aws

AWS CloudFormation Hooks User Guide

AWS CloudFormation Hooks

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

AWS CloudFormation Hooks: AWS CloudFormation Hooks User Guide

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS CloudFormation Hooks?

AWS CloudFormation Hooks is a feature that you can use to ensure that your CloudFormation resources are compliant with your organization's security, operational, and cost optimization best practices. With CloudFormation Hooks, you can provide code that proactively inspects the configuration of your AWS resources before provisioning. If non-compliant resources are found, AWS CloudFormation either fails the operation and prevents the resources from being provisioned, or emits a warning and allows the provisioning operation to continue.

You can use Hooks to enforce a variety of requirements and guidelines. For example, a security-related Hook can verify security groups for the appropriate inbound and outbound traffic rules for your Amazon Virtual Private Cloud (Amazon VPC). A cost-related Hook can restrict development environments to only use smaller Amazon Elastic Compute Cloud (Amazon EC2) instance types. A Hook designed for data availability can enforce automatic backups for Amazon Relational Database Service (Amazon RDS) .

CloudFormation Hooks is a supported extension type in the AWS CloudFormation registry. The registry makes it easy to distribute and activate Hooks both publicly and privately. Versioning, and resource and module extension types are also supported by the registry. You can use pre-built Hooks, or build your own Hooks using the CloudFormation CLI.

This guide provides an overview of the structure of AWS CloudFormation Hooks, and guides for developing, registering, testing, managing, and publishing your own Hooks.

Topics

Getting started with AWS CloudFormation Hooks

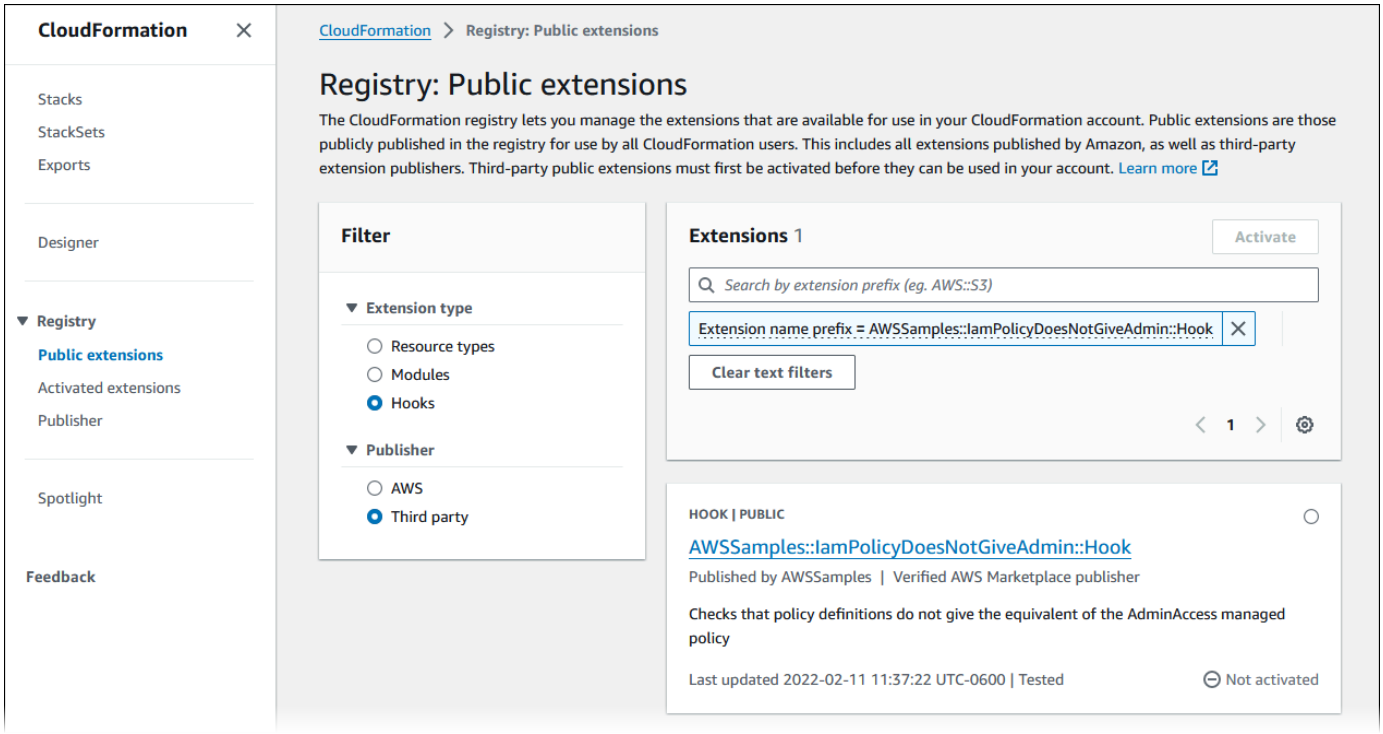
This tutorial provides a quick introduction to AWS CloudFormation Hooks. Follow these steps to activate a sample Hook from the Public Registry.

Step 1: Find a Hook in the Public Registry

To locate a sample Hook:

1. Access the CloudFormation console.
2. In the **CloudFormation** menu, in the **Registry** group, choose **Public extensions** to display the Registry: Public extensions dashboard.
3. In the **Filter** panel, from the **Extension type** list, select **Hooks**.
4. In the **Filter** panel, from the **Publisher** list, select **Third party**. A list of third-party Hooks is displayed. You can scroll the list to browse the available Hooks. Each Hook displays publisher and update information, a summary, and activation status.
5. In the **Extensions** list, find `AWSSamples::IamPolicyDoesNotGiveAdmin::Hook`. This Hook checks that policy definitions do not give the equivalent of the `AdminAccess` managed policy. It specifically checks that `AWS::IAM::Policy`, `AWS::IAM::Role`, `AWS::IAM::Group`, and `AWS::IAM::User` resources don't contain the following `PolicyDocument` statement.

```
- Effect: Allow
  Action: '*'
  Resource: '*'
```



6. Click the `AWSSamples::IamPolicyDoesNotGiveAdmin::Hook` link to display the Hook overview. The overview provides the following information about the Hook.

   - Amazon Resource Name (ARN)
   - Publisher
   - Release date
   - Description
   - Hook behavior
   - Target stacks
   - Registry

   The bottom panel has the following three tabs.

   Target resource types

   The Target resource types tab lists the resource types the Hook targets, and the handlers available in the Hook.

   Schema

   The Schema tab displays the Hook's schema.

Configuration

The Configuration tab displays the Hook's configuration schema, and the configuration for the active Hook. You can also edit the configuration for the active Hook in this tab.

7. Click the **View documentation** link to view the `README.md` for the Hook. The documentation provides example CloudFormation templates, including one that causes the Hook to fail, and an example configuration.
8. Copy the example template that causes the Hook to fail, and save it to a YAML file so that you can test the Hook in a later step.
9. Copy the configuration example to a file so that you can configure the Hook in a later step.
10. Return to the CloudFormation console.

Step 2: Enable the Hook

To enable the Hook:

- Choose **Activate** so that you can configure the Hook and enable it for your CloudFormation account.

Step 3: Set the Execution role ARN

In **Extension details**, enter the **Execution role ARN** of the IAM execution role for CloudFormation to assume when the Hook is invoked in your account and region. In order for CloudFormation to assume the execution role, the role must have a trust policy defined with CloudFormation. Create a new role in the IAM Console with the following custom trust policy. Be sure to specify your `account` and `region:account` in trust policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "resources.cloudformation.amazonaws.com",
                    "hooks.cloudformation.amazonaws.com"
```

```
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "account"
                },
                "StringLike": {
                    "aws:SourceArn": [
                        "arn:aws:cloudformation:region:account:type/hook/AWSSamples-
IamPolicyDoesNotGiveAdmin-Hook/*"
                    ]
                }
            }
        }
    ]
}
```

Step 4: (Optional) Enable logging

In **Logging config**, specify logging configuration information for the Hook, if desired. In the following example, be sure to include your `account`, `rolename`, and `log-group-name`.

```
{
    "logRoleArn": "arn:aws:iam::account:role/rolename",
    "logGroupName": "log-group-name"
}
```

The logging role requires the custom trust policy in Step 3: Set the Execution role ARN and the following permissions:

- cloudwatch:ListMetrics
- cloudwatch:PutMetricData
- logs:CreateLogStream
- logs:DescribeLogGroups
- logs:DescribeLogStreams

- logs:CreateLogGroup
- logs:PutLogEvents

Step 5: Activate the Hook

1. Ensure that **Configure now** is selected, so that you can configure the Hook in the next step.
2. Choose **Activate extension**.

Step 6: Configure the Hook

Once a public Hook is activated in your account, you must set the configuration so that it applies to your stack operations. Configuring a Hook allows you to turn the Hook on or off (`TargetStacks` = `ALL` or `NONE`), set `FailureMode` as `FAIL` or `WARN`, and set any type configuration properties specific to the Hook. For more information on `TargetStacks`, and `FailureMode` options, refer to Hook configuration schema.

To configure a Hook:

1. Ensure that the **Configuration alias** is set to *`default`*.
2. Use the following code for the Hook's configuration JSON. This configuration targets all stacks and fails when invoked for a stack that has a policy definition that gives the equivalent of the `AdminAccess` managed policy.

```
{
    "CloudFormationConfiguration": {
        "HookConfiguration": {
            "TargetStacks":"ALL",
            "FailureMode":"FAIL",
            "Properties": {}
        }
    }
}
```

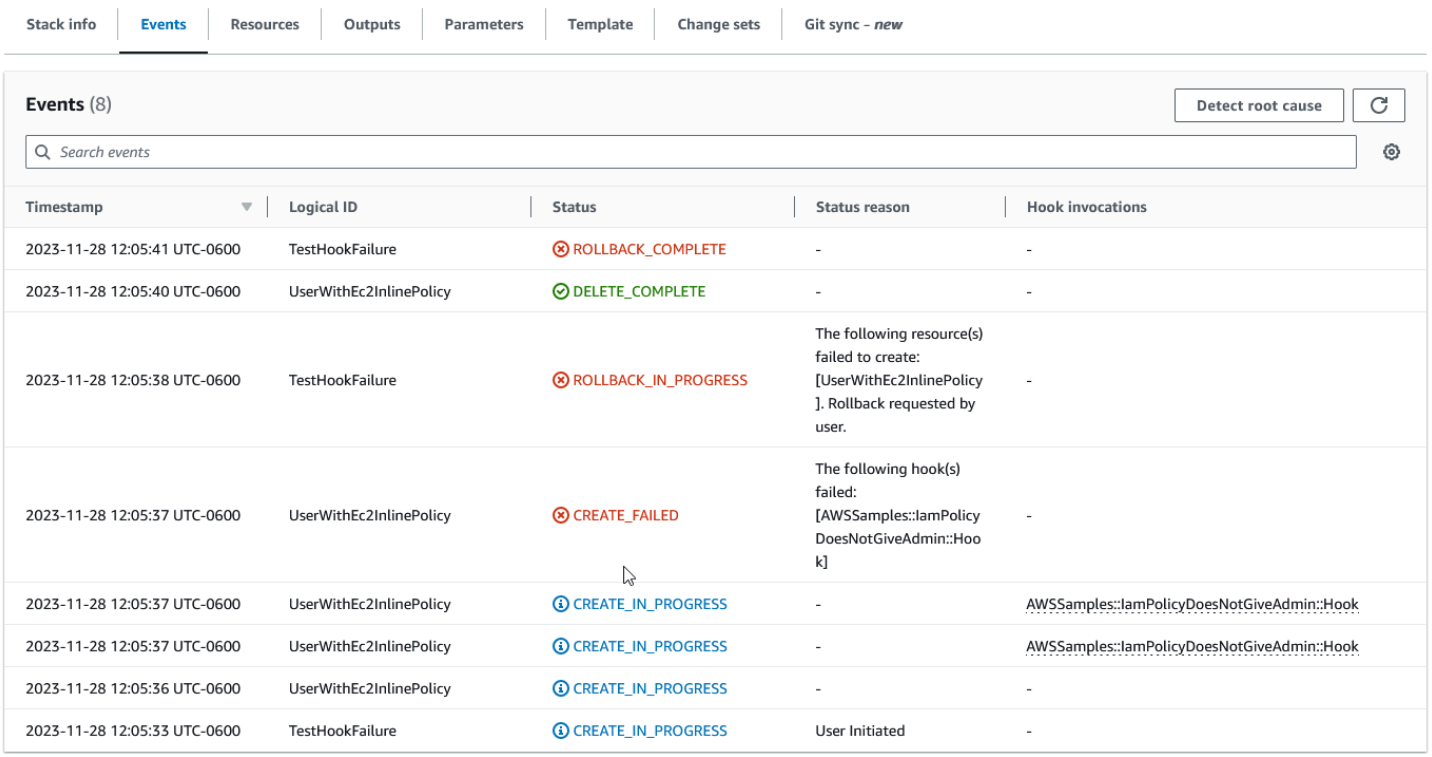
3. Choose **Configure extension** to proceed.

Step 7: Test the Hook

The Hook is now configured and activated for your CloudFormation account. To test the Hook, create a stack with the example CloudFormation template that you saved in Step 1: Find a Hook in the Public Registry.

1. In the CloudFormation menu, choose **Stacks**.
2. In the Stacks console, choose **Create stack**.
3. In the Prerequisite - Prepare template pane, choose **Template is ready**.
4. In the Specify template pane, choose **Upload a template file**.
5. Use **Choose file** to upload the failure example CloudFormation template that you saved in Step 1: Find a Hook in the Public Registry.
6. Choose **Next** to continue.
7. In Specify stack details, provide a name for your test stack, such as `TestHookFailure`.
8. Choose **Next** to continue.
9. In Configure stack options, the default settings are sufficient for testing the Hook. Choose **Next** to continue.
10. Review the stack settings, and choose **Submit** to continue.

CloudFormation attempts to create the stack but fails when the Hook is invoked. The operation is rolled back, and the stack is deleted. You can verify this in the stack's Events list.



Step 7: Deactivate the Hook

To deactivate the Hook, do the following.

1. In the CloudFormation menu, choose **Activated extensions**.
2. In Registry: Activated extensions, choose the **Hooks** tab.
3. From the **Filter extension type** list, choose **Activated third-party**.
4. In the list, click the radio button to the left of `AWSSamples::IamPolicyDoesNotGiveAdmin::Hook` to select it.
5. From the **Actions** list, choose **Deactivate**.
6. In the information popup, choose **Deactivate** again to disable the Hook.

Conclusion

Having completed the preceding steps, `AWSSamples::IamPolicyDoesNotGiveAdmin::Hook` runs when you create a stack with an applicable Hook target. For additional information about testing and managing Hooks, refer the chapter Developing Hooks.

AWS CloudFormation Hooks concepts and terminology

This section introduces fundamental terminology and concepts to help you get started with AWS CloudFormation Hooks.

On this page

Hooks terminology

AWS CloudFormation Hooks uses the following terminology in addition to common AWS concepts and terminology.

Hook

A Hook contains code that is invoked immediately before CloudFormation creates, updates, or deletes specific resources. Hooks can inspect the resources that CloudFormation is about to provision. If any resources don't comply with the organizational guidelines defined in your Hook logic, then you may choose to either `WARN` users or `FAIL`, preventing CloudFormation from provisioning the resource.

Hook target

Hook targets are the CloudFormation resources that you want to run a Hook against. They can be general resources that CloudFormation supports, or third-party resources in the registry. You specify targets while authoring a Hook. For example, you can author a Hook targeting the `AWS::S3::Bucket` resource. A Hook can support multiple targets, and there is no limit on the number of resource targets.

target invocation point

Target invocation points are the exact point in the provisioning logic where Hooks run. CloudFormation supports a `PRE` (before) target invocation point. This means that you can write a Hook that runs before the provisioning logic for the target begins. For example, a Hook with a

PRE target invocation point for an Amazon S3 target runs before the service starts provisioning an Amazon S3 bucket in your account.

target action

Target action is the type of operation that triggers a Hook. The action can be `CREATE`, `UPDATE`, or `DELETE`.

Valid values: (`CREATE` | `UPDATE` | `DELETE`)

Hook handler

The code that handles evaluation. It is associated with a target invocation point and a target action that mark an exact point where a Hook runs. You write handlers that host logic for these specific points. For example, a PRE target invocation point with `CREATE` target action makes a `preCreate` Hook handler. Code within the Hook handler runs when a matching target invocation point and service are performing an associated target action.

Valid values: (`preCreate` | `preUpdate` | `preDelete`)

> **Important**
>
> Stack operations that result in the status of `UpdateCleanup` do not invoke a Hook. For example, during the following two scenarios, the Hook's `preDelete` handler is not invoked:
>
> - the stack is updated after removing one resource from the template.
> - a resource with the update type of replacement is deleted.

Characteristics of Hooks

Hooks have the following characteristics:

- **Proactive validation** – Reduces risk, operational overhead, and cost by identifying noncompliant resources before they're created, updated, or deleted.
- **Automatic enforcement** – Provides enforcement in your AWS account to prevent noncompliant resources from being provisioned by CloudFormation.

Your Hook logic can return success or failure. A success response will allow the operation to continue. A failure for non-compliant resources can result in the following:

- `FAIL` – Stops provisioning resources.
- `WARN` – Allows provisioning to continue with a warning message.

You can register your Hooks as private or third-party public extensions in the CloudFormation registry. For more information, see Using the AWS CloudFormation registry.

Hooks quotas

For more information on Hooks quotas, see AWS CloudFormation quotas.

AWS CloudFormation Hooks structure overview

The properties and attributes of a Hook are defined in a *Hook schema*, which is a JSON-formatted text file. The following sections provide details of Hook schemas, including configuration schemas and using wildcards in Hook target names.

On this page

Hook schema

A Hook includes a Hook specification represented by a JSON schema and Hook handlers.

The first step in creating a Hook is modeling a schema that defines the Hook, its properties, and their attributes. When you initialize a Hook project using the CloudFormation CLI `init` command, a Hook schema file is created for you. Use this schema file as a starting point for defining the shape and semantics of your Hook.

In this section

Hook schema properties

The following schema is the structure for a Hook.

```
{
"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`

The unique name for your Hook. Specifies a three-part namespace for your Hook, with a recommended pattern of `Organization::Service::Hook`.

> **Note**
>
> The following organization namespaces are reserved and can't be used in your Hook type names:
>
> - `Alexa`
> - `AMZN`
> - `Amazon`
> - `ASK`
> - `AWS`
> - `Custom`
> - `Dev`

Required: Yes

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

Minimum: `10`

Maximum: `196`

`description`

A short description of the Hook that's displayed in the CloudFormation console.

Required: Yes

`sourceUrl`

The URL of the source code for the Hook, if public.

Required: No

Maximum: `4096`

`documentationUrl`

The URL of a page providing detailed documentation for the Hook.

Required: Yes

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

Maximum: 4096

> **Note**
>
> Although the Hook schema should include complete and accurate property descriptions, you can use the documentationURL property to provide users with more details, including examples, use cases, and other detailed information.

`definitions`

Use the `definitions` block to provide shared Hook property schemas.

It's considered a best practice to use the `definitions` section to define schema elements that can be used at multiple points in your Hook type schema. You can then use a JSON pointer to reference that element at the appropriate places in your Hook type schema.

Required: No

`typeConfiguration`

The definition of a Hook's configuration data.

Required: Yes

`properties`

The properties of the Hook. All properties of a Hook must be expressed in the schema. Align the Hook schema properties with the Hook type configuration properties.

> **Note**
>
> Nested properties aren't allowed. Instead, define any nested properties in the `definitions` element, and use a `$ref` pointer to reference them in the desired property.

`additionalProperties`

`additionalProperties` must be set to `false`. All properties of a Hook must be expressed in the schema: arbitrary inputs aren't allowed.

Required: Yes

Valid values: `false`

`handlers`

Handlers specify the operations which can initiate the Hook defined in the schema, such as Hook invocation points. For example, a `preUpdate` handler is invoked before the update operations for all specified targets in the handler.

Valid values: `preCreate` | `preUpdate` | `preDelete`

> **Note**
>
> At least one value must be specified for the handler.

> **Important**
>
> Stack operations that result in the status of `UpdateCleanup` do not invoke a Hook. For example, during the following two scenarios, the Hook's `preDelete` handler is not invoked:
>
> - the stack is updated after removing one resource from the template.
> - a resource with the update type of replacement is deleted.

`targetNames`

A string array of type names that Hook targets. For example, if a `preCreate` handler has an `AWS::S3::Bucket` target, the Hook runs for Amazon S3 buckets during the preprovisioning phase.

- `TargetName`

  Specify at least one target name for each implemented handler.

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

  Minimum: 1

  Required: Yes

> **Warning**
>
> SSM SecureString and Secrets Manager dynamic references are not resolved before they are passed to Hooks.

`permissions`

A string array that specifies the AWS permissions needed to invoke the handler.

Required: Yes

`additionalProperties`

`additionalProperties` must be set to `false`. All properties of a Hook must be expressed in the schema: arbitrary inputs aren't allowed.

Required: Yes

Valid values: `false`

Example Hooks schema

The Java and the Python walkthroughs use the following code example. The following is an example structure for a Hook called `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
}
```

Hook configuration schema

CloudFormation supports Hook configurations which are set by the Hook users. CloudFormation refers to this configuration schema at runtime when invoking a Hook in an AWS account. Hooks are configured when the type configuration is set to initiate from the stacks. To enable your Hook to proactively inspect the configuration of your stack, set the `TargetStacks` to `ALL`, after the Hook has been registered and activated in your account. The maximum amount of data that a Hook's configuration can store is 300 KB. This is in addition to all the constraints imposed on `Configuration` request parameter of `SetTypeConfiguration` operation.

In this section

Hook configuration schema properties

The following schema is the structure for a Hook configuration schema.

```
{
    "CloudFormationConfiguration": {
        "HookConfiguration": {
            "TargetStacks": "ALL",
            "FailureMode": "FAIL",
            "Properties": {
                ...
            }
            "TargetFilters": {
                ...
            }
        }
    }
}
```

`HookConfiguration`

Hook configuration supports activating or deactivating Hooks at stack level, failure modes, and Hook properties values.

The Hook configuration supports the following properties.

`TargetStacks`

If the mode is set to `ALL`, the Hook applies to all stacks in your account during a `CREATE`, `UPDATE`, or `DELETE` resource operation.

If the mode is set to `NONE`, the Hook won't apply to stacks in your account.

Valid values: `ALL` | `NONE`

`FailureMode`

This field tells the service how to treat Hook failures.

- If the mode is set to `FAIL`, and the Hook fails, then the fail configuration stops provisioning resources and rolls back the stack.
- If the mode is set to `WARN` and the Hook fails, then the warn configuration allows provisioning to continue with a warning message.

Valid values: `FAIL` | `WARN`

`Properties`

Specifies Hook runtime properties. These should match the shape of the properties supported by Hooks schema.

`TargetFilters`

Specifies the target filters for the Hook. You can use target filters to invoke the Hook only when specified target from the targets that the Hook is registered for in the Hook schema. For example, if a Hook targets `AWS::S3::Bucket` and `AWS::DynamoDB::Table` in the Hook schema, then target filters in the Hook configuration will apply only to those targets.

The target filter configuration supports the following properties.

`Targets`

An object array that specifies the list of targets to use for target filtering.

Each target in the targets array has the following properties.

`TargetNames`

The resource type name to target.

`Actions`

The action for the specified target.

Valid values: `CREATE` | `UPDATE` | `DELETE`

`InvocationPoints`

The invocation point for the specified target.

Valid values: `PRE_PROVISION`

> **Important**
>
> The Hook configuration schema doesn't support using the `Targets` object array in conjunction with the following `TargetNames`, `Actions`, and `InvocationPoints` list arrays. For more information, see the section called "Specifying target names, action, and invocation points for Hook targets".

`TargetNames`

A string array that specifies the resource type names to target.

Target names support concrete target names and full wildcard matching.

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

Maximum: `50`

`Actions`

A string array that specifies the target actions for the targets that you list in `TargetNames`.

Valid values: `CREATE` | `UPDATE` | `DELETE`

`InvocationPoints`

A string array that specifies the invocation points for the targets that you list in `TargetNames`.

Valid values: PRE_PROVISION

Specifying target names, action, and invocation points for Hook targets

You can use either the Targets object array to list specific target, action, and invocation point combinations, or you can use the individual TargetNames, Actions, and InvocationPoints list arrays to specify a list of target names, and the actions and invocation points that apply to all of those targets. The following example configurations have the same result:

Example 1: Using the Targets object array

```
{
   "CloudFormationConfiguration": {
        "HookConfiguration": {
            "TargetStacks": "ALL",
            "FailureMode": "FAIL",
            "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",
                        "InvocationPoint": "PRE_PROVISION"
                    },
                    {
                        "TargetName": "AWS::DynamoDB::Table",
                        "Action": "UPDATE",
                        "InvocationPoint": "PRE_PROVISION"
                    }
                ]
            }
        }
```

```
   }
}
```

Example 2: Using the TargetNames, Actions, and InvocationPoints list arrays

```
{
   "CloudFormationConfiguration": {
        "HookConfiguration": {
            "TargetStacks": "ALL",
            "FailureMode": "FAIL",
            "Properties": {},
            "TargetFilters":{
                "TargetNames": [
                    "AWS::S3::Bucket",
                    "AWS::DynamoDB::Table"
                ],
                "Actions": [
                    "CREATE",
                    "UPDATE"
                ],
                "InvocationPoints": [
                    "PRE_PROVISION"
                ]
            }
        }
   }
}
```

In these examples, the configuration that uses the list arrays is more concise, but both configurations set the Hook to be invoked only before a CREATE or UPDATE operation on `AWS::S3::Bucket` or `AWS::DynamoDB::Table`.

However, if you want to set the Hook to be invoked before CREATE operations only on `AWS::S3::Bucket`, and before UPDATE operations on `AWS::DynamoDB::Table`, you must use the `Targets` object array for more precise control over your Hook configuration. For example:

Example 3: Using Targets object array with different Actions for each TargetName

```
{
   "CloudFormationConfiguration": {
        "HookConfiguration": {
            "TargetStacks": "ALL",
```

```
            "FailureMode": "FAIL",
            "Properties": {},
            "TargetFilters":{
                 "Targets": [
                    {
                        "TargetName": "AWS::S3::Bucket",
                        "Action": "CREATE",
                        "InvocationPoint": "PRE_PROVISION"
                    },
                    {
                        "TargetName": "AWS::DynamoDB::Table",
                        "Action": "UPDATE",
                        "InvocationPoint": "PRE_PROVISION"
                    }
                ]
            }
        }
   }
}
```

Using wildcards with Hook target names

You can use wildcards as part of the target name. You can use wildcard characters (* and ?) within your Hook target names. The asterisk (*) represents any combination of characters. The question mark (?) represents any single character. You can use multiple * and ? characters in a target name.

Example : Examples of target name wildcards in Hook schemas

The following example targets all resource types supported by Amazon S3.

```
{
...
    "handlers": {
       "preCreate": {
            "targetNames": [
                "AWS::S3::*"
            ],
            "permissions": []
       }
    }
...
}
```

The following example matches all resource types that have "Bucket" in the name.

```
{
...
    "handlers": {
       "preCreate": {
            "targetNames": [
                "AWS::*::Bucket*"
            ],
            "permissions": []
       }
    }
...
}
```

The `AWS::*::Bucket*` might resolve to any of the following concrete resource types:

- `AWS::Lightsail::Bucket`
- `AWS::S3::Bucket`
- `AWS::S3::BucketPolicy`
- `AWS::S3Outpost::Bucket`
- `AWS::S3Outpost::BucketPolicy`

Example : Examples of target name wildcards in Hook configuration schemas

The following example configuration invokes the Hook for CREATE operations on all Amazon S3 resource types, and for UPDATE operations on all named table resource types, such as `AWS::DynamobDB::Table` or `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"
                    }
                ]
            }
        }
    }
}
```

The following example configuration invokes the Hook for CREATE and UPDATE operations on all Amazon S3 resource types, and also for CREATE and UPDATE operations on all named table resource types, such as `AWS::DynamobDB::Table` or `AWS::Glue::Table`.

```
{
   "CloudFormationConfiguration": {
        "HookConfiguration": {
            "TargetStacks": "ALL",
            "FailureMode": "FAIL",
            "Properties": {},
            "TargetFilters":{
                "TargetNames": [
                    "AWS::S3::*",
                    "AWS::*::Table"
                ],
                "Actions": [
                    "CREATE",
                    "UPDATE"
                ],
                "InvocationPoints": [
                    "PRE_PROVISION"
                ]
            }
        }
   }
}
```

AWS CloudFormation Hooks stack level filtering

With stack level filtering, you can add filters to your CloudFormation Hooks to target specific stacks. This is useful in cases where you have multiple stacks with the same resource types, but the Hook is written for specific stacks. With the `StackFilters` field under `HookConfiguration`, you can target specific stacks depending on their names and roles.

Stack level filtering components

Suppose you have a test Hook called `CFN::Test::Hook`. You can call `batch-describe-type-configuration` to check the existing configuration.

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "Properties": {},
      "FailureMode": "FAIL"
    }
  }
}
```

To use Stack Level Filtering, you need to add a `StackFilters` field under `HookConfiguration`. `StackFilters` field takes one required parameter and has two optional parameters.

`FilteringCriteria`

`FilteringCriteria` is required. It can be `ALL` or `ANY`.

`ALL` invokes the Hook if all the filters are matched.

`ANY` invokes the Hook if any one filter is matched.

```
...
# required parameter
"FilteringCriteria": {
  "description": "Attribute to specify the filtering behavior. ANY makes the Hook
 invokable if one filter matches. ALL makes the Hook invokable if all filters match",
```

```
  "type": "string",
  "default": "ALL",
  "enum": [
    "ALL",
    "ANY"
  ]
}
...
```

StackNames

With the `StackNames` field, you can specify one or more stack names as a filter.

```
...
"StackNames": {
  "description": "List of stack names as filters",
  "type": "object",
  "additionalProperties": false,
  "minProperties": 1,
  "properties": {
    "Include": {
      "description": "List of stack names that the hook is going to target",
      "type": "array",
      "maxItems": 50,
      "minItems": 1,
      "uniqueItems": true,
      "insertionOrder": false,
      "items": {
        "$ref": "#/definitions/StackName"
      }
    },
    "Exclude": {
      "description": "List of stack names that the hook is going to be excluded from",
      "type": "array",
      "maxItems": 50,
      "minItems": 1,
      "uniqueItems": true,
      "insertionOrder": false,
      "items": {
        "$ref": "#/definitions/StackName"
      }
    }
  }
```

```
}
...

...
# with StackName defined below
"StackName": {
  "description": "CloudFormation Stack name",
  "type": "string",
  "pattern": "^[a-zA-Z][-a-zA-Z0-9]*$",
  "maxLength": 128
}
...
```

StackRoles

If service roles are defined for the stacks, the `StackRoles` field can be used to filter the stacks.

```
...
"StackRoles": {
  "description": "List of stack roles that are performing the stack operations.",
  "type": "object",
  "additionalProperties": false,
  "minProperties": 1,
  "properties": {
    "Include": {
      "description": "List of stack roles that the hook is going to target",
      "type": "array",
      "maxItems": 50,
      "minItems": 1,
      "uniqueItems": true,
      "insertionOrder": false,
      "items": {
        "$ref": "#/definitions/StackRole"
      }
    },
    "Exclude": {
      "description": "List of stack roles that the hook is going to be excluded from",
      "type": "array",
      "maxItems": 50,
      "minItems": 1,
      "uniqueItems": true,
      "insertionOrder": false,
      "items": {
```

```
        "$ref": "#/definitions/StackRole"
      }
    }
  }
}
...

...
# with StackRole defined below
"StackRole": {
  "description": "The Amazon Resource Name (ARN) of the IAM execution role to use to
 perform stack operations",
  "type": "string",
  "pattern": "arn:.+:iam::[0-9]{12}:role/.+",
  "maxLength": 256
}
...
```

`Include` and `Exclude`

Each filter has an `Include` list and `Exclude` list. Using `StackNames` as an example, the Hook is only invoked on the stacks that are specified in `Include` list. If stack names are only specified in the `Exclude` list, the hook is only invoked on stacks that are *not* in the `Exclude` list. If both `Include` and `Exclude` are specified, the Hook targets what's in the `Include` list and not what's in the `Exclude` list.

For example, suppose you have four stacks: A, B, C, and D.

- `"Include": ["A","B"]` The Hook is invoked on A and B.
- `"Exclude": ["B"]` The Hook is invoked on A, C, and D.
- `"Include": ["A","B","C"], "Exclude": ["A","D"]` The Hook is invoked on B and C.
- `"Include": ["A","B","C"], "Exclude": ["A”,"B","C"]` The Hook is not invoked on any stack.

Stack level filtering examples

The following example targets all stacks, but specifies and `Include` list. Despite targeting all stacks, the Hook is only invoked on stacks named `stack-test-1`, `stack-test-2` and `stack-test-3`.

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "Properties": {},
      "FailureMode": "FAIL",
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        }
      }
    }
  }
}
```

If the stack names are instead added to the `Exclude` list, the Hook is invoked on any stack that is *not* named `stack-test-1`, `stack-test-2` or `stack-test-3`.

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "Properties": {},
      "FailureMode": "FAIL",
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Exclude": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        }
      }
    }
  }
}
```

If `Include` and `Exclude` lists aren't specified, the Hook is only invoked on the stacks in the `Include` that aren't in the `Exclude` list. In the following example, the Hook is only invoked on `stack-test-3`.

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "Properties": {},
      "FailureMode": "FAIL",
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ],
          "Exclude": [
            "stack-test-1",
            "stack-test-2"
          ]
        }
      }
    }
  }
}
```

The following Hook includes three `StackNames`, and one `StackRole`. Because the `FilteringCriteria` is specified as `ALL`, the Hook is only invoked for stack that have *both* a matching stack name *and* the matching stack role.

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "Properties": {},
      "FailureMode": "FAIL",
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
```

```
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        },
        "StackRoles": {
          "Include": ["arn:aws:iam::123456789012:role/hook-role"]
        }
      }
    }
  }
}
```

The following Hook includes three `StackNames`, and one `StackRole`. Because the `FilteringCriteria` is specified as ANY, the Hook is invoked for stack that have *either* a matching stack name *or* the matching stack role.

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "TargetStacks": "ALL",
      "Properties": {},
      "FailureMode": "FAIL",
      "StackFilters": {
        "FilteringCriteria": "ANY",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        },
        "StackRoles": {
            "Include": ["arn:aws:iam::123456789012:role/hook-role"]
        }
      }
    }
  }
}
```

Developing AWS CloudFormation Hooks

The topics in this chapter guide you through the process of developing, registering, and publishing your own Hooks with Python or Java.

Topics

AWS CloudFormation Hooks development overview

There are three major steps in developing a Hook:

1. **Initiate**

   To initiate a Hook's project and its required files, use the CloudFormation CLI `init` command and specify that you want to create a Hook. For more information, see Initiating an AWS CloudFormation Hooks project.

2. **Model**

   To model, author, and validate your Hook schema, define the Hook, its properties, and their attributes.

   The CloudFormation CLI creates empty handler functions which correspond to a specific Hook invocation point. Add your own logic to these handlers to control what happens during your Hook invocation at each stage of its target lifecycle. For more information, see Modeling AWS CloudFormation Hooks.

3. **Register**

To register a Hook, submit your Hook to be registered either as a private or a public third-party extension. Register your Hook with the `submit` operation. For more information, see Registering Hooks.

The following tasks are associated with registering your Hook:

a. *Publish* – Hooks are published to the registry.

b. *Configure* – Hooks are configured when the type configuration invokes against stacks.

There are two required type configuration properties:

- `TargetStacks` is used to turn the Hook on or off.
  - `TargetStacks` set to `ALL` turns the Hook on for all stack operations.
  - `TargetStacks` set to `NONE` turns the Hook off, so it doesn't apply to stack operations.
- `FailureMode` determines how CloudFormation responds to a failure response.
  - `FailureMode` set to `FAIL` prevents the operation.
  - `FailureMode` set to `WARN` allows the operation to continue and sends a warning.

> **Note**
>
> Hooks time out after 30 seconds.

Prerequisites for developing AWS CloudFormation Hooks

Hooks can be developed with Java or Python. The following are the prerequisites for developing Hooks:

Java prerequisites

- Apache Maven
- JDK 8 or JDK 11

> **Note**
>
> If you intend to use the CloudFormation Command Line Interface (CLI) to initiate a Hooks project for Java, it's recommended that you install Python 3.7 or later as well. The

Java plugin for the CloudFormation CLI can be installed through `pip` (Python's package manager), which is distrubted with Python.

To implement Hook handlers for your Java Hooks project, you can download the Java Hook handler example files.

Python prerequisites

- Python version 3.7 or later.

To implement Hook handlers for your Python Hooks project, you can download the Python Hook handler example files.

Permissions for developing Hooks

In addition to the CloudFormation `Create`, `Update`, and `Delete` stack permissions, you'll need access to the following AWS CloudFormation operations. Access to these operations is managed through your IAM role's CloudFormation policy.

- `register-type`
- `list-types`
- `deregister-type`
- `set-type-configuration`

Set up a development environment for Hooks

To develop Hooks, you should be familiar with AWS CloudFormation templates, and either Python or Java.

To install the CloudFormation CLI, and the associated plugins:

1. Install the the CloudFormation CLI with `pip`, the Python package manager.

```
pip3 install cloudformation-cli
```

2. Install either the Python or Java plugin for the CloudFormation CLI.

Python

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

Java

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

To upgrade the CloudFormation CLI and the plugin, you can use the upgrade option.

Python

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

Java

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

Initiating an AWS CloudFormation Hooks project

The first step in creating your Hooks project is to initiate the project. You can use the CloudFormation CLI `init` command to initiate your Hooks project.

The `init` command launches a wizard that walks you through setting up the project, including a Hooks schema file. Use this schema file as a starting point for defining the shape and semantics of your Hooks. For more information about Hook schemas, see Hook schema.

To inititate a Hook project:

1. Create a directory for the project.

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

2. Navigate to the new directory.

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

3. Use the CloudFormation CLI `init` command to initiate the project.

```
cfn init
```

The command returns the following output.

```
Initializing new project
```

4. The `init` command launches a wizard that walks you through setting up the project. When prompted, enter h to specify a Hooks project.

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

```
h
```

5. Enter a a name for your Hook type.

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

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

6. If only one language plugin is installed, it is selected by default. If more than one language plugin is installed, you can choose your desired language. Enter a number selection for the language of your choice.

```
Select a language for code generation:
[1] java
[2] python37
[3] python38
[4] python39
(enter an integer):
```

7. Set up packaging based on chosen development lanaguage.

   Python

   (*Optional*) Choose Docker for platform-independent packaging. While Docker isn't required, it's highly recommended to make packaging easier.

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

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

Java

Set Java package name and choose a codegen model. You can use the default package name, or create a new one.

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

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

Results: You have successfully initiated the project and have generated the files needed to develop a Hook. The following is an example of the directories and files that make up a Hooks project for Python 3.8.

```
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

The files in the `src` directory are created based on your language selection. There are some useful comments and examples in the generated files. Some files, such as `models.py`, are automatically updated in a later step when you run the `generate` command to add runtime code for your handlers.

Modeling AWS CloudFormation Hooks

Modeling AWS CloudFormation Hooks involves creating a schema that defines the Hook, its properties, and their attributes. When you create your Hook project using the `cfn init` command, an example Hook schema is created as a JSON-formatted text file, *`hook-name`*`.json`.

Target invocation points and target actions specify the exact point where the Hook is invoked. *Hook handlers* host executable custom logic for these points. For example, a target action of the CREATE operation uses a `preCreate` handler. Your code written in the handler will invoke when Hook targets and services perform a matching action. *Hook targets* are the destination where hooks are invoked. You can specify targets such as, AWS CloudFormation public resources, private resources, or custom resources. Hooks support an unlimited number of Hook targets.

The schema contains permissions required for the Hook. Authoring the Hook requires you to specify permissions for each Hook handler. CloudFormation encourages authors to write policies that follow the standard security advice of granting *least privilege*, or granting only the permissions required to perform a task. Determine what users (and roles) need to do, and then craft policies that allow them to perform *only* those tasks for Hook operations. CloudFormation uses these permissions to scope-down Hook users provided permissions. These permissions are passed down to the Hook. Hook handlers use these permissions to access AWS resources.

You can use the following schema file as a starting point to define your Hook. Use the Hook schema to specify which handlers you want to implement. If you choose not to implement a specific handler, remove it from the handlers' section of the Hook schema.For more details on the schema, see Hook schema.

```
{
    "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
}
```

Add project dependencies (Java)

> **Note**
>
> If you are using Python, continue to Generate the Hook project package.

Java based Hooks projects rely on Maven's `pom.xml` file as a dependency. Expand the following section and copy the source code intto the `pom.xml` file in the root of the project.

Hook project dependencies (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>
        <cfn.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>
        <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>
                    <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>
```

Generate the Hook project package

Generate your hook project package. The CloudFormation CLI creates empty handler functions that correspond to specific Hook actions in the target lifecycle as defined in the Hook specification.

```
cfn generate
```

The command returns the following output.

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

> **Note**
>
> Make sure your Lambda runtimes are up-to-date to avoid using a deprecated version. For more information, see Updating Lambda runtimes for resource types and hooks.

Add Hook handlers

Add your own hook handler runtime code to the handlers that you choose to implement. For example, you can add the following code for logging.

Python

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

The CloudFormation CLI generates the `src/handlers.py` file from the the section called "Hook configuration schema".

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
```

Java

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

The CloudFormation CLI generates a Plain Old Java Objects (Java POJO). The following are output examples generated from `AWS::S3::Bucket`.

Example AwsS3BucketTargetModel.java

```
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&lt;ServerSideEncryptionRule&gt; 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("KMSMasterKeyID")
    private String kMSMasterKeyID;

}
```

With the POJOs generated, you can now write the handlers that actually implement the hook's functionality. For this example, implement the `preCreate` and `preUpdate` invocation point for the handlers.

Implement Hook handlers (Python)

With the Python data classes generated, you can write the handlers that actually implement the Hook's functionality. In this example, you'll implement the `preCreate`, `preUpdate`, and `preDelete` invocation points for the handlers.

The `preCreate` handler verifies the server-side encryption settings for either an `AWS::S3::Bucket` or `AWS::SQS::Queue` resource.

- For an `AWS::S3::Bucket` resource, the Hook will only pass if the following is true.
  - The Amazon S3 bucket encryption is set.
  - The Amazon S3 bucket key is enabled for the bucket.
  - The encryption algorithm set for the Amazon S3 bucket is the correct algorithm required.

- The AWS Key Management Service key ID is set.
- For an `AWS::SQS::Queue` resource, the Hook will only pass if the following is true.
  - The AWS Key Management Service key ID is set.

Implement a `preUpdate` handler, which initiates before the update operations for all specified targets in the handler. The `preUpdate` handler accomplishes the following:

- For an `AWS::S3::Bucket` resource, the Hook will only pass if the following is true:
  - The bucket encryption algorithm for an Amazon S3 bucket hasn't been modified.

Implement a `preDelete` handler, which initiates before the delete operations for all specified targets in the handler. The `preDelete` handler accomplishes the following:

- For an `AWS::S3::Bucket` resource, the Hook will only pass if the following is true:
  - Verifies that the minimum required compliant resources will exist in the account after delete the resource.
  - The minimum required compliant resources amount is set in the Hook's configuration.

Implement a Hook handler

1. In your IDE, open the `handlers.py` file, located in the `src` folder.
2. Replace the entire contents of the `handlers.py` file with the following code.

   Example handlers.py

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

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}")
```

Continue to the next topic Registering Hooks.

Implement Hook handlers (Java)

Coding the API client builder

1. In your IDE, open the `ClientBuilder.java` file, located in the `src/main/java/com/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `ClientBuilder.java` file with the following code.

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();
    }
}
```

Coding the API request maker

1. In your IDE, open the `Translator.java` file, located in the `src/main/java/com/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `Translator.java` file with the following code.

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();
    }
}
```

Implementing the helper code

1. In your IDE, open the `AbstractTestBase.java` file, located in the `src/main/java/com/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `AbstractTestBase.java` file with the following code.

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
                )
        );
    }
}
```

Implementing the base handler

1. In your IDE, open the `BaseHookHandlerStd.java` file, located in the `src/main/java/com/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `BaseHookHandlerStd.java` file with the following code.

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);
```

```
        }
    }
}
```

Implementing the `preCreate` handler

The `preCreate` handler verifies the server-side encryption settings for either an `AWS::S3::Bucket` or `AWS::SQS::Queue` resource.

- For an `AWS::S3::Bucket` resource, the hook will only pass if the following is true:
  - The Amazon S3 bucket encryption is set.
  - The Amazon S3 bucket key is enabled for the bucket.
  - The encryption algorithm set for the Amazon S3 bucket is the correct algorithm required.
  - The AWS Key Management Service key ID is set.
- For an `AWS::SQS::Queue` resource, the hook will only pass if the following is true:
  - The AWS Key Management Service key ID is set.

Coding the `preCreate` handler

1. In your IDE, open the `PreCreateHookHandler.java` file, located in the `src/main/java/software/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `PreCreateHookHandler.java` file with the following code.

```
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();
    }
}
```

Updating the `preCreate` test

1. In your IDE, open the `PreCreateHandlerTest.java` file, located in the `src/test/java/software/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of `PreCreateHandlerTest.java` file with the following code.

```
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("MyBucket", 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("MyBucket", 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: MyBucket");
    }

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

        final AwsS3Bucket bucket = buildAwsS3Bucket("MyBucket", 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: MyBucket");
    }

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

        final AwsS3Bucket bucket = buildAwsS3Bucket("MyBucket", 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: MyBucket");
    }

    @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(targ
```

```
            .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) //
 "KMSMasterKeyID" is name of the property for an AWS::S3::Bucket
                                            .build()
                                    ).build()
                            )
                        ).build()
                ).build();
    }
}
```

Implementing the `preUpdate` handler

Implement a `preUpdate` handler, which initiates before the update operations for all specified targets in the handler. The `preUpdate` handler accomplishes the following:

- For an `AWS::S3::Bucket` resource, the hook will only pass if the following is true:
  - The bucket encryption algorithm for an Amazon S3 bucket hasn't been modified.

Coding the `preUpdate` handler

1. In your IDE, open the `PreUpdateHookHandler.java` file, located in the `src/main/java/software/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `PreUpdateHookHandler.java` file with the following code.

```
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();
    }
}
```

Updating the `preUpdate` test

1. In your IDE, open the `PreUpdateHandlerTest.java` file in the `src/main/java/com/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `PreUpdateHandlerTest.java` file with the following code.

```
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("MyBucket",
 serverSideEncryptionRule);
        final AwsS3Bucket previousResourceProperties = buildAwsS3Bucket("MyBucket",
 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("MyBucket",
serverSideEncryptionRules[0]);
        final AwsS3Bucket previousResourceProperties = buildAwsS3Bucket("MyBucket",
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("MyBucket",
buildServerSideEncryptionRule("SHA512"));
```

```
        final AwsS3Bucket previousResourceProperties = buildAwsS3Bucket("MyBucket",
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(targ
            .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();
    }
}
```

Implementing the `preDelete` handler

Implement a `preDelete` handler, which initiates before the delete operations for all specified targets in the handler. The `preDelete` handler accomplishes the following:

- For an `AWS::S3::Bucket` resource, the hook will only pass if the following is true:
  - Verifies that the minimum required complaint resources will exist in the account after delete the resource.
  - The minimum required complaint resources amount is set in the hook's type configuration.

Coding the `preDelete` handler

1. In your IDE, open the `PreDeleteHookHandler.java` file in the `src/main/java/com/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `PreDeleteHookHandler.java` file with the following code.

```
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);
        }
    }
}
```

Updating the `preDelete` handler

1. In your IDE, open the `PreDeleteHookHandler.java` file in the `src/main/java/com/mycompany/testing/mytesthook` folder.
2. Replace the entire contents of the `PreDeleteHookHandler.java` file with the following code.

```
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())
```

```
                .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();
    }
}
```

Registering Hooks

In this section, you'll learn to package and register your Hook for use in your AWS account.

Package a Hook (Java)

If you've developed your Hook with Java, use Maven to package it.

In the directory of your Hook project, run the following command to build your Hook, run unit tests, and package your project as a JAR file that you can use to submit your Hook to the CloudFormation registry.

```
mvn clean package
```

Register a Hook

To register a Hook

1. (Optional) Configure your default AWS Region name to `us-west-2`, by submitting the `configure` operation.

   ```
   $ 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. (Optional) The following command builds and packages your Hook project without registering it.

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

3. Register your Hook by using the CloudFormation CLI `submit` operation.

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

The command returns the following command.

```
{‘ProgressStatus’: ‘COMPLETE’}
```

Results: You've successfully registered your Hook.

Verifying Hooks are accessible in your account

Verify that your Hook is available in your AWS account and in the regions to which you have submitted it.

1. To verify your Hook, use the `list-types` command to list your newly registered Hook and return a summary description of it.

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

The command returns the following output and will also show you publicly available Hooks you can activate in your AWS account and regions.

```
{
    "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. Retrieve the `TypeArn` from the `list-type` output for your Hook and save it.

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

To learn how to publish Hooks for public use, see Publishing Hooks for public use.

Configure Hooks

After you've developed and registered your Hook, you can configure your Hook in your AWS account by publishing it to the registry.

- To configure a Hook in your account, use the `SetTypeConfiguration` operation. This operation enables the Hook's properties that are defined in the Hook's schema `properties` section. In the following example, the `minBuckets` property is set to 1 in the configuration.

  > **Note**
  >
  > By enabling Hooks in your account, you are authorizing a Hook to use defined permissions from your AWS account. CloudFormation removes non-required permissions before passing your permissions to the Hook. CloudFormation recommends customers or Hook users to review the Hook permissions and be aware of what permissions the Hooks are allowed to before enabling Hooks in your account.

  Specify the configuration data for your registered Hook extension in the same account and AWS Region.

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

Important

To enable your Hook to proactively inspect the configuration of your stack, you must set the `TargetStacks` to `ALL` in the `HookConfiguration` section, after the Hook has been registered and activated in your account.

Accessing AWS APIs in handlers

If your Hooks uses an AWS API in any of its handlers, the CFN-CLI automatically creates an IAM execution role template, `hook-role.yaml`. The `hook-role.yaml` template is based on the permissions specified for each handler in the handler's section of the hook schema. If the `--role-arn` flag is not used during the `generate` operation, the role in this stack will be provisioned and used as the execution role of the Hook.

For more information, see Accessing AWS APIs from a resource type.

hook-role.yaml template

Note

If you choose to create your own execution role, we highly recommend practicing the principle of least privilege by allow listing only `hooks.cloudformation.amazonaws.com` and `resources.cloudformation.amazonaws.com`.

The following template uses the IAM, Amazon S3, and Amazon SQS permissions.

```
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
```

Testing Hooks in your AWS account

Now that you've coded your handler functions that correspond to an invocation point, it's time to test your Hook on a CloudFormation stack.

In Implementing Hook handlers, the Hook failure mode was set to `FAIL` if the CloudFormation template didn't provision an S3 bucket with the following:

- The Amazon S3 bucket encryption is set.
- The Amazon S3 bucket key is enabled for the bucket.
- The encryption algorithm set for the Amazon S3 bucket is the correct algorithm required.
- The AWS Key Management Service key ID is set.

In the following example, create a template called `my-failed-bucket-stack.yml` with a stack name of `my-hook-stack` that fails the stack configuration and stops before the resource provisions.

Testing Hooks by provisioning a stack

To provision a stack (example 1)

Provision a non-compliant stack

1. Author a template that specifies an S3 bucket. For example, `my-failed-bucket-stack.yml`.

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

2. Create a stack, and specify your template in the AWS Command Line Interface (AWS CLI). In the following example, specify the stack name as `my-hook-stack` and the template name as `my-failed-bucket-stack.yml`.

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

3. (Optional) View your stack progress by specifying your stack name. In the following example, specify the stack name `my-hook-stack`.

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

Use the `describe-stack-events` operation to see the Hook failure while creating the bucket. The following is an example output of the command.

```
{
    "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"
        },
    ...
    ]
}
```

Results: The Hook invocation failed the stack configuration and stopped the resource from provisioning.

Use a CloudFormation template to pass Hook validation

1. To create a stack and pass the Hook validation, update the template so that your resource uses an encrypted S3 bucket. This example uses the template `my-encrypted-bucket-stack.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: '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**
>
> Hooks won't be invoked for skipped resources.

2. Create a stack and specify your template. In this example, the stack name is `my-encrypted-bucket-stack`.

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

3. (Optional) View your stack progress by specifying the stack name.

```
$ aws cloudformation describe-stack-events \
```

```
  --stack-name my-encrypted-bucket-stack
```

Use the `describe-stack-events` command to view the response. The following is an example of the `describe-stack-events` command.

```
{
    "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"
        },
    ...
    ]
}
```

Results: CloudFormation successfully created the stack. The Hook's logic verified that the `AWS::S3::Bucket` resource contained server-side encryption before provisioning the resource.

To provision a stack (example 2)

Provision a non-compliant stack

1. Author a template that specifies an S3 bucket. For example `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. Create a stack, and specify your template in the AWS CLI. In the following example, specify the stack name as `my-hook-stack` and the template name as `aes256-bucket.yml`.

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

3. (Optional) View your stack progress by specifying your stack name. In the following example, specify the stack name `my-hook-stack`.

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

Use the `describe-stack-events` operation to see the Hook failure while creating the bucket. The following is an example output of the command.

```
{
    "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"
        },
```

```
    ...
    ]
}
```

Results: The Hook invocation failed the stack configuration and stopped the resource from provisioning. The stack failed due to the S3 bucket encryption configured incorrectly. The Hook type configuration requires `aws:kms` while this bucket uses `AES256`.

Use a CloudFormation template to pass Hook validation

1. To create a stack and pass the Hook validation, update the template so that your resource uses an encrypted S3 bucket. This example uses the template `kms-bucket-and-queue.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: '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**
>
> Hooks won't be invoked for skipped resources.

2. Create a stack and specify your template. In this example, the stack name is `my-encrypted-bucket-stack`.

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

3. (Optional) View your stack progress by specifying the stack name.

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

Use the `describe-stack-events` command to view the response. The following is an example of the `describe-stack-events` command.

```
{
    "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"
        },
```

```
    ...
    ]
}
```

Results: CloudFormation successfully created the stack. The Hook's logic verified that the `AWS::S3::Bucket` resource contained server-side encryption before provisioning the resource.

Managing Hooks

Once you've registered a Hook, you can manage it through the CloudFormation registry, including versioning, account, and region availability.

Updating a Hook

Updating a Hook allows revisions in the Hook to be made available in the CloudFormation registry.

To update a Hook, submit your revisions to the CloudFormation registry through the CloudFormation CLI `submit` operation.

```
$ cfn submit
```

To specify the default version of your Hook in your account, use the `set-type-default-version` command and specify the type, type name, and version ID.

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

To retrieve information about the versions of a Hook, use `list-type-versions`.

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

Deactivating a Hook

Deactivating a Hook prevents the Hook from running your AWS account.

`TargetStacks` set to `NONE` turns the Hook off in your account, so it doesn't apply to stack operations. Use the `set-type-configuration` operation and specify `TargetStacks` as `NONE` to deactivate a Hook.

The following example specifies the AWS Region and the Amazon Resource Name (ARN) of the Hook that's being deactivated.

```
$ aws cloudformation set-type-configuration \
  --region us-west-2 \
  --configuration '{"CloudFormationConfiguration": {"HookConfiguration":
 {"TargetStacks": "NONE", "FailureMode": "FAIL", "Properties":{}}}}' \
  --type-arn HOOK_TYPE_ARN
```

Deregistering a Hook

Deregistering a Hook marks the extension or extension version as `DEPRECATED` in the CloudFormation registry, which removes it from active use. Once deprecated, the Hook can't be used in a CloudFormation operation.

> **Note**
>
> Before deregistering the Hook, you must individually deregister all previous active versions of that extension. For more information, see DeregisterType.

To deregister a Hook, use the `deregister-type` operation and specify your Hook ARN.

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

This command doesn't produce an output.

Publishing Hooks for public use

Developing a public extension for CloudFormation

To develop a public third-party Hook, develop your Hook as a private extension. Then, in each AWS Region in which you want to make the extension publicly available:

1. Register your Hook as a private extension in the CloudFormation registry.

2. Test your Hook to make sure it meets all necessary requirements for being published in the CloudFormation registry.
3. Publish your Hook to the CloudFormation registry.

> **Note**
>
> Before you publish any extension in a given Region, you must first register as an extension publisher in that Region. To do this in multiple Regions simultaneously, see Publishing your extension in multiple Regions using AWS CloudFormation StackSets.

After you've developed and registered your Hook, you can make it publicly available to general CloudFormation users by *publishing* it to the CloudFormation registry, as a third-party public extension.

Public third-party Hooks enable you to offer CloudFormation users to proactively inspect the configuration of AWS resources before provisioning. As with private Hooks, public Hooks are treated the same as any Hook published by AWS within CloudFormation.

Hooks published to the registry are visible by all CloudFormation users in the AWS Regions in which they're published. Users can then *activate* your extension in their account, which makes it available for use in their templates. For more information, see Using public extensions in CloudFormation in the *CloudFormation User Guide*.

Testing registered Hooks

In order to publish your Hook, it must pass all test requirements defined for it. The following is a list of requirements needed before publishing your Hook as a third-party extension.

Each handler and target is tested twice. Once for `SUCCESS` and once for `FAILED`.

- For `SUCCESS` response case:
  - Status must be `SUCCESS`.
  - Must not return an error code.
  - Callback delay should be set to `0` seconds, if specified.
- For `FAILED` response case:
  - Status must be `FAILED`.
  - Must return an error code.

- Must have a message in response.
- Callback delay should be set to `0` seconds, if specified.

- For `IN_PROGRESS` response case:
  - Must not return an error code.
  - `Result` field must not be set in response.

Specifying input data for use in contract tests

By default, the CloudFormation performs contract tests using input properties generated from the patterns you define in your Hook schema. However, most Hooks are complex enough that the input properties for precreating or preupdating provisioning stacks requires an understanding of the resource being provisioned. To address this, you can specify the input the CloudFormation uses when performing its contract tests.

CloudFormation offers two ways for you to specify the input data for it to use when performing contract tests:

- Overrides file

  Using an `overrides` file provides a light-weight way of specifying input data for certain specific properties for the CloudFormation to use during `preCreate`, `preUpdate` and `preDelete` operations testing.
- Input files

  You can also use multiple `input` files to specify contract test input data if:
  - You want or need to specify different input data for create, update, and delete operations, or invalid data with which to test.
  - You want to specify multiple different input data sets.

Specifying input data using an override file

The following is an example of Amazon S3 Hook's input data using the `overrides` file.

```
{
    "CREATE_PRE_PROVISION": {
        "AWS::S3::Bucket": {
            "resourceProperties": {
```

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

Specifying input data using input files

Use `input` files to specify different kinds of input data for the CloudFormation to use: `preCreate` input, `preUpdate` input, and invalid input. Each kind of data is specified in a separate file. You can also specify multiple sets of input data for contract tests.

To specify `input` files for the CloudFormation to use in contract testing, add an `inputs` folder to the root directory of your Hooks project. Then add your input files.

Specify which kind of input data a file contains by using the following naming conventions, where *n* is an integer:

- `inputs_`*n*`_pre_create.json`: Use files with `preCreate` handlers for specifying inputs for creating the resource.
- `inputs_`*n*`_pre_update.json`: Use files with `preUpdate` handlers for specifying inputs for updating the resource.
- `inputs_`*n*`_pre_delete.json`: Use files with `preDelete` handlers for specifying inputs for deleting the resource.
- `inputs_`*n*`_invalid.json`: For specifying invalid inputs to test.

To specify multiple sets of input data for contract tests, increment the integer in the file names to order your input data sets. For example, your first set of input files should be named `inputs_1_pre_create.json`, `inputs_1_pre_update.json`, and `inputs_1_pre_invalid.json`. Your next set would be named `inputs_2_pre_create.json`, `inputs_2_pre_update.json`, and `inputs_2_pre_invalid.json`, and so on.

Each input file is a JSON file containing only the resource properties to be used in testing.

The following is an example directory for `inputs` for Amazon S3 specifying input data using input files.

`inputs_1_pre_create.json`

The following is an example of the `inputs_1_pre_create.json` contract test.

```
{
    "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`

The following is an example of the `inputs_1_pre_update.json` contract test.

```
{
    "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`

The following is an example of the `inputs_1_invalid.json` contract test.

```
{
    "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`

The following is an example of the `inputs_1_invalid_pre_update.json` contract test.

```
{
    "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"
        }
    }
}
```

For more information, see Testing your public extension before publishing.

Document history for the AWS CloudFormation Hooks user guide

The following table describes the important changes to the documentation since the last release of AWS CloudFormation Hooks. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** December 8, 2023.

Change	Description	Date
Hooks User Guide	Initial version of the AWS CloudFormation Hooks User Guide. Updates include a new introduction, getting started walkthrough, concepts and terminology, stack level filtering, and updated topics on prerequisites, setup, and CloudFormation Hooks development.	December 8, 2023

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.