特定版本。


每次调用 Hook 时，Guard Hook 都会从 S3 下载您的规则。为防止意外更改或删除，我们建议在配置 Guard Hook 时使用版本。

5. (可选) 对于 S3 存储桶 for Guard 输出报告，请指定一个 S3 存储桶来存储 Guard 输出报告。此报告包含您的 Guard 规则验证结果。

要配置输出报告目标，请选择以下选项之一：

- 选中“使用我的 Guard 规则存储在同一个存储桶”复选框以使用您的 Guard 规则所在的同一个存储桶。
 - 选择不同的 S3 存储桶名称来存储 Guard 输出报告。
6. (可选) 展开防护规则输入参数，然后在“将您的防护规则输入参数存储在 S3 中”下提供以下信息：
 - 对于 S3 URI，请指定参数文件的 S3 路径，或者使用浏览 S3 按钮打开对话框来浏览和选择 S3 对象。
 - (可选) 对于对象版本，如果您的 S3 存储桶启用了版本控制，则可以选择 S3 对象的特定版本。
 7. 选择下一步。
 8. 在 Hook 名称中，选择以下选项之一：

- 提供一个简短的描述性名称，该名称将在之后`Private::Guard::`添加。例如，如果输入 *MyTestHook*，则完整的 Hook 名称变为 `Private::Guard::MyTestHook`。
 - 使用以下格式提供完整的 Hook 名称（也称为别名）：*Provider::ServiceName::HookName*
9. 对于 Hook 目标，请选择要评估的内容：
 - 堆栈-在用户创建、更新或删除堆栈时评估堆栈模板。
 - 资源-在用户更新堆栈时评估各个资源的变化。
 - 变更集-当用户创建变更集时，评估计划的更新。
 - 云控制 API-评估[云控制API](#)启动的创建、更新或删除操作。
 10. 在“操作”中，选择哪些操作（创建、更新、删除）将调用您的 Hook。
 11. 对于 Hook 模式，选择规则评估失败时挂钩的响应方式：
 - 警告-向用户发出警告，但允许继续执行操作。这对于非关键验证或信息检查很有用。
 - 失败-阻止操作继续进行。这有助于执行严格的合规或安全政策。
 12. 对于执行角色，选择 CloudFormation Hook 扮演的角色来从 S3 检索您的警卫规则，并可以选择写回详细的 Guard 输出报告。您可以 CloudFormation 允许自动为您创建执行角色，也可以指定已创建的角色。
 13. 选择下一步。
 14. （可选）对于 Hook 过滤器，请执行以下操作：
 - a. 在资源筛选器中，指定哪些资源类型可以调用 Hook。这样可以确保仅针对相关资源调用 Hook。
 - b. 在筛选条件中，选择应用堆栈名称和堆栈角色筛选器的逻辑：
 - 所有堆栈名称和堆栈角色 — 只有当所有指定的过滤器都匹配时，才会调用 Hook。
 - 任何堆栈名称和堆栈角色 — 如果指定的过滤器中至少有一个匹配，则将调用 Hook。

 Note

对于 Cloud Control API 操作，所有堆栈名称和堆栈角色筛选器都将被忽略。

- c. 对于堆栈名称，在 Hook 调用中包含或排除特定堆栈。

- 对于“包含”，指定要包含的堆栈名称。当你想要瞄准一小部分特定的堆栈时，请使用此选项。只有此列表中指定的堆栈才会调用 Hook。
 - 对于排除，请指定要排除的堆栈名称。当你想在大多数堆栈上调用 Hook 但排除一些特定的堆栈时，请使用此选项。除此列出的堆栈外，所有堆栈都将调用 Hook。
- d. 对于堆栈角色，请根据其关联角色在 Hook 调用中包含或排除特定堆栈。IAM
- 对于 Include，指定一个或多个 IAM 角色 ARNs 来定位与这些角色关联的堆栈。只有由这些角色启动的堆栈操作才会调用 Hook。
 - 在“排除”中，ARNs 为要排除的堆栈指定一个或多个 IAM 角色。Hook 将在除指定角色启动的堆栈之外的所有堆栈上调用。
15. 选择下一步。
 16. 在“查看并激活”页面上，查看您的选择。要进行更改，请在相关部分选择编辑。
 17. 准备好继续操作时，选择“激活挂钩”。

激活警戒钩 (AWS CLI)

在继续操作之前，请确认您已创建守卫规则和将在此 Hook 中使用的执行角色。有关更多信息，请参阅[编写 Guard 规则来评估防护挂钩的资源](#)和[为 Guard Hook 创建执行角色](#)。

激活 Guard Hook 以在你的账户中使用 (AWS CLI)

1. 要开始激活 Hook，请使用以下命令 `activate-type` 命令，用您的特定值替换占位符。此命令授权 Hook 使用您的 AWS 账户指定执行角色。

```
aws cloudformation activate-type --type HOOK \
  --type-name AWS::Hooks::GuardHook \
  --publisher-id aws-hooks \
  --type-name-alias Private::Guard::MyTestHook \
  --execution-role arn:aws:iam::123456789012:role/my-execution-role \
  --region us-west-2
```

2. 要完成激活 Hook，必须使用配置文件对其进行 JSON 配置。

使用 cat 命令创建具有以下结构的 JSON 文件。有关更多信息，请参阅[挂钩配置架构语法参考](#)。

```
$ cat > config.json
{
  "CloudFormationConfiguration": {
```

```

"HookConfiguration": {
  "HookInvocationStatus": "ENABLED",
  "TargetOperations": [
    "STACK",
    "RESOURCE",
    "CHANGE_SET"
  ],
  "FailureMode": "WARN",
  "Properties": {
    "ruleLocation": "s3://amzn-s3-demo-bucket/MyGuardRules.guard",
    "logBucket": "amzn-s3-demo-logging-bucket"
  }
}
}
}
}

```

- `HookInvocationStatus` : 设置ENABLED为可启用挂钩。
 - `TargetOperations` : 指定要对哪些操作运行 Hook。
 - `FailureMode` : 设置为 FAIL 或 WARN。
 - `ruleLocation` : 替换为URI存储规则的 S3。存储在 S3 中的对象必须具有以下文件扩展名之一 : `.guard.zip`、和 `.tar.gz`
 - `logBucket`: (可选) 为 Guard JSON 报告指定 S3 存储桶的名称。
3. 使用以下内容 [set-type-configuration](#)命令以及您创建JSON的文件一起应用配置。用您的特定值替换占位符。

```

aws cloudformation set-type-configuration \
  --configuration file://config.json \
  --type-arn "arn:aws:cloudformation:us-west-2:123456789012:type/hook/MyTestHook" \
  --region us-west-2

```

相关资源

我们提供了模板示例，您可以使用这些示例来了解如何在 CloudFormation 堆栈模板中声明 Guard Hook。有关更多信息，请参阅 [AWS::CloudFormation::GuardHook](#) (在 AWS CloudFormation 用户指南中)。

查看您账户中防御挂钩的日志

激活 Guard Hook 时，您可以将 Amazon S3 存储桶指定为挂钩输出报告的目标。激活后，Hook 会自动将您的 Guard 规则验证结果存储在指定的存储桶中。然后，您可以在 Amazon S3 控制台中查看这些结果。

在 Amazon S3 控制台中查看 Guard Hook 日志

查看 Guard Hook 输出日志文件

1. 登录 <https://console.aws.amazon.com/s3/>
2. 在屏幕顶部的导航栏中，选择您的 AWS 区域。
3. 选择 Buckets。
4. 选择您为 Guard 输出报告选择的存储桶。
5. 选择所需的验证输出报告日志文件。
6. 选择是要下载文件还是打开文件进行查看。

在你的账户中删除防护挂钩

当您不再需要已激活的 Guard Hook 时，请按照以下步骤将其从您的账户中删除。

要暂时禁用 Hook 而不是将其删除，请参阅[禁用和启用 AWS CloudFormation Hook](#)。

主题

- [删除账号中的警卫挂钩 \(主机\)](#)
- [删除你账户中的警报挂钩 \(AWS CLI\)](#)

删除账号中的警卫挂钩 (主机)

删除您账户中的警报挂钩

1. 登录 AWS Management Console 并在 <https://console.aws.amazon.com/cloudformation> 上打开 AWS CloudFormation 控制台。
2. 在屏幕顶部的导航栏上，选择 Hook 所在 AWS 区域 的位置。
3. 从导航窗格中选择 Hooks。

4. 在挂钩页面上，找到要删除的防护挂钩。
5. 选中 Hook 旁边的复选框并选择“删除”。
6. 当系统提示您确认时，键入挂钩名称以确认删除指定的挂钩，然后选择 Delete。

删除你账户中的警报挂钩 (AWS CLI)

Note

在删除挂钩之前，必须先将其禁用。有关更多信息，请参阅 [在您的账户中禁用并启用挂钩 \(AWS CLI\)](#)。

使用以下内容 [deactivate-type](#) 命令停用 Hook，这会将其从您的帐户中删除。用您的特定值替换占位符。

```
aws cloudformation deactivate-type \  
  --type-arn "arn:aws:cloudformation:us-west-2:123456789012:type/hook/MyTestHook" \  
  --region us-west-2
```

Lambda Hooks

要在您的账户中使用 AWS Lambda 挂钩，您必须先为要使用该挂钩的账户和区域激活挂钩。激活 Hook 后，它可以在激活该挂钩的账户和区域的堆栈操作中使用。

激活 Lambda 挂钩时，CloudFormation 会在账户的注册表中为已激活的挂钩创建一个条目作为私有挂钩。这允许您设置 Hook 包含的任何配置属性。配置属性定义了如何为给定 AWS 账户 和区域配置挂钩。

主题

- [AWS CLI 用于使用 Lambda 挂钩的命令](#)
- [创建 Lambda 函数来评估 Lambda 挂钩的资源](#)
- [准备创建 Lambda 挂钩](#)
- [在您的账户中激活 Lambda 挂钩](#)
- [查看您账户中 Lambda 挂钩的日志](#)
- [删除账户中的 Lambda 挂钩](#)

AWS CLI 用于使用 Lambda 挂钩的命令

使用 Lambda 挂钩的 AWS CLI 命令包括：

- [activate-type](#) 启动 Lambda 挂钩的激活过程。
- [set-type-configuration](#) 来指定您账户中的 Hook 的配置数据。
- [list-types](#) 列出你账户中的 Hook。
- [describe-type](#) 返回有关特定 Hook 或特定 Hook 版本的详细信息，包括当前配置数据。
- [deactivate-type](#) 从您的账户中移除之前激活的 Hook。

创建 Lambda 函数来评估 Lambda 挂钩的资源

AWS CloudFormation Lambda Hooks 允许您对自己的自定义代码进行评估 CloudFormation 和 AWS Cloud Control API 操作。您的 Hook 可以阻止操作继续进行，或者向调用者发出警告并允许操作继续进行。创建 Lambda 挂钩时，可以将其配置为拦截和评估以下操作：CloudFormation

- 资源操作
- 堆栈操作
- 更改集合操作

主题

- [开发 Lambda 挂钩](#)
- [使用 Lambda 挂钩评估资源操作](#)
- [使用 Lambda 挂钩评估堆栈操作](#)
- [使用 Lambda 挂钩评估变更集操作](#)

开发 Lambda 挂钩

当 Hook 调用你的 Lambda 时，它将等待长达 30 秒钟让 Lambda 评估输入。Lambda 将返回一个 JSON 响应，指示挂钩是成功还是失败。

主题

- [请求输入](#)
- [响应输入](#)

- [示例](#)

请求输入

传递给 Lambda 函数的输入取决于 Hook 目标操作（例如：堆栈、资源或更改集）。

响应输入

为了在您的请求成功或失败时与 Hook 进行通信，您的 Lambda 函数需要返回响应。JSON

以下是 Hooks 期望的响应的示例形状：

```
{
  "hookStatus": "SUCCESS" or "FAILED" or "IN_PROGRESS",
  "errorCode": None or "NonCompliant" or "InternalFailure"
  "message": String,
  "clientRequestToken": String
  "callbackContext": None,
  "callbackDelaySeconds": Integer,
}
```

hookStatus

挂钩的状态。此字段为必填字段。

有效值：(SUCCESS| FAILED |IN_PROGRESS)

Note

Hook 可以返回 IN_PROGRESS 3 次。如果未返回任何结果，Hook 将失败。对于 Lambda 挂钩，这意味着您的 Lambda 函数最多可以被调用 3 次。

errorCode

显示操作是否经过评估并被确定为无效，或者挂钩内是否发生了错误，从而阻止了评估。如果挂钩失败，则此字段为必填字段。

有效值：(NonCompliant|InternalFailure)

message

给调用者的消息，说明挂钩成功或失败的原因。

Note

计算 CloudFormation 操作时，此字段被截断为 4096 个字符。
在评估云控制API操作时，此字段被截断为 1024 个字符。

clientRequestToken

作为 Hook 请求的输入而提供的请求令牌。此字段为必填字段。

callbackContext

如果您指明hookStatus是，则IN_PROGRESS会传递一个额外的上下文，该上下文在重新调用 Lambda 函数时作为输入提供。

callbackDelaySeconds

Hook 应该等多久才能再次调用这个 Hook。

示例

以下是成功响应的示例：

```
{
  "hookStatus": "SUCCESS",
  "message": "compliant",
  "clientRequestToken": "123avjdjk31"
}
```

以下是响应失败的示例：

```
{
  "hookStatus": "FAILED",
  "errorCode": "NON_COMPLIANT",
  "message": "S3 Bucket Versioning must be enabled.",
  "clientRequestToken": "123avjdjk31"
}
```

使用 Lambda 挂钩评估资源操作

每当您创建、更新或删除资源时，都被视为资源操作。例如，如果您运行更新 CloudFormation 堆栈以创建新资源，则表示您已完成资源操作。当您使用 Cloud Control 创建API、更新或删除资源

时，这也被视为资源操作。您可以将 CloudFormation Lambda 挂钩配置为目标RESOURCE和挂钩TargetOperations配置中的CLOUD_CONTROL操作。

Note

仅当使用cloud-control delete-resource或中的操作触发器删除资源时，才会调用delete Hook 处理程序cloudformation delete-stack。

主题

- [Lambda Hook 资源输入语法](#)
- [Lambda Hook 资源变更输入示例](#)
- [用于资源操作的 Lambda 函数示例](#)

Lambda Hook 资源输入语法

当您的 Lambda 被调用进行资源操作时，您将收到一个JSON输入，其中包含资源属性、建议的属性以及围绕 Hook 调用的上下文。

以下是JSON输入的示例形状：

```
{
  "awsAccountId": String,
  "stackId": String,
  "changeSetId": String,
  "hookTypeName": String,
  "hookTypeVersion": String,
  "hookModel": {
    "LambdaFunction": String
  },
  "actionInvocationPoint": "CREATE_PRE_PROVISION" or "UPDATE_PRE_PROVISION" or
  "DELETE_PRE_PROVISION"
  "requestData": {
    "targetName": String,
    "targetType": String,
    "targetLogicalId": String,
    "targetModel": {
      "resourceProperties": {...},
      "previousResourceProperties": {...}
    }
  }
}
```

```
    },  
    "requestContext": {  
      "##": 1,  
      "callbackContext": null  
    }  
  }  
}
```

awsAccountId

AWS 账户 包含正在评估的资源的 ID。

stackId

此操作所属 CloudFormation 堆栈的堆栈 ID。如果来电者是云控制，则此字段为空API。

changeSetId

启动 Hook 调用的更改集的 ID。如果资源变更是由 Cloud Control 或API、或delete-stack操作启动的 create-stackupdate-stack，则此值为空。

hookTypeName

正在运行的 Hook 的名称。

hookTypeVersion

正在运行的 Hook 的版本。

hookModel

LambdaFunction

挂钩ARN调用的当前 Lambda。

actionInvocationPoint

配置逻辑中挂钩运行的确切位置。

有效值：(CREATE_PRE_PROVISION| UPDATE_PRE_PROVISION |DELETE_PRE_PROVISION)

requestData

targetName

正在创建的目标资源的名称。

targetType

例如，正在创建的目标类型AWS::S3::Bucket。

targetLogicalId

正在评估的资源的逻辑 ID。如果 Hook 调用的来源是 CloudFormation，则这将是 CloudFormation 模板中定义的逻辑资源 ID。如果此 Hook 调用的来源是 Cloud ControlAPI，则这将是构造值。

targetModel

resourceProperties

正在修改的资源的建议属性。如果要删除资源，则此值将为空。

previousResourceProperties

当前与正在修改的资源关联的属性。如果正在创建资源，则此值将为空。

requestContext

调用

当前执行挂钩的尝试。

callbackContext

如果 Hook 设置为 IN_PROGRESS，并且 callbackContext 已返回，则它将在重新调用后出现在这里。

Lambda Hook 资源变更输入示例

在以下示例输入中，Guard Hook 将收到正在更改的 `AWS::DynamoDB::Table` 资源的定义。ProvisionedThroughput 和 ReadCapacityUnits 参数正在从 3 更新到 10。

有关资源可用属性的更多信息，请参阅 [AWS::DynamoDB::Table](#)。

```
{
  "awsAccountId": "123456789",
  "stackId": "arn:aws:cloudformation:eu-central-1:123456789:stack/test-stack/123456abcd",
  "hookTypeName": "my::lambda::resourcehookfunction",
  "hookTypeVersion": "00000008",
  "hookModel": {
    "LambdaFunction": "arn:aws:lambda:eu-central-1:123456789:function:resourcehookfunction"
  },
  "actionInvocationPoint": "UPDATE_PRE_PROVISION",
  "requestData": {
```

```
"targetName": "AWS::DynamoDB::Table",
"targetType": "AWS::DynamoDB::Table",
"targetLogicalId": "DDBTable",
"targetModel": {
  "resourceProperties": {
    "AttributeDefinitions": [
      {
        "AttributeType": "S",
        "AttributeName": "Album"
      },
      {
        "AttributeType": "S",
        "AttributeName": "Artist"
      }
    ],
    "ProvisionedThroughput": {
      "WriteCapacityUnits": 5,
      "ReadCapacityUnits": 10
    },
    "KeySchema": [
      {
        "KeyType": "HASH",
        "AttributeName": "Album"
      },
      {
        "KeyType": "RANGE",
        "AttributeName": "Artist"
      }
    ]
  },
  "previousResourceProperties": {
    "AttributeDefinitions": [
      {
        "AttributeType": "S",
        "AttributeName": "Album"
      },
      {
        "AttributeType": "S",
        "AttributeName": "Artist"
      }
    ],
    "ProvisionedThroughput": {
      "WriteCapacityUnits": 5,
      "ReadCapacityUnits": 5
    }
  }
}
```

```

    },
    "KeySchema": [
      {
        "KeyType": "HASH",
        "AttributeName": "Album"
      },
      {
        "KeyType": "RANGE",
        "AttributeName": "Artist"
      }
    ]
  }
},
"requestContext": {
  "invocation": 1,
  "callbackContext": null
}
}

```

用于资源操作的 Lambda 函数示例

以下 Lambda 挂钩的目标示例。Node.js 这是一个简单的函数，它会导致对 DynamoDB 的任何资源更新失败，DynamoDB 会尝试将设置为大于 ProvisionedThroughput ReadCapacity 10 的值。如果挂接成功，则将向呼叫者显示消息“ReadCapacity 配置正确”。如果请求验证失败，则挂钩将失败，状态为“ReadCapacity 不能超过 10”。

```

export const handler = async (event, context) => {
  var targetModel = event?.requestData?.targetModel;
  var targetName = event?.requestData?.targetName;
  var response = {
    "hookStatus": "SUCCESS",
    "message": "ReadCapacity is correctly configured.",
    "clientRequestToken": event.clientRequestToken
  };

  if (targetName == "AWS::DynamoDB::Table") {
    var readCapacity =
targetModel?.resourceProperties?.ProvisionedThroughput?.ReadCapacityUnits;
    if (readCapacity > 10) {
      response.hookStatus = "FAILED";
      response.errorCode = "NonCompliant";
      response.message = "ReadCapacity must be cannot be more than 10.";
    }
  }
}

```

```
    }  
  }  
  return response;  
};
```

使用 Lambda 挂钩评估堆栈操作

每当您使用新模板创建、更新或删除堆栈时，都可以将您的 CloudFormation Lambda Hook 配置为从评估新模板开始，并可能阻止堆栈操作继续进行。您可以将 CloudFormation Lambda 挂钩配置为挂钩配置中的STACK操作目标。TargetOperations

主题

- [Lambda Hook 堆栈输入语法](#)
- [Lambda Hook 堆栈更改输入示例](#)
- [用于堆栈操作的 Lambda 函数示例](#)

Lambda Hook 堆栈输入语法

当您的 Lambda 被调用以执行堆栈操作时，您将收到一个包含 Hook 调用上下文和JSON请求上下文的请求。actionInvocationPoint由于 CloudFormation 模板的大小以及 Lambda 函数接受的输入大小有限，因此实际模板存储在 Amazon S3 对象中。的输入requestData包括已分配URL给另一个对象的 Amazon S3，该对象包含当前和以前的模板版本。

以下是JSON输入的示例形状：

```
{  
  "clientRequestToken": String,  
  "awsAccountId": String,  
  "stackID": String,  
  "changeSetId": String,  
  "hookTypeName": String,  
  "hookTypeVersion": String,  
  "hookModel": {  
    "LambdaFunction":String  
  },  
  "actionInvocationPoint": "CREATE_PRE_PROVISION" or "UPDATE_PRE_PROVISION" or  
  "DELETE_PRE_PROVISION"  
  "requestData": {  
    "targetName": "STACK",  
    "targetType": "STACK",  
    "targetLogicalId": String,  
  }  
}
```



```

    "payload": String (S3 Presigned URL)
  },
  "requestContext": {
    "invocation": Integer,
    "callbackContext": String
  }
}

```

clientRequestToken

作为 Hook 请求的输入而提供的请求令牌。此字段为必填字段。

awsAccountId

AWS 账户 包含正在评估的堆栈的 ID。

stackID

堆栈的堆 CloudFormation 栈 ID。

changeSetId

启动 Hook 调用的更改集的 ID。如果堆栈更改由 Cloud Control 或 API、或 delete-stack 操作启动 create-stackupdate-stack，则此值为空。

hookTypeName

正在运行的 Hook 的名称。

hookTypeVersion

正在运行的 Hook 的版本。

hookModel

LambdaFunction

挂钩ARN调用的当前 Lambda。

actionInvocationPoint

配置逻辑中挂钩运行的确切位置。

有效值：(CREATE_PRE_PROVISION| UPDATE_PRE_PROVISION |DELETE_PRE_PROVISION)

requestData

targetName

这个值将是 STACK。

targetType

这个值将是 STACK。

targetLogicalId

堆栈名称。

payload

预签名的 Amazon S3，URL 其中包含具有当前和先前模板定义的JSON对象。

requestContext

如果正在重新调用 Hook，则将设置此对象。

invocation

当前执行挂钩的尝试。

callbackContext

如果 Hook 设置为 IN_PROGRESS 并 callbackContext 已返回，则它将在重新调用时出现在这里。

请求数据中的 payload 属性 URL 是您的代码需要获取的。一旦它收到了 URL，你就会得到一个具有以下架构的对象：

```
{
  "template": String,
  "previousTemplate": String
}
```

template

提供给 create-stack 或的完整 CloudFormation 模板 update-stack。它可以是 JSON 或 YAML 字符串，具体取决于提供的内容 CloudFormation。

在 delete-stack 操作中，此值将为空。

previousTemplate

之前的 CloudFormation 模板。它可以是 JSON 或 YAML 字符串，具体取决于提供的内容 CloudFormation。

在delete-stack操作中，此值将为空。

Lambda Hook 堆栈更改输入示例

以下是堆栈变更输入的示例。Hook 正在评估一项更改，该更改ObjectLockEnabled将更新为 true，并添加了一个 Amazon SQS 队列：

```
{
  "clientRequestToken": "f8da6d11-b23f-48f4-814c-0fb6a667f50e",
  "awsAccountId": "123456789",
  "stackId": "arn:aws:cloudformation:eu-central-1:123456789:stack/david-ddb-test-stack/400b40f0-8e72-11ef-80ab-02f2902f0df1",
  "changeSetId": null,
  "hookTypeName": "my::lambda::stackhook",
  "hookTypeVersion": "00000008",
  "hookModel": {
    "LambdaFunction": "arn:aws:lambda:eu-central-1:123456789:function:stackhookfunction"
  },
  "actionInvocationPoint": "UPDATE_PRE_PROVISION",
  "requestData": {
    "targetName": "STACK",
    "targetType": "STACK",
    "targetLogicalId": "my-cloudformation-stack",
    "payload": "https://s3....."
  },
  "requestContext": {
    "invocation": 1,
    "callbackContext": null
  }
}
```

以下是以下payload示例requestData：

```
{
  "template": "{\"Resources\":{\"S3Bucket\":{\"Type\":\"AWS::S3::Bucket\", \"Properties\":{\"ObjectLockEnabled\":true}},\"SQSQueue\":{\"Type\":\"AWS::SQS::Queue\", \"Properties\":{\"QueueName\":\"NewQueue\"}}}}",
  "previousTemplate": "{\"Resources\":{\"S3Bucket\":{\"Type\":\"AWS::S3::Bucket\", \"Properties\":{\"ObjectLockEnabled\":false}}}}"
```

用于堆栈操作的 Lambda 函数示例

以下 Lambda Hook 的目标示例。Node.js 这是一个简单的函数，可以下载堆栈操作有效负载，解析模板JSON并返回SUCCESS。

```
export const handler = async (event, context) => {
  var targetType = event?.requestData?.targetType;
  var payloadUrl = event?.requestData?.payload;

  var response = {
    "hookStatus": "SUCCESS",
    "message": "Stack update is compliant",
    "clientRequestToken": event.clientRequestToken
  };
  try {
    const templateHookPayloadRequest = await fetch(payloadUrl);
    const templateHookPayload = await templateHookPayloadRequest.json()
    if (templateHookPayload.template) {
      // Do something with the template templateHookPayload.template
      // JSON or YAML
    }
    if (templateHookPayload.previousTemplate) {
      // Do something with the template templateHookPayload.previousTemplate
      // JSON or YAML
    }
  } catch (error) {
    console.log(error);
    response.hookStatus = "FAILED";
    response.message = "Failed to evaluate stack operation.";
    response.errorCode = "InternalFailure";
  }
  return response;
};
```

使用 Lambda 挂钩评估变更集操作

每当您创建更改集时，都可以将您的 CloudFormation Lambda Hook 配置为首先评估新的更改集并可能阻止其执行。您可以将 CloudFormation Lambda 挂钩配置为挂钩配置中的CHANGE_SET操作目标。TargetOperations

主题

- [Lambda Hook 更改集输入语法](#)

- [示例 Lambda 挂钩更改集更改输入](#)
- [用于变更集操作的 Lambda 函数示例](#)

Lambda Hook 更改集输入语法

变更集操作的输入与堆栈操作类似，但是的有效载荷requestData还包括变更集引入的资源变更列表。

以下是JSON输入的示例形状：

```
{
  "clientRequesttoken": String,
  "awsAccountId": String,
  "stackID": String,
  "changeSetId": String,
  "hookTypeName": String,
  "hookTypeVersion": String,
  "hookModel": {
    "LambdaFunction": String
  },
  "requestData": {
    "targetName": "CHANGE_SET",
    "targetType": "CHANGE_SET",
    "targetLogicalId": String,
    "payload": String (S3 Presigned URL)
  },
  "requestContext": {
    "invocation": Integer,
    "callbackContext": String
  }
}
```

clientRequesttoken

作为 Hook 请求的输入而提供的请求令牌。此字段为必填字段。

awsAccountId

AWS 账户 包含正在评估的堆栈的 ID。

stackID

堆栈的堆 CloudFormation 栈 ID。

changeSetId

启动 Hook 调用的更改集的 ID。

hookTypeName

正在运行的 Hook 的名称。

hookTypeVersion

正在运行的 Hook 的版本。

hookModel

LambdaFunction

挂钩ARN调用的当前 Lambda。

requestData

targetName

这个值将是 CHANGE_SET。

targetType

这个值将是 CHANGE_SET。

targetLogicalId

变更集ARN...

payload

预签名的 Amazon S3，URL其中包含使用当前模板的JSON对象，以及此更改集引入的更改列表。

requestContext

如果正在重新调用 Hook，则将设置此对象。

invocation

当前执行挂钩的尝试。

callbackContext

如果 Hook 设置为 IN_PROGRESS 并 callbackContext 已返回，则它将在重新调用时出现在这里。

请求数据中的payload属性URL是您的代码需要获取的。一旦它收到了URL，你就会得到一个具有以下架构的对象：

```
{
  "template": String,
  "changedResources": [
    {
      "action": String,
      "beforeContext": JSON String,
      "afterContext": JSON String,
      "lineNumber": Integer,
      "logicalResourceId": String,
      "resourceType": String
    }
  ]
}
```

template

提供给create-stack或的完整 CloudFormation 模板update-stack。它可以是JSON或YAML字符串，具体取决于提供的内容 CloudFormation。

changedResources

已更改的资源列表。

action

应用于资源的更改类型。

有效值：(CREATE| UPDATE |DELETE)

beforeContext

更改前的资源属性的JSON字符串。创建资源时，此值为空。此JSON字符串中的所有布尔值和数字值均为STRINGS。

afterContext

如果执行了此更改集，则为资源属性的JSON字符串。删除资源时，此值为空。此JSON字符串中的所有布尔值和数字值均为STRINGS。

lineNumber

模板中导致此更改的行号。如果操作是，则DELETE此值将为空。

logicalResourceId

正在更改的资源的逻辑资源 ID。

resourceType

正在更改的资源类型。

示例 Lambda 挂钩更改集更改输入

以下是更改集更改输入的示例。在以下示例中，您可以看到变更集引入的更改。第一个更改是删除名为的队列CoolQueue。第二个更改是添加一个名为的新队列NewCoolQueue。最后一项更改是对的新DynamoDBTable。

```
{
  "clientRequestToken": "f8da6d11-b23f-48f4-814c-0fb6a667f50e",
  "awsAccountId": "123456789",
  "stackId": "arn:aws:cloudformation:eu-central-1:123456789:stack/david-ddb-test-stack/400b40f0-8e72-11ef-80ab-02f2902f0df1",
  "changeSetId": "arn:aws:cloudformation:eu-central-1:123456789:changeSet/davids-change-set/59ebd63c-7c89-4771-a576-74c3047c15c6",
  "hookTypeName": "my::lambda::changesethook",
  "hookTypeVersion": "00000008",
  "hookModel": {
    "LambdaFunction": "arn:aws:lambda:eu-central-1:123456789:function:changesethookfunction"
  },
  "actionInvocationPoint": "CREATE_PRE_PROVISION",
  "requestData": {
    "targetName": "CHANGE_SET",
    "targetType": "CHANGE_SET",
    "targetLogicalId": "arn:aws:cloudformation:eu-central-1:123456789:changeSet/davids-change-set/59ebd63c-7c89-4771-a576-74c3047c15c6",
    "payload": "https://s3....."
  },
  "requestContext": {
    "invocation": 1,
    "callbackContext": null
  }
}
```

以下是以下payload示例requestData.payload：


```

{
  template: 'Resources:\n' +
    '  DynamoDBTable:\n' +
    '    Type: AWS::DynamoDB::Table\n' +
    '    Properties:\n' +
    '      AttributeDefinitions:\n' +
    '        - AttributeName: "PK"\n' +
    '          AttributeType: "S"\n' +
    '      BillingMode: "PAY_PER_REQUEST"\n' +
    '      KeySchema:\n' +
    '        - AttributeName: "PK"\n' +
    '          KeyType: "HASH"\n' +
    '      PointInTimeRecoverySpecification:\n' +
    '        PointInTimeRecoveryEnabled: false\n' +
    '    NewSQSQueue:\n' +
    '      Type: AWS::SQS::Queue\n' +
    '      Properties:\n' +
    '        QueueName: "NewCoolQueue"',
  changedResources: [
    {
      logicalResourceId: 'SQSQueue',
      resourceType: 'AWS::SQS::Queue',
      action: 'DELETE',
      lineNumber: null,
      beforeContext: '{"Properties":{"QueueName":"CoolQueue"}}',
      afterContext: null
    },
    {
      logicalResourceId: 'NewSQSQueue',
      resourceType: 'AWS::SQS::Queue',
      action: 'CREATE',
      lineNumber: 14,
      beforeContext: null,
      afterContext: '{"Properties":{"QueueName":"NewCoolQueue"}}'
    },
    {
      logicalResourceId: 'DynamoDBTable',
      resourceType: 'AWS::DynamoDB::Table',
      action: 'UPDATE',
      lineNumber: 2,
      beforeContext: '{"Properties":
{"BillingMode":"PAY_PER_REQUEST","AttributeDefinitions":

```

```
[{"AttributeType":"S","AttributeName":"PK"}],"KeySchema":
[{"KeyType":"HASH","AttributeName":"PK"}]}' ,
  afterContext: '{"Properties":
{"BillingMode":"PAY_PER_REQUEST","PointInTimeRecoverySpecification":
{"PointInTimeRecoveryEnabled":"false"},"AttributeDefinitions":
[{"AttributeType":"S","AttributeName":"PK"}],"KeySchema":
[{"KeyType":"HASH","AttributeName":"PK"}]}'
  }
]
}
```

用于变更集操作的 Lambda 函数示例

以下 Lambda Hook 的目标示例。Node.js 这是一个简单的函数，它下载变更集操作有效负载，循环浏览每个更改，然后在返回 a 之前打印出之前和之后的属性SUCCESS。

```
export const handler = async (event, context) => {
  var payloadUrl = event?.requestData?.payload;
  var response = {
    "hookStatus": "SUCCESS",
    "message": "Change set changes are compliant",
    "clientRequestToken": event.clientRequestToken
  };
  try {
    const changeSetHookPayloadRequest = await fetch(payloadUrl);
    const changeSetHookPayload = await changeSetHookPayloadRequest.json();
    const changes = changeSetHookPayload.changedResources || [];
    for(const change of changes) {
      var beforeContext = {};
      var afterContext = {};
      if(change.beforeContext) {
        beforeContext = JSON.parse(change.beforeContext);
      }
      if(change.afterContext) {
        afterContext = JSON.parse(change.afterContext);
      }
      console.log(beforeContext)
      console.log(afterContext)
      // Evaluate Change here
    }
  } catch (error) {
    console.log(error);
    response.hookStatus = "FAILED";
  }
}
```

```
        response.message = "Failed to evaluate change set operation.";
        response.errorCode = "InternalFailure";
    }
    return response;
};
```

准备创建 Lambda 挂钩

在创建 Lambda 挂钩之前，必须完成以下先决条件：

- 您必须已经创建了一个 Lambda 函数。有关更多信息，请参阅 [为挂钩创建 Lambda 函数](#)。
- 创建 Hook 的用户或角色必须具有足够的权限才能激活 Hook。
- 要使用 AWS CLI 或创建 Lambda 挂钩，您必须手动创建具有 IAM 权限的执行角色和允许 CloudFormation 调用 Lambda 挂钩的信任策略。SDK

为 Lambda 挂钩创建执行角色

Hook 使用执行角色来获得在你中调用该 Hook 所需的权限 AWS 账户。

如果您从中创建 Lambda Hook，则可以自动创建此角色 AWS Management Console；否则，您必须自己创建此角色。

以下部分向您展示如何设置创建 Lambda 挂钩的权限。

所需的权限

按照 IAM 用户指南中的 [使用自定义信任策略创建角色](#) 中的指导，使用自定义信任策略创建角色。

然后，完成以下步骤来设置您的权限：

1. 将以下最低权限策略附加到要用于创建 Lambda Hook 的 IAM 角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function-  
name"
    }
  ]
}
```

```
}
```

2. 通过向角色添加信任策略，授予您的 Hook 代入该角色的权限。以下显示了您可以使用的信任策略示例。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "hooks.cloudformation.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

在您的账户中激活 Lambda 挂钩

以下主题向您展示了如何在您的账户中激活 Lambda Hook，从而使其在激活该挂钩的账户和区域中可用。

主题

- [激活 Lambda 挂钩 \(控制台\)](#)
- [激活 Lambda 挂钩 \(AWS CLI\)](#)
- [相关资源](#)

激活 Lambda 挂钩 (控制台)

激活 Lambda 挂钩以在您的账户中使用


1. 登录 AWS Management Console 并在 <https://console.aws.amazon.com/cloudformation> 上打开 AWS CloudFormation 控制台。
2. 在屏幕顶部的导航栏上，选择要创建 Hook in AWS 区域的位置。
3. 如果您尚未为挂钩创建 Lambda 函数，请执行以下操作：

- 打开 Lambda 控制台的 [Functions](#) (函数) 页面。
- 创建您将用于此 Hook 的 Lambda 函数，然后返回此过程。有关更多信息，请参阅 [创建 Lambda 函数来评估 Lambda 挂钩的资源](#)。

如果您已经创建了 Lambda 函数，请继续执行下一步。

4. 在 Hook 名称中，选择以下选项之一：
 - 提供一个简短的描述性名称，该名称将在之后 `Private::Lambda::` 添加。例如，如果输入 `MyTestHook`，则完整的 Hook 名称变为 `Private::Lambda::MyTestHook`。
 - 使用以下格式提供完整的 Hook 名称 (也称为别名) : `Provider::ServiceName::HookName`
5. 对于 Lambda 函数，请提供用于此挂钩的 Lambda 函数。您可以使用：
 - 不带后缀的完整亚马逊资源名称 (ARN)。
 - ARN 带有版本或别名后缀的限定。
6. 对于 Hook 目标，请选择要评估的内容：
 - 堆栈-在用户创建、更新或删除堆栈时评估堆栈模板。
 - 资源-在用户更新堆栈时评估各个资源的变化。
 - 更改集-在用户创建更改集时评估计划的更新。
 - 云控制 API-评估 [云控制API](#) 启动的创建、更新或删除操作。
7. 在“操作”中，选择哪些操作 (创建、更新、删除) 将调用您的 Hook。
8. 对于挂钩模式，选择挂钩调用的 Lambda 函数返回响应时挂钩的响应方式：FAILED
 - 警告-向用户发出警告，但允许继续执行操作。这对于非关键验证或信息检查很有用。
 - 失败-阻止操作继续进行。这有助于执行严格的合规或安全政策。
9. 对于执行角色，选择挂钩在调用您的 Lambda 函数时 IAM 扮演的角色。您可以 CloudFormation 允许自动为您创建执行角色，也可以指定已创建的角色。
10. 选择下一步。
11. (可选) 对于 Hook 过滤器，请执行以下操作：
 - a. 在资源筛选器中，指定哪些资源类型可以调用 Hook。这样可以确保仅针对相关资源调用 Hook。
 - b. 在筛选条件中，选择应用堆栈名称和堆栈角色筛选器的逻辑：

- 所有堆栈名称和堆栈角色 — 只有当所有指定的过滤器都匹配时，才会调用 Hook。
- 任何堆栈名称和堆栈角色 — 如果指定的过滤器中至少有一个匹配，则将调用 Hook。

 Note

对于 Cloud Control API 操作，所有堆栈名称和堆栈角色筛选器都将被忽略。

- c. 对于堆栈名称，在 Hook 调用中包含或排除特定堆栈。
- 对于“包含”，指定要包含的堆栈名称。当你想要瞄准一小部分特定的堆栈时，请使用此选项。只有此列表中指定的堆栈才会调用 Hook。
 - 对于排除，请指定要排除的堆栈名称。当你想在大多数堆栈上调用 Hook 但排除一些特定的堆栈时，请使用此选项。除此外列出的堆栈外，所有堆栈都将调用 Hook。
- d. 对于堆栈角色，请根据其关联角色在 Hook 调用中包含或排除特定堆栈。IAM
- 对于 Include，指定一个或多个 IAM 角色 ARNs 来定位与这些角色关联的堆栈。只有由这些角色启动的堆栈操作才会调用 Hook。
 - 在“排除”中，ARNs 为要排除的堆栈指定一个或多个 IAM 角色。Hook 将在除指定角色启动的堆栈之外的所有堆栈上调用。

12. 选择下一步。

13. 在“查看并激活”页面上，查看您的选择。要进行更改，请在相关部分选择编辑。

14. 准备好继续操作时，选择“激活挂钩”。

激活 Lambda 挂钩 (AWS CLI)

在继续操作之前，请确认您已创建 Lambda 函数和将用于此 Hook 的执行角色。有关更多信息，请参阅[创建 Lambda 函数来评估 Lambda 挂钩的资源](#)和[为 Lambda 挂钩创建执行角色](#)。

激活 Lambda 挂钩以在您的账户中使用 (AWS CLI)

1. 要开始激活 Hook，请使用以下命令 `activate-type` 命令，用您的特定值替换占位符。此命令授权 Hook 使用您的 AWS 账户指定执行角色。

```
aws cloudformation activate-type --type HOOK \
  --type-name AWS::Hooks::LambdaHook \
  --publisher-id aws-hooks \
```

```
--execution-role arn:aws:iam::123456789012:role/my-execution-role \  
--type-name-alias Private::Lambda::MyTestHook \  
--region us-west-2
```

2. 要完成激活 Hook，必须使用配置文件对其进行JSON配置。

使用cat命令创建具有以下结构的JSON文件。有关更多信息，请参阅 [挂钩配置架构语法参考](#)。

```
$ cat > config.json  
{  
  "CloudFormationConfiguration": {  
    "HookConfiguration": {  
      "HookInvocationStatus": "ENABLED",  
      "TargetOperations": [  
        "CLOUD_CONTROL"  
      ],  
      "FailureMode": "WARN",  
      "Properties": {  
        "LambdaFunction": "arn:aws:lambda:us-  
west-2:123456789012:function:MyFunction"  
      }  
    }  
  }  
}
```

- HookInvocationStatus：设置ENABLED为可启用挂钩。
 - TargetOperations：指定要对哪些操作运行 Hook。
 - FailureMode：设置为 FAIL 或 WARN。
 - LambdaFunction：指定 Lambda 函数的 ARN。
3. 使用以下内容 [set-type-configuration](#) 命令以及您创建JSON的文件一起应用配置。用您的特定值替换占位符。

```
aws cloudformation set-type-configuration \  
--configuration file://config.json \  
--type-arn "arn:aws:cloudformation:us-west-2:123456789012:type/hook/MyTestHook" \  
--region us-west-2
```

相关资源

我们提供了模板示例，您可以使用这些示例来了解如何在 CloudFormation 堆栈模板中声明 Lambda Hook。有关更多信息，请参阅 [AWS::CloudFormation::LambdaHook](#)（在 AWS CloudFormation 用户指南中）。

查看您账户中 Lambda 挂钩的日志

使用 Lambda 挂钩时，可以在 Lambda 控制台中找到您的验证输出报告日志文件。

在 Lambda 控制台中查看 Lambda 挂钩日志

查看 Lambda 挂钩输出日志文件

1. 登录 Lambda 控制台。
2. 在屏幕顶部的导航栏中，选择您的 AWS 区域。
3. 选择函数。
4. 选择所需的 Lambda 函数。
5. 选择测试选项卡。
6. 选择 L CloudWatch ogs 实时跟踪
7. 选择下拉菜单并选择要查看的日志组。
8. 选择启动。日志将显示在 Log CloudWatch s Live Trail 窗口中。根据您的喜好，选择“按列查看”或“以纯文本形式查看”。
 - 您可以通过在“添加筛选器模式”字段中添加更多筛选器来向结果中添加更多过滤器。此字段允许您筛选结果，使其仅包含与指定模式匹配的事件。

有关查看 Lambda 函数日志的更多信息，请参阅[查看 Lambda 函数的 CloudWatch 日志](#)。

删除账户中的 Lambda 挂钩

当您不再需要已激活的 Lambda 挂钩时，请使用以下步骤将其从您的账户中删除。

要暂时禁用 Hook 而不是将其删除，请参阅[禁用和启用 AWS CloudFormation Hook](#)。

主题

- [删除您账户中的 Lambda 挂钩（控制台）](#)

- [删除您账户中的 Lambda 挂钩 \(AWS CLI\)](#)

删除您账户中的 Lambda 挂钩 (控制台)

删除账户中的 Lambda 挂钩

1. 登录 AWS Management Console 并在 <https://console.aws.amazon.com/cloudformation> 上打开 AWS CloudFormation 控制台。
2. 在屏幕顶部的导航栏上，选择 Hook 所在 AWS 区域 的位置。
3. 从导航窗格中选择 Hooks。
4. 在挂钩页面上，找到要删除的 Lambda 挂钩。
5. 选中 Hook 旁边的复选框并选择“删除”。
6. 当系统提示您确认时，键入挂钩名称以确认删除指定的挂钩，然后选择 Delete。

删除您账户中的 Lambda 挂钩 (AWS CLI)

Note

在删除挂钩之前，必须先将其禁用。有关更多信息，请参阅 [在您的账户中禁用并启用挂钩 \(AWS CLI\)](#)。

使用以下内容 [deactivate-type](#) 命令停用 Hook，这会将其从您的帐户中删除。用您的特定值替换占位符。

```
aws cloudformation deactivate-type \  
  --type-arn "arn:aws:cloudformation:us-west-2:123456789012:type/hook/MyTestHook" \  
  --region us-west-2
```

使用开发自定义 Hook CloudFormation CLI

本节适用于想要开发自定义 Hook 并将其注册到 AWS CloudFormation 注册表的客户。

开发自定义 Hook 有三个主要步骤：

1. 启动

要开发自定义 Hook，必须配置并使用 CloudFormation CLI。要启动 Hook 的项目及其必需的文件，请使用 CloudFormation CLI [init](#) 命令并指定要创建挂钩。有关更多信息，请参阅 [启动自定义 AWS CloudFormation Hooks 项目](#)。

2. 模型

要建模、创作和验证您的 Hook 架构，请定义 Hook 及其属性和属性。

会 CloudFormation CLI 创建与特定的 Hook 调用点相对应的空处理函数。将您自己的逻辑添加到这些处理程序中，以控制在目标生命周期的每个阶段调用 Hook 期间发生的情况。有关更多信息，请参阅 [建模自定义 AWS CloudFormation 挂钩](#)。

3. 注册

要注册 Hook，请提交您的 Hook 以注册为私有或公共第三方扩展。在 [submit](#) 操作中注册您的 Hook。有关更多信息，请参阅 [向注册自定义 Hook AWS CloudFormation](#)。

以下任务与注册 Hook 有关：

- a. 发布 — 挂钩发布到注册表。
- b. 配置 — 在对堆栈调用类型配置时配置挂钩。

Note

Hooks 在 30 秒后超时。

以下主题将指导您完成使用 Python 或 Java 开发、注册和发布自定义 Hook 的过程。

主题

- [开发自定义 AWS CloudFormation Hook 的先决条件](#)
- [启动自定义 AWS CloudFormation Hooks 项目](#)
- [建模自定义 AWS CloudFormation 挂钩](#)
- [向注册自定义 Hook AWS CloudFormation](#)
- [在你的 Hook 中测试自定义 Hook AWS 账户](#)
- [更新自定义 Hook](#)
- [从注册表中取消注册自定义 Hook CloudFormation](#)
- [发布 Hook 供公众使用](#)

- [H AWS CloudFormation hooks 的架构语法参考](#)

开发自定义 AWS CloudFormation Hook 的先决条件

你可以用 Java 或 Python 开发一个自定义 Hook。以下是开发自定义 Hook 的先决条件：

Java 先决条件

- [Apache Maven](#)
- [JDK17](#)

Note

如果你打算使用[CloudFormation 命令行界面 \(CLI\)](#) 启动 Java 的 Hooks 项目，则还必须安装 Python 3.8 或更高版本。的 Java 插件 CloudFormation CLI 可以通过 pip (Python 的包管理器) 进行安装，该插件由 Python 分发。

要为您的 Java Hooks 项目实现 Hook 处理程序，您可以下载 [Java Hook 处理程序示例文件](#)。

Python 先决条件

- [Python 版本 3.8](#) 或更高版本。

要为你的 Python Hooks 项目实现钩子处理程序，你可以下载 [Python Hook 处理程序示例文件](#)。

开发 Hook 的权限

除了 CloudFormation CreateUpdate、和Delete堆栈权限外，您还需要访问以下 AWS CloudFormation 操作。对这些操作的访问权限通过您的IAM角色 CloudFormation策略进行管理。

- [register-type](#)
- [list-types](#)
- [deregister-type](#)
- [set-type-configuration](#)

为 Hooks 设置开发环境

要开发 Hook，你应该熟悉[CloudFormation 模板](#)以及 Python 或 Java。

要安装和相关的插件 CloudFormation CLI，请执行以下操作：

1. 使用 Python 包管理器安装。CloudFormation CLI pip

```
pip3 install cloudformation-cli
```

2. 安装适用的 Python 或 Java 插件 CloudFormation CLI。

Python

```
pip3 install cloudformation-cli-python-plugin
```

Java

```
pip3 install cloudformation-cli-java-plugin
```

要升级 CloudFormation CLI和插件，您可以使用升级选项。

Python

```
pip3 install --upgrade cloudformation-cli cloudformation-cli-python-plugin
```

Java

```
pip3 install --upgrade cloudformation-cli cloudformation-cli-java-plugin
```

启动自定义 AWS CloudFormation Hooks 项目

创建自定义 Hooks 项目的第一步是启动该项目。您可以使用 CloudFormation CLI `init` 命令启动您的自定义 Hooks 项目。

该 `init` 命令将启动一个向导，引导您完成项目的设置，包括 Hooks 架构文件。使用此架构文件作为起点来定义 Hook 的形状和语义。有关更多信息，请参阅 [架构语法](#)。

要启动 Hook 项目，请执行以下操作：

1. 为项目创建目录。

```
mkdir ~/mycompany-testing-mytesthook
```

2. 导航到新目录。

```
cd ~/mycompany-testing-mytesthook
```

3. 使用 CloudFormation CLInit 命令启动项目。

```
cfn init
```

该命令将返回以下输出。

```
Initializing new project
```

4. 该init命令将启动一个向导，引导您完成项目的设置。出现提示时，输入h以指定 Hooks 项目。

```
Do you want to develop a new resource(r) a module(m) or a hook(h)?
```

```
h
```

5. 输入挂钩类型的名称。

```
What's the name of your hook type?  
(Organization::Service::Hook)
```

```
MyCompany::Testing::MyTestHook
```

6. 如果只安装了一个语言插件，则默认选择该插件。如果安装了多个语言插件，则可以选择所需的语言。输入您选择的语言的数字选择。

```
Select a language for code generation:  
[1] java  
[2] python38  
[3] python39  
(enter an integer):
```

7. 根据所选的开发语言设置打包。

Python

(可选) 选择 Docker 进行独立于平台的打包。虽然不需要 Docker , 但强烈建议让打包变得更简单。

```
Use docker for platform-independent packaging (Y/n)?
```

```
This is highly recommended unless you are experienced with cross-platform Python packaging.
```

Java

设置 Java 包名称并选择代码生成模型。您可以使用默认的软件包名称 , 也可以创建一个新的软件包名称。

```
Enter a package name (empty for default 'com.mycompany.testing.mytesthook'):
```

```
Choose codegen model - 1 (default) or 2 (guided-aws):
```

结果 : 您已成功启动项目并生成了开发 Hook 所需的文件。以下是构成 Python 3.8 的 Hooks 项目的目录和文件的示例。

```
mycompany-testing-mytesthook.json
rpdk.log
README.md
requirements.txt
hook-role.yaml
template.yml
docs
  README.md
src
  __init__.py
  handlers.py
  models.py
target_models
  aws_s3_bucket.py
```

Note

src目录中的文件是根据您的语言选择创建的。生成的文件中有一些有用的注释和示例。当您运行generate命令为处理程序添加运行时代码时models.py，某些文件（例如）会在稍后的步骤中自动更新。

建模自定义 AWS CloudFormation 挂钩

对自定义 AWS CloudFormation Hook 进行建模涉及创建一个用于定义 Hook、其属性和属性的架构。使用cfn init命令创建自定义 Hook 项目时，会将示例 Hook 架构创建为JSON格式的文本文件。*hook-name.json*

目标调用点和目标操作指定了调用 Hook 的确切位置。Hook 处理程序为这些点托管可执行的自定义逻辑。例如，CREATE操作的目标操作使用preCreate处理程序。当 Hook 目标和服务执行匹配操作时，您在处理程序中编写的代码将调用。挂钩目标是调用挂钩的目的地。您可以指定目标，例如 AWS CloudFormation 公共资源、私有资源或自定义资源。Hook 支持无限数量的钩子目标。

该架构包含挂钩所需的权限。创作 Hook 需要您为每个 Hook 处理程序指定权限。CloudFormation 鼓励作者编写遵循标准安全建议的策略，即授予最低权限或仅授予执行任务所需的权限。确定用户（和角色）需要做什么，然后制定策略，仅允许他们为 Hook 操作执行这些任务。CloudFormation 使用这些权限来限定 Hook 用户提供的权限。这些权限将传递给 Hook。Hook 处理程序使用这些权限来访问 AWS 资源。

您可以使用以下架构文件作为起点来定义您的 Hook。使用 Hook 架构来指定要实现的处理程序。如果您选择不实现特定的处理程序，请将其从 Hook 架构的处理程序部分中删除。有关架构的更多详细信息，请参阅[架构语法](#)。

```
{
  "typeName": "MyCompany::Testing::MyTestHook",
  "description": "Verifies S3 bucket and SQS queues properties before create and update",
  "sourceUrl": "https://mycorp.com/my-repo.git",
  "documentationUrl": "https://mycorp.com/documentation",
  "typeConfiguration": {
    "properties": {
      "minBuckets": {
        "description": "Minimum number of compliant buckets",
        "type": "string"
      }
    }
  },
}
```

```
    "minQueues":{
      "description":"Minimum number of compliant queues",
      "type":"string"
    },
    "encryptionAlgorithm":{
      "description":"Encryption algorithm for SSE",
      "default":"AES256",
      "type":"string"
    }
  },
  "required":[

  ],
  "additionalProperties":false
},
"handlers":{
  "preCreate":{
    "targetNames":[
      "AWS::S3::Bucket",
      "AWS::SQS::Queue"
    ],
    "permissions":[

    ]
  },
  "preUpdate":{
    "targetNames":[
      "AWS::S3::Bucket",
      "AWS::SQS::Queue"
    ],
    "permissions":[

    ]
  },
  "preDelete":{
    "targetNames":[
      "AWS::S3::Bucket",
      "AWS::SQS::Queue"
    ],
    "permissions":[
      "s3:ListBucket",
      "s3:ListAllMyBuckets",
      "s3:GetEncryptionConfiguration",
      "sqs:ListQueues",
```



```
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl"
    ]
}
},
"additionalProperties":false
}
```

主题

- [使用 Java 对自定义 AWS CloudFormation 挂钩进行建模](#)
- [使用 Python 对自定义 AWS CloudFormation 挂钩进行建模](#)

使用 Java 对自定义 AWS CloudFormation 挂钩进行建模

对自定义 AWS CloudFormation Hook 进行建模涉及创建一个用于定义 Hook、其属性和属性的架构。本教程将引导你使用 Java 对自定义 Hook 进行建模。

步骤 1：添加项目依赖关系

基于 Java 的 Hooks 项目依赖于 Maven pom.xml 的文件作为依赖项。展开以下部分，将源代码复制到项目根目录下的 pom.xml 文件中。

挂钩项目依赖关系 (pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.testing.mytesthook</groupId>
  <artifactId>mycompany-testing-mytesthook-handler</artifactId>
  <name>mycompany-testing-mytesthook-handler</name>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<aws.java.sdk.version>2.16.1</aws.java.sdk.version>
<checkstyle.version>8.36.2</checkstyle.version>
<commons-io.version>2.8.0</commons-io.version>
<jackson.version>2.11.3</jackson.version>
<maven-checkstyle-plugin.version>3.1.1</maven-checkstyle-plugin.version>
<mockito.version>3.6.0</mockito.version>
<spotbugs.version>4.1.4</spotbugs.version>
<spotless.version>2.5.0</spotless.version>
<maven-javadoc-plugin.version>3.2.0</maven-javadoc-plugin.version>
<maven-source-plugin.version>3.2.1</maven-source-plugin.version>
<cfm.generate.args/>
</properties>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>bom</artifactId>
      <version>2.16.1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <!-- https://mvnrepository.com/artifact/software.amazon.cloudformation/aws-
cloudformation-rpdk-java-plugin -->
  <dependency>
    <groupId>software.amazon.cloudformation</groupId>
    <artifactId>aws-cloudformation-rpdk-java-plugin</artifactId>
    <version>[2.0.0,3.0.0)</version>
  </dependency>

  <!-- AWS Java SDK v2 Dependencies -->
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>sdk-core</artifactId>
  </dependency>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>cloudformation</artifactId>
  </dependency>
</dependencies>
```

```
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>s3</artifactId>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>utils</artifactId>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>apache-client</artifactId>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>sqs</artifactId>
</dependency>

<!-- Test dependency for Java Providers -->
<dependency>
  <groupId>software.amazon.cloudformation</groupId>
  <artifactId>cloudformation-cli-java-plugin-testing-support</artifactId>
  <version>1.0.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-s3 -->
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk-s3</artifactId>
  <version>1.12.85</version>
</dependency>

<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>${commons-io.version}</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.9</version>
</dependency>
```

```
    <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-collections4
-->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-collections4</artifactId>
      <version>4.4</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.google.guava/guava -->
    <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>29.0-jre</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-
cloudformation -->
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-cloudformation</artifactId>
      <version>1.11.555</version>
      <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->
    <dependency>
      <groupId>commons-codec</groupId>
      <artifactId>commons-codec</artifactId>
      <version>1.14</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/software.amazon.cloudformation/aws-
cloudformation-resource-schema -->
    <dependency>
      <groupId>software.amazon.cloudformation</groupId>
      <artifactId>aws-cloudformation-resource-schema</artifactId>
      <version>[2.0.5, 3.0.0)</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/
jackson-databind -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>${jackson.version}</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/
jackson-dataformat-cbor -->
```

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-cbor</artifactId>
  <version>${jackson.version}</version>
</dependency>

<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-jsr310</artifactId>
  <version>${jackson.version}</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.module/jackson-
modules-java8 -->
<dependency>
  <groupId>com.fasterxml.jackson.module</groupId>
  <artifactId>jackson-modules-java8</artifactId>
  <version>${jackson.version}</version>
  <type>pom</type>
  <scope>runtime</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/org.json/json -->
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20180813</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-core -->
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk-core</artifactId>
  <version>1.11.1034</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-lambda-java-core -->
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-lambda-java-core</artifactId>
  <version>1.2.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-lambda-java-log4j2 --
>

<dependency>
  <groupId>com.amazonaws</groupId>
```

```
        <artifactId>aws-lambda-java-log4j2</artifactId>
        <version>1.2.0</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.8</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.4</version>
        <scope>provided</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.17.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core --
>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.17.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-
impl -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j-impl</artifactId>
        <version>2.17.1</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.assertj/assertj-core -->
    <dependency>
        <groupId>org.assertj</groupId>
        <artifactId>assertj-core</artifactId>
```

```
        <version>3.12.2</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.5.0-M1</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.mockito/mockito-core -->
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>3.6.0</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.mockito/mockito-junit-jupiter -->
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-junit-jupiter</artifactId>
        <version>3.6.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <compilerArgs>
                    <arg>-Xlint:all, -options, -processing</arg>
                </compilerArgs>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>2.3</version>
            <configuration>
                <createDependencyReducedPom>>false</createDependencyReducedPom>
            </configuration>
        </plugin>
    </plugins>
</build>
```

```

        <filters>
          <filter>
            <artifact>*:*</artifact>
            <excludes>
              <exclude>**/Log4j2Plugins.dat</exclude>
            </excludes>
          </filter>
        </filters>
      </configuration>
    <executions>
      <execution>
        <phase>package</phase>
        <goals>
          <goal>shade</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.6.0</version>
    <executions>
      <execution>
        <id>generate</id>
        <phase>generate-sources</phase>
        <goals>
          <goal>exec</goal>
        </goals>
        <configuration>
          <executable>cfn</executable>
          <commandlineArgs>generate ${cfn.generate.args}</
commandlineArgs>
          <workingDirectory>${project.basedir}</workingDirectory>
        </configuration>
      </execution>
    </executions>
  </plugin>
  <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>3.0.0</version>
    <executions>
      <execution>

```



```

        <id>add-source</id>
        <phase>generate-sources</phase>
        <goals>
            <goal>add-source</goal>
        </goals>
        <configuration>
            <sources>
                <source>${project.basedir}/target/generated-sources/
rpdk</source>
            </sources>
        </configuration>
    </execution>
</executions>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>2.4</version>
</plugin>
<plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M3</version>
</plugin>
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.4</version>
    <configuration>
        <excludes>
            <exclude>**/BaseHookConfiguration*</exclude>
            <exclude>**/BaseHookHandler*</exclude>
            <exclude>**/HookHandlerWrapper*</exclude>
            <exclude>**/ResourceModel*</exclude>
            <exclude>**/TypeConfigurationModel*</exclude>
            <exclude>**/model/**/*</exclude>
        </excludes>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
    </execution>

```

```
        <id>report</id>
        <phase>test</phase>
        <goals>
            <goal>report</goal>
        </goals>
    </execution>
    <execution>
        <id>jacoco-check</id>
        <goals>
            <goal>check</goal>
        </goals>
        <configuration>
            <rules>
                <rule>
                    <element>PACKAGE</element>
                    <limits>
                        <limit>
                            <counter>BRANCH</counter>
                            <value>COVEREDRATIO</value>
                            <minimum>0.8</minimum>
                        </limit>
                        <limit>
                            <counter>INSTRUCTION</counter>
                            <value>COVEREDRATIO</value>
                            <minimum>0.8</minimum>
                        </limit>
                    </limits>
                </rule>
            </rules>
        </configuration>
    </execution>
</executions>
</plugin>
</plugins>
<resources>
    <resource>
        <directory>${project.basedir}</directory>
        <includes>
            <include>mycompany-testing-mytesthook.json</include>
        </includes>
    </resource>
    <resource>
        <directory>${project.basedir}/target/loaded-target-schemas</directory>
        <includes>
```

```
        <include>**/*.json</include>
      </includes>
    </resource>
  </resources>
</build>
</project>
```

第 2 步：生成 Hook 项目包

生成你的 Hook 项目包。CloudFormation CLI 会创建空的处理函数，这些函数对应于 Hook 规范中定义的目标生命周期中的特定 Hook 操作。

```
cfn generate
```

该命令将返回以下输出。

```
Generated files for MyCompany::Testing::MyTestHook
```

Note

确保您的 Lambda 运行时 up-to-date 避免使用已弃用的版本。有关更多信息，请参阅[更新资源类型的 Lambda 运行时和挂钩](#)。

第 3 步：添加 Hook 处理程序

将您自己的 Hook 处理程序运行时代码添加到您选择实现的处理程序中。例如，您可以添加以下用于记录的代码。

```
logger.log("Internal testing Hook triggered for target: " +
  request.getHookContext().getTargetName());
```

CloudFormation CLI 生成一个普通的旧 Java 对象 (JavaPOJO)。以下是生成的输出示例 `AWS::S3::Bucket`。

Example `awss3.java` `BucketTargetModel`

```
package com.mycompany.testing.mytesthook.model.aws.s3.bucket;
```

```
import...

@Data
@NoArgsConstructor
@EqualsAndHashCode(callSuper = true)
@ToString(callSuper = true)
@JsonAutoDetect(fieldVisibility = Visibility.ANY, getterVisibility = Visibility.NONE,
    setterVisibility = Visibility.NONE)
public class AwsS3BucketTargetModel extends ResourceHookTargetModel<AwsS3Bucket> {

    @JsonIgnore
    private static final TypeReference<AwsS3Bucket> TARGET_REFERENCE =
        new TypeReference<AwsS3Bucket>() {};

    @JsonIgnore
    private static final TypeReference<AwsS3BucketTargetModel> MODEL_REFERENCE =
        new TypeReference<AwsS3BucketTargetModel>() {};

    @JsonIgnore
    public static final String TARGET_TYPE_NAME = "AWS::S3::Bucket";

    @JsonIgnore
    public TypeReference<AwsS3Bucket> getHookTargetTypeReference() {
        return TARGET_REFERENCE;
    }

    @JsonIgnore
    public TypeReference<AwsS3BucketTargetModel> getTargetModelTypeReference() {
        return MODEL_REFERENCE;
    }
}
```

Example AwsS3Bucket.java

```
package com.mycompany.testing.mytesthook.model.aws.s3.bucket;

import ...

@Data
@Builder
```

```
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode(callSuper = true)
@ToString(callSuper = true)
@JsonAutoDetect(fieldVisibility = Visibility.ANY, getterVisibility = Visibility.NONE,
    setterVisibility = Visibility.NONE)
public class AwsS3Bucket extends ResourceHookTarget {
    @JsonIgnore
    public static final String TYPE_NAME = "AWS::S3::Bucket";

    @JsonIgnore
    public static final String IDENTIFIER_KEY_ID = "/properties/Id";

    @JsonProperty("InventoryConfigurations")
    private List<InventoryConfiguration> inventoryConfigurations;

    @JsonProperty("WebsiteConfiguration")
    private WebsiteConfiguration websiteConfiguration;

    @JsonProperty("DualStackDomainName")
    private String dualStackDomainName;

    @JsonProperty("AccessControl")
    private String accessControl;

    @JsonProperty("AnalyticsConfigurations")
    private List<AnalyticsConfiguration> analyticsConfigurations;

    @JsonProperty("AccelerateConfiguration")
    private AccelerateConfiguration accelerateConfiguration;

    @JsonProperty("PublicAccessBlockConfiguration")
    private PublicAccessBlockConfiguration publicAccessBlockConfiguration;

    @JsonProperty("BucketName")
    private String bucketName;

    @JsonProperty("RegionalDomainName")
    private String regionalDomainName;

    @JsonProperty("OwnershipControls")
    private OwnershipControls ownershipControls;

    @JsonProperty("ObjectLockConfiguration")
```

```
private ObjectLockConfiguration objectLockConfiguration;

@JsonProperty("ObjectLockEnabled")
private Boolean objectLockEnabled;

@JsonProperty("LoggingConfiguration")
private LoggingConfiguration loggingConfiguration;

@JsonProperty("ReplicationConfiguration")
private ReplicationConfiguration replicationConfiguration;

@JsonProperty("Tags")
private List<Tag> tags;

@JsonProperty("DomainName")
private String domainName;

@JsonProperty("BucketEncryption")
private BucketEncryption bucketEncryption;

@JsonProperty("WebsiteURL")
private String websiteURL;

@JsonProperty("NotificationConfiguration")
private NotificationConfiguration notificationConfiguration;

@JsonProperty("LifecycleConfiguration")
private LifecycleConfiguration lifecycleConfiguration;

@JsonProperty("VersioningConfiguration")
private VersioningConfiguration versioningConfiguration;

@JsonProperty("MetricsConfigurations")
private List<MetricsConfiguration> metricsConfigurations;

@JsonProperty("IntelligentTieringConfigurations")
private List<IntelligentTieringConfiguration> intelligentTieringConfigurations;

@JsonProperty("CorsConfiguration")
private CorsConfiguration corsConfiguration;

@JsonProperty("Id")
private String id;
```

```

@JsonProperty("Arn")
private String arn;

@JsonIgnore
public JSONObject getPrimaryIdentifier() {
    final JSONObject identifier = new JSONObject();
    if (this.getId() != null) {
        identifier.put(IDENTIFIER_KEY_ID, this.getId());
    }

    // only return the identifier if it can be used, i.e. if all components are
    present
    return identifier.length() == 1 ? identifier : null;
}

@JsonIgnore
public List<JSONObject> getAdditionalIdentifiers() {
    final List<JSONObject> identifiers = new ArrayList<JSONObject>();
    // only return the identifiers if any can be used
    return identifiers.isEmpty() ? null : identifiers;
}
}

```

Example BucketEncryption.java

```

package software.amazon.testing.mytesthook.model.aws.s3.bucket;

import ...

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@JsonAutoDetect(fieldVisibility = Visibility.ANY, getterVisibility = Visibility.NONE,
    setterVisibility = Visibility.NONE)
public class BucketEncryption {
    @JsonProperty("ServerSideEncryptionConfiguration")
    private List<ServerSideEncryptionRule> serverSideEncryptionConfiguration;
}

```

Example ServerSideEncryptionRule.java

```
package com.mycompany.testing.mytesthook.model.aws.s3.bucket;

import ...

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@JsonAutoDetect(fieldVisibility = Visibility.ANY, getterVisibility = Visibility.NONE,
    setterVisibility = Visibility.NONE)
public class ServerSideEncryptionRule {
    @JsonProperty("BucketKeyEnabled")
    private Boolean bucketKeyEnabled;

    @JsonProperty("ServerSideEncryptionByDefault")
    private ServerSideEncryptionByDefault serverSideEncryptionByDefault;
}
```

Example ServerSideEncryptionByDefault.java

```
package com.mycompany.testing.mytesthook.model.aws.s3.bucket;

import ...

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@JsonAutoDetect(fieldVisibility = Visibility.ANY, getterVisibility = Visibility.NONE,
    setterVisibility = Visibility.NONE)
public class ServerSideEncryptionByDefault {
    @JsonProperty("SSEAlgorithm")
    private String sSEAlgorithm;

    @JsonProperty("KMSEMasterKeyID")
    private String kMSEMasterKeyID;
}
```


使用POJOs生成的，您现在可以编写实际实现 Hook 功能的处理程序。在此示例中，为处理程序实现preCreate和preUpdate调用点。

第 4 步：实现 Hook 处理程序

主题

- [对API客户端生成器进行编码](#)
- [对API请求生成器进行编码](#)
- [实现帮助程序代码](#)
- [实现基本处理程序](#)
- [实现处理preCreate程序](#)
- [对preCreate处理程序进行编码](#)
- [更新preCreate测试](#)
- [实现处理preUpdate程序](#)
- [对preUpdate处理程序进行编码](#)
- [更新preUpdate测试](#)
- [实现处理preDelete程序](#)
- [对preDelete处理程序进行编码](#)
- [更新处理preDelete程序](#)

对API客户端生成器进行编码

1. 在您的ClientBuilder.java文件中IDE，打开位于src/main/java/com/mycompany/testing/mytesthook文件夹中的文件。
2. 用以下代码替换ClientBuilder.java文件的全部内容。

Example ClientBuilder.java

```
package com.awscommunity.kms.encryptionsettings;

import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.cloudformation.HookLambdaWrapper;

/**
 * Describes static HTTP clients (to consume less memory) for API calls that
```

```
* this hook makes to a number of AWS services.
*/
public final class ClientBuilder {

    private ClientBuilder() {
    }

    /**
     * Create an HTTP client for Amazon EC2.
     *
     * @return Ec2Client An {@link Ec2Client} object.
     */
    public static Ec2Client getEc2Client() {
        return
        Ec2Client.builder().httpClient(HookLambdaWrapper.HTTP_CLIENT).build();
    }
}
```

对API请求生成器进行编码

1. 在您的Translator.java文件中IDE，打开位于src/main/java/com/mycompany/testing/mytesthook文件夹中的文件。
2. 用以下代码替换Translator.java文件的全部内容。

Example Translator.java

```
package com.mycompany.testing.mytesthook;

import software.amazon.awssdk.services.s3.model.GetBucketEncryptionRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesRequest;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;

/**
 * This class is a centralized placeholder for
 * - api request construction
 * - object translation to/from aws sdk
 */

public class Translator {
```

```
static ListBucketsRequest translateToListBucketsRequest(final HookTargetModel
targetModel) {
    return ListBucketsRequest.builder().build();
}

static ListQueuesRequest translateToListQueuesRequest(final String nextToken) {
    return ListQueuesRequest.builder().nextToken(nextToken).build();
}

static ListBucketsRequest createListBucketsRequest() {
    return ListBucketsRequest.builder().build();
}

static ListQueuesRequest createListQueuesRequest() {
    return createListQueuesRequest(null);
}

static ListQueuesRequest createListQueuesRequest(final String nextToken) {
    return ListQueuesRequest.builder().nextToken(nextToken).build();
}

static GetBucketEncryptionRequest createGetBucketEncryptionRequest(final String
bucket) {
    return GetBucketEncryptionRequest.builder().bucket(bucket).build();
}
}
```

实现帮助程序代码

1. 在您的AbstractTestBase.java文件中IDE，打开位于src/main/java/com/mycompany/testing/mytesthook文件夹中的文件。
2. 用以下代码替换AbstractTestBase.java文件的全部内容。

Example Translator.java

```
package com.mycompany.testing.mytesthook;

import com.google.common.collect.ImmutableMap;
import org.mockito.Mockito;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.AwsSessionCredentials;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
```

```
import software.amazon.awssdk.awscore.AwsRequest;
import software.amazon.awssdk.awscore.AwsRequestOverrideConfiguration;
import software.amazon.awssdk.awscore.AwsResponse;
import software.amazon.awssdk.core.SdkClient;
import software.amazon.awssdk.core.pagination.sync.SdkIterable;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
import software.amazon.cloudformation.proxy.Credentials;
import software.amazon.cloudformation.proxy.LoggerProxy;
import software.amazon.cloudformation.proxy.OperationStatus;
import software.amazon.cloudformation.proxy.ProgressEvent;
import software.amazon.cloudformation.proxy.ProxyClient;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;

import javax.annotation.Nonnull;
import java.time.Duration;
import java.util.concurrent.CompletableFuture;
import java.util.function.Function;
import java.util.function.Supplier;

import static org.assertj.core.api.Assertions.assertThat;

@lombok.Getter
public class AbstractTestBase {
    protected final AwsSessionCredentials awsSessionCredential;
    protected final AwsCredentialsProvider v2CredentialsProvider;
    protected final AwsRequestOverrideConfiguration configuration;
    protected final LoggerProxy loggerProxy;
    protected final Supplier<Long> awsLambdaRuntime = () ->
Duration.ofMinutes(15).toMillis();
    protected final AmazonWebServicesClientProxy proxy;
    protected final Credentials mockCredentials =
        new Credentials("mockAccessId", "mockSecretKey", "mockSessionToken");

    @lombok.Setter
    private SdkClient serviceClient;

    protected AbstractTestBase() {
        loggerProxy = Mockito.mock(LoggerProxy.class);
        awsSessionCredential =
AwsSessionCredentials.create(mockCredentials.getAccessKeyId(),
        mockCredentials.getSecretAccessKey(),
mockCredentials.getSessionToken());
        v2CredentialsProvider =
StaticCredentialsProvider.create(awsSessionCredential);
    }
}
```

```

configuration = AwsRequestOverrideConfiguration.builder()
    .credentialsProvider(v2CredentialsProvider)
    .build();
proxy = new AmazonWebServicesClientProxy(
    loggerProxy,
    mockCredentials,
    awsLambdaRuntime
) {
    @Override
    public <ClientT> ProxyClient<ClientT> newProxy(@NonNull
Supplier<ClientT> client) {
        return new ProxyClient<ClientT>() {
            @Override
            public <RequestT extends AwsRequest, ResponseT extends
AwsResponse>
                ResponseT injectCredentialsAndInvokeV2(RequestT request,
Function<RequestT,
ResponseT> requestFunction) {
                return proxy.injectCredentialsAndInvokeV2(request,
requestFunction);
            }

            @Override
            public <RequestT extends AwsRequest, ResponseT extends
AwsResponse> CompletableFuture<ResponseT>
                injectCredentialsAndInvokeV2Async(RequestT request,
Function<RequestT, CompletableFuture<ResponseT>> requestFunction) {
                return proxy.injectCredentialsAndInvokeV2Async(request,
requestFunction);
            }

            @Override
            public <RequestT extends AwsRequest, ResponseT extends
AwsResponse, IterableT extends SdkIterable<ResponseT>>
                IterableT
                injectCredentialsAndInvokeIterableV2(RequestT request,
Function<RequestT, IterableT> requestFunction) {
                return proxy.injectCredentialsAndInvokeIterableV2(request,
requestFunction);
            }

            @SuppressWarnings("unchecked")
            @Override
            public ClientT client() {

```

```
        return (ClientT) serviceClient;
    }
};
}

protected void assertResponse(final ProgressEvent<HookTargetModel,
CallbackContext> response, final OperationStatus expectedStatus, final String
expectedMsg) {
    assertThat(response).isNotNull();
    assertThat(response.getStatus()).isEqualTo(expectedStatus);
    assertThat(response.getCallbackContext()).isNull();
    assertThat(response.getCallbackDelaySeconds()).isEqualTo(0);
    assertThat(response.getMessage()).isNotNull();
    assertThat(response.getMessage()).isEqualTo(expectedMsg);
}

protected HookTargetModel createHookTargetModel(final Object
resourceProperties) {
    return HookTargetModel.of(ImmutableMap.of("ResourceProperties",
resourceProperties));
}

protected HookTargetModel createHookTargetModel(final Object
resourceProperties, final Object previousResourceProperties) {
    return HookTargetModel.of(
        ImmutableMap.of(
            "ResourceProperties", resourceProperties,
            "PreviousResourceProperties", previousResourceProperties
        )
    );
}
}
```

实现基本处理程序

1. 在您的BaseHookHandlerStd.java文件中IDE，打开位于src/main/java/com/mycompany/testing/mytesthook文件夹中的文件。
2. 用以下代码替换BaseHookHandlerStd.java文件的全部内容。

Example Translator.java

```
package com.mycompany.testing.mytesthook;

import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;
import com.mycompany.testing.mytesthook.model.aws.sqs.queue.AwsSqsQueue;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.cloudformation.exceptions.UnsupportedTargetException;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
import software.amazon.cloudformation.proxy.Logger;
import software.amazon.cloudformation.proxy.ProgressEvent;
import software.amazon.cloudformation.proxy.ProxyClient;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;

public abstract class BaseHookHandlerStd extends BaseHookHandler<CallbackContext,
    TypeConfigurationModel> {
    public static final String HOOK_TYPE_NAME = "MyCompany::Testing::MyTestHook";

    protected Logger logger;

    @Override
    public ProgressEvent<HookTargetModel, CallbackContext> handleRequest(
        final AmazonWebServicesClientProxy proxy,
        final HookHandlerRequest request,
        final CallbackContext callbackContext,
        final Logger logger,
        final TypeConfigurationModel typeConfiguration
    ) {
        this.logger = logger;

        final String targetName = request.getHookContext().getTargetName();

        final ProgressEvent<HookTargetModel, CallbackContext> result;
        if (AwsS3Bucket.TYPE_NAME.equals(targetName)) {
            result = handleS3BucketRequest(
                proxy,
                request,
                callbackContext != null ? callbackContext : new
                CallbackContext(),
                proxy.newProxy(ClientBuilder::createS3Client),
```

```

        typeConfiguration
    );
} else if (AwsSqsQueue.TYPE_NAME.equals(targetName)) {
    result = handleSqsQueueRequest(
        proxy,
        request,
        callbackContext != null ? callbackContext : new
CallbackContext(),
        proxy.newProxy(ClientBuilder::createSqsClient),
        typeConfiguration
    );
} else {
    throw new UnsupportedTargetException(targetName);
}

log(
    String.format(
        "Result for [%s] invocation for target [%s] returned status [%s]
with message [%s]",
        request.getHookContext().getInvocationPoint(),
        targetName,
        result.getStatus(),
        result.getMessage()
    )
);

return result;
}

protected abstract ProgressEvent<HookTargetModel, CallbackContext>
handleS3BucketRequest(
    final AmazonWebServicesClientProxy proxy,
    final HookHandlerRequest request,
    final CallbackContext callbackContext,
    final ProxyClient<S3Client> proxyClient,
    final TypeConfigurationModel typeConfiguration
);

protected abstract ProgressEvent<HookTargetModel, CallbackContext>
handleSqsQueueRequest(
    final AmazonWebServicesClientProxy proxy,
    final HookHandlerRequest request,
    final CallbackContext callbackContext,
    final ProxyClient<SqsClient> proxyClient,

```



```
        final TypeConfigurationModel typeConfiguration
    );

    protected void log(final String message) {
        if (logger != null) {
            logger.log(message);
        } else {
            System.out.println(message);
        }
    }
}
```

实现处理preCreate程序

preCreate处理程序验证AWS::S3::Bucket或AWS::SQS::Queue资源的服务器端加密设置。

- 对于AWS::S3::Bucket资源，只有在满足以下条件时，Hook 才会通过：
 - Amazon S3 存储桶加密已设置。
 - 已为该存储桶启用了 Amazon S3 存储桶密钥。
 - 为 Amazon S3 存储桶设置的加密算法是所需的正确算法。
 - 密 AWS Key Management Service 钥 ID 已设置。
- 对于AWS::SQS::Queue资源，只有在满足以下条件时，Hook 才会通过：
 - 密 AWS Key Management Service 钥 ID 已设置。

对preCreate处理程序进行编码

1. 在您的PreCreateHookHandler.java文件中IDE，打开位于src/main/java/software/mycompany/testing/mytesthook文件夹中的文件。
2. 用以下代码替换PreCreateHookHandler.java文件的全部内容。

```
package com.mycompany.testing.mytesthook;

import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3BucketTargetModel;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.BucketEncryption;
import
    com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionByDefault;
```

```
import
    com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionRule;
import com.mycompany.testing.mytesthook.model.aws.sqs.queue.AwsSqsQueue;
import com.mycompany.testing.mytesthook.model.aws.sqs.queue.AwsSqsQueueTargetModel;
import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.lang3.StringUtils;
import software.amazon.cloudformation.exceptions.UnsupportedTargetException;
import software.amazon.cloudformation.proxy.HandlerErrorCode;
import software.amazon.cloudformation.proxy.Logger;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
import software.amazon.cloudformation.proxy.hook.HookStatus;
import software.amazon.cloudformation.proxy.hook.HookProgressEvent;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import
    software.amazon.cloudformation.proxy.hook.targetmodel.ResourceHookTargetModel;

import java.util.List;

public class PreCreateHookHandler extends BaseHookHandler<TypeConfigurationModel,
    CallbackContext> {

    @Override
    public HookProgressEvent<CallbackContext> handleRequest(
        final AmazonWebServicesClientProxy proxy,
        final HookHandlerRequest request,
        final CallbackContext callbackContext,
        final Logger logger,
        final TypeConfigurationModel typeConfiguration) {

        final String targetName = request.getHookContext().getTargetName();
        if ("AWS::S3::Bucket".equals(targetName)) {
            final ResourceHookTargetModel<AwsS3Bucket> targetModel =
request.getHookContext().getTargetModel(AwsS3BucketTargetModel.class);

            final AwsS3Bucket bucket = targetModel.getResourceProperties();
            final String encryptionAlgorithm =
typeConfiguration.getEncryptionAlgorithm();

            return validateS3BucketEncryption(bucket, encryptionAlgorithm);

        } else if ("AWS::SQS::Queue".equals(targetName)) {
            final ResourceHookTargetModel<AwsSqsQueue> targetModel =
request.getHookContext().getTargetModel(AwsSqsQueueTargetModel.class);
```

```

        final AwsSqsQueue queue = targetModel.getResourceProperties();
        return validateSQSQueueEncryption(queue);
    } else {
        throw new UnsupportedTargetException(targetName);
    }
}

private HookProgressEvent<CallbackContext> validateS3BucketEncryption(final
    AwsS3Bucket bucket, final String requiredEncryptionAlgorithm) {
    HookStatus resultStatus = null;
    String resultMessage = null;

    if (bucket != null) {
        final BucketEncryption bucketEncryption = bucket.getBucketEncryption();
        if (bucketEncryption != null) {
            final List<ServerSideEncryptionRule> serverSideEncryptionRules =
                bucketEncryption.getServerSideEncryptionConfiguration();
            if (CollectionUtils.isNotEmpty(serverSideEncryptionRules)) {
                for (final ServerSideEncryptionRule rule :
                    serverSideEncryptionRules) {
                    final Boolean bucketKeyEnabled =
                        rule.getBucketKeyEnabled();
                    if (bucketKeyEnabled) {
                        final ServerSideEncryptionByDefault
                            serverSideEncryptionByDefault = rule.getServerSideEncryptionByDefault();

                        final String encryptionAlgorithm =
                            serverSideEncryptionByDefault.getSSEAlgorithm();
                        final String kmsKeyId =
                            serverSideEncryptionByDefault.getKMSMasterKeyID(); // "KMSMasterKeyID" is name of
                            the property for an AWS::S3::Bucket;

                        if (!StringUtils.equals(encryptionAlgorithm,
                            requiredEncryptionAlgorithm) && StringUtils.isBlank(kmsKeyId)) {
                            resultStatus = HookStatus.FAILED;
                            resultMessage = "KMS Key ID not set
                            and SSE Encryption Algorithm is incorrect for bucket with name: " +
                                bucket.getBucketName();
                        } else if (!StringUtils.equals(encryptionAlgorithm,
                            requiredEncryptionAlgorithm)) {
                            resultStatus = HookStatus.FAILED;
                            resultMessage = "SSE Encryption Algorithm is
                            incorrect for bucket with name: " + bucket.getBucketName();
                        } else if (StringUtils.isBlank(kmsKeyId)) {

```

```

        resultStatus = HookStatus.FAILED;
        resultMessage = "KMS Key ID not set for bucket with
name: " + bucket.getBucketName();
    } else {
        resultStatus = HookStatus.SUCCESS;
        resultMessage = "Successfully invoked
PreCreateHookHandler for target: AWS::S3::Bucket";
    }
} else {
    resultStatus = HookStatus.FAILED;
    resultMessage = "Bucket key not enabled for bucket with
name: " + bucket.getBucketName();
}

    if (resultStatus == HookStatus.FAILED) {
        break;
    }
}
} else {
    resultStatus = HookStatus.FAILED;
    resultMessage = "No SSE Encryption configurations for bucket
with name: " + bucket.getBucketName();
}
} else {
    resultStatus = HookStatus.FAILED;
    resultMessage = "Bucket Encryption not enabled for bucket with
name: " + bucket.getBucketName();
}
} else {
    resultStatus = HookStatus.FAILED;
    resultMessage = "Resource properties for S3 Bucket target model are
empty";
}

    return HookProgressEvent.<CallbackContext>builder()
        .status(resultStatus)
        .message(resultMessage)
        .errorCode(resultStatus == HookStatus.FAILED ?
HandlerErrorCode.ResourceConflict : null)
        .build();
}

    private HookProgressEvent<CallbackContext> validateSQSQueueEncryption(final
AwsSqsQueue queue) {

```

```
        if (queue == null) {
            return HookProgressEvent.<CallbackContext>builder()
                .status(HookStatus.FAILED)
                .message("Resource properties for SQS Queue target model are
empty")
                .errorCode(HandlerErrorCode.ResourceConflict)
                .build();
        }

        final String kmsKeyId = queue.getKmsMasterKeyId(); // "KmsMasterKeyId" is
name of the property for an AWS::SQS::Queue
        if (StringUtils.isBlank(kmsKeyId)) {
            return HookProgressEvent.<CallbackContext>builder()
                .status(HookStatus.FAILED)
                .message("Server side encryption turned off for queue with
name: " + queue.getQueueName())
                .errorCode(HandlerErrorCode.ResourceConflict)
                .build();
        }

        return HookProgressEvent.<CallbackContext>builder()
            .status(HookStatus.SUCCESS)
            .message("Successfully invoked PreCreateHookHandler for target:
AWS::SQS::Queue")
            .build();
    }
}
```

更新preCreate测试

1. 在您的PreCreateHandlerTest.java文件中IDE，打开位于src/test/java/software/mycompany/testing/mytesthook文件夹中的文件。
2. 用以下代码替换PreCreateHandlerTest.java文件的全部内容。

```
package com.mycompany.testing.mytesthook;

import com.google.common.collect.ImmutableMap;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.BucketEncryption;
import
    com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionByDefault;
```

```
import
    com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionRule;
import com.mycompany.testing.mytesthook.model.aws.sqs.queue.AwsSqsQueue;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import software.amazon.cloudformation.exceptions.UnsupportedTargetException;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
import software.amazon.cloudformation.proxy.HandlerErrorCode;
import software.amazon.cloudformation.proxy.Logger;
import software.amazon.cloudformation.proxy.hook.HookContext;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import software.amazon.cloudformation.proxy.hook.HookProgressEvent;
import software.amazon.cloudformation.proxy.hook.HookStatus;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;

import java.util.Collections;
import java.util.Map;

import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatExceptionOfType;
import static org.mockito.Mockito.mock;

@ExtendWith(MockitoExtension.class)
public class PreCreateHookHandlerTest {

    @Mock
    private AmazonWebServicesClientProxy proxy;

    @Mock
    private Logger logger;

    @BeforeEach
    public void setup() {
        proxy = mock(AmazonWebServicesClientProxy.class);
        logger = mock(Logger.class);
    }

    @Test
    public void handleRequest_awsSqsQueueSuccess() {
        final PreCreateHookHandler handler = new PreCreateHookHandler();
```

```
        final AwsSqsQueue queue = buildSqsQueue("MyQueue", "KmsKey");
        final HookTargetModel targetModel = createHookTargetModel(queue);
        final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

        final HookHandlerRequest request = HookHandlerRequest.builder()

        .hookContext(HookContext.builder().targetName("AWS::SQS::Queue").targetModel(targetModel)
                .build());

        final HookProgressEvent<CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);
        assertResponse(response, HookStatus.SUCCESS, "Successfully invoked
PreCreateHookHandler for target: AWS::SQS::Queue");
    }

    @Test
    public void handleRequest_awsS3BucketSuccess() {
        final PreCreateHookHandler handler = new PreCreateHookHandler();

        final AwsS3Bucket bucket = buildAwsS3Bucket("amzn-s3-demo-bucket", true,
"AES256", "KmsKey");
        final HookTargetModel targetModel = createHookTargetModel(bucket);
        final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

        final HookHandlerRequest request = HookHandlerRequest.builder()

        .hookContext(HookContext.builder().targetName("AWS::S3::Bucket").targetModel(targetModel)
                .build());

        final HookProgressEvent<CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);
        assertResponse(response, HookStatus.SUCCESS, "Successfully invoked
PreCreateHookHandler for target: AWS::S3::Bucket");
    }

    @Test
    public void handleRequest_awsS3BucketFail_bucketKeyNotEnabled() {
        final PreCreateHookHandler handler = new PreCreateHookHandler();

        final AwsS3Bucket bucket = buildAwsS3Bucket("amzn-s3-demo-bucket", false,
"AES256", "KmsKey");
        final HookTargetModel targetModel = createHookTargetModel(bucket);
```

```
        final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

        final HookHandlerRequest request = HookHandlerRequest.builder()

.hookContext(HookContext.builder().targetName("AWS::S3::Bucket").targetModel(targetModel)
        .build());

        final HookProgressEvent<CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);
        assertResponse(response, HookStatus.FAILED, "Bucket key not enabled for
bucket with name: amzn-s3-demo-bucket");
    }

    @Test
    public void handleRequest_awsS3BucketFail_incorrectSSEEncryptionAlgorithm() {
        final PreCreateHookHandler handler = new PreCreateHookHandler();

        final AwsS3Bucket bucket = buildAwsS3Bucket("amzn-s3-demo-bucket", true,
"SHA512", "KmsKey");
        final HookTargetModel targetModel = createHookTargetModel(bucket);
        final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

        final HookHandlerRequest request = HookHandlerRequest.builder()

.hookContext(HookContext.builder().targetName("AWS::S3::Bucket").targetModel(targetModel)
        .build());

        final HookProgressEvent<CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);
        assertResponse(response, HookStatus.FAILED, "SSE Encryption Algorithm is
incorrect for bucket with name: amzn-s3-demo-bucket");
    }

    @Test
    public void handleRequest_awsS3BucketFail_kmsKeyIdNotSet() {
        final PreCreateHookHandler handler = new PreCreateHookHandler();

        final AwsS3Bucket bucket = buildAwsS3Bucket("amzn-s3-demo-bucket", true,
"AES256", null);
        final HookTargetModel targetModel = createHookTargetModel(bucket);
        final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();
```



```
        final HookHandlerRequest request = HookHandlerRequest.builder()

        .hookContext(HookContext.builder().targetName("AWS::S3::Bucket").targetModel(targetModel)
            .build());

        final HookProgressEvent<CallbackContext> response =
        handler.handleRequest(proxy, request, null, logger, typeConfiguration);
        assertResponse(response, HookStatus.FAILED, "KMS Key ID not set for bucket
        with name: amzn-s3-demo-bucket");
    }

    @Test
    public void handleRequest_awsSqsQueueFail_serverSideEncryptionOff() {
        final PreCreateHookHandler handler = new PreCreateHookHandler();

        final AwsSqsQueue queue = buildSqsQueue("MyQueue", null);
        final HookTargetModel targetModel = createHookTargetModel(queue);
        final TypeConfigurationModel typeConfiguration =
        TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

        final HookHandlerRequest request = HookHandlerRequest.builder()

        .hookContext(HookContext.builder().targetName("AWS::SQS::Queue").targetModel(targetModel)
            .build());

        final HookProgressEvent<CallbackContext> response =
        handler.handleRequest(proxy, request, null, logger, typeConfiguration);
        assertResponse(response, HookStatus.FAILED, "Server side encryption turned
        off for queue with name: MyQueue");
    }

    @Test
    public void handleRequest_unsupportedTarget() {
        final PreCreateHookHandler handler = new PreCreateHookHandler();

        final Map<String, Object> unsupportedTarget =
        ImmutableMap.of("ResourceName", "MyUnsupportedTarget");
        final HookTargetModel targetModel =
        createHookTargetModel(unsupportedTarget);
        final TypeConfigurationModel typeConfiguration =
        TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

        final HookHandlerRequest request = HookHandlerRequest.builder()
```

```

.hookContext(HookContext.builder().targetName("AWS::Unsupported::Target").targetModel(targetModel)
    .build());

    assertThatExceptionOfType(UnsupportedTargetException.class)
        .isThrownBy(() -> handler.handleRequest(proxy, request, null,
logger, typeConfiguration))
        .withMessageContaining("Unsupported target")
        .withMessageContaining("AWS::Unsupported::Target")
        .satisfies(e ->
assertThat(e.getErrorCode()).isEqualTo(HandlerErrorCode.InvalidRequest));
    }

    private void assertResponse(final HookProgressEvent<CallbackContext> response,
final HookStatus expectedStatus, final String expectedErrorMsg) {
        assertThat(response).isNotNull();
        assertThat(response.getStatus()).isEqualTo(expectedStatus);
        assertThat(response.getCallbackContext()).isNull();
        assertThat(response.getCallbackDelaySeconds()).isEqualTo(0);
        assertThat(response.getMessage()).isNotNull();
        assertThat(response.getMessage()).isEqualTo(expectedErrorMsg);
    }

    private HookTargetModel createHookTargetModel(final Object resourceProperties)
{
        return HookTargetModel.of(ImmutableMap.of("ResourceProperties",
resourceProperties));
    }

    @SuppressWarnings("SameParameterValue")
    private AwsSqsQueue buildSqsQueue(final String queueName, final String
kmsKeyId) {
        return AwsSqsQueue.builder()
            .queueName(queueName)
            .kmsMasterKeyId(kmsKeyId) // "KmsMasterKeyId" is name of the
property for an AWS::SQS::Queue
            .build();
    }

    @SuppressWarnings("SameParameterValue")
    private AwsS3Bucket buildAwsS3Bucket(
        final String bucketName,
        final Boolean bucketKeyEnabled,
        final String sseAlgorithm,

```

```

        final String kmsKeyId
    ) {
        return AwsS3Bucket.builder()
            .bucketName(bucketName)
            .bucketEncryption(
                BucketEncryption.builder()
                    .serverSideEncryptionConfiguration(
                        Collections.singletonList(
                            ServerSideEncryptionRule.builder()
                                .bucketKeyEnabled(bucketKeyEnabled)
                                .serverSideEncryptionByDefault(
                                    ServerSideEncryptionByDefault.builder()
                                        .sSEAlgorithm(sseAlgorithm)
                                        .kMSMasterKeyID(kmsKeyId) //
                                )
                            )
                        )
                    )
                )
            )
        )
    }.build();
}

```

"KMSMasterKeyID" is name of the property for an AWS::S3::Bucket

实现处理preUpdate程序

实现一个preUpdate处理程序，该处理程序在处理程序中所有指定目标的更新操作之前启动。该preUpdate处理程序完成以下任务：

- 对于AWS::S3::Bucket资源，只有在满足以下条件时，Hook 才会通过：
 - Amazon S3 存储桶的存储桶加密算法尚未修改。

对preUpdate处理程序进行编码

1. 在您的PreUpdateHookHandler.java文件中IDE，打开位于src/main/java/software/mycompany/testing/mytesthook文件夹中的文件。
2. 用以下代码替换PreUpdateHookHandler.java文件的全部内容。

```

package com.mycompany.testing.mytesthook;

import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;

```

```
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3BucketTargetModel;
import
    com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionRule;
import org.apache.commons.lang3.StringUtils;
import software.amazon.cloudformation.exceptions.UnsupportedTargetException;
import software.amazon.cloudformation.proxy.HandlerErrorCode;
import software.amazon.cloudformation.proxy.Logger;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
import software.amazon.cloudformation.proxy.hook.HookStatus;
import software.amazon.cloudformation.proxy.hook.HookProgressEvent;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import
    software.amazon.cloudformation.proxy.hook.targetmodel.ResourceHookTargetModel;

import java.util.List;

public class PreUpdateHookHandler extends BaseHookHandler<TypeConfigurationModel,
    CallbackContext> {

    @Override
    public HookProgressEvent<CallbackContext> handleRequest(
        final AmazonWebServicesClientProxy proxy,
        final HookHandlerRequest request,
        final CallbackContext callbackContext,
        final Logger logger,
        final TypeConfigurationModel typeConfiguration) {

        final String targetName = request.getHookContext().getTargetName();
        if ("AWS::S3::Bucket".equals(targetName)) {
            final ResourceHookTargetModel<AwsS3Bucket> targetModel =
                request.getHookContext().getTargetModel(AwsS3BucketTargetModel.class);

            final AwsS3Bucket bucketProperties =
                targetModel.getResourceProperties();
            final AwsS3Bucket previousBucketProperties =
                targetModel.getPreviousResourceProperties();

            return validateBucketEncryptionRulesNotUpdated(bucketProperties,
                previousBucketProperties);
        } else {
            throw new UnsupportedTargetException(targetName);
        }
    }
}
```

```

private HookProgressEvent<CallbackContext>
validateBucketEncryptionRulesNotUpdated(final AwsS3Bucket resourceProperties,
final AwsS3Bucket previousResourceProperties) {
    final List<ServerSideEncryptionRule> bucketEncryptionConfigs =
resourceProperties.getBucketEncryption().getServerSideEncryptionConfiguration();
    final List<ServerSideEncryptionRule> previousBucketEncryptionConfigs =
previousResourceProperties.getBucketEncryption().getServerSideEncryptionConfiguration();

    if (bucketEncryptionConfigs.size() !=
previousBucketEncryptionConfigs.size()) {
        return HookProgressEvent.<CallbackContext>builder()
            .status(HookStatus.FAILED)
            .errorCode(HandlerErrorCode.NotUpdatable)
            .message(
                String.format(
                    "Current number of bucket encryption configs does not
match previous. Current has %d configs while previously there were %d configs",
                    bucketEncryptionConfigs.size(),
                    previousBucketEncryptionConfigs.size()
                )
            ).build();
    }

    for (int i = 0; i < bucketEncryptionConfigs.size(); ++i) {
        final String currentEncryptionAlgorithm =
bucketEncryptionConfigs.get(i).getServerSideEncryptionByDefault().getSSEAlgorithm();
        final String previousEncryptionAlgorithm =
previousBucketEncryptionConfigs.get(i).getServerSideEncryptionByDefault().getSSEAlgorithm();

        if (!StringUtils.equals(currentEncryptionAlgorithm,
previousEncryptionAlgorithm)) {
            return HookProgressEvent.<CallbackContext>builder()
                .status(HookStatus.FAILED)
                .errorCode(HandlerErrorCode.NotUpdatable)
                .message(
                    String.format(
                        "Bucket Encryption algorithm can not be changed once
set. The encryption algorithm was changed to '%s' from '%s'.",
                        currentEncryptionAlgorithm,
                        previousEncryptionAlgorithm
                    )
                )
                .build();
        }
    }
}

```

```
    }

    return HookProgressEvent.<CallbackContext>builder()
        .status(HookStatus.SUCCESS)
        .message("Successfully invoked PreUpdateHookHandler for target:
AWS::SQS::Queue")
        .build();
    }
}
```

更新preUpdate测试

1. 在您的IDE，打开src/main/java/com/mycompany/testing/mytesthook文件夹中的PreUpdateHandlerTest.java文件。
2. 用以下代码替换PreUpdateHandlerTest.java文件的全部内容。

```
package com.mycompany.testing.mytesthook;

import com.google.common.collect.ImmutableMap;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.BucketEncryption;
import
    com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionByDefault;
import
    com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionRule;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import software.amazon.cloudformation.exceptions.UnsupportedTargetException;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
import software.amazon.cloudformation.proxy.HandlerErrorCode;
import software.amazon.cloudformation.proxy.Logger;
import software.amazon.cloudformation.proxy.hook.HookContext;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import software.amazon.cloudformation.proxy.hook.HookProgressEvent;
import software.amazon.cloudformation.proxy.hook.HookStatus;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;

import java.util.Arrays;
import java.util.stream.Stream;
```

```
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatExceptionOfType;
import static org.mockito.Mockito.mock;

@ExtendWith(MockitoExtension.class)
public class PreUpdateHookHandlerTest {

    @Mock
    private AmazonWebServicesClientProxy proxy;

    @Mock
    private Logger logger;

    @BeforeEach
    public void setup() {
        proxy = mock(AmazonWebServicesClientProxy.class);
        logger = mock(Logger.class);
    }

    @Test
    public void handleRequest_awsS3BucketSuccess() {
        final PreUpdateHookHandler handler = new PreUpdateHookHandler();

        final ServerSideEncryptionRule serverSideEncryptionRule =
            buildServerSideEncryptionRule("AES256");
        final AwsS3Bucket resourceProperties = buildAwsS3Bucket("amzn-s3-demo-
bucket", serverSideEncryptionRule);
        final AwsS3Bucket previousResourceProperties = buildAwsS3Bucket("amzn-s3-
demo-bucket", serverSideEncryptionRule);
        final HookTargetModel targetModel =
            createHookTargetModel(resourceProperties, previousResourceProperties);
        final TypeConfigurationModel typeConfiguration =
            TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

        final HookHandlerRequest request = HookHandlerRequest.builder()

            .hookContext(HookContext.builder().targetName("AWS::S3::Bucket").targetModel(targetModel)
                .build());

        final HookProgressEvent<CallbackContext> response =
            handler.handleRequest(proxy, request, null, logger, typeConfiguration);
        assertResponse(response, HookStatus.SUCCESS, "Successfully invoked
PreUpdateHookHandler for target: AWS::SQS::Queue");
    }
}
```

```
}

@Test
public void handleRequest_awsS3BucketFail_bucketEncryptionConfigsDontMatch() {
    final PreUpdateHookHandler handler = new PreUpdateHookHandler();

    final ServerSideEncryptionRule[] serverSideEncryptionRules =
Stream.of("AES256", "SHA512", "AES32")
        .map(this::buildServerSideEncryptionRule)
        .toArray(ServerSideEncryptionRule[]::new);

    final AwsS3Bucket resourceProperties = buildAwsS3Bucket("amzn-s3-demo-
bucket", serverSideEncryptionRules[0]);
    final AwsS3Bucket previousResourceProperties = buildAwsS3Bucket("amzn-s3-
demo-bucket", serverSideEncryptionRules);
    final HookTargetModel targetModel =
createHookTargetModel(resourceProperties, previousResourceProperties);
    final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

    final HookHandlerRequest request = HookHandlerRequest.builder()

.hookContext(HookContext.builder().targetName("AWS::S3::Bucket").targetModel(targetModel)
        .build());

    final HookProgressEvent<CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);
    assertResponse(response, HookStatus.FAILED, "Current number of bucket
encryption configs does not match previous. Current has 1 configs while previously
there were 3 configs");
}

@Test
public void
handleRequest_awsS3BucketFail_bucketEncryptionAlgorithmDoesNotMatch() {
    final PreUpdateHookHandler handler = new PreUpdateHookHandler();

    final AwsS3Bucket resourceProperties = buildAwsS3Bucket("amzn-s3-demo-
bucket", buildServerSideEncryptionRule("SHA512"));
    final AwsS3Bucket previousResourceProperties = buildAwsS3Bucket("amzn-s3-
demo-bucket", buildServerSideEncryptionRule("AES256"));
    final HookTargetModel targetModel =
createHookTargetModel(resourceProperties, previousResourceProperties);
```



```

    final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

    final HookHandlerRequest request = HookHandlerRequest.builder()

.hookContext(HookContext.builder().targetName("AWS::S3::Bucket").targetModel(targetModel)
    .build());

    final HookProgressEvent<CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);
    assertResponse(response, HookStatus.FAILED, String.format("Bucket
Encryption algorithm can not be changed once set. The encryption algorithm was
changed to '%s' from '%s'.", "SHA512", "AES256"));
}

@Test
public void handleRequest_unsupportedTarget() {
    final PreUpdateHookHandler handler = new PreUpdateHookHandler();

    final Object resourceProperties = ImmutableMap.of("FileSizeLimit", 256);
    final Object previousResourceProperties = ImmutableMap.of("FileSizeLimit",
512);
    final HookTargetModel targetModel =
createHookTargetModel(resourceProperties, previousResourceProperties);
    final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder().encryptionAlgorithm("AES256").build();

    final HookHandlerRequest request = HookHandlerRequest.builder()

.hookContext(HookContext.builder().targetName("AWS::Unsupported::Target").targetModel(targetModel)
    .build());

    assertThatExceptionOfType(UnsupportedTargetException.class)
        .isThrownBy(() -> handler.handleRequest(proxy, request, null,
logger, typeConfiguration))
        .withMessageContaining("Unsupported target")
        .withMessageContaining("AWS::Unsupported::Target")
        .satisfies(e ->
assertThat(e.getErrorCode()).isEqualTo(HandlerErrorCode.InvalidRequest));
}

private void assertResponse(final HookProgressEvent<CallbackContext> response,
final HookStatus expectedStatus, final String expectedErrorMsg) {
    assertThat(response).isNotNull();
}

```

```
        assertThat(response.getStatus()).isEqualTo(expectedStatus);
        assertThat(response.getCallbackContext()).isNull();
        assertThat(response.getCallbackDelaySeconds()).isEqualTo(0);
        assertThat(response.getMessage()).isNotNull();
        assertThat(response.getMessage()).isEqualTo(expectedErrorMsg);
    }

    private HookTargetModel createHookTargetModel(final Object resourceProperties,
final Object previousResourceProperties) {
        return HookTargetModel.of(
            ImmutableMap.of(
                "ResourceProperties", resourceProperties,
                "PreviousResourceProperties", previousResourceProperties
            )
        );
    }

    @SuppressWarnings("SameParameterValue")
    private AwsS3Bucket buildAwsS3Bucket(
        final String bucketName,
        final ServerSideEncryptionRule ...serverSideEncryptionRules
    ) {
        return AwsS3Bucket.builder()
            .bucketName(bucketName)
            .bucketEncryption(
                BucketEncryption.builder()
                    .serverSideEncryptionConfiguration(
                        Arrays.asList(serverSideEncryptionRules)
                    ).build()
            ).build();
    }

    private ServerSideEncryptionRule buildServerSideEncryptionRule(final String
encryptionAlgorithm) {
        return ServerSideEncryptionRule.builder()
            .bucketKeyEnabled(true)
            .serverSideEncryptionByDefault(
                ServerSideEncryptionByDefault.builder()
                    .sSEAlgorithm(encryptionAlgorithm)
                    .build()
            ).build();
    }
}
```

实现处理preDelete程序

实现一个preDelete处理程序，该处理程序在处理程序中所有指定目标的删除操作之前启动。该preDelete处理程序完成以下任务：

- 对于AWS::S3::Bucket资源，只有在满足以下条件时，Hook 才会通过：
 - 验证删除资源后，账户中是否存在所需的最低投诉资源。
 - 所需的最低投诉资源数量在 Hook 的类型配置中设置。

对preDelete处理程序进行编码

1. 在您的IDE，打开src/main/java/com/mycompany/testing/mytesthook文件夹中的PreDeleteHookHandler.java文件。
2. 用以下代码替换PreDeleteHookHandler.java文件的全部内容。

```
package com.mycompany.testing.mytesthook;

import com.google.common.annotations.VisibleForTesting;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3BucketTargetModel;
import com.mycompany.testing.mytesthook.model.aws.sqs.queue.AwsSqsQueue;
import com.mycompany.testing.mytesthook.model.aws.sqs.queue.AwsSqsQueueTargetModel;
import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.math.NumberUtils;
import software.amazon.awssdk.services.cloudformation.CloudFormationClient;
import
    software.amazon.awssdk.services.cloudformation.model.CloudFormationException;
import
    software.amazon.awssdk.services.cloudformation.model.DescribeStackResourceRequest;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.Bucket;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesResponse;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;
import software.amazon.awssdk.services.sqs.model.SqsException;
import software.amazon.cloudformation.exceptions.CfnGeneralServiceException;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
```

```
import software.amazon.cloudformation.proxy.HandlerErrorCode;
import software.amazon.cloudformation.proxy.OperationStatus;
import software.amazon.cloudformation.proxy.ProgressEvent;
import software.amazon.cloudformation.proxy.ProxyClient;
import software.amazon.cloudformation.proxy.hook.HookContext;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;
import
    software.amazon.cloudformation.proxy.hook.targetmodel.ResourceHookTargetModel;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

public class PreDeleteHookHandler extends BaseHookHandlerStd {

    private ProxyClient<S3Client> s3Client;
    private ProxyClient<SqsClient> sqsClient;

    @Override
    protected ProgressEvent<HookTargetModel, CallbackContext>
    handleS3BucketRequest(
        final AmazonWebServicesClientProxy proxy,
        final HookHandlerRequest request,
        final CallbackContext callbackContext,
        final ProxyClient<S3Client> proxyClient,
        final TypeConfigurationModel typeConfiguration
    ) {
        final HookContext hookContext = request.getHookContext();
        final String targetName = hookContext.getTargetName();
        if (!AwsS3Bucket.TYPE_NAME.equals(targetName)) {
            throw new RuntimeException(String.format("Request target type [%s] is
not 'AWS::S3::Bucket'", targetName));
        }
        this.s3Client = proxyClient;

        final String encryptionAlgorithm =
typeConfiguration.getEncryptionAlgorithm();
        final int minBuckets =
NumberUtils.toInt(typeConfiguration.getMinBuckets());
```

```

        final ResourceHookTargetModel<AwsS3Bucket> targetModel =
hookContext.getTargetModel(AwsS3BucketTargetModel.class);
        final List<String> buckets = listBuckets().stream()
            .filter(b -> !StringUtils.equals(b,
targetModel.getResourceProperties().getBucketName()))
            .collect(Collectors.toList());

        final List<String> compliantBuckets = new ArrayList<>();
        for (final String bucket : buckets) {
            if (getBucketSSEAlgorithm(bucket).contains(encryptionAlgorithm)) {
                compliantBuckets.add(bucket);
            }

            if (compliantBuckets.size() >= minBuckets) {
                return ProgressEvent.<HookTargetModel, CallbackContext>builder()
                    .status(OperationStatus.SUCCESS)
                    .message("Successfully invoked PreDeleteHookHandler for
target: AWS::S3::Bucket")
                    .build();
            }
        }

        return ProgressEvent.<HookTargetModel, CallbackContext>builder()
            .status(OperationStatus.FAILED)
            .errorCode(HandlerErrorCode.NonCompliant)
            .message(String.format("Failed to meet minimum of [%d] encrypted
buckets.", minBuckets))
            .build();
    }

    @Override
    protected ProgressEvent<HookTargetModel, CallbackContext>
handleSqsQueueRequest(
        final AmazonWebServicesClientProxy proxy,
        final HookHandlerRequest request,
        final CallbackContext callbackContext,
        final ProxyClient<SqsClient> proxyClient,
        final TypeConfigurationModel typeConfiguration
    ) {
        final HookContext hookContext = request.getHookContext();
        final String targetName = hookContext.getTargetName();
        if (!AwsSqsQueue.TYPE_NAME.equals(targetName)) {
            throw new RuntimeException(String.format("Request target type [%s] is
not 'AWS::SQS::Queue'", targetName));
        }
    }

```

```
    }
    this.sqsClient = proxyClient;
    final int minQueues = NumberUtils.toInt(typeConfiguration.getMinQueues());

    final ResourceHookTargetModel<AwsSqsQueue> targetModel =
hookContext.getTargetModel(AwsSqsQueueTargetModel.class);

    final String queueName =
Objects.toString(targetModel.getResourceProperties().get("QueueName"), null);

    String targetQueueUrl = null;
    if (queueName != null) {
        try {
            targetQueueUrl = sqsClient.injectCredentialsAndInvokeV2(
                GetQueueUrlRequest.builder().queueName(
                    queueName
                ).build(),
                sqsClient.client()::getQueueUrl
            ).queueUrl();
        } catch (SqsException e) {
            log(String.format("Error while calling GetQueueUrl API for queue
name [%s]: %s", queueName, e.getMessage()));
        }
    } else {
        log("Queue name is empty, attempting to get queue's physical ID");
        try {
            final ProxyClient<CloudFormationClient> cfnClient =
proxy.newProxy(ClientBuilder::createCloudFormationClient);
            targetQueueUrl = cfnClient.injectCredentialsAndInvokeV2(
                DescribeStackResourceRequest.builder()
                    .stackName(hookContext.getTargetLogicalId())
                    .logicalResourceId(hookContext.getTargetLogicalId())
                    .build(),
                cfnClient.client()::describeStackResource
            ).stackResourceDetail().physicalResourceId();
        } catch (CloudFormationException e) {
            log(String.format("Error while calling DescribeStackResource API
for queue name: %s", e.getMessage()));
        }
    }

    // Creating final variable for the filter lambda
    final String finalTargetQueueUrl = targetQueueUrl;
```

```
final List<String> compliantQueues = new ArrayList<>();

String nextToken = null;
do {
    final ListQueuesRequest req =
Translator.createListQueuesRequest(nextToken);
    final ListQueuesResponse res =
sqsClient.injectCredentialsAndInvokeV2(req, sqsClient.client()::listQueues);
    final List<String> queueUrls = res.queueUrls().stream()
        .filter(q -> !StringUtils.equals(q, finalTargetQueueUrl))
        .collect(Collectors.toList());

    for (final String queueUrl : queueUrls) {
        if (isQueueEncrypted(queueUrl)) {
            compliantQueues.add(queueUrl);
        }

        if (compliantQueues.size() >= minQueues) {
            return ProgressEvent.<HookTargetModel,
CallbackContext>builder()
                .status(OperationStatus.SUCCESS)
                .message("Successfully invoked PreDeleteHookHandler for
target: AWS::SQS::Queue")
                .build();
        }
        nextToken = res.nextToken();
    }
} while (nextToken != null);

return ProgressEvent.<HookTargetModel, CallbackContext>builder()
    .status(OperationStatus.FAILED)
    .errorCode(HandlerErrorCode.NonCompliant)
    .message(String.format("Failed to meet minimum of [%d] encrypted
queues.", minQueues))
    .build();
}

private List<String> listBuckets() {
    try {
        return
s3Client.injectCredentialsAndInvokeV2(Translator.createListBucketsRequest(),
s3Client.client()::listBuckets)
            .buckets()
    }
}
```

```

        .stream()
        .map(Bucket::name)
        .collect(Collectors.toList());
    } catch (S3Exception e) {
        throw new CfnGeneralServiceException("Error while calling S3
ListBuckets API", e);
    }
}

@VisibleForTesting
Collection<String> getBucketSSEAlgorithm(final String bucket) {
    try {
        return
s3Client.injectCredentialsAndInvokeV2(Translator.createGetBucketEncryptionRequest(bucket),
s3Client.client()::getBucketEncryption)
            .serverSideEncryptionConfiguration()
            .rules()
            .stream()
            .filter(r ->
Objects.nonNull(r.applyServerSideEncryptionByDefault()))
            .map(r ->
r.applyServerSideEncryptionByDefault().sseAlgorithmAsString())
            .collect(Collectors.toSet());
    } catch (S3Exception e) {
        return new HashSet<>();
    }
}

@VisibleForTesting
boolean isQueueEncrypted(final String queueUrl) {
    try {
        final GetQueueAttributesRequest request =
GetQueueAttributesRequest.builder()
            .queueUrl(queueUrl)
            .attributeNames(QueueAttributeName.KMS_MASTER_KEY_ID)
            .build();
        final String kmsKeyId = sqsClient.injectCredentialsAndInvokeV2(request,
sqsClient.client()::getQueueAttributes)
            .attributes()
            .get(QueueAttributeName.KMS_MASTER_KEY_ID);

        return StringUtils.isNotBlank(kmsKeyId);
    } catch (SqsException e) {

```



```
        throw new CfnGeneralServiceException("Error while calling SQS
        GetQueueAttributes API", e);
    }
}
}
```

更新处理preDelete程序

1. 在您的IDE，打开src/main/java/com/mycompany/testing/mytesthook文件夹中的PreDeleteHookHandler.java文件。
2. 用以下代码替换PreDeleteHookHandler.java文件的全部内容。

```
package com.mycompany.testing.mytesthook;

import com.google.common.collect.ImmutableList;
import com.google.common.collect.ImmutableMap;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.Bucket;
import software.amazon.awssdk.services.s3.model.GetBucketEncryptionRequest;
import software.amazon.awssdk.services.s3.model.GetBucketEncryptionResponse;
import software.amazon.awssdk.services.s3.model.ListBucketsRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.ServerSideEncryptionByDefault;
import software.amazon.awssdk.services.s3.model.ServerSideEncryptionConfiguration;
import software.amazon.awssdk.services.s3.model.ServerSideEncryptionRule;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.ListQueuesRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesResponse;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;
import software.amazon.cloudformation.proxy.Logger;
```

```
import software.amazon.cloudformation.proxy.OperationStatus;
import software.amazon.cloudformation.proxy.ProgressEvent;
import software.amazon.cloudformation.proxy.hook.HookContext;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;

import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.stream.Collectors;

import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.never;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

@ExtendWith(MockitoExtension.class)
public class PreDeleteHookHandlerTest extends AbstractTestBase {

    @Mock private S3Client s3Client;
    @Mock private SqsClient sqsClient;
    @Mock private Logger logger;

    @BeforeEach
    public void setup() {
        s3Client = mock(S3Client.class);
        sqsClient = mock(SqsClient.class);
        logger = mock(Logger.class);
    }

    @Test
    public void handleRequest_awsS3BucketSuccess() {
        final PreDeleteHookHandler handler = Mockito.spy(new
PreDeleteHookHandler());

        final List<Bucket> bucketList = ImmutableList.of(
            Bucket.builder().name("bucket1").build(),
            Bucket.builder().name("bucket2").build(),
            Bucket.builder().name("toBeDeletedBucket").build(),
            Bucket.builder().name("bucket3").build(),
            Bucket.builder().name("bucket4").build(),
```

```

        Bucket.builder().name("bucket5").build()
    );
    final ListBucketsResponse mockResponse =
ListBucketsResponse.builder().buckets(bucketList).build();

when(s3Client.listBuckets(any(ListBucketsRequest.class))).thenReturn(mockResponse);
    when(s3Client.getBucketEncryption(any(GetBucketEncryptionRequest.class)))
        .thenReturn(buildGetBucketEncryptionResponse("AES256"))
        .thenReturn(buildGetBucketEncryptionResponse("AES256", "aws:kms"))
        .thenThrow(S3Exception.builder().message("No Encrypt").build())
        .thenReturn(buildGetBucketEncryptionResponse("aws:kms"))
        .thenReturn(buildGetBucketEncryptionResponse("AES256"));
    setServiceClient(s3Client);

    final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder()
        .encryptionAlgorithm("AES256")
        .minBuckets("3")
        .build();

    final HookHandlerRequest request = HookHandlerRequest.builder()
        .hookContext(
            HookContext.builder()
                .targetName("AWS::S3::Bucket")
                .targetModel(
                    createHookTargetModel(
                        AwsS3Bucket.builder()
                            .bucketName("toBeDeletedBucket")
                            .build()
                    )
                )
            )
        .build();

    final ProgressEvent<HookTargetModel, CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);

    verify(s3Client,
times(5)).getBucketEncryption(any(GetBucketEncryptionRequest.class));
    verify(handler, never()).getBucketSSEAlgorithm("toBeDeletedBucket");

    assertResponse(response, OperationStatus.SUCCESS, "Successfully invoked
PreDeleteHookHandler for target: AWS::S3::Bucket");
}

```

```
@Test
public void handleRequest_awsSqsQueueSuccess() {
    final PreDeleteHookHandler handler = Mockito.spy(new
PreDeleteHookHandler());

    final List<String> queueUrls = ImmutableList.of(
        "https://queue1.queue",
        "https://queue2.queue",
        "https://toBeDeletedQueue.queue",
        "https://queue3.queue",
        "https://queue4.queue",
        "https://queue5.queue"
    );

    when(sqsClient.getQueueUrl(any(GetQueueUrlRequest.class)))
        .thenReturn(GetQueueUrlResponse.builder().queueUrl("https://
toBeDeletedQueue.queue").build());
    when(sqsClient.listQueues(any(ListQueuesRequest.class)))

.thenReturn(ListQueuesResponse.builder().queueUrls(queueUrls).build());
    when(sqsClient.getQueueAttributes(any(GetQueueAttributesRequest.class)))

.thenReturn(GetQueueAttributesResponse.builder().attributes(ImmutableMap.of(QueueAttribute
"kmsKeyId")).build())
        .thenReturn(GetQueueAttributesResponse.builder().attributes(new
HashMap<>()).build())

.thenReturn(GetQueueAttributesResponse.builder().attributes(ImmutableMap.of(QueueAttribute
"kmsKeyId")).build())
        .thenReturn(GetQueueAttributesResponse.builder().attributes(new
HashMap<>()).build())

.thenReturn(GetQueueAttributesResponse.builder().attributes(ImmutableMap.of(QueueAttribute
"kmsKeyId")).build());
    setServiceClient(sqsClient);

    final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder()
        .minQueues("3")
        .build();

    final HookHandlerRequest request = HookHandlerRequest.builder()
```

```

        .hookContext(
            HookContext.builder()
                .targetName("AWS::SQS::Queue")
                .targetModel(
                    createHookTargetModel(
                        ImmutableMap.of("QueueName", "toBeDeletedQueue")
                    )
                )
                .build()
        ).build();

        final ProgressEvent<HookTargetModel, CallbackContext> response =
            handler.handleRequest(proxy, request, null, logger, typeConfiguration);

        verify(sqsClient,
            times(5)).getQueueAttributes(any(GetQueueAttributesRequest.class));
        verify(handler, never()).isQueueEncrypted("toBeDeletedQueue");

        assertResponse(response, OperationStatus.SUCCESS, "Successfully invoked
        PreDeleteHookHandler for target: AWS::SQS::Queue");
    }

    @Test
    public void handleRequest_awsS3BucketFailed() {
        final PreDeleteHookHandler handler = Mockito.spy(new
        PreDeleteHookHandler());

        final List<Bucket> bucketList = ImmutableList.of(
            Bucket.builder().name("bucket1").build(),
            Bucket.builder().name("bucket2").build(),
            Bucket.builder().name("toBeDeletedBucket").build(),
            Bucket.builder().name("bucket3").build(),
            Bucket.builder().name("bucket4").build(),
            Bucket.builder().name("bucket5").build()
        );

        final ListBucketsResponse mockResponse =
            ListBucketsResponse.builder().buckets(bucketList).build();

        when(s3Client.listBuckets(any(ListBucketsRequest.class))).thenReturn(mockResponse);
        when(s3Client.getBucketEncryption(any(GetBucketEncryptionRequest.class)))
            .thenReturn(buildGetBucketEncryptionResponse("AES256"))
            .thenReturn(buildGetBucketEncryptionResponse("AES256", "aws:kms"))
            .thenThrow(S3Exception.builder().message("No Encrypt").build())
            .thenReturn(buildGetBucketEncryptionResponse("aws:kms"))
    }

```

```
        .thenReturn(buildGetBucketEncryptionResponse("AES256"));
        setServiceClient(s3Client);

        final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder()
        .encryptionAlgorithm("AES256")
        .minBuckets("10")
        .build();

        final HookHandlerRequest request = HookHandlerRequest.builder()
        .hookContext(
            HookContext.builder()
                .targetName("AWS::S3::Bucket")
                .targetModel(
                    createHookTargetModel(
                        AwsS3Bucket.builder()
                            .bucketName("toBeDeletedBucket")
                            .build()
                    )
                )
            ).build()
        .build();

        final ProgressEvent<HookTargetModel, CallbackContext> response =
handler.handleRequest(proxy, request, null, logger, typeConfiguration);

        verify(s3Client,
times(5)).getBucketEncryption(any(GetBucketEncryptionRequest.class));
        verify(handler, never()).getBucketSSEAlgorithm("toBeDeletedBucket");

        assertResponse(response, OperationStatus.FAILED, "Failed to meet minimum of
[10] encrypted buckets.");
    }

    @Test
    public void handleRequest_awsSqsQueueFailed() {
        final PreDeleteHookHandler handler = Mockito.spy(new
PreDeleteHookHandler());

        final List<String> queueUrls = ImmutableList.of(
            "https://queue1.queue",
            "https://queue2.queue",
            "https://toBeDeletedQueue.queue",
```

```
        "https://queue3.queue",
        "https://queue4.queue",
        "https://queue5.queue"
    );

    when(sqsClient.getQueueUrl(any(GetQueueUrlRequest.class)))
        .thenReturn(GetQueueUrlResponse.builder().queueUrl("https://
toBeDeletedQueue.queue").build());
    when(sqsClient.listQueues(any(ListQueuesRequest.class)))

    .thenReturn(ListQueuesResponse.builder().queueUrls(queueUrls).build());
    when(sqsClient.getQueueAttributes(any(GetQueueAttributesRequest.class)))

    .thenReturn(GetQueueAttributesResponse.builder().attributes(ImmutableMap.of(QueueAttribute
"kmsKeyId")).build())
        .thenReturn(GetQueueAttributesResponse.builder().attributes(new
HashMap<>()).build())

    .thenReturn(GetQueueAttributesResponse.builder().attributes(ImmutableMap.of(QueueAttribute
"kmsKeyId")).build())
        .thenReturn(GetQueueAttributesResponse.builder().attributes(new
HashMap<>()).build())

    .thenReturn(GetQueueAttributesResponse.builder().attributes(ImmutableMap.of(QueueAttribute
"kmsKeyId")).build());
    setServiceClient(sqsClient);

    final TypeConfigurationModel typeConfiguration =
TypeConfigurationModel.builder()
        .minQueues("10")
        .build();

    final HookHandlerRequest request = HookHandlerRequest.builder()
        .hookContext(
            HookContext.builder()
                .targetName("AWS::SQS::Queue")
                .targetModel(
                    createHookTargetModel(
                        ImmutableMap.of("QueueName", "toBeDeletedQueue")
                    )
                )
                .build()
        ).build();
```

```

        final ProgressEvent<HookTargetModel, CallbackContext> response =
            handler.handleRequest(proxy, request, null, logger, typeConfiguration);

        verify(sqsClient,
            times(5)).getQueueAttributes(any(GetQueueAttributesRequest.class));
        verify(handler, never()).isQueueEncrypted("toBeDeletedQueue");

        assertResponse(response, OperationStatus.FAILED, "Failed to meet minimum of
[10] encrypted queues.");
    }

    private GetBucketEncryptionResponse buildGetBucketEncryptionResponse(final
String ...sseAlgorithm) {
        return buildGetBucketEncryptionResponse(
            Arrays.stream(sseAlgorithm)
                .map(a ->
                    ServerSideEncryptionRule.builder().applyServerSideEncryptionByDefault(
                        ServerSideEncryptionByDefault.builder()
                            .sseAlgorithm(a)
                            .build()
                    ).build()
                )
            ).collect(Collectors.toList())
        );
    }

    private GetBucketEncryptionResponse buildGetBucketEncryptionResponse(final
Collection<ServerSideEncryptionRule> rules) {
        return GetBucketEncryptionResponse.builder()
            .serverSideEncryptionConfiguration(
                ServerSideEncryptionConfiguration.builder().rules(
                    rules
                ).build()
            ).build();
    }
}

```

使用 Python 对自定义 AWS CloudFormation 挂钩进行建模

对自定义 AWS CloudFormation Hook 进行建模涉及创建一个用于定义 Hook、其属性和属性的架构。本教程将引导你使用 Python 对自定义挂钩进行建模。

第 1 步：生成 Hook 项目包

生成你的 Hook 项目包。CloudFormation CLI 会创建空的处理函数，这些函数对应于 Hook 规范中定义的目标生命周期中的特定 Hook 操作。

```
cfn generate
```

该命令将返回以下输出。

```
Generated files for MyCompany::Testing::MyTestHook
```

Note

确保您的 Lambda 运行时 up-to-date 避免使用已弃用的版本。有关更多信息，请参阅[更新资源类型的 Lambda 运行时和挂钩](#)。

第 2 步：添加 Hook 处理程序

将您自己的 Hook 处理程序运行时代码添加到您选择实现的处理程序中。例如，您可以添加以下用于记录的代码。

```
LOG.setLevel(logging.INFO)
LOG.info("Internal testing Hook triggered for target: " +
        request.hookContext.targetName);
```

从 CloudFormation CLI 生成 `src/models.py` 文件[配置架构](#)。

Example models.py

```
import sys
from dataclasses import dataclass
from inspect import getmembers, isclass
from typing import (
    AbstractSet,
    Any,
    Generic,
    Mapping,
    MutableMapping,
    Optional,
    Sequence,
```

```
    Type,
    TypeVar,
)

from cloudformation_cli_python_lib.interface import (
    BaseModel,
    BaseHookHandlerRequest,
)
from cloudformation_cli_python_lib.recast import recast_object
from cloudformation_cli_python_lib.utils import deserialize_list

T = TypeVar("T")

def set_or_none(value: Optional[Sequence[T]]) -> Optional[AbstractSet[T]]:
    if value:
        return set(value)
    return None

@dataclass
class HookHandlerRequest(BaseHookHandlerRequest):
    pass

@dataclass
class TypeConfigurationModel(BaseModel):
    limitSize: Optional[str]
    cidr: Optional[str]
    encryptionAlgorithm: Optional[str]

    @classmethod
    def _deserialize(
        cls: Type["_TypeConfigurationModel"],
        json_data: Optional[Mapping[str, Any]],
    ) -> Optional["_TypeConfigurationModel"]:
        if not json_data:
            return None
        return cls(
            limitSize=json_data.get("limitSize"),
            cidr=json_data.get("cidr"),
            encryptionAlgorithm=json_data.get("encryptionAlgorithm"),
        )
```

```
_TypeConfigurationModel = TypeConfigurationModel
```

第 3 步：实现挂钩处理程序

生成的 Python 数据类后，你可以编写实际实现 Hook 功能的处理程序。在此示例中，您将为处理程序实现preCreatepreUpdate、和preDelete调用点。

主题

- [实现处理preCreate程序](#)
- [实现处理preUpdate程序](#)
- [实现处理preDelete程序](#)
- [实现一个 Hook 处理程序](#)

实现处理preCreate程序

preCreate处理程序验证AWS::S3::Bucket 或AWS::SQS::Queue资源的服务器端加密设置。

- 对于AWS::S3::Bucket资源，只有在满足以下条件时，Hook 才会通过。
 - Amazon S3 存储桶加密已设置。
 - 已为该存储桶启用了 Amazon S3 存储桶密钥。
 - 为 Amazon S3 存储桶设置的加密算法是所需的正确算法。
 - 密 AWS Key Management Service 键 ID 已设置。
- 对于AWS::SQS::Queue资源，只有在满足以下条件时，Hook 才会通过。
 - 密 AWS Key Management Service 键 ID 已设置。

实现处理preUpdate程序

实现一个preUpdate处理程序，该处理程序在处理程序中所有指定目标的更新操作之前启动。该preUpdate处理程序完成以下任务：

- 对于AWS::S3::Bucket资源，只有在满足以下条件时，Hook 才会通过：
 - Amazon S3 存储桶的存储桶加密算法尚未修改。

实现处理preDelete程序

实现一个preDelete处理程序，该处理程序在处理程序中所有指定目标的删除操作之前启动。该preDelete处理程序完成以下任务：

- 对于AWS::S3::Bucket资源，只有在满足以下条件时，Hook 才会通过：
 - 验证删除资源后，账户中是否存在所需的最低合规资源。
 - 所需的最低合规资源量在 Hook 的配置中设置。

实现一个 Hook 处理程序

1. 在您的IDE，打开位于src文件夹中的handlers.py文件。
2. 用以下代码替换handlers.py文件的全部内容。

Example handlers.py

```
import logging
from typing import Any, MutableMapping, Optional
import boto3

from cloudformation_cli_python_lib import (
    BaseHookHandlerRequest,
    HandlerErrorCode,
    Hook,
    HookInvocationPoint,
    OperationStatus,
    ProgressEvent,
    SessionProxy,
    exceptions,
)

from .models import HookHandlerRequest, TypeConfigurationModel

# Use this logger to forward log messages to CloudWatch Logs.
LOG = logging.getLogger(__name__)
TYPE_NAME = "MyCompany::Testing::MyTestHook"

LOG.setLevel(logging.INFO)

hook = Hook(TYPE_NAME, TypeConfigurationModel)
test_entrypoint = hook.test_entrypoint
```

```

def _validate_s3_bucket_encryption(
    bucket: MutableMapping[str, Any], required_encryption_algorithm: str
) -> ProgressEvent:
    status = None
    message = ""
    error_code = None

    if bucket:
        bucket_name = bucket.get("BucketName")

        bucket_encryption = bucket.get("BucketEncryption")
        if bucket_encryption:
            server_side_encryption_rules = bucket_encryption.get(
                "ServerSideEncryptionConfiguration"
            )
            if server_side_encryption_rules:
                for rule in server_side_encryption_rules:
                    bucket_key_enabled = rule.get("BucketKeyEnabled")
                    if bucket_key_enabled:
                        server_side_encryption_by_default = rule.get(
                            "ServerSideEncryptionByDefault"
                        )

                        encryption_algorithm =
server_side_encryption_by_default.get(
                            "SSEAlgorithm"
                        )
                        kms_key_id = server_side_encryption_by_default.get(
                            "KMSMasterKeyID"
                        ) # "KMSMasterKeyID" is name of the property for an
AWS::S3::Bucket

                        if encryption_algorithm == required_encryption_algorithm:
                            if encryption_algorithm == "aws:kms" and not
kms_key_id:
                                status = OperationStatus.FAILED
                                message = f"KMS Key ID not set for bucket with
name: f{bucket_name}"
                            else:
                                status = OperationStatus.SUCCESS
                                message = f"Successfully invoked
PreCreateHookHandler for AWS::S3::Bucket with name: {bucket_name}"

```

```

        else:
            status = OperationStatus.FAILED
            message = f"SSE Encryption Algorithm is incorrect for
bucket with name: {bucket_name}"
        else:
            status = OperationStatus.FAILED
            message = f"Bucket key not enabled for bucket with name:
{bucket_name}"

        if status == OperationStatus.FAILED:
            break
    else:
        status = OperationStatus.FAILED
        message = f"No SSE Encryption configurations for bucket with name:
{bucket_name}"
    else:
        status = OperationStatus.FAILED
        message = (
            f"Bucket Encryption not enabled for bucket with name:
{bucket_name}"
        )
    else:
        status = OperationStatus.FAILED
        message = "Resource properties for S3 Bucket target model are empty"

    if status == OperationStatus.FAILED:
        error_code = HandlerErrorCode.NonCompliant

    return ProgressEvent(status=status, message=message, errorCode=error_code)

def _validate_sqs_queue_encryption(queue: MutableMapping[str, Any]) ->
ProgressEvent:
    if not queue:
        return ProgressEvent(
            status=OperationStatus.FAILED,
            message="Resource properties for SQS Queue target model are empty",
            errorCode=HandlerErrorCode.NonCompliant,
        )
    queue_name = queue.get("QueueName")

    kms_key_id = queue.get(
        "KmsMasterKeyId"
    ) # "KmsMasterKeyId" is name of the property for an AWS::SQS::Queue

```

```

    if not kms_key_id:
        return ProgressEvent(
            status=OperationStatus.FAILED,
            message=f"Server side encryption turned off for queue with name:
{queue_name}",
            errorCode=HandlerErrorCode.NonCompliant,
        )

    return ProgressEvent(
        status=OperationStatus.SUCCESS,
        message=f"Successfully invoked PreCreateHookHandler for
targetAWS::SQS::Queue with name: {queue_name}",
    )

@hook.handler(HookInvocationPoint.CREATE_PRE_PROVISION)
def pre_create_handler(
    session: Optional[SessionProxy],
    request: HookHandlerRequest,
    callback_context: MutableMapping[str, Any],
    type_configuration: TypeConfigurationModel,
) -> ProgressEvent:
    target_name = request.hookContext.targetName
    if "AWS::S3::Bucket" == target_name:
        return _validate_s3_bucket_encryption(
            request.hookContext.targetModel.get("resourceProperties"),
            type_configuration.encryptionAlgorithm,
        )
    elif "AWS::SQS::Queue" == target_name:
        return _validate_sqs_queue_encryption(
            request.hookContext.targetModel.get("resourceProperties")
        )
    else:
        raise exceptions.InvalidRequest(f"Unknown target type: {target_name}")

def _validate_bucket_encryption_rules_not_updated(
    resource_properties, previous_resource_properties
) -> ProgressEvent:
    bucket_encryption_configs = resource_properties.get("BucketEncryption",
{}).get(
        "ServerSideEncryptionConfiguration", []
    )
    previous_bucket_encryption_configs = previous_resource_properties.get(

```

```
        "BucketEncryption", {}
    ).get("ServerSideEncryptionConfiguration", [])

    if len(bucket_encryption_configs) != len(previous_bucket_encryption_configs):
        return ProgressEvent(
            status=OperationStatus.FAILED,
            message=f"Current number of bucket encryption configs does not
match previous. Current has {str(len(bucket_encryption_configs))} configs while
previously there were {str(len(previous_bucket_encryption_configs))} configs",
            errorCode=HandlerErrorCode.NonCompliant,
        )

    for i in range(len(bucket_encryption_configs)):
        current_encryption_algorithm = (
            bucket_encryption_configs[i]
            .get("ServerSideEncryptionByDefault", {})
            .get("SSEAlgorithm")
        )
        previous_encryption_algorithm = (
            previous_bucket_encryption_configs[i]
            .get("ServerSideEncryptionByDefault", {})
            .get("SSEAlgorithm")
        )

        if current_encryption_algorithm != previous_encryption_algorithm:
            return ProgressEvent(
                status=OperationStatus.FAILED,
                message=f"Bucket Encryption algorithm can not be changed once
set. The encryption algorithm was changed to {current_encryption_algorithm} from
{previous_encryption_algorithm}.",
                errorCode=HandlerErrorCode.NonCompliant,
            )

    return ProgressEvent(
        status=OperationStatus.SUCCESS,
        message="Successfully invoked PreUpdateHookHandler for target:
AWS::SQS::Queue",
    )

def _validate_queue_encryption_not_disabled(
    resource_properties, previous_resource_properties
) -> ProgressEvent:
    if previous_resource_properties.get(
```



```

        "KmsMasterKeyId"
    ) and not resource_properties.get("KmsMasterKeyId"):
        return ProgressEvent(
            status=OperationStatus.FAILED,
            errorCode=HandlerErrorCode.NonCompliant,
            message="Queue encryption can not be disable",
        )
    else:
        return ProgressEvent(status=OperationStatus.SUCCESS)

@hook.handler(HookInvocationPoint.UPDATE_PRE_PROVISION)
def pre_update_handler(
    session: Optional[SessionProxy],
    request: BaseHookHandlerRequest,
    callback_context: MutableMapping[str, Any],
    type_configuration: MutableMapping[str, Any],
) -> ProgressEvent:
    target_name = request.hookContext.targetName
    if "AWS::S3::Bucket" == target_name:
        resource_properties =
request.hookContext.targetModel.get("resourceProperties")
        previous_resource_properties = request.hookContext.targetModel.get(
            "previousResourceProperties"
        )

        return _validate_bucket_encryption_rules_not_updated(
            resource_properties, previous_resource_properties
        )
    elif "AWS::SQS::Queue" == target_name:
        resource_properties =
request.hookContext.targetModel.get("resourceProperties")
        previous_resource_properties = request.hookContext.targetModel.get(
            "previousResourceProperties"
        )

        return _validate_queue_encryption_not_disabled(
            resource_properties, previous_resource_properties
        )
    else:
        raise exceptions.InvalidRequest(f"Unknown target type: {target_name}")

```

继续进入下一个主题 [向注册自定义 Hook AWS CloudFormation](#)。

向注册自定义 Hook AWS CloudFormation

创建自定义 Hook 后，需要向其注册 AWS CloudFormation 才能使用它。在本节中，你将学习如何打包和注册你的 Hook，以便在你的 AWS 账户。

Package a Hook (Java)

如果你是用 Java 开发的 Hook，请使用 Maven 对其进行打包。

在 Hook 项目的目录中，运行以下命令来构建 Hook，运行单元测试，并将项目打包为可用于将 Hook 提交到 CloudFormation 注册表 JAR 的文件中。

```
mvn clean package
```

注册一个自定义 Hook

注册挂钩

1. (可选) 将您的默认 AWS 区域名称配置为 us-west-2，方法是提交 [configure](#) 操作。

```
$ aws configure
AWS Access Key ID [None]: <Your Access Key ID>
AWS Secret Access Key [None]: <Your Secret Key>
Default region name [None]: us-west-2
Default output format [None]: json
```

2. (可选) 以下命令无需注册即可构建和打包您的 Hook 项目。

```
$ cfn submit --dry-run
```

3. 使用 Hook 注册你的 Hook CloudFormation CLI [submit](#) 操作。

```
$ cfn submit --set-default
```

该命令将返回以下命令。

```
{'ProgressStatus': 'COMPLETE'}
```

结果：您已成功注册您的 Hook。

验证您的账户中是否可以访问 Hook

确认您的 Hook 在您 AWS 账户 和您已提交该挂钩的区域中可用。

1. 要验证您的 Hook，请使用 [list-types](#) 命令列出你新注册的 Hook 并返回它的摘要描述。

```
$ aws cloudformation list-types
```

该命令返回以下输出，还将向您显示可在 AWS 账户 和区域中激活的公开可用的 Hook。

```
{
  "TypeSummaries": [
    {
      "Type": "HOOK",
      "TypeName": "MyCompany::Testing::MyTestHook",
      "DefaultVersionId": "00000001",
      "TypeArn": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID/type/hook/MyCompany-Testing-MyTestHook",
      "LastUpdated": "2021-08-04T23:00:03.058000+00:00",
      "Description": "Verifies S3 bucket and SQS queues properties before creating or updating"
    }
  ]
}
```

2. TypeArn从 Hook 的list-type输出中检索并保存。

```
export HOOK_TYPE_ARN=arn:aws:cloudformation:us-west-2:ACCOUNT_ID/type/hook/MyCompany-Testing-MyTestHook
```

要了解如何发布 Hook 供公众使用，请参阅[发布 Hook 供公众使用](#)。

配置挂钩

开发并注册了 Hook 后，您可以 AWS 账户 通过将其发布到注册表来配置您的 Hook。

- 要在您的账户中配置 Hook，请使用 [SetTypeConfiguration](#) 操作。此操作启用挂钩架构 `properties` 部分中定义的挂钩属性。在以下示例中，该 `minBuckets` 属性在配置1中设置为。

Note

通过在您的账户中启用 Hook，即授权 Hook 使用您的 AWS 账户已定义权限。CloudFormation 在将您的权限传递给 Hook 之前，会移除不需要的权限。CloudFormation 建议客户或 Hook 用户在账户中启用 Hook 之前，先查看 Hook 权限并了解允许 Hook 拥有哪些权限。

在同一个账户中为您注册的 Hook 扩展指定配置数据，然后 AWS 区域。

```
$ aws cloudformation set-type-configuration --region us-west-2
  --configuration '{"CloudFormationConfiguration":{"HookConfiguration":
{"HookInvocationStatus":"ENABLED","FailureMode":"FAIL","Properties":{"minBuckets":
"1","minQueues": "1", "encryptionAlgorithm": "aws:kms"}}}}'
  --type-arn $HOOK_TYPE_ARN
```

Important

要使您的 Hook 能够主动检查堆栈的配置，您必须在账户ENABLED中注册并激活 Hook 之后，在该HookConfiguration部分中将设置为。HookInvocationStatus

AWS APIs在处理程序中访问

如果您的 Hook AWS API 在其任何处理程序中使用了，则 CFN-CLI 会自动创建IAM执行角色模板。hook-role.yaml该hook-role.yaml模板基于在 Hook 架构的处理程序部分中为每个处理程序指定的权限。如果在此期间未使用该--role-arn标志 [generate](#)操作，则此堆栈中的角色将被配置并用作 Hook 的执行角色。

有关更多信息，请参阅[AWS APIs从资源类型访问](#)。

hook-role.yaml 模板

Note

如果您选择创建自己的执行角色，我们强烈建议您遵循最低权限原则，即仅允许上架hooks.cloudformation.amazonaws.com和resources.cloudformation.amazonaws.com。

以下模板使用IAM、Amazon S3 和亚马逊SQS权限。

```
AWSTemplateFormatVersion: 2010-09-09
Description: >
  This CloudFormation template creates a role assumed by CloudFormation during
  Hook operations on behalf of the customer.
Resources:
  ExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      MaxSessionDuration: 8400
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - resources.cloudformation.amazonaws.com
                - hooks.cloudformation.amazonaws.com
            Action: 'sts:AssumeRole'
            Condition:
              StringEquals:
                aws:SourceAccount: !Ref AWS::AccountId
              StringLike:
                aws:SourceArn: !Sub arn:${AWS::Partition}:cloudformation:
${AWS::Region}:${AWS::AccountId}:type/hook/MyCompany-Testing-MyTestHook/*
            Path: /
    Policies:
      - PolicyName: HookTypePolicy
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            - Effect: Allow
              Action:
                - 's3:GetEncryptionConfiguration'
                - 's3:ListBucket'
                - 's3:ListAllMyBuckets'
                - 'sqs:GetQueueAttributes'
                - 'sqs:GetQueueUrl'
                - 'sqs:ListQueues'
              Resource: '*'
Outputs:
  ExecutionRoleArn:
    Value: !GetAtt
```

- ExecutionRole
- Arn

在你的 Hook 中测试自定义 Hook AWS 账户

既然你已经编码了与调用点对应的处理函数，那么是时候在 CloudFormation 堆栈上测试你的自定义 Hook 了。

FAIL 如果 CloudFormation 模板未使用以下内容配置 S3 存储桶，则挂钩失败模式将设置为：

- Amazon S3 存储桶加密已设置。
- 已为该存储桶启用了 Amazon S3 存储桶密钥。
- 为 Amazon S3 存储桶设置的加密算法是所需的正确算法。
- 密 AWS Key Management Service 钥 ID 已设置。

在以下示例中，创建一个名为的模板，其堆栈 `my-failed-bucket-stack.yml` 名称为 `my-hook-stack`，该模板无法配置堆栈并在资源配置之前停止。

通过配置堆栈来测试 Hook

示例 1：配置堆栈

配置不合规的堆栈

1. 创作一个指定 S3 存储桶的模板。例如，`my-failed-bucket-stack.yml`。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  S3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties: {}
```

2. 创建堆栈，然后在 AWS Command Line Interface (AWS CLI) 中指定您的模板。在以下示例中，将堆栈名称指定为 `my-hook-stack`，将模板名称指定为 `my-failed-bucket-stack.yml`。

```
$ aws cloudformation create-stack \
  --stack-name my-hook-stack \
  --template-body file:///my-failed-bucket-stack.yml
```

3. (可选) 通过指定堆栈名称来查看堆栈进度。在以下示例中，指定堆栈名称 `my-hook-stack`。

```
$ aws cloudformation describe-stack-events \  
  --stack-name my-hook-stack
```

创建存储桶时，使用describe-stack-events操作查看挂钩失败。以下是该命令的输出示例。

```
{  
  "StackEvents": [  
    ...  
    {  
      "StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-hook-stack/2c693970-f57e-11eb-a0fb-061a2a83f0b9",  
      "EventId": "S3Bucket-CREATE_FAILED-2021-08-04T23:47:03.305Z",  
      "StackName": "my-hook-stack",  
      "LogicalResourceId": "S3Bucket",  
      "PhysicalResourceId": "",  
      "ResourceType": "AWS::S3::Bucket",  
      "Timestamp": "2021-08-04T23:47:03.305000+00:00",  
      "ResourceStatus": "CREATE_FAILED",  
      "ResourceStatusReason": "The following hook(s) failed:  
[MyCompany::Testing::MyTestHook]",  
      "ResourceProperties": "{}",  
      "ClientRequestToken": "Console-CreateStack-abe71ac2-ade4-a762-0499-8d34d91d6a92"  
    },  
    ...  
  ]  
}
```

结果：Hook 调用使堆栈配置失败，并停止了资源置备。

使用 CloudFormation 模板通过 Hook 验证

1. 要创建堆栈并通过 Hook 验证，请更新模板，使您的资源使用加密的 S3 存储桶。此示例使用 my-encrypted-bucket-stack.yml 模板。

```
AWSTemplateFormatVersion: 2010-09-09  
Description: |  
  This CloudFormation template provisions an encrypted S3 Bucket  
Resources:  
  EncryptedS3Bucket:
```

```

Type: 'AWS::S3::Bucket'
Properties:
  BucketName: !Sub 'encryptedbucket-${AWS::Region}-${AWS::AccountId}'
  BucketEncryption:
    ServerSideEncryptionConfiguration:
      - ServerSideEncryptionByDefault:
          SSEAlgorithm: 'aws:kms'
          KMSMasterKeyID: !Ref EncryptionKey
          BucketKeyEnabled: true
EncryptionKey:
  Type: 'AWS::KMS::Key'
  DeletionPolicy: Retain
  Properties:
    Description: KMS key used to encrypt the resource type artifacts
    EnableKeyRotation: true
    KeyPolicy:
      Version: 2012-10-17
      Statement:
        - Sid: Enable full access for owning account
          Effect: Allow
          Principal:
            AWS: !Ref 'AWS::AccountId'
          Action: 'kms:*'
          Resource: '*'
Outputs:
  EncryptedBucketName:
    Value: !Ref EncryptedS3Bucket

```

Note

不会为跳过的资源调用 Hook。

2. 创建堆栈并指定您的模板。在此示例中，堆栈名称为my-encrypted-bucket-stack。

```

$ aws cloudformation create-stack \
  --stack-name my-encrypted-bucket-stack \
  --template-body file:///my-encrypted-bucket-stack.yml \

```

3. (可选) 通过指定堆栈名称来查看堆栈进度。

```

$ aws cloudformation describe-stack-events \
  --stack-name my-encrypted-bucket-stack

```


使用describe-stack-events命令查看响应。以下是 describe-stack-events 命令的示例。

```
{
  "StackEvents": [
    ...
    {
      "StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-encrypted-bucket-stack/82a97150-f57a-11eb-8eb2-06a6bdcc7779",
      "EventId": "EncryptedS3Bucket-CREATE_COMPLETE-2021-08-04T23:23:20.973Z",
      "StackName": "my-encrypted-bucket-stack",
      "LogicalResourceId": "EncryptedS3Bucket",
      "PhysicalResourceId": "encryptedbucket-us-west-2-ACCOUNT_ID",
      "ResourceType": "AWS::S3::Bucket",
      "Timestamp": "2021-08-04T23:23:20.973000+00:00",
      "ResourceStatus": "CREATE_COMPLETE",
      "ResourceProperties": "{\"BucketName\":\"encryptedbucket-us-west-2-071617338693\", \"BucketEncryption\":{\"ServerSideEncryptionConfiguration\": [{\"BucketKeyEnabled\":\"true\", \"ServerSideEncryptionByDefault\":{\"SSEAlgorithm\":\"aws:kms\", \"KMSMasterKeyID\":\"ENCRYPTION_KEY_ARN\"}}]}}",
      "ClientRequestToken": "Console-CreateStack-39df35ac-ca00-b7f6-5661-4e917478d075"
    },
    {
      "StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-encrypted-bucket-stack/82a97150-f57a-11eb-8eb2-06a6bdcc7779",
      "EventId": "EncryptedS3Bucket-CREATE_IN_PROGRESS-2021-08-04T23:22:59.410Z",
      "StackName": "my-encrypted-bucket-stack",
      "LogicalResourceId": "EncryptedS3Bucket",
      "PhysicalResourceId": "encryptedbucket-us-west-2-ACCOUNT_ID",
      "ResourceType": "AWS::S3::Bucket",
      "Timestamp": "2021-08-04T23:22:59.410000+00:00",
      "ResourceStatus": "CREATE_IN_PROGRESS",
      "ResourceStatusReason": "Resource creation Initiated",
      "ResourceProperties": "{\"BucketName\":\"encryptedbucket-us-west-2-071617338693\", \"BucketEncryption\":{\"ServerSideEncryptionConfiguration\": [{\"BucketKeyEnabled\":\"true\", \"ServerSideEncryptionByDefault\":{\"SSEAlgorithm\":\"aws:kms\", \"KMSMasterKeyID\":\"ENCRYPTION_KEY_ARN\"}}]}}",
      "ClientRequestToken": "Console-CreateStack-39df35ac-ca00-b7f6-5661-4e917478d075"
    }
  ]
}
```

```

    },
    {
      "StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-
encrypted-bucket-stack/82a97150-f57a-11eb-8eb2-06a6bdcc7779",
      "EventId": "EncryptedS3Bucket-6516081f-c1f2-4bfe-a0f0-cefa28679994",
      "StackName": "my-encrypted-bucket-stack",
      "LogicalResourceId": "EncryptedS3Bucket",
      "PhysicalResourceId": "",
      "ResourceType": "AWS::S3::Bucket",
      "Timestamp": "2021-08-04T23:22:58.349000+00:00",
      "ResourceStatus": "CREATE_IN_PROGRESS",
      "ResourceStatusReason": "Hook invocations complete. Resource creation
initiated",
      "ClientRequestToken": "Console-CreateStack-39df35ac-ca00-
b7f6-5661-4e917478d075"
    },
    ...
  ]
}

```

结果：CloudFormation 成功创建堆栈。在配置资源之前，Hook 的逻辑验证了AWS::S3::Bucket资源是否包含服务器端加密。

示例 2：配置堆栈

配置不合规的堆栈

1. 创作一个指定 S3 存储桶的模板。例如 aes256-bucket.yml。

```

AWSTemplateFormatVersion: 2010-09-09
Description: |
  This CloudFormation template provisions an encrypted S3 Bucket
Resources:
  EncryptedS3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: !Sub 'encryptedbucket-${AWS::Region}-${AWS::AccountId}'
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: AES256
            BucketKeyEnabled: true

```

```
Outputs:
  EncryptedBucketName:
    Value: !Ref EncryptedS3Bucket
```

2. 创建堆栈，并在其中指定您的模板 AWS CLI。在以下示例中，将堆栈名称指定为 `my-hook-stack`，将模板名称指定为 `aes256-bucket.yml`。

```
$ aws cloudformation create-stack \
  --stack-name my-hook-stack \
  --template-body file://aes256-bucket.yml
```

3. (可选) 通过指定堆栈名称来查看堆栈进度。在以下示例中，指定堆栈名称 `my-hook-stack`。

```
$ aws cloudformation describe-stack-events \
  --stack-name my-hook-stack
```

创建存储桶时，使用 `describe-stack-events` 操作查看挂钩失败。以下是该命令的输出示例。

```
{
  "StackEvents": [
    ...
    {
      "StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-hook-stack/2c693970-f57e-11eb-a0fb-061a2a83f0b9",
      "EventId": "S3Bucket-CREATE_FAILED-2021-08-04T23:47:03.305Z",
      "StackName": "my-hook-stack",
      "LogicalResourceId": "S3Bucket",
      "PhysicalResourceId": "",
      "ResourceType": "AWS::S3::Bucket",
      "Timestamp": "2021-08-04T23:47:03.305000+00:00",
      "ResourceStatus": "CREATE_FAILED",
      "ResourceStatusReason": "The following hook(s) failed:
[MyCompany::Testing::MyTestHook]",
      "ResourceProperties": "{}",
      "ClientRequestToken": "Console-CreateStack-abe71ac2-ade4-a762-0499-8d34d91d6a92"
    },
    ...
  ]
}
```

结果：Hook 调用使堆栈配置失败，并停止了资源置备。由于 S3 存储桶加密配置不正确，堆栈失败。挂钩类型配置需要在此存储桶使用aws:kms时使用AES256。

使用 CloudFormation 模板通过 Hook 验证

1. 要创建堆栈并通过 Hook 验证，请更新模板，使您的资源使用加密的 S3 存储桶。此示例使用 kms-bucket-and-queue.yml 模板。

```
AWSTemplateFormatVersion: 2010-09-09
Description: |
  This CloudFormation template provisions an encrypted S3 Bucket
Resources:
  EncryptedS3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: !Sub 'encryptedbucket-${AWS::Region}-${AWS::AccountId}'
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: 'aws:kms'
              KMSMasterKeyID: !Ref EncryptionKey
              BucketKeyEnabled: true
  EncryptedQueue:
    Type: 'AWS::SQS::Queue'
    Properties:
      QueueName: 'encryptedqueue-${AWS::Region}-${AWS::AccountId}'
      KmsMasterKeyId: !Ref EncryptionKey
  EncryptionKey:
    Type: 'AWS::KMS::Key'
    DeletionPolicy: Retain
    Properties:
      Description: KMS key used to encrypt the resource type artifacts
      EnableKeyRotation: true
      KeyPolicy:
        Version: 2012-10-17
        Statement:
          - Sid: Enable full access for owning account
            Effect: Allow
            Principal:
              AWS: !Ref 'AWS::AccountId'
            Action: 'kms:*'
```

```

Resource: '*'

Outputs:
  EncryptedBucketName:
    Value: !Ref EncryptedS3Bucket
  EncryptedQueueName:
    Value: !Ref EncryptedQueue

```

Note

不会为跳过的资源调用 Hook。

2. 创建堆栈并指定您的模板。在此示例中，堆栈名称为 `my-encrypted-bucket-stack`。

```

$ aws cloudformation create-stack \
  --stack-name my-encrypted-bucket-stack \
  --template-body file://kms-bucket-and-queue.yml

```

3. (可选) 通过指定堆栈名称来查看堆栈进度。

```

$ aws cloudformation describe-stack-events \
  --stack-name my-encrypted-bucket-stack

```

使用 `describe-stack-events` 命令查看响应。以下是 `describe-stack-events` 命令的示例。

```

{
  "StackEvents": [
    ...
    {
      "StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-encrypted-bucket-stack/82a97150-f57a-11eb-8eb2-06a6bdcc7779",
      "EventId": "EncryptedS3Bucket-CREATE_COMPLETE-2021-08-04T23:23:20.973Z",
      "StackName": "my-encrypted-bucket-stack",
      "LogicalResourceId": "EncryptedS3Bucket",
      "PhysicalResourceId": "encryptedbucket-us-west-2-ACCOUNT_ID",
      "ResourceType": "AWS::S3::Bucket",
      "Timestamp": "2021-08-04T23:23:20.973000+00:00",
      "ResourceStatus": "CREATE_COMPLETE",
      "ResourceProperties": "{\"BucketName\": \"encryptedbucket-us-west-2-071617338693\", \"BucketEncryption\": {\"ServerSideEncryptionConfiguration\":

```

```
[{"BucketKeyEnabled": "true", "ServerSideEncryptionByDefault": {"SSEAlgorithm": "aws:kms", "KMSMasterKeyID": "ENCRYPTION_KEY_ARM"}}, {"ClientRequestToken": "Console-CreateStack-39df35ac-ca00-b7f6-5661-4e917478d075"}, {"StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-encrypted-bucket-stack/82a97150-f57a-11eb-8eb2-06a6bdcc7779", "EventId": "EncryptedS3Bucket-CREATE_IN_PROGRESS-2021-08-04T23:22:59.410Z", "StackName": "my-encrypted-bucket-stack", "LogicalResourceId": "EncryptedS3Bucket", "PhysicalResourceId": "encryptedbucket-us-west-2-ACCOUNT_ID", "ResourceType": "AWS::S3::Bucket", "Timestamp": "2021-08-04T23:22:59.410000+00:00", "ResourceStatus": "CREATE_IN_PROGRESS", "ResourceStatusReason": "Resource creation initiated", "ResourceProperties": {"BucketName": "encryptedbucket-us-west-2-071617338693", "BucketEncryption": {"ServerSideEncryptionConfiguration": [{"BucketKeyEnabled": "true", "ServerSideEncryptionByDefault": {"SSEAlgorithm": "aws:kms", "KMSMasterKeyID": "ENCRYPTION_KEY_ARM"}}]}, {"ClientRequestToken": "Console-CreateStack-39df35ac-ca00-b7f6-5661-4e917478d075"}, {"StackId": "arn:aws:cloudformation:us-west-2:ACCOUNT_ID:stack/my-encrypted-bucket-stack/82a97150-f57a-11eb-8eb2-06a6bdcc7779", "EventId": "EncryptedS3Bucket-6516081f-c1f2-4bfe-a0f0-cefa28679994", "StackName": "my-encrypted-bucket-stack", "LogicalResourceId": "EncryptedS3Bucket", "PhysicalResourceId": "", "ResourceType": "AWS::S3::Bucket", "Timestamp": "2021-08-04T23:22:58.349000+00:00", "ResourceStatus": "CREATE_IN_PROGRESS", "ResourceStatusReason": "Hook invocations complete. Resource creation initiated", "ClientRequestToken": "Console-CreateStack-39df35ac-ca00-b7f6-5661-4e917478d075"}, ... ] }
```

结果：CloudFormation 成功创建堆栈。在配置资源之前，Hook 的逻辑验证了AWS::S3::Bucket资源是否包含服务器端加密。

更新自定义 Hook

更新自定义 Hook 允许在 CloudFormation注册表中提供 Hook 中的修订版本。

要更新自定义 Hook，请通过以下方式将您的修订版提交到 CloudFormation 注册表 CloudFormation CLI [submit](#)操作。

```
$ cfn submit
```

要在账户中指定 Hook 的默认版本，请使用 [set-type-default-version](#)命令并指定类型、类型名称和版本 ID。

```
$ aws cloudformation set-type-default-version \  
  --type HOOK \  
  --type-name MyCompany::Testing::MyTestHook \  
  --version-id 00000003
```

要检索有关 Hook 版本的信息，请使用 [list-type-versions](#).

```
$ aws cloudformation list-type-versions \  
  --type HOOK \  
  --type-name "MyCompany::Testing::MyTestHook"
```

从注册表中取消注册自定义 Hook CloudFormation

取消注册自定义 Hook 会将扩展或扩展版本标记为 CloudFormation 注册表DEPRECATED中的版本，这会将其从活动使用中删除。自定义 Hook 一旦被弃用，就无法在 CloudFormation 操作中使用。

Note

在取消注册 Hook 之前，必须单独取消注册该扩展的所有先前活动版本。有关更多信息，请参阅 [DeregisterType](#).

要取消注册 Hook，请使用 [deregister-type](#)操作并指定您的 Hook ARN。

```
$ aws cloudformation deregister-type \  
  --arn HOOK_TYPE_ARN
```

此命令不产生输出。

发布 Hook 供公众使用

要开发公共的第三方 Hook，请将您的 Hook 开发为私有扩展。然后，AWS 区域 在每个你想公开扩展程序的内容中：

1. 在注册表中将您的 Hook CloudFormation 注册为私有扩展。
2. 测试您的 Hook，确保它满足在 CloudFormation 注册表中发布的所有必要要求。
3. 将您的 Hook 发布到 CloudFormation 注册表。

Note

在给定的区域发布任何扩展之前，必须先在该地区注册为扩展发布者。要同时在多个区域执行此操作，请参阅 [AWS CloudFormation CLI 用户指南中的使用 StackSets 在多个区域发布扩展程序](#)。

开发并注册了 Hook 后，您可以将其作为第三方公共扩展发布到 CloudFormation 注册表，使其向普通 CloudFormation 用户公开。

公共第三方 Hook 使您能够让 CloudFormation 用户在配置之前主动检查 AWS 资源配置。与私有 Hook 一样，公共 Hook 的处理方式与 within 发布的 AWS 任何 Hook 相同 CloudFormation。

发布到注册表的挂钩对发布到注册表 AWS 区域的所有 CloudFormation 用户都可见。然后，用户可以在其帐户中激活您的扩展程序，从而使其可在模板中使用。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的 [“使用 CloudFormation 注册表中的第三方公共扩展”](#)。

测试供公众使用的自定义 Hook

要发布您注册的自定义 Hook，它必须通过为其定义的所有测试要求。以下是将您的自定义 Hook 发布为第三方扩展之前所需的要求列表。

每个处理程序和目标都要经过两次测试。一次为 SUCCESS，一次为 FAILED。

- 对于 SUCCESS 响应案例：

- 状态必须是SUCCESS。
- 不得返回错误代码。
- 如果已指定，则应将回调延迟设置为0秒。
- 对于FAILED响应案例：
 - 状态必须是FAILED。
 - 必须返回错误代码。
 - 必须有消息作为回应。
 - 如果已指定，则应将回调延迟设置为0秒。
- 对于IN_PROGRESS响应案例：
 - 不得返回错误代码。
 - Result不得在响应时设置字段。

指定用于合约测试的输入数据

默认情况下，使用根据您在 Hook 架构中定义的模式生成的输入属性来 CloudFormation 执行合约测试。但是，大多数 Hook 都非常复杂，以至于用于预创建或预更新配置堆栈的输入属性需要了解所配置的资源。为了解决这个问题，你可以指定在执行合约测试时 CloudFormation 使用的输入。

CloudFormation 提供了两种方法供您指定在执行合约测试时使用的输入数据：

- 覆盖文件

使用overrides文件提供了一种轻量级的方法来指定某些特定属性的输入数据preCreate，CloudFormation 以便在操作测试preUpdate和preDelete操作测试期间使用。

- 输入文件

在以下情况下，您也可以使用多个input文件来指定合约测试输入数据：

- 您想要或需要为创建、更新和删除操作指定不同的输入数据，或者为测试指定无效数据。
- 您想要指定多个不同的输入数据集。

使用替代文件指定输入数据

以下是 Amazon S3 Hook 使用该overrides文件输入数据的示例。

```
{  
  "CREATE_PRE_PROVISION": {
```

```
"AWS::S3::Bucket": {
  "resourceProperties": {
    "/BucketName": "encryptedbucket-us-west-2-contractor",
    "/BucketEncryption/ServerSideEncryptionConfiguration": [
      {
        "BucketKeyEnabled": true,
        "ServerSideEncryptionByDefault": {
          "KMSEncryptionContext": "KMS-KEY-ARN",
          "SSEAlgorithm": "aws:kms"
        }
      }
    ]
  }
},
"AWS::SQS::Queue": {
  "resourceProperties": {
    "/QueueName": "MyQueueContract",
    "/KmsMasterKeyId": "hellocontract"
  }
},
"UPDATE_PRE_PROVISION": {
  "AWS::S3::Bucket": {
    "resourceProperties": {
      "/BucketName": "encryptedbucket-us-west-2-contractor",
      "/BucketEncryption/ServerSideEncryptionConfiguration": [
        {
          "BucketKeyEnabled": true,
          "ServerSideEncryptionByDefault": {
            "KMSEncryptionContext": "KMS-KEY-ARN",
            "SSEAlgorithm": "aws:kms"
          }
        }
      ]
    },
    "previousResourceProperties": {
      "/BucketName": "encryptedbucket-us-west-2-contractor",
      "/BucketEncryption/ServerSideEncryptionConfiguration": [
        {
          "BucketKeyEnabled": true,
          "ServerSideEncryptionByDefault": {
            "KMSEncryptionContext": "KMS-KEY-ARN",
            "SSEAlgorithm": "aws:kms"
          }
        }
      ]
    }
  }
}
```

```

    }
  ]
}
},
"INVALID_UPDATE_PRE_PROVISION": {
  "AWS::S3::Bucket": {
    "resourceProperties": {
      "/BucketName": "encryptedbucket-us-west-2-contractor",
      "/BucketEncryption/ServerSideEncryptionConfiguration": [
        {
          "BucketKeyEnabled": true,
          "ServerSideEncryptionByDefault": {
            "KMSEncryptionKeyId": "KMS-KEY-ARN",
            "SSEAlgorithm": "AES256"
          }
        }
      ]
    },
    "previousResourceProperties": {
      "/BucketName": "encryptedbucket-us-west-2-contractor",
      "/BucketEncryption/ServerSideEncryptionConfiguration": [
        {
          "BucketKeyEnabled": true,
          "ServerSideEncryptionByDefault": {
            "KMSEncryptionKeyId": "KMS-KEY-ARN",
            "SSEAlgorithm": "aws:kms"
          }
        }
      ]
    }
  }
},
"INVALID": {
  "AWS::SQS::Queue": {
    "resourceProperties": {
      "/QueueName": "MyQueueContract",
      "/KmsMasterKeyId": "KMS-KEY-ARN"
    }
  }
}
}

```

使用输入文件指定输入数据

使用input文件来指定要使用的不同类型的输入数据：preCreate输入、preUpdate输入和无效输入。CloudFormation 每种数据都在单独的文件中指定。您还可以为合约测试指定多组输入数据。

要指定要在合约测试中使用的input文件，请在 Hooks 项目的根目录中添加一个inputs文件夹。CloudFormation 然后添加您的输入文件。

使用以下命名约定指定文件包含哪种输入数据，其中*n*为整数：

- `inputs_n_pre_create.json`：使用带有preCreate处理程序的文件来指定用于创建资源的输入。
- `inputs_n_pre_update.json`：使用带有preUpdate处理程序的文件来指定用于更新资源的输入。
- `inputs_n_pre_delete.json`：使用带有preDelete处理程序的文件来指定删除资源的输入。
- `inputs_n_invalid.json`：用于指定要测试的无效输入。

要为合约测试指定多组输入数据，请增加文件名中的整数以对输入数据集进行排序。例如，您的第一组输入文件应命名为`inputs_1_pre_create.json`、`inputs_1_pre_update.json`和`inputs_1_pre_invalid.json`。您的下一组将命名为`inputs_2_pre_create.json`、`inputs_2_pre_update.json`和`inputs_2_pre_invalid.json`，依此类推。

每个输入文件都是一个仅包含用于测试的资源属性的JSON文件。

以下是使用输入文件inputs Amazon S3 指定输入数据的示例目录。

`inputs_1_pre_create.json`

以下是`inputs_1_pre_create.json`合同测试的示例。

```
{
  "AWS::S3::Bucket": {
    "resourceProperties": {
      "AccessControl": "BucketOwnerFullControl",
      "AnalyticsConfigurations": [],
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
          {
            "BucketKeyEnabled": true,
            "ServerSideEncryptionByDefault": {
```

```

        "KMSMasterKeyId": "KMS-KEY-ARN",
        "SSEAlgorithm": "aws:kms"
      }
    ]
  },
  "BucketName": "encryptedbucket-us-west-2"
}
},
"AWS::SQS::Queue": {
  "resourceProperties": {
    "QueueName": "MyQueue",
    "KmsMasterKeyId": "KMS-KEY-ARN"
  }
}
}
}

```

inputs_1_pre_update.json

以下是inputs_1_pre_update.json合同测试的示例。

```

{
  "AWS::S3::Bucket": {
    "resourceProperties": {
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
          {
            "BucketKeyEnabled": true,
            "ServerSideEncryptionByDefault": {
              "KMSMasterKeyId": "KMS-KEY-ARN",
              "SSEAlgorithm": "aws:kms"
            }
          }
        ]
      },
      "BucketName": "encryptedbucket-us-west-2"
    },
    "previousResourceProperties": {
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
          {
            "BucketKeyEnabled": true,
            "ServerSideEncryptionByDefault": {
              "KMSMasterKeyId": "KMS-KEY-ARN",

```

```

        "SSEAlgorithm": "aws:kms"
      }
    }
  ],
  },
  "BucketName": "encryptedbucket-us-west-2"
}
}
}

```

inputs_1_invalid.json

以下是inputs_1_invalid.json合同测试的示例。

```

{
  "AWS::S3::Bucket": {
    "resourceProperties": {
      "AccessControl": "BucketOwnerFullControl",
      "AnalyticsConfigurations": [],
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
          {
            "ServerSideEncryptionByDefault": {
              "SSEAlgorithm": "AES256"
            }
          }
        ]
      },
      "BucketName": "encryptedbucket-us-west-2"
    }
  },
  "AWS::SQS::Queue": {
    "resourceProperties": {
      "NotValid": "The property of this resource is not valid."
    }
  }
}

```

inputs_1_invalid_pre_update.json

以下是inputs_1_invalid_pre_update.json合同测试的示例。

```

{

```

```
"AWS::S3::Bucket": {
  "resourceProperties": {
    "BucketEncryption": {
      "ServerSideEncryptionConfiguration": [
        {
          "BucketKeyEnabled": true,
          "ServerSideEncryptionByDefault": {
            "KMSMasterKeyID": "KMS-KEY-ARN",
            "SSEAlgorithm": "AES256"
          }
        }
      ]
    },
    "BucketName": "encryptedbucket-us-west-2"
  },
  "previousResourceProperties": {
    "BucketEncryption": {
      "ServerSideEncryptionConfiguration": [
        {
          "BucketKeyEnabled": true,
          "ServerSideEncryptionByDefault": {
            "KMSMasterKeyID": "KMS-KEY-ARN",
            "SSEAlgorithm": "aws:kms"
          }
        }
      ]
    },
    "BucketName": "encryptedbucket-us-west-2"
  }
}
}
```

有关更多信息，请参阅 [《AWS CloudFormation CLI 用户指南》](#) 中的发布扩展程序以使其可供公众使用。

H AWS CloudFormation Hooks 的架构语法参考

本节介绍用于开发 AWS CloudFormation Hook 的架构的语法。

Hook 包括由 JSON 架构和 Hook 处理程序表示的 Hook 规范。创建自定义 Hook 的第一步是对定义挂钩及其属性的架构进行建模。当你使用初始化自定义 Hook 项目时 CloudFormation CLI [init](#) 命令，将为您创建一个 Hook 架构文件。使用此架构文件作为定义自定义 Hook 的形状和语义的起点。

架构语法

以下架构是 Hook 的结构。

```
{
  "typeName": "string",
  "description": "string",
  "sourceUrl": "string",
  "documentationUrl": "string",
  "definitions": {
    "definitionName": {
      . . .
    }
  },
  "typeConfiguration": {
    "properties": {
      "propertyName": {
        "description": "string",
        "type": "string",
        . . .
      },
    },
  },
  "required": [
    "propertyName"
    . . .
  ],
  "additionalProperties": false
},
"handlers": {
  "preCreate": {
    "targetNames": [
    ],
    "permissions": [
    ]
  },
  "preUpdate": {
    "targetNames": [
    ],
    "permissions": [
    ]
  },
  "preDelete": {
    "targetNames": [
    ],
  },
}
```



```

        "permissions": [
        ]
    },
    "additionalProperties": false
}

```

typeName

您的 Hook 的唯一名称。为 Hook 指定一个由三部分组成的命名空间，推荐的 `Organization::Service::Hook` 模式为。

Note

以下组织命名空间已预留，不能用于您的 Hook 类型名称：

- Alexa
- AMZN
- Amazon
- ASK
- AWS
- Custom
- Dev

必需：是

图案：`^[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}$`

最小值：10

最大值：196

description

CloudFormation 控制台中显示的 Hook 的简短描述。

必需：是

sourceUrl

Hook URL 的源代码（如果是公开的）。

必需：否

最大值：4096

documentationUrl

提供 Hook 详细文档的页面中的 URL。

必需：是

图案：`^https\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])(\:[0-9]*)*([\?/#].*)?$`

最大值：4096

Note

尽管 Hook 架构应包含完整而准确的属性描述，但您可以使用该 `documentationUrl` 属性为用户提供更多详细信息，包括示例、用例和其他详细信息。

definitions

使用该 `definitions` 块提供共享的 Hook 属性架构。

使用该 `definitions` 部分来定义可在您的 Hook 类型架构中的多个位置使用的架构元素被认为是一种最佳实践。然后，您可以使用指 JSON 针在 Hook 类型架构中的相应位置引用该元素。

必需：否

typeConfiguration

Hook 配置数据的定义。

必需：是

properties

挂钩的属性。Hook 的所有属性都必须在架构中表达。将 Hook 架构属性与挂钩类型配置属性对齐。

Note

不允许使用嵌套属性。相反，在 `definitions` 元素中定义任何嵌套属性，然后使用 `$ref` 指针在所需的属性中引用它们。

additionalProperties

`additionalProperties` 必须设置为 `false`。Hook 的所有属性都必须在架构中表达：不允许任意输入。

必需：是

有效值：`false`

handlers

处理程序指定可以启动架构中定义的 Hook 的操作，例如 Hook 调用点。例如，`preUpdate` 处理程序是在处理程序中所有指定目标的更新操作之前调用的。

有效值：`preCreate` | `preUpdate` | `preDelete`

Note

必须为处理程序指定至少一个值。

Important

导致状态为的堆栈操作 `UpdateCleanup` 不会调用 Hook。例如，在以下两个场景中，不会调用 Hook 的 `preDelete` 处理程序：

- 从模板中移除一个资源后，堆栈即会更新。
- 更新类型为 [替换](#) 的资源被删除。

targetNames

Hook 瞄准的类型名称的字符串数组。例如，如果 `preCreate` 处理程序有 `AWS::S3::Bucket` 目标，则挂钩将在预配置阶段为 Amazon S3 存储桶运行。

- `TargetName`

为每个已实现的处理程序指定至少一个目标名称。

图案：`^[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}$`

最小值：1

必需：是

⚠ Warning

SSM SecureString 而且 Secrets Manager 动态引用在传递给 Hooks 之前不会被解析。

permissions

一个字符串数组，它指定调用处理程序所需的 AWS 权限。

必需：是

additionalProperties

additionalProperties 必须设置为 false。Hook 的所有属性都必须在架构中表达：不允许任意输入。

必需：是

有效值：false

Hooks 架构示例

示例 1

Java 和 Python 演练使用以下代码示例。以下是名为的 Hook 的示例结构mycompany-testing-mytesthook.json。

```
{
  "typeName": "MyCompany::Testing::MyTestHook",
  "description": "Verifies S3 bucket and SQS queues properties before create and update",
  "sourceUrl": "https://mycorp.com/my-repo.git",
  "documentationUrl": "https://mycorp.com/documentation",
  "typeConfiguration": {
    "properties": {
      "minBuckets": {
        "description": "Minimum number of compliant buckets",
        "type": "string"
      },
      "minQueues": {
        "description": "Minimum number of compliant queues",
```

```
        "type":"string"
    },
    "encryptionAlgorithm":{
        "description":"Encryption algorithm for SSE",
        "default":"AES256",
        "type":"string"
    }
},
"required":[

],
"additionalProperties":false
},
"handlers":{
    "preCreate":{
        "targetNames":[
            "AWS::S3::Bucket",
            "AWS::SQS::Queue"
        ],
        "permissions":[]
    }
},
    "preUpdate":{
        "targetNames":[
            "AWS::S3::Bucket",
            "AWS::SQS::Queue"
        ],
        "permissions":[]
    }
},
    "preDelete":{
        "targetNames":[
            "AWS::S3::Bucket",
            "AWS::SQS::Queue"
        ],
        "permissions":[
            "s3:ListBucket",
            "s3:ListAllMyBuckets",
            "s3:GetEncryptionConfiguration",
            "sqs:ListQueues",
            "sqs:GetQueueAttributes",
            "sqs:GetQueueUrl"
        ]
    }
}
```

```

    ]
  }
},
"additionalProperties":false
}

```

示例 2

以下示例是一个使用STACK和 CHANGE_SET for targetNames 来定位堆栈模板和更改集操作的架构。

```

{
  "typeName":"MyCompany::Testing::MyTestHook",
  "description":"Verifies Stack and Change Set properties before create and update",
  "sourceUrl":"https://mycorp.com/my-repo.git",
  "documentationUrl":"https://mycorp.com/documentation",
  "typeConfiguration":{
    "properties":{
      "minBuckets":{
        "description":"Minimum number of compliant buckets",
        "type":"string"
      },
      "minQueues":{
        "description":"Minimum number of compliant queues",
        "type":"string"
      },
      "encryptionAlgorithm":{
        "description":"Encryption algorithm for SSE",
        "default":"AES256",
        "type":"string"
      }
    },
    "required":[
    ],
    "additionalProperties":false
  },
  "handlers":{
    "preCreate":{
      "targetNames":[
        "STACK",
        "CHANGE_SET"
      ],
      "permissions":[
      ]
    },
  },
}

```

```
    "preUpdate":{
      "targetNames":[
        "STACK"
      ],
      "permissions":[
      ]
    },
    "preDelete":{
      "targetNames":[
        "STACK"
      ],
      "permissions":[
      ]
    }
  },
  "additionalProperties":false
}
```

禁用和启用 AWS CloudFormation Hook

本主题介绍如何禁用 Hook 然后重新启用，以暂时阻止其在您的账户中处于活动状态。当您需要不受 Hook 干扰的情况下调查问题时，禁用 Hook 会很有用。

在您的账户中禁用和启用挂钩（控制台）

在您的账户中禁用 Hook

1. 登录 AWS Management Console 并在 <https://console.aws.amazon.com/cloudformation> 上打开 AWS CloudFormation 控制台。
2. 在屏幕顶部的导航栏上，选择 Hook 所在 AWS 区域 的位置。
3. 从导航窗格中选择 Hooks。
4. 选择要禁用的挂钩的名称。
5. 在挂钩详细信息页面上，选择挂钩名称右侧的禁用按钮。
6. 当提示您确认时，选择“禁用挂钩”。

重新启用先前禁用的 Hook

1. 登录 AWS Management Console 并在 <https://console.aws.amazon.com/cloudformation> 上打开 AWS CloudFormation 控制台。
2. 在屏幕顶部的导航栏上，选择 Hook 所在 AWS 区域 的位置。
3. 从导航窗格中选择 Hooks。
4. 选择要启用的挂钩的名称。
5. 在挂钩详细信息页面上，选择挂钩名称右侧的启用按钮。
6. 当提示您确认时，选择“启用挂机”。

在您的账户中禁用并启用挂钩 (AWS CLI)

⚠ Important

用于禁用和启用 Hook 的 AWS CLI 命令将整个 Hook 配置替换为 `--configuration` 选项中指定的值。为避免意外更改，在运行这些命令时，必须包括所有希望保留的现有设置。要查看当前配置数据，请使用 [describe-type](#) 命令。

禁用挂钩

使用以下内容 [set-type-configuration](#) 命令并指定 `HookInvocationStatusDISABLED` 为禁用 Hook。用您的特定值替换占位符。

```
aws cloudformation set-type-configuration \  
  --configuration '{"CloudFormationConfiguration":{"HookConfiguration":  
{"HookInvocationStatus": "DISABLED", "FailureMode": "FAIL",  
"TargetOperations": ["STACK", "RESOURCE", "CHANGE_SET"], "Properties":{}}}}' \  
  --type-arn "arn:aws:cloudformation:us-west-2:123456789012:type/hook/MyTestHook" \  
  --region us-west-2
```

重新启用先前禁用的 Hook

使用以下内容 [set-type-configuration](#) 命令并指定 `ENABLED` 要 `HookInvocationStatus` 重新启用 Hook。用您的特定值替换占位符。

```
aws cloudformation set-type-configuration \  
  --configuration '{"CloudFormationConfiguration":{"HookConfiguration":  
{"HookInvocationStatus": "ENABLED", "FailureMode": "FAIL",  
"TargetOperations": ["STACK", "RESOURCE", "CHANGE_SET"], "Properties":{}}}}' \  
  --type-arn "arn:aws:cloudformation:us-west-2:123456789012:type/hook/MyTestHook" \  
  --region us-west-2
```

有关更多信息，请参阅 [挂钩配置架构语法参考](#)。

挂钩配置架构语法参考

本节概述了用于配置 Hook 的架构语法。CloudFormation 在中调用 Hook 时，在运行时使用此配置架构。AWS 账户

要使您的 Hook 能够主动检查堆栈的配置，请在HookInvocationStatus您的账户中注册并激活挂钩ENABLED后将设置为。

主题

- [挂钩配置架构属性](#)
- [挂钩配置示例](#)
- [AWS CloudFormation Hooks 堆栈级别过滤器](#)
- [AWS CloudFormation 挂钩目标过滤器](#)
- [使用带有 Hook 目标名称的通配符](#)

Note

Hook 的配置可以存储的最大数据量为 300 KB。这是对Configuration请求参数施加的所有约束的补充 [SetTypeConfiguration](#)操作。

挂钩配置架构属性

以下架构是 Hook 配置架构的结构。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": "STACK",
      "FailureMode": "FAIL",
      "Properties": {
        ...
      }
    }
  }
}
```

HookConfiguration

挂钩配置支持在堆栈级别激活或停用 Hook、故障模式和挂钩属性值。

Hook 配置支持以下属性。

HookInvocationStatus

指定 Hook 是否为ENABLED或DISABLED。

有效值：ENABLED | DISABLED

TargetOperations

指定 Hook 所针对的操作类型。

有效值：STACK | RESOURCE | CHANGE_SET | CLOUD_CONTROL

TargetStacks

可用于向后兼容。*HookInvocationStatus*改用。

如果将模式设置为ALL，则在、或DELETE资源操作期间，挂钩将应用于您账户中的所有堆栈。CREATE UPDATE

如果将模式设置为NONE，则挂钩将不适用于您账户中的堆栈。

有效值：ALL | NONE

FailureMode

此字段告诉服务如何处理 Hook 故障。

- 如果模式设置为FAIL，但挂钩失败，则失败配置将停止配置资源并回滚堆栈。
- 如果模式设置为WARN，但挂钩失败，则警告配置允许继续配置并显示警告消息。

有效值：FAIL | WARN

Properties

指定 Hook 运行时属性。它们应与 Hooks 架构支持的属性的形状相匹配。

挂钩配置示例

有关从中配置 Hook 的示例 AWS CLI，请参阅以下各节：

- [激活警戒钩 \(AWS CLI\)](#)
- [激活 Lambda 挂钩 \(AWS CLI\)](#)

AWS CloudFormation Hooks 堆栈级别过滤器

您可以在 CloudFormation Hook 中添加堆栈级别过滤器，根据堆栈名称和角色定位特定的堆栈。如果您有多个资源类型相同的堆栈，但是 Hook 适用于特定的堆栈，则此功能很有用。

本节说明了这些过滤器的工作原理，并提供了您可以遵循的示例。

没有堆栈级别筛选的 Hook 配置的基本结构如下所示：

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {}
    }
  }
}
```

有关HookConfiguration语法的更多信息，请参阅[挂钩配置架构语法参考](#)。

要使用堆栈级别筛选器，请在下方添加一个StackFilters密钥HookConfiguration。

该StackFilters密钥有一个必填成员和两个可选成员。

- FilteringCriteria (必需)
- StackNames (可选)
- StackRoles (可选)

StackNames或StackRoles属性是可选的。但是，您必须指定其中至少一个属性。

如果您创建了以 [Cloud Control API](#) 操作为目标的 Hook，则所有堆栈级别的过滤器都将被忽略。

FilteringCriteria

FilteringCriteria是指定筛选行为的必填参数。可以将其设置为ALL或ANY。

- ALL如果所有过滤器都匹配，则调用 Hook。
- ANY如果匹配任何一个过滤器，则调用 Hook。

StackNames

要在 Hooks 配置中将一个或多个堆栈名称指定为过滤器，请使用以下JSON结构：

```
"StackNames": {
  "Include": [
    "string"
  ],
  "Exclude": [
    "string"
  ]
}
```

您必须指定以下各项之一：

- Include：要包含的堆栈名称列表。只有此列表中指定的堆栈才会调用 Hook。
 - 类型：字符串数组
 - 最大物品数量:50
 - 最少物品:1
- Exclude：要排除的堆栈名称列表。除此处列出的堆栈外，所有堆栈都将调用 Hook。
 - 类型：字符串数组
 - 最大物品数量:50
 - 最少物品:1

Include和Exclude数组中的每个堆栈名称都必须符合以下模式和长度要求：

- 模式：`^[a-zA-Z][-a-zA-Z0-9]*$`
- 最大长度：128

StackRoles

要在 Hook 配置中将一个或多个 [IAM角色](#) 指定为筛选器，请使用以下JSON结构：

```
"StackRoles": {
  "Include": [
    "string"
  ],
  "Exclude": [
    "string"
  ]
}
```

您必须指定以下各项之一：

- **Include:** 与这些IAM角色关联ARNs的目标堆栈的角色列表。只有由这些角色启动的堆栈操作才会调用 Hook。
 - 类型：字符串数组
 - 最大物品数量:50
 - 最少物品:1
- **Exclude:** 要排除的堆栈ARNs的IAM角色列表。Hook 将在除指定角色启动的堆栈之外的所有堆栈上调用。
 - 类型：字符串数组
 - 最大物品数量:50
 - 最少物品:1

Include和Exclude数组中的每个堆栈角色都必须遵守以下模式和长度要求：

- 模式：arn:.:+:iam::[0-9]{12}:role/.+
- 最大长度：256

Include 和 Exclude

每个筛选器 (StackNames和StackRoles) 都有一个Include列表和Exclude列表。StackNames举个例子，Hook 只能在Include列表中指定的堆栈上调用。如果仅

在Exclude列表中指定堆栈名称，则仅在不在Exclude列表中的堆栈上调用该挂钩。如果同时指定了Include和Exclude，则 Hook 会瞄准Include列表中的内容，而不是Exclude列表中的内容。

例如，假设你有四个堆栈：A、B、C 和 D。

- "Include": ["A", "B"]在 A 和 B 上调用 Hook
- "Exclude": ["B"]在 A、C 和 D 上调用 Hook
- "Include": ["A", "B", "C"], "Exclude": ["A", "D"]在 B 和 C 上调用 Hook
- "Include": ["A", "B", "C"], "Exclude": ["A", "B", "C"]不会在任何堆栈上调用 Hook。

堆栈级过滤器示例

本节提供了一些示例，您可以按照这些示例为 AWS CloudFormation Hook 创建堆栈级过滤器。

示例 1：包括特定的堆栈

以下示例指定了一个Include列表。Hook 只能在名为stack-test-1、stack-test-2和stack-test-3的堆栈上调用。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        }
      }
    }
  }
}
```

```

}
}

```

示例 2：排除特定堆栈

如果改为将堆栈名称添加到Exclude列表中，则会在任何未命名的堆栈上调用 Hookstack-test-1，stack-test-2或stack-test-3。

```

{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Exclude": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        }
      }
    }
  }
}

```

示例 3：合并包含和排除

如果未指定Include和Exclude列表，则仅在Exclude列表中未Include包含的堆栈上调用 Hook。在以下示例中，Hook 仅在上调用stack-test-3。

```

{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [

```



```

    "STACK",
    "RESOURCE"
  ],
  "FailureMode": "WARN",
  "Properties": {},
  "StackFilters": {
    "FilteringCriteria": "ALL",
    "StackNames": {
      "Include": [
        "stack-test-1",
        "stack-test-2",
        "stack-test-3"
      ],
      "Exclude": [
        "stack-test-1",
        "stack-test-2"
      ]
    }
  }
}
}
}
}
}
}

```

示例 4：将堆栈名称和角色与ALL标准相结合

以下 Hook 包括三个堆栈名称和一个堆栈角色。由于指定FilteringCriteria为ALL，因此只有同时具有匹配堆栈名称和匹配堆栈角色的堆栈才会调用 Hook。

```

{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-1",

```

```

        "stack-test-2",
        "stack-test-3"
    ]
  },
  "StackRoles": {
    "Include": ["arn:aws:iam::123456789012:role/hook-role"]
  }
}
}
}
}
}
}
}

```

示例 5：将堆栈名称和角色与ANY标准相结合

以下 Hook 包括三个堆栈名称和一个堆栈角色。由于指定 *FilteringCriteria* 为 *ANY*，因此会为具有匹配堆栈名称或匹配堆栈角色的堆栈调用 Hook。

```

{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ANY",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        }
      }
    },
    "StackRoles": {
      "Include": ["arn:aws:iam::123456789012:role/hook-role"]
    }
  }
}
}
}
}
}
}
}
}

```

AWS CloudFormation 挂钩目标过滤器

本主题提供有关为 AWS CloudFormation Hook 配置目标过滤器的指南。您可以使用目标过滤器来更精细地控制何时以及在哪些资源上调用 Hook。您可以配置过滤器，从简单的资源类型定位到更复杂的资源类型、操作和调用点组合。

要在 Hooks 配置中将一个或多个堆栈名称指定为过滤器，请在下方添加一个 TargetFilters 密钥 HookConfiguration。

TargetFilters 支持以下属性。

TargetNames

一个字符串数组，它指定要定位的资源类型名称。有关示例，请参阅 [示例 1：基本目标过滤器](#)。

目标名称支持具体的目标名称和完整的通配符匹配。有关更多信息，请参阅 [使用带有 Hook 目标名称的通配符](#)。

图案：`^[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}$`

最大值：50

Actions

一个字符串数组，用于指定您在目标列出的目标操作 TargetNames。

有效值：CREATE | UPDATE | DELETE

Note

使用 RESOURCESTACK、和 CLOUD_CONTROL Hook 目标时，所有目标操作都适用。使用 CHANGE_SET Hook 目标时，只有 CREATE 操作才适用。

InvocationPoints

一个字符串数组，它为你列出的目标指定调用点。TargetNames

有效值：PRE_PROVISION

Targets

一个对象数组，它指定用于目标筛选的目标列表。

`targets` 数组中的每个目标都具有以下属性。

TargetNames

要定位的资源类型名称。

Actions

针对指定目标的操作。

有效值：CREATE | UPDATE | DELETE

InvocationPoints

指定目标的调用点。

有效值：PRE_PROVISION

Note

不能同时包含 `Targets` 对象数组和 `TargetNamesActions`、或 `InvocationPoints` 数组。如果要使用这三个项目和 `Targets`，则必须将它们包含在 `Targets` 对象数组中。有关示例，请参阅 [示例 2：使用 `Targets` 对象数组](#)。

目标过滤器示例

本节提供了一些示例，您可以按照这些示例为 AWS CloudFormation Hook 创建目标过滤器。

示例 1：基本目标过滤器

要创建侧重于特定资源类型的基本目标筛选器，请使用带有 `TargetNames` 数组的 `TargetFilters` 对象。以下目标筛选器配置将对 S3 存储桶和 DynamoDB 表的所有操作调用挂钩。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
    },
  },
}
```

```

    "FailureMode": "WARN",
    "Properties": {},
    "TargetFilters": {
      "TargetNames": [
        "AWS::S3::Bucket",
        "AWS::DynamoDB::Table"
      ]
    }
  }
}
}
}

```

示例 2：使用Targets对象数组

要获得更高级的过滤器，您可以使用Targets对象数组来列出特定的目标、操作和调用点组合。以下目标筛选器配置将调用 Hook 之前的 Hook 以及对 S3 存储桶CREATE和 DynamoDB 表的UPDATE操作。

```

{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "TargetFilters": {
        "Targets": [
          {
            "TargetName": "AWS::S3::Bucket",
            "Action": "CREATE",
            "InvocationPoint": "PRE_PROVISION"
          },
          {
            "TargetName": "AWS::S3::Bucket",
            "Action": "UPDATE",
            "InvocationPoint": "PRE_PROVISION"
          },
          {
            "TargetName": "AWS::DynamoDB::Table",
            "Action": "CREATE",

```



```

        "AWS::*::Bucket*"
    ],
    "permissions": []
}
}
...
}

```

AWS::*::Bucket*可能会解析为以下任何一种具体的资源类型：

- AWS::Lightsail::Bucket
- AWS::S3::Bucket
- AWS::S3::BucketPolicy
- AWS::S3Outpost::Bucket
- AWS::S3Outpost::BucketPolicy

Example : Hook 配置架构中的目标名称通配符示例

以下示例配置调用 Hook 来执行所有 Amazon S3 资源类型的UPDATE操作，以及对所有命名表资源类型（例如AWS::DynamoDB::Table或）的操作。CREATE AWS::Glue::Table

```

{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "FailureMode": "FAIL",
      "Properties": {},
      "TargetFilters": {
        "Targets": [
          {
            "TargetName": "AWS::S3::*",
            "Action": "CREATE",
            "InvocationPoint": "PRE_PROVISION"
          },
          {
            "TargetName": "AWS::*::Table",
            "Action": "UPDATE",
            "InvocationPoint": "PRE_PROVISION"
          }
        ]
      }
    }
  }
}

```

```

    }
  }
}

```

以下示例配置调用所有 Amazon S3 资源类型的挂钩CREATE和UPDATE操作，以及对所有命名表资源类型（例如AWS::DynamoDB::Table或）的CREATE和UPDATE操作。AWS::Glue::Table

```

{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "FailureMode": "FAIL",
      "Properties": {},
      "TargetFilters": {
        "TargetNames": [
          "AWS::S3::*",
          "AWS::*::Table"
        ],
        "Actions": [
          "CREATE",
          "UPDATE"
        ],
        "InvocationPoints": [
          "PRE_PROVISION"
        ]
      }
    }
  }
}

```


使用 CloudFormation 模板创建 Hook

本页提供了 Hooks 的示例 CloudFormation 模板和技术参考主题的连接。

通过使用 CloudFormation 模板创建 Hook，您可以重复使用模板来一致且重复地设置 Hook。这种方法允许你定义一次你的 Hook，然后在多个 AWS 账户和区域中一遍又一遍地配置相同的 Hook。

CloudFormation 为创建防护和 Lambda 挂钩提供了以下专门的资源类型。

Task	解决方案	链接
创建防御挂钩	使用 <code>AWS::CloudFormation::GuardHook</code> 资源类型创建和激活防护挂钩。	样本模板 技术参考
创建一个 Lambda 挂钩	使用 <code>AWS::CloudFormation::LambdaHook</code> 资源类型创建和激活 Lambda 挂钩。	样本模板 技术参考

CloudFormation 还提供以下资源类型，您可以在堆栈模板中使用这些资源类型来创建自定义 Hook。

Task	解决方案	链接
设置 Hook 的默认版本	使用 <code>AWS::CloudFormation::HookDefaultVersion</code> 资源类型指定自定义 Hook 的默认版本。	示例模板 技术参考
设置挂钩的配置	使用 <code>AWS::CloudFormation::HookTypeConfig</code> 资源类型指定自定义 Hook 的配置。	示例模板 技术参考
注册挂钩	使用 <code>AWS::CloudFormation::HookVersion</code> 资源类型将自定义 Hook 的新版本或第一个版本发布到 CloudFormation 注册表。	示例模板 技术参考
公开发布 Hook	使用 <code>AWS::CloudFormation::PublicTypeVersion</code> 资源	技术参考

Task	解决方案	链接
	类型来测试注册的自定义 Hook 并将其发布为公共的第三方 Hook。	
将您的账号注册为出版商	使用AWS:: <code>CloudFormation::Publisher</code> 资源类型在注册 CloudFormation 表中将您的账户注册为公共扩展 (挂钩、模块和资源类型) 的发布者。	技术参考
激活公共、第三方的 Hook	AWS:: <code>CloudFormation::TypeActivation</code> 资源类型与AWS:: <code>CloudFormation::HookTypeConfig</code> 资源类型配合使用，可在您的账户中激活公共的第三方自定义 Hook。	技术参考

AWS CloudFormation Hooks 用户指南的文档历史记录

下表描述了自上次发布 AWS CloudFormation Hooks 以来对文档所做的重要更改。要获得有关本文档更新的通知，您可以订阅 RSS Feed。

- 最新文档更新：2023 年 12 月 8 日。

变更	说明	日期
堆栈级挂钩	现在在堆栈级别支持挂钩，允许客户使用 CloudFormation Hook 来评估新模板，并可能阻止堆栈操作继续进行。	2024 年 11 月 13 日
AWS Cloud Control API Hooks 集成	Hook 现在已与 Cloud Control 集成API，允许客户在配置之前使用 CloudFormation Hook 主动检查资源的配置。如果发现不合规的资源，Hook 要么使操作失败并阻止资源置备，要么发出警告并允许配置操作继续进行。	2024 年 11 月 13 日
AWS CloudFormation Guard 挂钩	AWS CloudFormation Guard 是一种开源的通用领域专用语言 (DSL)，你可以用来创作 policy-as-code。Guard Hooks 可以在配置之前评估云控制API 和 CloudFormation 操作，以检查资源的配置。如果发现不合规的资源，Hook 要么使操作失败并阻止资源置备，要么发出警告并允许配置操作继续进行。	2024 年 11 月 13 日
AWS Lambda 挂钩	AWS CloudFormation Lambda Hooks 允许您根据自己的自定义	2024 年 11 月 13 日

义代码评估 CloudFormation 和云控制API操作。您的 Hook 可以阻止操作继续进行，或者向调用者发出警告并允许操作继续进行。

[Hooks 用户指南](#)

AWS CloudFormation Hooks 用户指南的初始版本。更新包括新的简介、入门演练、概念和术语、堆栈级别筛选以及有关先决条件、设置和 CloudFormation Hook 开发的更新主题。

2023 年 12 月 8 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。