Hooks ユーザーガイド

AWS CloudFormation

AWS CloudFormation: Hooks ユーザーガイド

Table of Contents

AWS CloudFormation フックとは

AWS CloudFormation フックは、CloudFormation リソース、スタック、変更セットが組織のセキュリティ、運用、コスト最適化のベストプラクティスに準拠していることを確認するために使用できる機能です。CloudFormation フックは、 AWS クラウドコントロール API リソースに対する同じレベルのコンプライアンスを確保することもできます。CloudFormation Hooks を使用すると、プロビジョニング前に AWS リソースの設定をプロアクティブに検査するコードを提供できます。非準拠のリソースが見つかった場合、 オペレーションが AWS CloudFormation 失敗してリソースのプロビジョニングが妨げられるか、警告を発行してプロビジョニングオペレーションを続行できます。

フックを使用して、さまざまな要件とガイドラインを適用できます。例えば、セキュリティ関連のフックは、Amazon Virtual Private Cloud (Amazon VPC) の適切なインバウンドトラフィックルールとアウトバウンドトラフィックルールのセキュリティグループを検証できます。コスト関連のフックは、より小さな Amazon Elastic Compute Cloud (Amazon EC2) インスタンスタイプのみを使用するように開発環境を制限できます。データ可用性のために設計されたフックは、Amazon Relational Database Service (Amazon RDS) の自動バックアップを適用できます。

CloudFormation フックは、AWS CloudFormation レジストリでサポートされている拡張機能タイプです。レジストリを使用すると、フックをパブリックとプライベートの両方で簡単に配布およびアクティブ化できます。構築済みのフックを使用するか、CloudFormation CLI を使用して独自のフックを構築できます。

このガイドでは、 AWS CloudFormation フックの構造の概要と、独自のフックを開発、登録、テスト、管理、公開するためのガイドについて説明します。

AWS CloudFormation フックの作成と管理

AWS CloudFormation フックは、スタックの作成、変更、または削除を許可する前に CloudFormation リソースを評価するメカニズムを提供します。この機能は、CloudFormation リソースが組織のセキュリティ、運用、コスト最適化のベストプラクティスに準拠していることを確認するのに役立ちます。

フックを作成するには、3 つのオプションがあります。

- ガードフック – AWS CloudFormation Guard ルールを使用してリソースを評価します。
- Lambda フック – リソース評価のリクエストを AWS Lambda 関数に転送します。
- カスタムフック – 手動で開発するカスタムフックハンドラーを使用します。

Guard Hook

ガードフックを作成するには、以下の主要なステップに従います。

1. Guard ドメイン固有の言語 (DSL) を使用して、リソース評価ロジックを Guard ポリシールールとして記述します。
2. ガードポリシールールを Amazon S3 バケットに保存します。
3. CloudFormation コンソールに移動し、ガードフックの作成を開始します。
4. ガードルールへの Amazon S3 パスを指定します。
5. フックが評価する特定のターゲットを選択します。
6. フックを呼び出すデプロイアクション (作成、更新、削除) を選択します。
7. 評価に失敗したときにフックがどのように応答するかを選択します。
8. 設定が完了したら、フックをアクティブ化して適用を開始します。

Lambda Hook

Lambda フックを作成するには、以下の主要なステップに従います。

1. リソース評価ロジックを Lambda 関数として記述します。
2. CloudFormation コンソールに移動し、Lambda フックの作成を開始します。
3. Lambda 関数の Amazon リソースネーム (ARN) を指定します。
4. フックが評価する特定のターゲットを選択します。

5. フックを呼び出すデプロイアクション (作成、更新、削除) を選択します。
6. 評価に失敗したときにフックがどのように応答するかを選択します。
7. 設定が完了したら、フックをアクティブ化して適用を開始します。

Custom Hook

カスタムフックは、CloudFormation コマンドラインインターフェイス (CFN-CLI) を使用して CloudFormation レジストリに登録する拡張機能です。

カスタムフックを作成するには、以下の主要なステップに従います。

1. プロジェクトを開始する – カスタムフックの開発に必要なファイルを生成します。
2. フックのモデル化 – フックを定義するスキーマと、フックを呼び出すことができるオペレーションを指定するハンドラーを記述します。
3. フックを登録してアクティブ化する – フックを作成したら、フックを使用するアカウントとリージョンに登録する必要があります。これによりフックがアクティブ化されます。

以下のトピックでは、フックの作成と管理について詳しく説明します。

トピック

AWS CloudFormation フックの概念

AWS CloudFormation フックの理解と使用には、以下の用語と概念が不可欠です。

フック

フックには、CloudFormation がスタックまたは特定のリソースを作成、更新、または削除する直前に呼び出されるコードが含まれています。また、変更セットの作成オペレーション中に呼び出すこともできます。フックは、CloudFormation がプロビジョニングしようとしているテンプレート、リソース、または変更セットを検査できます。さらに、Cloud Control API が特定のリソースを作成、更新、または削除する直前にフックを呼び出すことができます。

Hook ロジックで定義されている組織ガイドラインに準拠していない設定が Hook によって識別された場合は、WARNユーザーまたは のいずれかを選択してFAIL、CloudFormation がリソースをプロビジョニングできないようにすることができます。

フックには次の特性があります。

- プロアクティブ検証 – 非準拠のリソースを作成、更新、または削除する前に特定することで、リスク、運用オーバーヘッド、コストを削減します。
- 自動適用 – で適用 AWS アカウント を行い、非準拠のリソースが CloudFormation によってプロビジョニングされないようにします。

失敗モード

フックロジックは成功または失敗を返す可能性があります。成功レスポンスにより、オペレーションを続行できます。非準拠のリソースに障害が発生すると、次のような結果になる可能性があります。

- FAIL – プロビジョニングオペレーションを停止します。
- WARN – プロビジョニングが警告メッセージで続行できるようにします。

WARN モードでフックを作成することは、スタックオペレーションに影響を与えずにフックの動作をモニタリングする効果的な方法です。まず、フックを WARN モードでアクティブ化して、影響を受けるオペレーションを理解します。潜在的な影響を評価したら、フックを FAIL モードに切り替えて、非準拠のオペレーションの防止を開始できます。

フックターゲット

フックターゲットは、フックが評価するオペレーションを指定します。これらは、以下に対するオペレーションです。

- CloudFormation でサポートされているリソース (RESOURCE)
- スタックテンプレート (STACK)
- 変更セット (CHANGE_SET)
- Cloud Control API でサポートされているリソース (CLOUD_CONTROL)

フックが評価する最も広範なオペレーションを指定する 1 つ以上のターゲットを定義します。たとえば、フックターゲットを作成して、すべての AWS リソースRESOURCEをターゲットにし、すべてのスタックテンプレートSTACKをターゲットにすることができます。

ターゲットアクション

ターゲットアクションはCREATE、フックを呼び出す特定のアクション (、UPDATE、または DELETE) を定義します。RESOURCE、STACK、および CLOUD_CONTROLターゲットの場合、すべてのターゲットアクションが適用されます。CHANGE_SET ターゲットの場合、 CREATEアクションのみが適用されます。

フックハンドラー

カスタムフックの場合、これは評価を処理するコードです。これは、ターゲット呼び出しポイントと、フックが実行される正確なポイントを示すターゲットアクションに関連付けられます。これらの特定のポイントのロジックをホストするハンドラーを記述します。たとえば、PREターゲットアクションを持つCREATEターゲット呼び出しポイントは、preCreateフックハンドラーを作成します。フックハンドラー内のコードは、一致するターゲット呼び出しポイントとサービスが関連するターゲットアクションを実行するときに実行されます。

有効な値: (preCreate | preUpdate | preDelete)

> **Important**
>
> ステータスが になるスタックオペレーションでは、フックは呼び出UpdateCleanupされません。たとえば、次の 2 つのシナリオでは、フックのpreDeleteハンドラーは呼び出されません。
>
> - スタックは、テンプレートから 1 つのリソースを削除した後に更新されます。
> - 置換の更新タイプのリソースが削除されます。

タイムアウトと再試行の制限

フックには呼び出しごとに 30 秒のタイムアウト制限があり、再試行回数は 3 回に制限されています。呼び出しがタイムアウトを超えると、フック実行がタイムアウトしたことを示すエラーメッセージが返されます。3 回目の再試行後、CloudFormation はフックの実行を失敗としてマークします。

Guardフック

アカウントで AWS CloudFormation Guard フックを使用するには、使用するアカウントとリージョンのフックをアクティブ化する必要があります。フックを有効にすると、フックがアクティブ化されているアカウントとリージョンのスタックオペレーションで使用できます。

ガードフックをアクティブ化すると、CloudFormation はアクティブ化されたフックのエントリをプライベートフックとしてアカウントのレジストリに作成します。これにより、フックに含まれる設定プロパティを設定できます。設定プロパティは、特定の AWS アカウント およびリージョンに対してフックを設定する方法を定義します。

トピック

AWS CLI ガードフックを操作するための コマンド

ガードフックを操作するための AWS CLI コマンドは次のとおりです。

- activate-type は、ガードフックのアクティベーションプロセスを開始します。
- set-type-configuration アカウント内のフックの設定データを指定します。
- list-types アカウント内のフックを一覧表示します。
- describe-type は、現在の設定データを含む、特定のフックまたは特定のフックバージョンに関する詳細情報を返します。
- deactivate-type は、以前にアクティブ化したフックをアカウントから削除します。

ガードフックのリソースを評価するためのガードルールを記述する

AWS CloudFormation Guard は、policy-as-codeの作成に使用できるオープンソースおよび汎用のドメイン固有の言語 (DSL) です。このトピックでは、Guard を使用して Guard Hook で実行して CloudFormation と AWS クラウドコントロール API オペレーションを自動的に評価できるルール例を作成する方法について説明します。また、ガードフックの実行時期に応じて、ガードルールで使用できるさまざまなタイプの入力にも焦点を当てます。ガードフックは、次のタイプのオペレーション中に実行するように設定できます。

- リソースオペレーション

- スタックオペレーション
- 変更セットオペレーション

ガードルールの記述の詳細については、「ルールの記述 AWS CloudFormation Guard」を参照してください。

トピック

- リソースオペレーションのガードルール
- スタックオペレーションのガードルール
- 変更セットオペレーションのガードルール

リソースオペレーションのガードルール

リソースを作成、更新、または削除するときはいつでも、リソースオペレーションと見なされます。たとえば、新しいリソースを作成する CloudFormation スタックの更新を実行すると、リソースオペレーションが完了しました。Cloud Control API を使用してリソースを作成、更新、または削除すると、リソースオペレーションとも見なされます。フックの設定で RESOURCEおよびCLOUD_CONTROLオペレーションをターゲットにするように ガードフックTargetOperationsを設定できます。ガードフックがリソースオペレーションを評価すると、ガードエンジンはリソース入力を評価します。

トピック

- ガードリソース入力構文
- Guard リソースオペレーション入力の例
- リソース変更のガードルール

ガードリソース入力構文

Guard リソース入力は、Guard ルールが評価できるようにするデータです。

リソース入力のシェイプの例を次に示します。

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

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

HookContext

AWSAccountID

評価対象のリソース AWS アカウント を含む の ID。

StackId

リソースオペレーションの一部である CloudFormation スタックのスタック ID。呼び出し元が Cloud Control API の場合、これは空です。

HookTypeName

実行中のフックの名前。

HookTypeVersion

実行中のフックのバージョン。

InvocationPoint

フックが実行されるプロビジョニングロジックの正確なポイント。

有効な値: (CREATE_PRE_PROVISION | UPDATE_PRE_PROVISION | DELETE_PRE_PROVISION)

TargetName

評価対象のターゲットタイプ。例: AWS::S3::Bucket。

`TargetType`

評価されるターゲットタイプ。例: `AWS::S3::Bucket`。Cloud Control API でプロビジョニングされたリソースの場合、この値は になりますRESOURCE。

`TargetLogicalId`

評価対象のリソースTargetLogicalIdの 。フックのオリジンが CloudFormation の場合、これはリソースの論理 ID (論理名とも呼ばれます) になります。フックのオリジンが Cloud Control API の場合、これは構築された値になります。

`ChangeSetId`

フック呼び出しを引き起こすために実行された変更セット ID。リソースの変更が Cloud Control API、、、create-stackupdate-stackまたは delete-stackオペレーションによって開始された場合、この値は空です。

`Resources`

`ResourceLogicalID`

オペレーションが CloudFormation によって開始されると、 ResourceLogicalIDは CloudFormation テンプレート内のリソースの論理 ID です。

オペレーションが Cloud Control API によって開始されると、 ResourceLogicalIDはリソースタイプ、名前、オペレーション ID、リクエスト ID の組み合わせになります。

`ResourceType`

リソースのタイプ名 (例: `AWS::S3::Bucket`) 。

`ResourceProperties`

変更されるリソースの提案されたプロパティ。Guard Hook が CloudFormation リソースに対して実行されている場合、関数、パラメータ、変換はすべて完全に解決されます。リソースが削除されている場合、この値は空になります。

`Previous`

`ResourceLogicalID`

オペレーションが CloudFormation によって開始されると、 ResourceLogicalIDは CloudFormation テンプレート内のリソースの論理 ID です。

オペレーションが Cloud Control API によって開始されると、 ResourceLogicalIDはリソースタイプ、名前、オペレーション ID、リクエスト ID の組み合わせになります。

`ResourceType`

リソースのタイプ名 (例: `AWS::S3::Bucket`) 。

`ResourceProperties`

変更中のリソースに関連付けられている現在のプロパティ。リソースが削除されている場合、この値は空になります。

Guard リソースオペレーション入力の例

次の入力例は、更新する`AWS::S3::Bucket`リソースの定義を受け取る Guard Hook を示しています。これは、Guard が評価に使用できるデータです。

```
HookContext:
  AwsAccountId: "123456789012"
  StackId: "arn:aws:cloudformation:us-west-2:123456789012:stack/
MyStack/1a2345b6-0000-00a0-a123-00abc0abc000"
  HookTypeName: org::s3policy::hook
  HookTypeVersion: "00001"
  InvocationPoint: UPDATE_PRE_PROVISION
  TargetName: AWS::S3::Bucket
  TargetType: RESOURCE
  TargetLogicalId: MyS3Bucket
  ChangeSetId: ""
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: amzn-s3-demo-bucket
      ObjectLockEnabled: true
Previous:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: amzn-s3-demo-bucket
      ObjectLockEnabled: false
```

リソースタイプで使用できるすべてのプロパティを確認するには、「」を参照してくださいAWS::S3::Bucket。

リソース変更のガードルール

ガードフックがリソースの変更を評価するとき、まずフックで設定されたすべてのルールをダウンロードします。これらのルールは、リソース入力に対して評価されます。いずれかのルールが評価に失敗すると、フックは失敗します。失敗がない場合、フックは成功します。

次の例は、 ObjectLockEnabledプロパティが任意のAWS::S3::Bucketリソースタイプtrue用であるかどうかを評価する Guard ルールです。

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

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

このルールは次の入力に対して実行されると、 ObjectLockEnabledプロパティが に設定されていないため失敗しますtrue。

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

このルールは次の入力に対して実行されると、 ObjectLockEnabledが に設定されているため、渡されますtrue。

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

フックが失敗すると、失敗したルールは CloudFormation または Cloud Control API に伝達されます。Guard Hook にログ記録バケットが設定されている場合、そこに追加のルールフィードバックが提供されます。この追加のフィードバックには、 `Violation`および `Fix`情報が含まれます。

スタックオペレーションのガードルール

CloudFormation スタックが作成、更新、または削除されると、新しいテンプレートを評価してスタックオペレーションの進行をブロックできるように Guard Hook を設定できます。フックの設定で`STACK`オペレーションをターゲットにするように Guard Hook `TargetOperations` を設定できます。

トピック

- ガードスタックの入力構文
- Guard スタックオペレーション入力の例
- スタック変更のガードルール

ガードスタックの入力構文

Guard スタックオペレーションの入力は、Guard ルールが評価する CloudFormation テンプレート全体を提供します。

スタック入力の形状の例を次に示します。

```
HookContext:
  AWSAccountID: String
  StackId: String
  HookTypeName: String
  HookTypeVersion: String
  InvocationPoint: [CREATE_PRE_PROVISION, UPDATE_PRE_PROVISION, DELETE_PRE_PROVISION]
  TargetName: String
  TargetType:STACK
  ChangeSetId: String
{Proposed CloudFormation Template}
Previous:
    {CloudFormation Template}
```

`HookContext`

`AWSAccountID`

リソース AWS アカウント を含む の ID。

`StackId`

スタックオペレーションの一部である CloudFormation スタックのスタック ID。

`HookTypeName`

実行中のフックの名前。

`HookTypeVersion`

実行中のフックのバージョン。

`InvocationPoint`

フックが実行されるプロビジョニングロジックの正確なポイント。

有効な値: (`CREATE_PRE_PROVISION` | `UPDATE_PRE_PROVISION` | `DELETE_PRE_PROVISION`)

`TargetName`

評価対象のスタックの名前。

`TargetType`

この値は、スタックレベルのフックとして を実行するSTACKときに になります。

`ChangeSetId`

フック呼び出しを引き起こすために実行された変更セット ID。スタックオペレーションが `create-stack`、、または `delete-stack`オペレーションによって開始された場合`update-stack`、この値は空です。

`Proposed CloudFormation Template`

CloudFormation `create-stack`または `update-stack`オペレーションに渡された CloudFormation テンプレートの完全な値。これには、`Resources`、、 `Outputs`などが含まれますProperties。CloudFormation に提供された内容に応じて、JSON または YAML 文字列にすることができます。

`delete-stack` オペレーションでは、この値は空になります。

`Previous`

最後に正常にデプロイされた CloudFormation テンプレート。スタックを作成または削除する場合、この値は空です。

delete-stack オペレーションでは、この値は空になります。

> **Note**
>
> 提供されるテンプレートは、 createまたは updateスタックオペレーションに渡されます。スタックを削除する場合、テンプレート値は指定されません。

Guard スタックオペレーション入力の例

次の入力例は、完全なテンプレートと以前にデプロイされたテンプレートを受け取る Guard Hook を示しています。この例のテンプレートは JSON 形式を使用しています。

```
HookContext:
  AwsAccountId: 123456789012
  StackId: "arn:aws:cloudformation:us-west-2:123456789012:stack/
MyStack/1a2345b6-0000-00a0-a123-00abc0abc000"
  HookTypeName: org::templatechecker::hook
  HookTypeVersion: "00001"
  InvocationPoint: UPDATE_PRE_PROVISION
  TargetName: MyStack
  TargetType: CHANGE_SET
  TargetLogicalId: arn:aws:cloudformation:us-west-2:123456789012:changeSet/
SampleChangeSet/1a2345b6-0000-00a0-a123-00abc0abc000
  ChangeSetId: arn:aws:cloudformation:us-west-2:123456789012:changeSet/
SampleChangeSet/1a2345b6-0000-00a0-a123-00abc0abc000
Resources: {
   "S3Bucket": {
        "Type": "AWS::S3::Bucket",
        "Properties": {
           "BucketEncryption": {
               "ServerSideEncryptionConfiguration": [
                {"ServerSideEncryptionByDefault":
                    {"SSEAlgorithm": "aws:kms",
                      "KMSMasterKeyID": "KMS-KEY-ARN" },
                      "BucketKeyEnabled": true }
                ]
           }
        }
}
Previous: {
    "AWSTemplateFormatVersion": "2010-09-09",
```

```
    "Resources": {
        "S3Bucket": {
            "Type": "AWS::S3::Bucket",
            "Properties": {}
        }
    }
}
```

スタック変更のガードルール

ガードフックがスタックの変更を評価するとき、まずフックで設定されたすべてのルールをダウンロードします。これらのルールは、リソース入力に対して評価されます。いずれかのルールが評価に失敗すると、フックは失敗します。失敗がない場合、フックは成功します。

次の例は、 が aws:kmsまたは SSEAlgorithmに設定されている BucketEncryptionというプロパティを含むAWS::S3::Bucketリソースタイプがあるかどうかを評価する Guard ルールですAES256。

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

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

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

ルールが次のテンプレートに対して実行されると、 になりますfail。

```
AWSTemplateFormatVersion: 2010-09-09
Description: S3 bucket without default encryption
Resources:
  EncryptedS3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: !Sub 'encryptedbucket-${AWS::Region}-${AWS::AccountId}'
```

ルールが次のテンプレートに対して実行されると、 になりますpass。

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

変更セットオペレーションのガードルール

CloudFormation 変更セットが作成されると、変更セットで提案されたテンプレートと変更を評価して変更セットの実行をブロックするように Guard Hook を設定できます。

トピック

ガード変更セットの入力構文

ガード変更セット入力は、Guard ルールが評価できるようにするデータです。

以下は、変更セット入力のシェイプの例です。

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

```
  TargetLogicalId:ChangeSet ID
  ChangeSetId: String
{Proposed CloudFormation Template}
Previous:
  {CloudFormation Template}
Changes: [{ResourceChange}]
```

ResourceChange モデル構文は次のとおりです。

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

HookContext

AWSAccountID

リソース AWS アカウント を含む の ID。

StackId

スタックオペレーションの一部である CloudFormation スタックのスタック ID。

HookTypeName

実行中のフックの名前。

HookTypeVersion

実行中のフックのバージョン。

InvocationPoint

フックが実行されるプロビジョニングロジックの正確なポイント。

有効な値: (CREATE_PRE_PROVISION | UPDATE_PRE_PROVISION | DELETE_PRE_PROVISION)

TargetName

評価対象のスタックの名前。

TargetType

この値は、変更セットレベルのフックとして を実行するCHANGE_SETときに になります。

TargetLogicalId

この値は、変更セットの ARN になります。

ChangeSetId

フック呼び出しを引き起こすために実行された変更セット ID。スタックオペレーションが create-stack、、または delete-stackオペレーションによって開始された場合update-stack、この値は空です。

Proposed CloudFormation Template

create-change-set オペレーションに提供された完全な CloudFormation テンプレート。CloudFormation に提供された内容に応じて、JSON または YAML 文字列にすることができます。

Previous

最後に正常にデプロイされた CloudFormation テンプレート。スタックを作成または削除する場合、この値は空です。

Changes

Changes モデル。これにより、リソースの変更が一覧表示されます。

変更

logicalResourceId

変更されたリソースの論理リソース名。

resourceType

変更されるリソースタイプ。

アクション

リソースで実行されるオペレーションのタイプ。

有効な値: (CREATE | UPDATE | DELETE)

lineNumber

変更に関連付けられたテンプレートの行番号。

beforeContext

変更前のリソースのプロパティの JSON 文字列：

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

afterContext

変更後のリソースのプロパティの JSON 文字列：

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

ガード変更セットオペレーション入力の例

次の入力例は、完全なテンプレート、以前にデプロイされたテンプレート、およびリソース変更のリストを受け取る Guard Hook を示しています。この例のテンプレートは JSON 形式を使用しています。

```
HookContext:
  AwsAccountId: "00000000"
  StackId: MyStack
  HookTypeName: org::templatechecker::hook
  HookTypeVersion: "00001"
  InvocationPoint: UPDATE_PRE_PROVISION
  TargetName: my-example-stack
  TargetType:STACK
  TargetLogicalId: arn...:changeSet/change-set
  ChangeSetId: ""
Resources: {
    "S3Bucket": {
       "Type": "AWS::S3::Bucket",
       "Properties": {
           "BucketName": "amzn-s3-demo-bucket",
           "VersioningConfiguration":{
              "Status": "Enabled"
            }
         }
    }
Previous: {
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
```

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

変更セットオペレーションのガードルール

次の例は、Amazon S3 バケットへの変更を評価し、 が無効にならないようにする Guard ルールVersionConfigurationです。

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

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

ガードフックを作成する準備をする

ガードフックを作成する前に、次の前提条件を満たす必要があります。

- ガードルールは作成済みである必要があります。詳細については、「フックの書き込みガードルール」を参照してください。
- フックを作成するユーザーまたはロールには、フックをアクティブ化するための十分なアクセス許可が必要です。
- AWS CLI または SDK を使用してガードフックを作成するには、IAM アクセス許可と信頼ポリシーを持つ実行ロールを手動で作成して、CloudFormation がガードフックを呼び出すことができるようにする必要があります。

ガードフックの実行ロールを作成する

フックは、 でそのフックを呼び出すために必要なアクセス許可の実行ロールを使用します AWS アカウント。

このロールは、 からガードフックを作成すると自動的に作成できます AWS Management Console。それ以外の場合は、このロールを自分で作成する必要があります。

次のセクションでは、ガードフックを作成するためのアクセス許可を設定する方法を示します。

必要なアクセス許可

「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用してロールを作成する」のガイダンスに従って、カスタム信頼ポリシーを使用してロールを作成します。

次に、次の手順を実行してアクセス許可を設定します。

1. 次の最小権限ポリシーを、ガードフックの作成に使用する IAM ロールにアタッチします。

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
```

```
      "Resource": [
        "arn:aws:s3:::my-guard-output-bucket/*",
        "arn:aws:s3:::my-guard-rules-bucket"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-guard-output-bucket/*"
      ]
    }
  ]
}
```

2. ロールに信頼ポリシーを追加して、ロールを引き受けるアクセス許可をフックに付与します。使用できる信頼ポリシーの例を次に示します。

   JSON

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

アカウントでガードフックをアクティブ化する

次のトピックでは、アカウントで Guard Hook をアクティブ化する方法を示します。これにより、アクティブ化されたアカウントとリージョンで使用できます。

トピック

ガードフックをアクティブ化する (コンソール)

アカウントで使用するガードフックをアクティブ化するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。
2. 画面上部のナビゲーションバーで、フックを作成する AWS リージョン を選択します。
3. ガードルールをまだ作成していない場合は、ガードルールを作成し、Amazon S3 に保存してから、この手順に戻ります。開始するには、ガードフックのリソースを評価するためのガードルールを記述する「」のルール例を参照してください。

   ガードルールを既に作成して S3 に保存している場合は、次のステップに進みます。

   > **Note**
   >
   > S3 に保存されるオブジェクトには、、、`.guard`、`.zip`または のいずれかのファイル拡張子が必要です`.tar.gz`。

4. Guard Hook ソースの場合は、S3 に Guard ルールを保存します。以下を実行します。

   - S3 URI の場合は、ルールファイルへの S3 パスを指定するか、S3 参照ボタンを使用してダイアログボックスを開き、S3 オブジェクトを参照して選択します。
   - (オプション) オブジェクトバージョンでは、S3 バケットでバージョニングが有効になっている場合、特定のバージョンの S3 オブジェクトを選択できます。

     ガードフックは、フックが呼び出されるたびに S3 からルールをダウンロードします。誤って変更または削除されないように、ガードフックを設定するときは バージョンを使用することをお勧めします。
5. (オプション) Guard 出力レポートの S3 バケットには、Guard 出力レポートを保存する S3 バケットを指定します。このレポートには、ガードルールの検証結果が含まれます。

   出力レポートの送信先を設定するには、次のいずれかのオプションを選択します。

- 「Guard ルールが保存されているのと同じバケットを使用する」チェックボックスをオンにして、Guard ルールがあるのと同じバケットを使用します。
- Guard 出力レポートを保存する別の S3 バケット名を選択します。

6. (オプション) ガードルール入力パラメータを展開し、S3 にガードルール入力パラメータを保存する の下に次の情報を入力します。

   - S3 URI の場合は、パラメータファイルへの S3 パスを指定するか、S3 参照ボタンを使用してダイアログボックスを開き、S3 オブジェクトを参照して選択します。
   - (オプション) オブジェクトバージョンでは、S3 バケットでバージョニングが有効になっている場合、特定のバージョンの S3 オブジェクトを選択できます。

7. [次へ] を選択します。
8. フック名で、次のいずれかのオプションを選択します。

   - の後に追加される短いわかりやすい名前を指定しますPrivate::Guard::。たとえば、 と入力すると*MyTestHook*、完全なフック名は になりますPrivate::Guard::*MyTestHook*。
   - 次の形式を使用して、完全なフック名 (エイリアスとも呼ばれます) を指定します。*Provider*::*ServiceName*::*HookName*

9. フックターゲットで、評価対象を選択します。

   - スタック — ユーザーがスタックを作成、更新、または削除するときにスタックテンプレートを評価します。
   - リソース — ユーザーがスタックを更新するときに、個々のリソースの変更を評価します。
   - 変更セット — ユーザーが変更セットを作成するときに、計画された更新を評価します。
   - Cloud Control API — Cloud Control API によって開始された作成、更新、または削除オペレーションを評価します。

10. アクション で、フックを呼び出すアクション (作成、更新、削除) を選択します。
11. フックモードでは、ルールが評価に失敗したときにフックがどのように応答するかを選択します。

    - 警告 — ユーザーに警告を発行しますが、アクションを続行できます。これは、重要でない検証や情報チェックに役立ちます。
    - 失敗 — アクションが続行されないようにします。これは、厳格なコンプライアンスまたはセキュリティポリシーを適用するのに役立ちます。

12. 実行ロールで、CloudFormation フックが S3 から Guard ルールを取得するために引き受ける IAM ロールを選択し、オプションで詳細な Guard 出力レポートを書き戻します。CloudFormation が実行ロールを自動的に作成できるようにするか、作成したロールを指定できます。
13. [次へ] を選択します。
14. (オプション) フックフィルターの場合は、次の操作を行います。

    a. リソースフィルターで、フックを呼び出すことができるリソースタイプを指定します。これにより、フックは関連するリソースに対してのみ呼び出されます。

    b. フィルタリング条件 で、スタック名とスタックロールフィルターを適用するためのロジックを選択します。

    - すべてのスタック名とスタックロール – フックは、指定されたすべてのフィルターが一致する場合にのみ呼び出されます。
    - スタック名とスタックロール – 指定されたフィルターの少なくとも 1 つが一致すると、フックが呼び出されます。

    > **Note**
    >
    > Cloud Control API オペレーションでは、すべてのスタック名とスタックロールフィルターは無視されます。

    c. スタック名には、フック呼び出しに特定のスタックを含めるか除外します。

    - 含める には、含めるスタック名を指定します。これは、ターゲットにする特定のスタックの小さなセットがある場合に使用します。このリストで指定されたスタックのみがフックを呼び出します。
    - Exclude には、除外するスタック名を指定します。これは、ほとんどのスタックでフックを呼び出すが、いくつかの特定のスタックを除外する場合に使用します。ここにリストされているスタックを除くすべてのスタックは、 フックを呼び出します。

    d. スタックロールの場合、関連付けられた IAM ロールに基づいてフック呼び出しに特定のスタックを含めるか除外します。

    - Include には、これらのロールに関連付けられたスタックをターゲットにする 1 つ以上の IAM ロール ARNs を指定します。これらのロールによって開始されたスタックオペレーションのみがフックを呼び出します。

- Exclude には、除外するスタックの 1 つ以上の IAM ロール ARNs を指定します。フックは、指定されたロールによって開始されたスタックを除くすべてのスタックで呼び出されます。

15. [次へ] を選択します。
16. 確認とアクティブ化ページで、選択内容を確認します。変更するには、関連セクションで [編集] をクリックします。
17. 続行する準備ができたら、フックのアクティブ化を選択します。

ガードフックをアクティブ化する (AWS CLI)

続行する前に、このフックで使用する Guard ルールと実行ロールが作成されていることを確認します。詳細については、「ガードフックのリソースを評価するためのガードルールを記述する」および「ガードフックの実行ロールを作成する」を参照してください。

アカウントで使用するガードフックをアクティブ化するには (AWS CLI)

1. フックのアクティブ化を開始するには、次のactivate-typeコマンドを使用して、プレースホルダーを特定の値に置き換えます。このコマンドは、フックが から指定された実行ロールを使用することを許可します AWS アカウント。

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

2. フックのアクティブ化を完了するには、JSON 設定ファイルを使用してフックを設定する必要があります。

   cat コマンドを使用して、次の構造で JSON ファイルを作成します。詳細については、「フック設定スキーマ構文リファレンス」を参照してください。

```
$ cat > config.json
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
```

```
        "STACK",
        "RESOURCE",
        "CHANGE_SET"
      ],
      "FailureMode": "WARN",
      "Properties": {
        "ruleLocation": "s3://amzn-s3-demo-bucket/MyGuardRules.guard",
        "logBucket": "amzn-s3-demo-logging-bucket"
      },
      "TargetFilters": {
        "Actions": [
          "CREATE",
          "UPDATE",
          "DELETE"
        ]
      }
    }
  }
}
```

- HookInvocationStatus: フックを有効にするENABLEDには、 に設定します。
- TargetOperations: フックが評価するオペレーションを指定します。
- FailureMode: FAIL または WARN に設定します。
- ruleLocation: を、ルールが保存されている S3 URI に置き換えます。S3 に保存されるオブジェクトには、、.guard、 .zipのいずれかのファイル拡張子が必要です.tar.gz。
- logBucket: (オプション) Guard JSON レポートの S3 バケットの名前を指定します。
- TargetFilters: フックを呼び出すアクションのタイプを指定します。

3. 次のset-type-configurationコマンドと作成した JSON ファイルを使用して、設定を適用します。プレースホルダーを特定の値に置き換えます。

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

関連リソース

CloudFormation スタックテンプレートで Guard Hook を宣言する方法を理解するために使用できるテンプレートの例を示します。詳細については、AWS CloudFormation ユーザーガイドの AWS::CloudFormation::GuardHookを参照してください。

アカウントのガードフックのログを表示する

ガードフックをアクティブ化すると、フック出力レポートの送信先として Amazon S3 バケットを指定できます。有効にすると、フックは Guard ルールの検証の結果を指定されたバケットに自動的に保存します。その後、Amazon S3 コンソールでこれらの結果を表示できます。

Amazon S3 コンソールで Guard Hook ログを表示する

Guard Hook 出力ログファイルを表示するには

1. https://console.aws.amazon.com/s3/ にサインインします。
2. 画面上部にあるナビゲーションバーで、 AWS リージョンを選択します。
3. バケットを選択します。
4. Guard 出力レポート用に選択したバケットを選択します。
5. 目的の検証出力レポートのログファイルを選択します。
6. ファイルをダウンロードするか、開くかを選択します。

アカウントのガードフックを削除する

アクティブ化された Guard Hook が不要になった場合は、次の手順を使用してアカウントで削除します。

フックを削除する代わりに一時的に無効にするには、「」を参照してくださいAWS CloudFormation フックの無効化と有効化。

トピック

- アカウントのガードフックを削除する (コンソール)
- アカウントのガードフックを削除する (AWS CLI)

アカウントのガードフックを削除する (コンソール)

アカウントのガードフックを削除するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。
2. 画面上部のナビゲーションバーで、フックがある を選択します AWS リージョン 。
3. ナビゲーションペインから、フックを選択します。
4. フックページで、削除するガードフックを見つけます。
5. フックの横にあるチェックボックスをオンにし、削除を選択します。
6. 確認を求められたら、フック名を入力して、指定されたフックの削除を確認し、削除を選択します。

アカウントのガードフックを削除する (AWS CLI)

> **Note**
>
> フックを削除する前に、まずフックを無効にする必要があります。詳細については、「アカウントでフックを無効化および有効化する (AWS CLI)」を参照してください。

次のdeactivate-typeコマンドを使用してフックを非アクティブ化し、アカウントから削除します。プレースホルダーを特定の値に置き換えます。

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

Lambda フック

アカウントで AWS Lambda フックを使用するには、まずそのフックを使用するアカウントとリージョンのフックをアクティブ化する必要があります。フックを有効にすると、フックがアクティブ化されているアカウントとリージョンのスタックオペレーションで使用できます。

Lambda フックをアクティブ化すると、CloudFormation はアクティブ化されたフックのエントリをプライベートフックとしてアカウントのレジストリに作成します。これにより、フックに含まれる設

定プロパティを設定できます。設定プロパティは、特定の AWS アカウント およびリージョンに対してフックを設定する方法を定義します。

トピック

AWS CLI Lambda Hooks を操作するための コマンド

Lambda フックを操作するための AWS CLI コマンドは次のとおりです。

- activate-type Lambda フックのアクティベーションプロセスを開始します。
- set-type-configuration アカウント内のフックの設定データを指定します。
- list-types アカウント内のフックを一覧表示します。
- describe-type は、現在の設定データを含む、特定のフックまたは特定のフックバージョンに関する詳細情報を返します。
- deactivate-type は、以前にアクティブ化したフックをアカウントから削除します。

Lambda フックのリソースを評価する Lambda 関数を作成する

AWS CloudFormation Lambda フックを使用すると、独自のカスタムコードに対して CloudFormation および AWS クラウドコントロール API オペレーションを評価できます。フックは、操作の進行をブロックしたり、呼び出し元に警告を発行して操作の進行を許可したりできます。Lambda フックを作成するときに、次の CloudFormation オペレーションを傍受して評価するように設定できます。

- リソースオペレーション
- スタックオペレーション
- 変更セットオペレーション

トピック

Lambda フックの開発

フックが Lambda を呼び出すと、Lambda が入力を評価するまで最大 30 秒待機します。Lambda は、フックが成功したか失敗したかを示す JSON レスポンスを返します。

トピック

リクエスト入力

Lambda 関数に渡される入力は、フックターゲットオペレーション (スタック、リソース、変更セットなど) によって異なります。

レスポンス入力

リクエストが成功または失敗した場合にフックと通信するには、Lambda 関数が JSON レスポンスを返す必要があります。

以下は、フックが期待するレスポンスの形状の例です。

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

hookStatus

フックのステータス。これは必須のフィールドです。

有効な値: (SUCCESS | FAILED | IN_PROGRESS)

> **Note**
>
> フックは 3 IN_PROGRESS 回返すことができます。結果が返されない場合、フックは失敗します。Lambda フックの場合、これは Lambda 関数を最大 3 回呼び出すことができることを意味します。

errorCode

オペレーションが評価されて無効であると判断された場合、またはフック内でエラーが発生し、評価が妨げられたかどうかを示します。フックが失敗した場合、このフィールドは必須です。

有効な値: (NonCompliant | InternalFailure)

message

フックが成功または失敗した理由を示す発信者へのメッセージ。

> **Note**
>
> CloudFormation オペレーションを評価する場合、このフィールドは 4096 文字に切り捨てられます。
> Cloud Control API オペレーションを評価する場合、このフィールドは 1024 文字に切り捨てられます。

clientRequestToken

Hook リクエストへの入力として提供されたリクエストトークン。これは必須のフィールドです。

callbackContext

が hookStatusであることを示すIN_PROGRESS場合は、Lambda 関数が再呼び出されたときに入力として提供される追加のコンテキストを渡します。

callbackDelaySeconds

フックがこのフックを再度呼び出すまで待機する時間。

例

以下は、成功したレスポンスの例です。

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

失敗したレスポンスの例を次に示します。

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

Lambda フックを使用したリソースオペレーションの評価

リソースを作成、更新、または削除するときはいつでも、リソースオペレーションと見なされます。たとえば、新しいリソースを作成する CloudFormation スタックの更新を実行すると、リソースオペレーションが完了しました。Cloud Control API を使用してリソースを作成、更新、または削除すると、リソースオペレーションとも見なされます。フック設定で `RESOURCE`および `CLOUD_CONTROL`オペレーションをターゲットにするように CloudFormation Lambda フック`TargetOperations`を設定できます。

> **Note**
>
> `delete` フックハンドラーは、Cloud Control API `delete-resource`または CloudFormation からオペレーショントリガーを使用してリソースが削除された場合にのみ呼び出されます`delete-stack`。

トピック

- Lambda Hook リソース入力構文
- Lambda Hook リソース変更入力の例
- リソースオペレーションの Lambda 関数の例

Lambda Hook リソース入力構文

Lambda がリソースオペレーションに対して呼び出されると、リソースプロパティ、提案されたプロパティ、およびフック呼び出しに関するコンテキストを含む JSON 入力を受け取ります。

JSON 入力のシェイプの例を次に示します。

```
{
    "awsAccountId": String,
    "stackId": String,
    "changeSetId": String,
    "hookTypeName": String,
    "hookTypeVersion": String,
    "hookModel": {
        "LambdaFunction": String
    },
    "actionInvocationPoint": "CREATE_PRE_PROVISION" or "UPDATE_PRE_PROVISION" or
 "DELETE_PRE_PROVISION"
    "requestData": {
        "targetName": String,
        "targetType": String,
        "targetLogicalId": String,
        "targetModel": {
            "resourceProperties": {...},
            "previousResourceProperties": {...}
        }
    },
    "requestContext": {
        "####": 1,
        "callbackContext": null
    }
}
```

`awsAccountId`

評価対象のリソース AWS アカウント を含む の ID。

`stackId`

このオペレーションが属する CloudFormation スタックのスタック ID。発信者が Cloud Control API の場合、このフィールドは空です。

`changeSetId`

フック呼び出しを開始した変更セットの ID。リソースの変更が Cloud Control API、、、`create-stack`または `delete-stack`オペレーションによって開始された場合、この値は空です。

`hookTypeName`

実行中のフックの名前。

`hookTypeVersion`

実行中のフックのバージョン。

`hookModel`

`LambdaFunction`

フックによって呼び出される現在の Lambda ARN。

`actionInvocationPoint`

フックが実行されるプロビジョニングロジックの正確なポイント。

有効な値: (`CREATE_PRE_PROVISION` | `UPDATE_PRE_PROVISION` | `DELETE_PRE_PROVISION`)

`requestData`

`targetName`

評価対象のターゲットタイプ。例: `AWS::S3::Bucket`。

`targetType`

評価されるターゲットタイプ。例: `AWS::S3::Bucket`。Cloud Control API でプロビジョニングされたリソースの場合、この値は になります`RESOURCE`。

`targetLogicalId`

評価対象のリソースの論理 ID。フック呼び出しのオリジンが CloudFormation の場合、これは CloudFormation テンプレートで定義されている論理リソース ID になります。このフック呼び出しのオリジンが Cloud Control API である場合、これは構築された値になります。

targetModel

resourceProperties

変更されるリソースの提案されたプロパティ。リソースが削除されている場合、この値は空になります。

previousResourceProperties

変更中のリソースに現在関連付けられているプロパティ。リソースが作成されている場合、この値は空になります。

requestContext

呼び出し

フックの現在の実行試行。

callbackContext

Hook が に設定されIN_PROGRESS、 callbackContextが返された場合、再呼び出し後にここに表示されます。

Lambda Hook リソース変更入力の例

次の入力例は、更新するAWS::DynamoDB::Tableリソースの定義を受け取る Lambda フックを示しています。ここで、 ReadCapacityUnits の ProvisionedThroughputは 3 から 10 に変更されています。これは、Lambda が評価に使用できるデータです。

```
{
    "awsAccountId": "123456789012",
    "stackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/
MyStack/1a2345b6-0000-00a0-a123-00abc0abc000",
    "hookTypeName": "my::lambda::resourcehookfunction",
    "hookTypeVersion": "00000008",
    "hookModel": {
        "LambdaFunction": "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
    },
    "actionInvocationPoint": "UPDATE_PRE_PROVISION",
    "requestData": {
        "targetName": "AWS::DynamoDB::Table",
        "targetType": "AWS::DynamoDB::Table",
        "targetLogicalId": "DDBTable",
        "targetModel": {
            "resourceProperties": {
```

```
                "AttributeDefinitions": [
                    {
                        "AttributeType": "S",
                        "AttributeName": "Album"
                    },
                    {
                        "AttributeType": "S",
                        "AttributeName": "Artist"
                    }
                ],
                "ProvisionedThroughput": {
                    "WriteCapacityUnits": 5,
                    "ReadCapacityUnits": 10
                },
                "KeySchema": [
                    {
                        "KeyType": "HASH",
                        "AttributeName": "Album"
                    },
                    {
                        "KeyType": "RANGE",
                        "AttributeName": "Artist"
                    }
                ]
            },
            "previousResourceProperties": {
                "AttributeDefinitions": [
                    {
                        "AttributeType": "S",
                        "AttributeName": "Album"
                    },
                    {
                        "AttributeType": "S",
                        "AttributeName": "Artist"
                    }
                ],
                "ProvisionedThroughput": {
                    "WriteCapacityUnits": 5,
                    "ReadCapacityUnits": 5
                },
                "KeySchema": [
                    {
                        "KeyType": "HASH",
                        "AttributeName": "Album"
```

```
                    },
                    {
                        "KeyType": "RANGE",
                        "AttributeName": "Artist"
                    }
                ]
            }
        }
    },
    "requestContext": {
        "invocation": 1,
        "callbackContext": null
    }
}
```

リソースタイプで使用できるすべてのプロパティを確認するには、「」を参照してくださいAWS::DynamoDB::Table。

リソースオペレーションの Lambda 関数の例

以下は、DynamoDB へのリソース更新に失敗するシンプルな関数です。DynamoDB は、ReadCapacity の ProvisionedThroughputを 10 より大きい値に設定しようとします。フックが成功すると、ReadCapacity が正しく設定されています」というメッセージが発信者に表示されます。リクエストの検証に失敗すると、フックはReadCapacity cannot be more than 10」というステータスで失敗します。

Node.js

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

    if (targetName == "AWS::DynamoDB::Table") {
        var readCapacity =
 targetModel?.resourceProperties?.ProvisionedThroughput?.ReadCapacityUnits;
        if (readCapacity > 10) {
            response.hookStatus = "FAILED";
```

```
            response.errorCode = "NonCompliant";
            response.message = "ReadCapacity must be cannot be more than 10.";
        }
    }
    return response;
};
```

Python

```
import json

def lambda_handler(event, context):
    # Using dict.get() for safe access to nested dictionary values
    request_data = event.get('requestData', {})
    target_model = request_data.get('targetModel', {})
    target_name = request_data.get('targetName', '')

    response = {
        "hookStatus": "SUCCESS",
        "message": "ReadCapacity is correctly configured.",
        "clientRequestToken": event.get('clientRequestToken')
    }

    if target_name == "AWS::DynamoDB::Table":
        # Safely navigate nested dictionary
        resource_properties = target_model.get('resourceProperties', {})
        provisioned_throughput = resource_properties.get('ProvisionedThroughput',
 {})
        read_capacity = provisioned_throughput.get('ReadCapacityUnits')

        if read_capacity and read_capacity > 10:
            response['hookStatus'] = "FAILED"
            response['errorCode'] = "NonCompliant"
            response['message'] = "ReadCapacity must be cannot be more than 10."

    return response
```

Lambda フックを使用したスタックオペレーションの評価

新しいテンプレートを使用してスタックを作成、更新、または削除するときはいつでも、新しいテンプレートを評価してスタックオペレーションの進行をブロックできるように CloudFormation

Lambda フックを設定できます。フック設定でSTACKオペレーションをターゲットにするように CloudFormation Lambda フックTargetOperationsを設定できます。

トピック

- Lambda フックスタックの入力構文
- Lambda フックスタック変更入力の例
- スタックオペレーションの Lambda 関数の例

Lambda フックスタックの入力構文

スタックオペレーションで Lambda が呼び出されると、フック呼び出しコンテキスト、、actionInvocationPointおよびリクエストコンテキストを含む JSON リクエストを受け取ります。CloudFormation テンプレートのサイズと Lambda 関数で受け入れられる入力サイズが限られているため、実際のテンプレートは Amazon S3 オブジェクトに保存されます。の入力requestDataには、現在および以前のテンプレートバージョンを含む別のオブジェクトへの Amazon S3 署名付き URL が含まれます。

JSON 入力のシェイプの例を次に示します。

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

```
    }
}
```

clientRequesttoken

Hook リクエストへの入力として提供されたリクエストトークン。これは必須のフィールドです。

awsAccountId

評価対象のスタック AWS アカウント を含む の ID。

stackID

CloudFormation スタックのスタック ID。

changeSetId

フック呼び出しを開始した変更セットの ID。スタックの変更が Cloud Control API、、、create-stackupdate-stackまたは delete-stackオペレーションによって開始された場合、この値は空です。

hookTypeName

実行中のフックの名前。

hookTypeVersion

実行中のフックのバージョン。

hookModel

LambdaFunction

フックによって呼び出される現在の Lambda ARN。

actionInvocationPoint

フックが実行されるプロビジョニングロジックの正確なポイント。

有効な値: (CREATE_PRE_PROVISION | UPDATE_PRE_PROVISION | DELETE_PRE_PROVISION)

requestData

targetName

この値は STACK になります。

`targetType`

この値は STACK になります。

`targetLogicalId`

スタック名。

`payload`

現在および以前のテンプレート定義を持つ JSON オブジェクトを含む Amazon S3 署名付き URL。

`requestContext`

フックが再呼び出されている場合、このオブジェクトが設定されます。

`invocation`

フックの現在の実行試行。

`callbackContext`

フックが に設定`IN_PROGRESS`され、返`callbackContext`された場合、再呼び出し時にここに表示されます。

リクエストデータの `payload`プロパティは、コードが取得する必要がある URL です。URL を受信すると、次のスキーマを持つオブジェクトを取得します。

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

`template`

`create-stack` または に提供された完全な CloudFormation テンプレート`update-stack`。CloudFormation に提供された内容に応じて、JSON または YAML 文字列にすることができます。

`delete-stack` オペレーションでは、この値は空になります。

`previousTemplate`

以前の CloudFormation テンプレート。CloudFormation に提供された内容に応じて、JSON または YAML 文字列にすることができます。

delete-stack オペレーションでは、この値は空になります。

Lambda フックスタック変更入力の例

以下は、スタック変更入力の例です。フックは、 を true ObjectLockEnabled に更新し、Amazon SQS キューを追加する変更を評価しています。

```
{
    "clientRequestToken": "f8da6d11-b23f-48f4-814c-0fb6a667f50e",
    "awsAccountId": "123456789012",
    "stackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/
MyStack/1a2345b6-0000-00a0-a123-00abc0abc000",
    "changeSetId": null,
    "hookTypeName": "my::lambda::stackhook",
    "hookTypeVersion": "00000008",
    "hookModel": {
        "LambdaFunction": "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
    },
    "actionInvocationPoint": "UPDATE_PRE_PROVISION",
    "requestData": {
        "targetName": "STACK",
        "targetType": "STACK",
        "targetLogicalId": "my-cloudformation-stack",
        "payload": "https://s3......"
    },
    "requestContext": {
        "invocation": 1,
        "callbackContext": null
    }
}
```

payload の例を次に示しますrequestData。

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

スタックオペレーションの Lambda 関数の例

次の例は、スタックオペレーションペイロードをダウンロードし、テンプレート JSON を解析してを返すシンプルな関数ですSUCCESS。

Node.js

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

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

Python

Python を使用するには、requestsライブラリをインポートする必要があります。これを行うには、Lambda 関数を作成するときに、デプロイパッケージに ライブラリを含める必要があります。詳細については、「 デベロッパーガイド」の「依存関係を持つ .zip デプロイパッケージの作成」を参照してください。 AWS Lambda

```
import json
import requests

def lamnbda_handler(event, context):
    # Safely access nested dictionary values
    request_data = event.get('requestData', {})
    target_type = request_data.get('targetType')
    payload_url = request_data.get('payload')

    response = {
        "hookStatus": "SUCCESS",
        "message": "Stack update is compliant",
        "clientRequestToken": event.get('clientRequestToken')
    }

    try:
        # Fetch the payload
        template_hook_payload_request = requests.get(payload_url)
        template_hook_payload_request.raise_for_status()  # Raise an exception for
 bad responses
        template_hook_payload = template_hook_payload_request.json()

        if 'template' in template_hook_payload:
            # Do something with the template template_hook_payload['template']
            # JSON or YAML
            pass

        if 'previousTemplate' in template_hook_payload:
            # Do something with the template
 template_hook_payload['previousTemplate']
            # JSON or YAML
            pass

    except Exception as error:
        print(error)
        response['hookStatus'] = "FAILED"
        response['message'] = "Failed to evaluate stack operation."
        response['errorCode'] = "InternalFailure"

    return response
```

Lambda フックを使用した変更セットオペレーションの評価

変更セットを作成するたびに、新しい変更セットを最初に評価し、その実行をブロックするように CloudFormation Lambda フックを設定できます。フック設定でCHANGE_SETオペレーションをターゲットにするように CloudFormation Lambda フックTargetOperationsを設定できます。

トピック

- Lambda フック変更セットの入力構文
- Lambda フック変更セット変更入力の例
- 変更セットオペレーションの Lambda 関数の例

Lambda フック変更セットの入力構文

変更セットオペレーションの入力はスタックオペレーションに似ていますが、 のペイロードには、変更セットによって導入されたリソース変更のリストrequestDataも含まれています。

JSON 入力のシェイプの例を次に示します。

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

clientRequesttoken

Hook リクエストへの入力として提供されたリクエストトークン。これは必須のフィールドです。

awsAccountId

評価対象のスタック AWS アカウント を含む の ID。

stackID

CloudFormation スタックのスタック ID。

changeSetId

フック呼び出しを開始した変更セットの ID。

hookTypeName

実行中のフックの名前。

hookTypeVersion

実行中のフックのバージョン。

hookModel

LambdaFunction

フックによって呼び出される現在の Lambda ARN。

requestData

targetName

この値は CHANGE_SET になります。

targetType

この値は CHANGE_SET になります。

targetLogicalId

変更セット ARN。

payload

現在のテンプレートを持つ JSON オブジェクトと、この変更セットによって導入された変更のリストを含む Amazon S3 署名付き URL。

`requestContext`

フックが再呼び出されている場合、このオブジェクトが設定されます。

`invocation`

フックの現在の実行試行。

`callbackContext`

フックが に設定`IN_PROGRESS`され、返`callbackContext`された場合、再呼び出し時にここに表示されます。

リクエストデータの `payload`プロパティは、コードが取得する必要がある URL です。URL を受信すると、次のスキーマを持つオブジェクトを取得します。

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

`template`

`create-stack` または に提供された完全な CloudFormation テンプレート`update-stack`。CloudFormation に提供された内容に応じて、JSON または YAML 文字列にすることができます。

`changedResources`

変更されたリソースのリスト。

`action`

リソースに適用される変更のタイプ。

有効な値: (CREATE | UPDATE | DELETE)

`beforeContext`

変更前のリソースプロパティの JSON 文字列。この値は、リソースの作成時に null になります。この JSON 文字列のすべてのブール値と数値は STRINGS です。

`afterContext`

この変更セットが実行された場合のリソースプロパティの JSON 文字列。リソースが削除されている場合、この値は null です。この JSON 文字列のすべてのブール値と数値は STRINGS です。

`lineNumber`

この変更の原因となったテンプレートの行番号。アクションが の場合、DELETEこの値は null になります。

`logicalResourceId`

変更するリソースの論理リソース ID。

`resourceType`

変更されるリソースタイプ。

Lambda フック変更セット変更入力の例

以下は、変更セットの変更入力の例です。次の例では、変更セットによって導入された変更を確認できます。最初の変更は、 というキューの削除ですCoolQueue。2 番目の変更は、 という新しいキューの追加ですNewCoolQueue。最後の変更は、 の更新ですDynamoDBTable。

```
{
    "clientRequestToken": "f8da6d11-b23f-48f4-814c-0fb6a667f50e",
    "awsAccountId": "123456789012",
    "stackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/
MyStack/1a2345b6-0000-00a0-a123-00abc0abc000",
    "changeSetId": "arn:aws:cloudformation:us-west-2:123456789012:changeSet/
SampleChangeSet/1a2345b6-0000-00a0-a123-00abc0abc000",
    "hookTypeName": "my::lambda::changesethook",
    "hookTypeVersion": "00000008",
    "hookModel": {
        "LambdaFunction": "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
    },
    "actionInvocationPoint": "CREATE_PRE_PROVISION",
```

```
    "requestData": {
        "targetName": "CHANGE_SET",
        "targetType": "CHANGE_SET",
        "targetLogicalId": "arn:aws:cloudformation:us-west-2:123456789012:changeSet/
SampleChangeSet/1a2345b6-0000-00a0-a123-00abc0abc000",
        "payload": "https://s3......"
    },
    "requestContext": {
        "invocation": 1,
        "callbackContext": null
    }
}
```

payload の例を次に示しますrequestData.payload。

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

```
      logicalResourceId: 'NewSQSQueue',
      resourceType: 'AWS::SQS::Queue',
      action: 'CREATE',
      lineNumber: 14,
      beforeContext: null,
      afterContext: '{"Properties":{"QueueName":"NewCoolQueue"}}'
    },
    {
      logicalResourceId: 'DynamoDBTable',
      resourceType: 'AWS::DynamoDB::Table',
      action: 'UPDATE',
      lineNumber: 2,
      beforeContext: '{"Properties":
{"BillingMode":"PAY_PER_REQUEST","AttributeDefinitions":
[{"AttributeType":"S","AttributeName":"PK"}],"KeySchema":
[{"KeyType":"HASH","AttributeName":"PK"}]}}',
      afterContext: '{"Properties":
{"BillingMode":"PAY_PER_REQUEST","PointInTimeRecoverySpecification":
{"PointInTimeRecoveryEnabled":"false"},"AttributeDefinitions":
[{"AttributeType":"S","AttributeName":"PK"}],"KeySchema":
[{"KeyType":"HASH","AttributeName":"PK"}]}}'
    }
  ]
}
```

変更セットオペレーションの Lambda 関数の例

次の例は、変更セットオペレーションペイロードをダウンロードし、各変更をループして、 を返す前に前後のプロパティを出力するシンプルな関数ですSUCCESS。

Node.js

```
export const handler = async (event, context) => {
    var payloadUrl = event?.requestData?.payload;
    var response = {
        "hookStatus": "SUCCESS",
        "message": "Change set changes are compliant",
        "clientRequestToken": event.clientRequestToken
    };
    try {
        const changeSetHookPayloadRequest = await fetch(payloadUrl);
        const changeSetHookPayload = await changeSetHookPayloadRequest.json();
        const changes = changeSetHookPayload.changedResources || [];
```

```
        for(const change of changes) {
            var beforeContext = {};
            var afterContext = {};
            if(change.beforeContext) {
                beforeContext = JSON.parse(change.beforeContext);
            }
            if(change.afterContext) {
                afterContext = JSON.parse(change.afterContext);
            }
            console.log(beforeContext)
            console.log(afterContext)
            // Evaluate Change here
        }
    } catch (error) {
        console.log(error);
        response.hookStatus = "FAILED";
        response.message = "Failed to evaluate change set operation.";
        response.errorCode = "InternalFailure";
    }
    return response;
};
```

Python

Python を使用するには、requestsライブラリをインポートする必要があります。これを行うには、Lambda 関数を作成するときに、デプロイパッケージに ライブラリを含める必要があります。詳細については、「 デベロッパーガイド」の「依存関係を持つ .zip デプロイパッケージの作成」を参照してください。 AWS Lambda

```
import json
import requests

def lambda_handler(event, context):
    payload_url = event.get('requestData', {}).get('payload')
    response = {
        "hookStatus": "SUCCESS",
        "message": "Change set changes are compliant",
        "clientRequestToken": event.get('clientRequestToken')
    }

    try:
        change_set_hook_payload_request = requests.get(payload_url)
```

```
        change_set_hook_payload_request.raise_for_status()  # Raises an HTTPError
for bad responses
        change_set_hook_payload = change_set_hook_payload_request.json()

        changes = change_set_hook_payload.get('changedResources', [])

        for change in changes:
            before_context = {}
            after_context = {}

            if change.get('beforeContext'):
                before_context = json.loads(change['beforeContext'])

            if change.get('afterContext'):
                after_context = json.loads(change['afterContext'])

            print(before_context)
            print(after_context)
            # Evaluate Change here

    except requests.RequestException as error:
        print(error)
        response['hookStatus'] = "FAILED"
        response['message'] = "Failed to evaluate change set operation."
        response['errorCode'] = "InternalFailure"
    except json.JSONDecodeError as error:
        print(error)
        response['hookStatus'] = "FAILED"
        response['message'] = "Failed to parse JSON payload."
        response['errorCode'] = "InternalFailure"

    return response
```

Lambda フックを作成する準備をする

Lambda フックを作成する前に、次の前提条件を満たす必要があります。

- Lambda 関数を既に作成しておく必要があります。詳細については、「フックの Lambda 関数を作成する」を参照してください。
- フックを作成するユーザーまたはロールには、フックをアクティブ化するための十分なアクセス許可が必要です。

- AWS CLI または SDK を使用して Lambda フックを作成するには、IAM アクセス許可と信頼ポリシーを持つ実行ロールを手動で作成して、CloudFormation が Lambda フックを呼び出せるようにする必要があります。

Lambda フックの実行ロールを作成する

フックは、 でそのフックを呼び出すために必要なアクセス許可の実行ロールを使用します AWS アカウント。

このロールは、 から Lambda フックを作成すると自動的に作成できます AWS Management Console。それ以外の場合は、このロールを自分で作成する必要があります。

次のセクションでは、Lambda フックを作成するためのアクセス許可を設定する方法を示します。

必要なアクセス許可

「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用してロールを作成する」のガイダンスに従って、カスタム信頼ポリシーを使用してロールを作成します。

次に、次の手順を実行してアクセス許可を設定します。

1. Lambda フックの作成に使用する IAM ロールに、次の最小権限ポリシーをアタッチします。

   JSON

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

2. ロールに信頼ポリシーを追加して、ロールを引き受けるアクセス許可をフックに付与します。使用できる信頼ポリシーの例を次に示します。

JSON

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

アカウントで Lambda フックをアクティブ化する

次のトピックでは、アカウントで Lambda フックをアクティブ化する方法を示します。これにより、アクティブ化されたアカウントとリージョンで使用できます。

トピック

- Lambda フックをアクティブ化する (コンソール)
- Lambda フックをアクティブ化する (AWS CLI)
- 関連リソース

Lambda フックをアクティブ化する (コンソール)

アカウントで使用する Lambda フックをアクティブ化するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。
2. 画面上部のナビゲーションバーで、フックを作成する AWS リージョン を選択します。
3. フックの Lambda 関数を作成していない場合は、次の操作を行います。
   - Lambda コンソールで [Functions (関数)] ページを開きます。

- このフックで使用する Lambda 関数を作成し、この手順に戻ります。詳細については、「Lambda フックのリソースを評価する Lambda 関数を作成する」を参照してください。

Lambda 関数を既に作成している場合は、次のステップに進みます。

4. 左側のナビゲーションペインで、フックを選択します。
5. フック名で、次のいずれかのオプションを選択します。

   - の後に追加される短いわかりやすい名前を指定しますPrivate::Lambda::。たとえば、 と入力すると*MyTestHook*、完全なフック名は になりますPrivate::Lambda::*MyTestHook*。
   - 次の形式を使用して、完全なフック名 (エイリアスとも呼ばれます) を指定します。*Provider*::*ServiceName*::*HookName*

6. Lambda 関数の場合は、このフックで使用する Lambda 関数を指定します。次を使用できます。

   - サフィックスのない完全な Amazon リソースネーム (ARN)。
   - バージョンまたはエイリアスのサフィックスを持つ修飾 ARN。

7. フックターゲットで、評価対象を選択します。

   - スタック — ユーザーがスタックを作成、更新、または削除するときにスタックテンプレートを評価します。
   - リソース — ユーザーがスタックを更新するときに、個々のリソースの変更を評価します。
   - 変更セット — ユーザーが変更セットを作成するときに、計画された更新を評価します。
   - Cloud Control API — Cloud Control API によって開始された作成、更新、または削除オペレーションを評価します。

8. アクション で、フックを呼び出すアクション (作成、更新、削除) を選択します。
9. フックモードでは、フックによって呼び出された Lambda 関数がレスポンスを返したときにフックがFAILED応答する方法を選択します。

   - 警告 — ユーザーに警告を発行しますが、アクションを続行できます。これは、重要でない検証や情報チェックに役立ちます。
   - 失敗 — アクションが続行されないようにします。これは、厳格なコンプライアンスまたはセキュリティポリシーを適用するのに役立ちます。

10. 実行ロールで、フックが Lambda 関数を呼び出すために引き受ける IAM ロールを選択します。CloudFormation が実行ロールを自動的に作成できるようにするか、作成したロールを指定できます。
11. [次へ] を選択します。
12. (オプション) フックフィルターの場合は、次の操作を行います。

    a. リソースフィルターで、フックを呼び出すことができるリソースタイプを指定します。これにより、フックは関連するリソースに対してのみ呼び出されます。

    b. フィルタリング条件で、スタック名とスタックロールフィルターを適用するロジックを選択します。

    - すべてのスタック名とスタックロール – フックは、指定されたすべてのフィルターが一致する場合にのみ呼び出されます。
    - スタック名とスタックロール – 指定されたフィルターの少なくとも 1 つが一致すると、フックが呼び出されます。

    > **Note**
    >
    > Cloud Control API オペレーションでは、すべてのスタック名とスタックロールフィルターは無視されます。

    c. スタック名には、フック呼び出しに特定のスタックを含めるか除外します。

    - 含める には、含めるスタック名を指定します。これは、ターゲットにする特定のスタックの小さなセットがある場合に使用します。このリストで指定されたスタックのみがフックを呼び出します。
    - Exclude には、除外するスタック名を指定します。これは、ほとんどのスタックでフックを呼び出しますが、いくつかの特定のスタックを除外する場合に使用します。ここにリストされているスタックを除くすべてのスタックは、 フックを呼び出します。

    d. スタックロールの場合、関連付けられた IAM ロールに基づいてフック呼び出しに特定のスタックを含めるか除外します。

    - Include には、これらのロールに関連付けられたスタックをターゲットにする 1 つ以上の IAM ロール ARNs を指定します。これらのロールによって開始されたスタックオペレーションのみがフックを呼び出します。

- Exclude には、除外するスタックの 1 つ以上の IAM ロール ARNs を指定します。フックは、指定されたロールによって開始されたスタックを除くすべてのスタックで呼び出されます。

13. [次へ] を選択します。
14. 確認とアクティブ化ページで、選択内容を確認します。変更するには、関連セクションで [編集] をクリックします。
15. 続行する準備ができたら、Activate Hook を選択します。

Lambda フックをアクティブ化する (AWS CLI)

続行する前に、Lambda 関数と、このフックで使用する実行ロールが作成されていることを確認します。詳細については、「Lambda フックのリソースを評価する Lambda 関数を作成する」および「Lambda フックの実行ロールを作成する」を参照してください。

アカウントで使用する Lambda フックをアクティブ化するには (AWS CLI)

1. フックのアクティブ化を開始するには、次のactivate-typeコマンドを使用して、プレースホルダーを特定の値に置き換えます。このコマンドは、フックが から指定された実行ロールを使用することを許可します AWS アカウント。

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

2. フックのアクティブ化を完了するには、JSON 設定ファイルを使用してフックを設定する必要があります。

   cat コマンドを使用して、次の構造で JSON ファイルを作成します。詳細については、「フック設定スキーマ構文リファレンス」を参照してください。

```
$ cat > config.json
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
```

```
        "CLOUD_CONTROL"
      ],
      "FailureMode": "WARN",
      "Properties": {
        "LambdaFunction": "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction"
      },
      "TargetFilters": {
        "Actions": [
          "CREATE",
          "UPDATE",
          "DELETE"
        ]
      }
    }
  }
}
```

- HookInvocationStatus: フックを有効にするENABLEDには、 に設定します。
- TargetOperations: フックが評価するオペレーションを指定します。
- FailureMode: FAIL または WARN に設定します。
- LambdaFunction: Lambda 関数の ARN を指定します。
- TargetFilters: フックを呼び出すアクションのタイプを指定します。

3. 次のset-type-configurationコマンドと作成した JSON ファイルを使用して、設定を適用します。プレースホルダーを特定の値に置き換えます。

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

関連リソース

CloudFormation スタックテンプレートで Lambda フックを宣言する方法を理解するために使用できるテンプレートの例を示します。詳細については、AWS CloudFormation ユーザーガイドのAWS::CloudFormation::LambdaHookを参照してください。

アカウントの Lambda フックのログを表示する

Lambda フックを使用する場合、検証出力レポートのログファイルは Lambda コンソールにあります。

Lambda コンソールで Lambda フックログを表示する

Lambda Hook 出力ログファイルを表示するには

1. Lambda コンソールにサインインします。
2. 画面上部にあるナビゲーションバーで、 AWS リージョンを選択します。
3. 関数を選択します。
4. 目的の Lambda 関数を選択します。
5. [テスト] タブを選択します。
6. CloudWatch Logs ライブ証跡を選択する
7. ドロップダウンメニューを選択し、表示するロググループを選択します。
8. [開始] を選択します。ログは CloudWatch Logs Live Trail ウィンドウに表示されます。列で表示またはプレーンテキストで表示を選択します。

    - フィルターパターンの追加フィールドに追加することで、結果にフィルターを追加できます。このフィールドでは、結果をフィルタリングして、指定されたパターンに一致するイベントのみを含めることができます。

Lambda 関数のログの表示の詳細については、「Lambda 関数の CloudWatch Logs の表示」を参照してください。

アカウントで Lambda フックを削除する

アクティブ化された Lambda フックが不要になった場合は、次の手順を使用してアカウントで削除します。

フックを削除する代わりに一時的に無効にするには、「」を参照してくださいAWS CloudFormation フックの無効化と有効化。

トピック

- アカウントで Lambda フックを削除する (コンソール)
- アカウントで Lambda フックを削除する (AWS CLI)

アカウントで Lambda フックを削除する (コンソール)

アカウントの Lambda フックを削除するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。
2. 画面上部のナビゲーションバーで、フックがある を選択します AWS リージョン 。
3. ナビゲーションペインから、フックを選択します。
4. フックページで、削除する Lambda フックを見つけます。
5. フックの横にあるチェックボックスをオンにし、削除を選択します。
6. 確認を求められたら、フック名を入力して、指定されたフックの削除を確認し、削除を選択します。

アカウントで Lambda フックを削除する (AWS CLI)

> **Note**
>
> フックを削除する前に、まずフックを無効にする必要があります。詳細については、「アカウントでフックを無効化および有効化する (AWS CLI)」を参照してください。

次のdeactivate-typeコマンドを使用してフックを非アクティブ化し、アカウントから削除します。プレースホルダーを特定の値に置き換えます。

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

CloudFormation CLI を使用したカスタムフックの開発

このセクションは、カスタムフックを開発してレジストリに登録したいお客様を対象としています AWS CloudFormation 。

カスタムフックの開発には、主に 3 つのステップがあります。

1. 開始

カスタムフックを開発するには、CloudFormation CLI を設定して使用する必要があります。Hook のプロジェクトとその必要なファイルを開始するには、CloudFormation CLI init コマンドを使用して、フックを作成することを指定します。詳細については、「カスタム AWS CloudFormation フックプロジェクトの開始」を参照してください。

2. [Model] (モデル)

   フックスキーマをモデル化、作成、検証するには、フック、そのプロパティ、および属性を定義します。

   CloudFormation CLI は、特定のフック呼び出しポイントに対応する空のハンドラー関数を作成します。これらのハンドラーに独自のロジックを追加して、ターゲットライフサイクルの各段階でフック呼び出し中に何が起こるかを制御します。詳細については、「カスタム AWS CloudFormation フックのモデリング」を参照してください。

3. 登録

   フックを登録するには、フックを送信してプライベートまたはパブリックのサードパーティー拡張機能として登録します。フックを `submit`オペレーションに登録します。詳細については、「カスタムフックを に登録する AWS CloudFormation」を参照してください。

   フックの登録には、次のタスクが関連しています。

   a. Publish – フックはレジストリに発行されます。

   b. Configure – フックは、タイプ設定がスタックに対して呼び出されたときに設定されます。

> **Note**
>
> フックは 30 秒後にタイムアウトし、最大 3 回再試行します。詳細については、「タイムアウトと再試行の制限」を参照してください。

以下のトピックでは、Python または Java でカスタムフックを開発、登録、公開するプロセスについて説明します。

トピック

- カスタム AWS CloudFormation フックを開発するための前提条件
- カスタム AWS CloudFormation フックプロジェクトの開始
- カスタム AWS CloudFormation フックのモデリング

- カスタムフックを に登録する AWS CloudFormation
- でのカスタムフックのテスト AWS アカウント
- カスタムフックの更新
- CloudFormation レジストリからのカスタムフックの登録解除
- 公開用フックの公開
- AWS CloudFormation フックのスキーマ構文リファレンス

カスタム AWS CloudFormation フックを開発するための前提条件

Java または Python を使用してカスタムフックを開発できます。カスタムフックを開発するための前提条件は次のとおりです。

Java の前提条件

- Apache Maven
- JDK 17

> **Note**
>
> CloudFormation コマンドラインインターフェイス (CLI) を使用して Java 用のフックプロジェクトを開始する場合は、Python 3.8 以降もインストールする必要があります。CloudFormation CLI 用の Java プラグインは、Python でディストリビューションされる `pip` (Python のパッケージマネージャー) を介してインストールできます。

Java Hooks プロジェクトのフックハンドラーを実装するには、Java Hook ハンドラーのサンプルファイルをダウンロードできます。

Python の前提条件

- Python バージョン 3.8 以降。

Python Hooks プロジェクトのフックハンドラーを実装するには、Python Hook ハンドラーのサンプルファイルをダウンロードします。

フックを開発するためのアクセス許可

CloudFormation `Create`、、`Update`および `Delete`スタックのアクセス許可に加えて、次の AWS CloudFormation オペレーションにアクセスする必要があります。これらのオペレーションへのアクセスは、IAM ロールの CloudFormation ポリシーを通じて管理されます。

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

フックの開発環境を設定する

フックを開発するには、CloudFormation テンプレートと Python または Java に精通している必要があります。

CloudFormation CLI および関連するプラグインをインストールするには：

1. Python パッケージマネージャー`pip`である を使用して CloudFormation CLI をインストールします。

   ```
   pip3 install cloudformation-cli
   ```

2. CloudFormation CLI 用の Python または Java プラグインをインストールします。

   Python

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

   Java

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

CloudFormation CLI とプラグインをアップグレードするには、アップグレードオプションを使用できます。

Python

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

Java

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

カスタム AWS CloudFormation フックプロジェクトの開始

カスタムフックプロジェクトを作成する最初のステップは、プロジェクトを開始することです。CloudFormation CLI `init` コマンドを使用して、カスタムフックプロジェクトを開始できます。

`init` コマンドは、フックスキーマファイルを含むプロジェクトの設定を順を追って説明するウィザードを起動します。このスキーマファイルをフックのシェイプとセマンティクスを定義するための開始点として使用します。詳細については、「スキーマ構文」を参照してください。

フックプロジェクトを開始するには：

1. プロジェクトのディレクトリを作成します。

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

2. 新しいディレクトリに移動します。

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

3. CloudFormation CLI `init` コマンドを使用してプロジェクトを開始します。

   ```
   cfn init
   ```

   このコマンドは、以下の出力を返します。

   ```
   Initializing new project
   ```

4. `init` コマンドは、プロジェクトの設定手順を示すウィザードを起動します。プロンプトが表示されたら、 hを入力してフックプロジェクトを指定します。

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

```
h
```

5. フックタイプの名前を入力します。

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

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

6. 言語プラグインが 1 つだけインストールされている場合、デフォルトで選択されます。複数の言語プラグインがインストールされている場合は、目的の言語を選択できます。選択した言語の番号選択を入力します。

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

7. 選択した開発言語に基づいてパッケージを設定します。

   Python

   (オプション) プラットフォームに依存しないパッケージングには Docker を選択します。Docker は必須ではありませんが、パッケージ化を容易にすることを強くお勧めします。

```
Use docker for platform-independent packaging (Y/n)?
This is highly recommended unless you are experienced with cross-platform Python
 packaging.
```

   Java

   Java パッケージ名を設定し、codegen モデルを選択します。デフォルトのパッケージ名を使用するか、新しいパッケージ名を作成できます。

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

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

結果: プロジェクトを正常に開始し、フックの開発に必要なファイルを生成しました。以下は、Python 3.8 のフックプロジェクトを構成するディレクトリとファイルの例です。

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

> **Note**
>
> src ディレクトリ内のファイルは、言語の選択に基づいて作成されます。生成されたファイルには、役立つコメントと例がいくつかあります。などの一部のファイルは、後のステップで generate コマンドを実行してハンドラーのランタイムコードを追加するときにmodels.py自動的に更新されます。

カスタム AWS CloudFormation フックのモデリング

カスタム AWS CloudFormation フックのモデリングには、フック、そのプロパティ、および属性を定義するスキーマの作成が含まれます。cfn init コマンドを使用してカスタムフックプロジェクトを作成すると、フックスキーマの例が JSON 形式のテキストファイル として作成されます*hook-name*.json。

ターゲット呼び出しポイントとターゲットアクションは、フックが呼び出される正確なポイントを指定します。フックハンドラーは、これらのポイントの実行可能なカスタムロジックをホストします。例えば、 CREATEオペレーションのターゲットアクションはpreCreateハンドラーを使用します。ハンドラーに記述されたコードは、フックターゲットとサービスが一致するアクションを実行するとを呼び出します。フックターゲットは、フックが呼び出される送信先です。 AWS CloudFormation パブリックリソース、プライベートリソース、カスタムリソースなどのターゲットを指定できます。フックは、フックターゲットを無制限にサポートします。

スキーマには、フックに必要なアクセス許可が含まれています。フックを作成するには、各フックハンドラーのアクセス許可を指定する必要があります。CloudFormation では、最小特権を付与するか、タスクの実行に必要なアクセス許可のみを付与するという標準的なセキュリティアドバイスに従うポリシーを作成することを作成者にお勧めします。ユーザー (およびロール) が実行する必要がある操作を決定し、フック操作のタスクのみを実行できるようにするポリシーを作成します。CloudFormation は、これらのアクセス許可を使用して、フックユーザーが提供するアクセス許可をスコープダウンします。これらのアクセス許可はフックに渡されます。フックハンドラーは、これらのアクセス許可を使用して AWS リソースにアクセスします。

フックを定義する開始点として、次のスキーマファイルを使用できます。フックスキーマを使用して、実装するハンドラーを指定します。特定のハンドラーを実装しない場合は、フックスキーマのハンドラーのセクションから削除します。スキーマの詳細については、「」を参照してくださいスキーマ構文。

```
{
    "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
}
```

トピック

- Java を使用したカスタム AWS CloudFormation フックのモデリング
- Python を使用したカスタム AWS CloudFormation フックのモデリング

Java を使用したカスタム AWS CloudFormation フックのモデリング

カスタム AWS CloudFormation フックのモデリングには、フック、そのプロパティ、および属性を定義するスキーマの作成が含まれます。このチュートリアルでは、Java を使用してカスタムフックをモデリングする手順について説明します。

ステップ 1: プロジェクトの依存関係を追加する

Java ベースのフックプロジェクトは、依存関係として Maven の pom.xml ファイルに依存します。次のセクションを展開し、ソースコードをプロジェクトのルートにある pom.xml ファイルにコピーします。

フックプロジェクトの依存関係 (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>
```

ステップ 2: Hook プロジェクトパッケージを生成する

Hook プロジェクトパッケージを生成します。CloudFormation CLI は、フック仕様で定義されているように、ターゲットライフサイクル内の特定のフックアクションに対応する空のハンドラー関数を作成します。

```
cfn generate
```

このコマンドは、以下の出力を返します。

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

> **Note**
>
> 廃止されたバージョンを使用しないように、Lambda ランタイムがup-to-dateであることを確認します。詳細については、「リソースタイプとフックの Lambda ランタイムの更新」を参照してください。

ステップ 3: フックハンドラーを追加する

実装するハンドラーに独自のフックハンドラーランタイムコードを追加します。例えば、ログ記録に次のコードを追加できます。

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

CloudFormation CLI は Plain Old Java Objects (Java POJO) を生成します。以下は、 から生成された出力例です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<ServerSideEncryptionRule> serverSideEncryptionConfiguration;

}
```

Example ServerSideEncryptionRule.java

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

import ...
```

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

    @JsonProperty("ServerSideEncryptionByDefault")
    private ServerSideEncryptionByDefault serverSideEncryptionByDefault;

}
```

Example ServerSideEncryptionByDefault.java

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

import ...


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

    @JsonProperty("KMSMasterKeyID")
    private String kMSMasterKeyID;

}
```

POJOs が生成されると、フックの機能を実際に実装するハンドラーを記述できるようになりました。この例では、ハンドラーの `preCreate`および `preUpdate` 呼び出しポイントを実装します。

ステップ 4: フックハンドラーを実装する

トピック

- API クライアントビルダーのコーディング
- API リクエストメーカーのコーディング
- ヘルパーコードの実装
- ベースハンドラーの実装
- preCreate ハンドラーの実装
- preCreate ハンドラーのコーディング
- preCreate テストの更新
- preUpdate ハンドラーの実装
- preUpdate ハンドラーのコーディング
- preUpdate テストの更新
- preDelete ハンドラーの実装
- preDelete ハンドラーのコーディング
- preDelete ハンドラーの更新

API クライアントビルダーのコーディング

1. IDE で、 `src/main/java/com/mycompany/testing/mytesthook`フォルダにある `ClientBuilder.java` ファイルを開きます。
2. `ClientBuilder.java` ファイルの内容全体を次のコードに置き換えます。

   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();
    }
}
```

API リクエストメーカーのコーディング

1. IDE で、 src/main/java/com/mycompany/testing/mytesthookフォルダにある Translator.java ファイルを開きます。
2. Translator.java ファイルの内容全体を次のコードに置き換えます。

   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();
    }
}
```

ヘルパーコードの実装

1. IDE で、 `src/main/java/com/mycompany/testing/mytesthook`フォルダにある `AbstractTestBase.java` ファイルを開きます。
2. `AbstractTestBase.java` ファイルの内容全体を次のコードに置き換えます。

   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
                )
        );
    }
}
```

ベースハンドラーの実装

1. IDE で、 src/main/java/com/mycompany/testing/mytesthookフォルダにある BaseHookHandlerStd.java ファイルを開きます。
2. BaseHookHandlerStd.java ファイルの内容全体を次のコードに置き換えます。

   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);
        }
```

```
    }
}
```

preCreate ハンドラーの実装

preCreate ハンドラーは、 AWS::S3::Bucketまたは AWS::SQS::Queueリソースのサーバー側の暗号化設定を検証します。

- AWS::S3::Bucket リソースの場合、フックは以下が true の場合にのみ渡されます。
  - Amazon S3 バケット暗号化が設定されています。
  - Amazon S3 バケットキーはバケットに対して有効になっています。
  - Amazon S3 バケットに設定された暗号化アルゴリズムは、必要な正しいアルゴリズムです。
  - AWS Key Management Service キー ID が設定されます。
- AWS::SQS::Queue リソースの場合、フックは以下が true の場合にのみ渡されます。
  - AWS Key Management Service キー ID が設定されます。

preCreate ハンドラーのコーディング

1. IDE で、 src/main/java/software/mycompany/testing/mytesthookフォルダにある PreCreateHookHandler.java ファイルを開きます。
2. PreCreateHookHandler.java ファイルの内容全体を次のコードに置き換えます。

```
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();
    }
}
```

preCreate テストの更新

1. IDE で、 src/test/java/software/mycompany/testing/mytesthookフォルダにある PreCreateHandlerTest.java ファイルを開きます。
2. PreCreateHandlerTest.java ファイルの内容全体を次のコードに置き換えます。

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

import com.google.common.collect.ImmutableMap;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.AwsS3Bucket;
import com.mycompany.testing.mytesthook.model.aws.s3.bucket.BucketEncryption;
import
 com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionByDefault;
import
 com.mycompany.testing.mytesthook.model.aws.s3.bucket.ServerSideEncryptionRule;
import com.mycompany.testing.mytesthook.model.aws.sqs.queue.AwsSqsQueue;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import software.amazon.cloudformation.exceptions.UnsupportedTargetException;
import software.amazon.cloudformation.proxy.AmazonWebServicesClientProxy;
```

```
import software.amazon.cloudformation.proxy.HandlerErrorCode;
import software.amazon.cloudformation.proxy.Logger;
import software.amazon.cloudformation.proxy.hook.HookContext;
import software.amazon.cloudformation.proxy.hook.HookHandlerRequest;
import software.amazon.cloudformation.proxy.hook.HookProgressEvent;
import software.amazon.cloudformation.proxy.hook.HookStatus;
import software.amazon.cloudformation.proxy.hook.targetmodel.HookTargetModel;

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

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

@ExtendWith(MockitoExtension.class)
public class PreCreateHookHandlerTest {

    @Mock
    private AmazonWebServicesClientProxy proxy;

    @Mock
    private Logger logger;

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

```
        assertResponse(response, HookStatus.FAILED, "Bucket key not enabled for
bucket with name: amzn-s3-demo-bucket");
    }

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

```
    }

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

preUpdate ハンドラーの実装

preUpdate ハンドラーを実装します。ハンドラーは、ハンドラー内のすべての指定されたターゲットの更新オペレーションの前に開始されます。preUpdate ハンドラーは以下を実行します。

- AWS::S3::Bucket リソースの場合、フックは以下が true の場合にのみ渡されます。
  - Amazon S3 バケットのバケット暗号化アルゴリズムは変更されていません。

preUpdate ハンドラーのコーディング

1. IDE で、 src/main/java/software/mycompany/testing/mytesthookフォルダにある PreUpdateHookHandler.java ファイルを開きます。
2. PreUpdateHookHandler.java ファイルの内容全体を次のコードに置き換えます。

```
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();
    }
```

```
}
```

preUpdate テストの更新

1. IDE で、 src/main/java/com/mycompany/testing/mytesthookフォルダの PreUpdateHandlerTest.java ファイルを開きます。
2. PreUpdateHandlerTest.java ファイルの内容全体を次のコードに置き換えます。

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

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

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

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

@ExtendWith(MockitoExtension.class)
public class PreUpdateHookHandlerTest {

```

```
    @Mock
    private AmazonWebServicesClientProxy proxy;

    @Mock
    private Logger logger;

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

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

        final ServerSideEncryptionRule[] serverSideEncryptionRules =
 Stream.of("AES256", "SHA512", "AES32")
```

```
                .map(this::buildServerSideEncryptionRule)
                .toArray(ServerSideEncryptionRule[]::new);

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

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

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

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

        final HookHandlerRequest request = HookHandlerRequest.builder()

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

preDelete ハンドラーの実装

preDelete ハンドラーを実装します。ハンドラーは、ハンドラー内のすべての指定されたターゲットの削除オペレーションの前に開始されます。preDelete ハンドラーは以下を実行します。

- AWS::S3::Bucket リソースの場合、フックは以下が true の場合にのみ渡されます。
  - リソースを削除した後、最小限必要な苦情リソースがアカウントに存在することを確認します。
  - 必要な苦情リソースの最小量は、フックのタイプ設定で設定されます。

preDelete ハンドラーのコーディング

1. IDE で、 src/main/java/com/mycompany/testing/mytesthookフォルダの PreDeleteHookHandler.java ファイルを開きます。
2. PreDeleteHookHandler.java ファイルの内容全体を次のコードに置き換えます。

```
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);
        }
    }
```

```
}
```

preDelete ハンドラーの更新

1. IDE で、 src/main/java/com/mycompany/testing/mytesthookフォルダの PreDeleteHookHandler.java ファイルを開きます。
2. PreDeleteHookHandler.java ファイルの内容全体を次のコードに置き換えます。

```
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();
    }
}
```

Python を使用したカスタム AWS CloudFormation フックのモデリング

カスタム AWS CloudFormation フックのモデリングには、フック、そのプロパティ、および属性を定義するスキーマの作成が含まれます。このチュートリアルでは、Python を使用してカスタムフックをモデリングする方法について説明します。

ステップ 1: Hook プロジェクトパッケージを生成する

Hook プロジェクトパッケージを生成します。CloudFormation CLI は、フック仕様で定義されているように、ターゲットライフサイクル内の特定のフックアクションに対応する空のハンドラー関数を作成します。

```
cfn generate
```

このコマンドは、以下の出力を返します。

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

> **Note**
>
> 廃止されたバージョンを使用しないように、Lambda ランタイムがup-to-dateであることを確認します。詳細については、「リソースタイプとフックの Lambda ランタイムの更新」を参照してください。

ステップ 2: フックハンドラーを追加する

実装するハンドラーに独自のフックハンドラーランタイムコードを追加します。例えば、ログ記録に次のコードを追加できます。

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

CloudFormation CLI は、 から `src/models.py` ファイルを生成します設定のスキーマ。

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
```

ステップ 3: フックハンドラーを実装する

Python データクラスが生成されると、フックの機能を実際に実装するハンドラーを記述できます。この例では、ハンドラーの `preCreate`、`preUpdate`、および `preDelete`呼び出しポイントを実装します。

トピック

- preCreate ハンドラーを実装する
- preUpdate ハンドラーを実装する
- preDelete ハンドラーを実装する
- フックハンドラーを実装する

preCreate ハンドラーを実装する

`preCreate` ハンドラーは、 `AWS::S3::Bucket` または `AWS::SQS::Queue`リソースのサーバー側の暗号化設定を検証します。

- `AWS::S3::Bucket` リソースの場合、フックは以下が true の場合にのみ渡されます。
  - Amazon S3 バケット暗号化が設定されています。
  - Amazon S3 バケットキーはバケットに対して有効になっています。
  - Amazon S3 バケットに設定された暗号化アルゴリズムは、必要な正しいアルゴリズムです。
  - AWS Key Management Service キー ID が設定されます。
- `AWS::SQS::Queue` リソースの場合、フックは以下が true の場合にのみ渡されます。
  - AWS Key Management Service キー ID が設定されます。

preUpdate ハンドラーを実装する

`preUpdate` ハンドラーを実装します。ハンドラーは、ハンドラー内のすべての指定されたターゲットの更新オペレーションの前に開始されます。`preUpdate` ハンドラーは以下を実行します。

- `AWS::S3::Bucket` リソースの場合、フックは以下が true の場合にのみ渡されます。

- Amazon S3 バケットのバケット暗号化アルゴリズムは変更されていません。

preDelete ハンドラーを実装する

preDelete ハンドラーを実装します。ハンドラーは、ハンドラー内のすべての指定されたターゲットの削除オペレーションの前に開始されます。preDelete ハンドラーは以下を実行します。

- AWS::S3::Bucket リソースの場合、フックは以下が true の場合にのみ渡されます。
  - リソースを削除した後、最低限必要な準拠リソースがアカウントに存在することを確認します。
  - 必要な最小準拠リソース量は、フックの設定で設定されます。

フックハンドラーを実装する

1. IDE で、 srcフォルダにある handlers.py ファイルを開きます。
2. handlers.py ファイルの内容全体を次のコードに置き換えます。

   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}")
```

次のトピック「カスタムフックを に登録する AWS CloudFormation」に進みます。

カスタムフックを に登録する AWS CloudFormation

カスタムフックを作成したら、使用 AWS CloudFormation できるように に登録する必要があります。このセクションでは、 で使用するフックをパッケージ化して登録する方法について説明します AWS アカウント。

フックをパッケージ化する (Java)

Java でフックを開発している場合は、Maven を使用してパッケージ化します。

Hook プロジェクトの ディレクトリで、次のコマンドを実行してフックを構築し、ユニットテストを実行して、プロジェクトをファイルとしてパッケージ化します。このJARファイルを使用して、フックを CloudFormation レジストリに送信できます。

```
mvn clean package
```

カスタムフックを登録する

フックを登録するには

1. (オプション) configureオペレーションを送信してus-west-2、デフォルト AWS リージョン名を に設定します。

   ```
   $ 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. (オプション) 次のコマンドは、フックプロジェクトを登録せずにビルドしてパッケージ化します。

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

3. CloudFormation CLI submitオペレーションを使用してフックを登録します。

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

このコマンドは以下のコマンドを返します。

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

結果: フックが正常に登録されました。

アカウントでフックにアクセスできることを確認する

フックが AWS アカウント と、フックを送信したリージョンで使用可能であることを確認します。

1. フックを確認するには、 list-types コマンドを使用して新しく登録したフックを一覧表示し、その概要の説明を返します。

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

コマンドは次の出力を返します。また、 AWS アカウント および リージョンでアクティブ化できる公開されているフックも表示されます。

```
{
    "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. フックのlist-type出力TypeArnから を取得し、保存します。

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

公開用にフックを発行する方法については、「」を参照してください公開用フックの公開。

フックの設定

フックを開発して登録したら、 レジストリに公開 AWS アカウント することで、 でフックを設定できます。

- アカウントでフックを設定するには、 SetTypeConfigurationオペレーションを使用します。このオペレーションにより、フックのスキーマ properties セクションで定義されているフックのプロパティが有効になります。次の例では、 設定1で minBucketsプロパティが に設定されています。

  > **Note**
  >
  > アカウントでフックを有効にすると、 から定義されたアクセス許可を使用するフックを承認します AWS アカウント。CloudFormation は、アクセス許可をフックに渡す前に、不要なアクセス許可を削除します。CloudFormation では、顧客またはフックユーザーがフックのアクセス許可を確認し、アカウントでフックを有効にする前にフックが許可されるアクセス許可に注意することをお勧めします。

  同じ アカウントと で登録されたフック拡張の設定データを指定します AWS リージョン。

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

  > **Important**
  >
  > フックがスタックの設定をプロアクティブに検査できるようにするには、フックがアカウントで登録およびアクティブ化された後、 HookConfigurationセクションHookInvocationStatusENABLEDで を に設定する必要があります。

ハンドラー AWS APIs へのアクセス

フックがいずれかのハンドラーで AWS API を使用する場合、CFN-CLI は IAM 実行ロールテンプレート を自動的に作成しますhook-role.yaml。hook-role.yaml テンプレートは、フックスキーマのハンドラーの セクションで各ハンドラーに指定されたアクセス許可に基づいています。generate オペレーション中に --role-arnフラグが使用されない場合、このスタックのロールはプロビジョニングされ、フックの実行ロールとして使用されます。

詳細については、「リソースタイプからの AWS APIs」を参照してください。

hook-role.yaml テンプレート

> **Note**
>
> 独自の実行ロールを作成する場合は、 hooks.cloudformation.amazonaws.comとのみを一覧表示して最小特権の原則を実践することを強くお勧めしますresources.cloudformation.amazonaws.com。

次のテンプレートでは、IAM、Amazon S3、および Amazon SQS アクセス許可を使用します。

```
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
```

でのカスタムフックのテスト AWS アカウント

呼び出しポイントに対応するハンドラー関数をコーディングしたので、CloudFormation スタックでカスタムフックをテストします。

CloudFormation テンプレートFAILが以下を使用して S3 バケットをプロビジョニングしなかった場合、フック失敗モードは に設定されます。

- Amazon S3 バケット暗号化が設定されています。
- Amazon S3 バケットキーはバケットに対して有効になっています。
- Amazon S3 バケットに設定された暗号化アルゴリズムは、必要な正しいアルゴリズムです。
- AWS Key Management Service キー ID が設定されます。

次の例では、スタック設定に失敗し、リソースがプロビジョニングされる前に停止my-hook-stackする というスタック名my-failed-bucket-stack.ymlで というテンプレートを作成します。

スタックをプロビジョニングしてフックをテストする

例 1: スタックをプロビジョニングするには

非準拠のスタックをプロビジョニングする

1. S3 バケットを指定するテンプレートを作成します。例えば、my-failed-bucket-stack.yml と指定します。

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

2. スタックを作成し、 AWS Command Line Interface () でテンプレートを指定しますAWS CLI。次の例では、スタック名を my-hook-stackに、テンプレート名を に指定しますmy-failed-bucket-stack.yml。

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

3. (オプション) スタック名を指定してスタックの進行状況を表示します。次の例では、スタック名 を指定しますmy-hook-stack。

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

describe-stack-events オペレーションを使用して、バケットの作成中にフックの失敗を確認します。コマンドの出力例を次に示します。

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

結果: フックの呼び出しがスタック設定に失敗し、リソースのプロビジョニングが停止しました。

CloudFormation テンプレートを使用してフック検証に合格する

1. スタックを作成してフック検証に合格するには、リソースが暗号化された S3 バケットを使用するようにテンプレートを更新します。この例では `my-encrypted-bucket-stack.yml` テンプレートを使用しています。

```
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**
>
> スキップされたリソースに対してフックは呼び出されません。

2. スタックを作成し、テンプレートを指定します。この例では、スタック名は ですmy-encrypted-bucket-stack。

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

3. (オプション) スタック名を指定してスタックの進行状況を表示します。

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

describe-stack-events コマンドを使用してレスポンスを表示します。次に describe-stack-events コマンドの例を示します。

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

結果: CloudFormation はスタックを正常に作成しました。フックのロジックは、AWS::S3::Bucketリソースをプロビジョニングする前に、リソースにサーバー側の暗号化が含まれていることを確認しました。

例 2: スタックをプロビジョニングするには

非準拠のスタックをプロビジョニングする

1. S3 バケットを指定するテンプレートを作成します。例えば、 です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. スタックを作成し、 でテンプレートを指定します AWS CLI。次の例では、スタック名を my-hook-stackに、テンプレート名を に指定しますaes256-bucket.yml。

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

3. (オプション) スタック名を指定してスタックの進行状況を表示します。次の例では、スタック名 を指定しますmy-hook-stack。

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

describe-stack-events オペレーションを使用して、バケットの作成中にフックの失敗を確認します。コマンドの出力例を次に示します。

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

結果: フックの呼び出しがスタック設定に失敗し、リソースのプロビジョニングが停止しました。S3 バケットの暗号化が正しく設定されていないため、スタックが失敗しました。フックタイプの設定では、このバケットが を使用しているaws:kms間に が必要ですAES256。

CloudFormation テンプレートを使用してフック検証に合格する

1. スタックを作成してフック検証に合格するには、リソースが暗号化された S3 バケットを使用するようにテンプレートを更新します。この例では kms-bucket-and-queue.yml テンプレートを使用しています。

```
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**
>
> スキップされたリソースに対してフックは呼び出されません。

2. スタックを作成し、テンプレートを指定します。この例では、スタック名は ですmy-encrypted-bucket-stack。

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

3. (オプション) スタック名を指定してスタックの進行状況を表示します。

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

describe-stack-events コマンドを使用してレスポンスを表示します。次に describe-stack-events コマンドの例を示します。

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

結果: CloudFormation はスタックを正常に作成しました。フックのロジックは、AWS::S3::Bucketリソースをプロビジョニングする前に、リソースにサーバー側の暗号化が含まれていることを確認しました。

カスタムフックの更新

カスタムフックを更新すると、フック内のリビジョンを CloudFormation レジストリで使用可能にできます。

カスタムフックを更新するには、CloudFormation CLI submitオペレーションを使用して CloudFormation レジストリにリビジョンを送信します。

```
$ cfn submit
```

アカウントでフックのデフォルトバージョンを指定するには、 set-type-default-version コマンドを使用して、タイプ、タイプ名、バージョン ID を指定します。

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

フックのバージョンに関する情報を取得するには、 を使用しますlist-type-versions。

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

CloudFormation レジストリからのカスタムフックの登録解除

カスタムフックの登録を解除すると、CloudFormation レジストリDEPRECATEDで として拡張機能または拡張機能バージョンがマークされ、アクティブな使用から削除されます。廃止されると、カスタムフックを CloudFormation オペレーションで使用することはできません。

> **Note**
>
> フックの登録を解除する前に、その拡張機能の以前のアクティブなバージョンをすべて個別に登録解除する必要があります。詳細については、「DeregisterType」を参照してください。

フックの登録を解除するには、 deregister-typeオペレーションを使用してフック ARN を指定します。

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

このコマンドは出力を生成しません。

公開用フックの公開

パブリックサードパーティーフックを開発するには、フックをプライベート拡張として開発します。次に、拡張機能を公開する各 AWS リージョン で、次の操作を行います。

1. フックをプライベート拡張として CloudFormation レジストリに登録します。
2. フックをテストして、CloudFormation レジストリで公開するために必要なすべての要件を満たしていることを確認します。
3. フックを CloudFormation レジストリに発行します。

> **Note**
>
> 特定のリージョンで拡張機能を公開する前に、まずそのリージョンで拡張機能パブリッシャーとして登録する必要があります。これを複数のリージョンで同時に実行するには、「 AWS CloudFormation CLI ユーザーガイド」のStackSets を使用した複数のリージョンでの拡張機能の発行」を参照してください。

フックを開発して登録したら、それをサードパーティーのパブリック拡張として CloudFormation レジストリに公開することで、一般的な CloudFormation ユーザーが公開できるようになります。

パブリックサードパーティーフックを使用すると、CloudFormation ユーザーにプロビジョニング前に AWS リソースの設定を事前に検査させることができます。プライベートフックと同様に、パブリックフックは CloudFormation AWS 内で によって発行されたフックと同じように扱われます。

レジストリに公開されたフックは、公開されている AWS リージョン のすべての CloudFormation ユーザーに表示されます。その後、ユーザーはアカウントで拡張機能をアクティブ化し、テンプレートで使用できるようにします。詳細については、「 ユーザーガイド」のCloudFormation レジストリからサードパーティーのパブリック拡張機能を使用する」を参照してください。 AWS CloudFormation

パブリック使用のためのカスタムフックのテスト

登録されたカスタムフックを公開するには、そのフックに定義されているすべてのテスト要件に合格する必要があります。以下は、カスタムフックをサードパーティー拡張機能として公開する前に必要な要件のリストです。

各ハンドラーとターゲットは 2 回テストされます。の場合は 1 回SUCCESS、 の場合は 1 回FAILED。

- SUCCESS レスポンスケースの場合：
  - ステータスは である必要がありますSUCCESS。
  - エラーコードを返さないでください。
  - コールバックの遅延は、指定されている場合は 0秒に設定する必要があります。
- FAILED レスポンスケースの場合：
  - ステータスは である必要がありますFAILED。
  - エラーコードを返す必要があります。
  - レスポンスにメッセージが必要です。
  - コールバックの遅延は、指定されている場合は 0秒に設定する必要があります。
- IN_PROGRESS レスポンスケースの場合：
  - エラーコードを返さないでください。
  - Result レスポンスで フィールドを設定することはできません。

契約テストで使用する入力データの指定

デフォルトでは、 はフックスキーマで定義したパターンから生成された入力プロパティを使用して契約テスト CloudFormation を実行します。ただし、ほとんどのフックは複雑であるため、プ

ロビジョニングスタックを事前作成または事前更新するための入力プロパティには、プロビジョニングされるリソースの理解が必要です。これに対処するには、 が契約テストを実行するときに CloudFormation 使用する入力を指定できます。

CloudFormation には、契約テストを実行するときに使用する入力データを指定する 2 つの方法があります。

- ファイルを上書きします

  overrides ファイルを使用すると、 の特定のプロパティの入力データを軽量に指定 CloudFormation しpreCreate、 および preUpdatepreDeleteオペレーションテストで使用します。
- 入力ファイル

  以下の場合は、複数のinputファイルを使用して契約テスト入力データを指定することもできます。
  - 作成、更新、削除オペレーションに異なる入力データを指定するか、テストする無効なデータを指定する必要があります。
  - 複数の異なる入力データセットを指定する場合。

オーバーライドファイルを使用した入力データの指定

以下は、 overrides ファイルを使用した Amazon S3 フックの入力データの例です。

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

入力ファイルを使用した入力データの指定

input ファイルを使用して、 CloudFormation が使用するさまざまな種類の入力データを指定します。preCreate入力、preUpdate入力、無効な入力です。各種類のデータは個別の ファイルで指定されます。契約テスト用に複数の入力データのセットを指定することもできます。

契約テスト CloudFormation で使用する のinputファイルを指定するには、フックプロジェクトのルートディレクトリに inputsフォルダを追加します。次に、入力ファイルを追加します。

次の命名規則を使用して、ファイルに含まれる入力データの種類を指定します。*n* は整数です。

- inputs_*n*_pre_create.json: preCreateハンドラーでファイルを使用して、リソースを作成するための入力を指定します。

- inputs_*n*_pre_update.json: preUpdateハンドラーでファイルを使用して、リソースを更新するための入力を指定します。
- inputs_*n*_pre_delete.json: preDeleteハンドラーでファイルを使用して、リソースを削除するための入力を指定します。
- inputs_*n*_invalid.json: テストする無効な入力を指定する場合。

契約テスト用に複数の入力データのセットを指定するには、ファイル名の整数をインクリメントして入力データセットを順序付けます。例えば、最初の入力ファイルのセットには、inputs_1_pre_create.json、inputs_1_pre_update.json、および inputs_1_pre_invalid.json という名前を付ける必要があります。次のセットの名前は inputs_2_pre_create.json、inputs_2_pre_update.json、inputs_2_pre_invalid.jsonなどになります。

各入力ファイルは、テストに使用されるリソースプロパティのみを含む JSON ファイルです。

以下は、入力ファイルを使用して入力データ Amazon S3 を指定inputsするための ディレクトリの例です。

inputs_1_pre_create.json

inputs_1_pre_create.json 契約テストの例を次に示します。

```
{
    "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

inputs_1_pre_update.json 契約テストの例を次に示します。

```
{
    "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

inputs_1_invalid.json 契約テストの例を次に示します。

```
{
    "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

inputs_1_invalid_pre_update.json 契約テストの例を次に示します。

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

詳細については、 AWS CloudFormation CLI ユーザーガイドの Publishing extensions to make them available for public use を参照してください。

AWS CloudFormation フックのスキーマ構文リファレンス

このセクションでは、 AWS CloudFormation フックの開発に使用するスキーマの構文について説明します。

フックには、JSON スキーマとフックハンドラーで表されるフック仕様が含まれています。カスタムフックを作成する最初のステップは、フック、そのプロパティ、および属性を定義するスキーマをモデル化することです。CloudFormation CLI init コマンドを使用してカスタムフックプロジェクトを初期化すると、フックスキーマファイルが作成されます。このスキーマファイルを、カスタムフックのシェイプとセマンティクスを定義するための開始点として使用します。

スキーマ構文

次のスキーマはフックの構造です。

```
{
"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`

フックの一意の名前。フックの 3 つの部分からなる名前空間を指定し、推奨パターンは ですOrganization::Service::Hook。

> **Note**
>
> 次の組織名前空間は予約されており、フックタイプ名では使用できません。
>
> - Alexa
> - AMZN
> - Amazon
> - ASK
> - AWS
> - Custom
> - Dev

必須: はい

パターン: ^[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}::[a-zA-Z0-9]{2,64}$

最小: 10

最大: 196

`description`

CloudFormation コンソールに表示されるフックの簡単な説明。

必須: はい

`sourceUrl`

パブリックの場合、フックのソースコードの URL。

必須: いいえ

最大: 4096

`documentationUrl`

フックの詳細なドキュメントを提供するページの URL。

必須: はい

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

最大: 4096

> **Note**
> フックスキーマには完全で正確なプロパティの説明を含める必要がありますが、documentationURLプロパティを使用して、例、ユースケース、その他の詳細情報などの詳細をユーザーに提供できます。

`definitions`

definitions ブロックを使用して、共有フックプロパティスキーマを提供します。

definitions セクションを使用して、フックタイプスキーマの複数のポイントで使用できるスキーマ要素を定義することがベストプラクティスと見なされます。その後、JSON ポインタを使用して、フックタイプスキーマの適切な場所でその要素を参照できます。

必須: いいえ

`typeConfiguration`

フックの設定データの定義。

必須: はい

`properties`

フックのプロパティ。フックのすべてのプロパティは、スキーマで表現する必要があります。フックスキーマプロパティをフックタイプの設定プロパティに合わせます。

> **Note**
>
> ネストされたプロパティは許可されません。代わりに、 definitions要素でネストされたプロパティを定義し、$refポインタを使用して目的のプロパティでそれらを参照します。

現在、次のプロパティがサポートされています。

- default – プロパティのデフォルト値。
- description – プロパティの説明。
- pattern – 入力の検証に使用される正規表現パターン。
- type – プロパティの受け入れられたタイプ。

additionalProperties

additionalProperties を false に設定する必要があります。フックのすべてのプロパティはスキーマで表現する必要があります。任意の入力は許可されません。

必須: はい

有効な値: false

handlers

ハンドラーは、フック呼び出しポイントなど、スキーマで定義されたフックを開始できるオペレーションを指定します。たとえば、preUpdateハンドラー内のすべての指定されたターゲットの更新オペレーションの前にハンドラーが呼び出されます。

有効な値: preCreate | preUpdate | preDelete

> **Note**
>
> ハンドラーには少なくとも 1 つの値を指定する必要があります。

> **Important**
>
> ステータスが になるスタックオペレーションでは、フックは呼び出UpdateCleanupされません。たとえば、次の 2 つのシナリオでは、フックのpreDeleteハンドラーは呼び出されません。

- スタックは、テンプレートから 1 つのリソースを削除した後に更新されます。
- 更新タイプの置換のリソースが削除されます。

`targetNames`

Hook がターゲットとする型名の文字列配列。たとえば、`preCreate`ハンドラーに `AWS::S3::Bucket`ターゲットがある場合、フックは事前プロビジョニングフェーズ中に Amazon S3 バケットに対して実行されます。

- `TargetName`

  実装されたハンドラーごとに少なくとも 1 つのターゲット名を指定します。

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

  最小: `1`

  必須: はい

  > **Warning**
  >
  > SSM SecureString と Secrets Manager の動的参照は、フックに渡される前に解決されません。

`permissions`

ハンドラーを呼び出すために必要な AWS アクセス許可を指定する文字列配列。

必須: はい

`additionalProperties`

`additionalProperties` を `false` に設定する必要があります。フックのすべてのプロパティはスキーマで表現する必要があります。任意の入力は許可されません。

必須: はい

有効な値: `false`

フックスキーマの例

例 1

Java と Python のチュートリアルでは、次のコード例を使用します。以下は、 というフックの構造例です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",
                "pattern": "[a-zA-Z]*[1-9]"
            }
        },
        "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
}
```

例 2

次の例は、 STACKと CHANGE_SET の を使用してスタックテンプレートと変更セットオペレーションtargetNamesをターゲットにするスキーマです。

```
{
    "typeName":"MyCompany::Testing::MyTestHook",
    "description":"Verifies Stack and Change Set properties before create and update",
    "sourceUrl":"https://mycorp.com/my-repo.git",
    "documentationUrl":"https://mycorp.com/documentation",
    "typeConfiguration":{
        "properties":{
            "minBuckets":{
                "description":"Minimum number of compliant buckets",
```

```
                "type":"string"
            },
            "minQueues":{
                "description":"Minimum number of compliant queues",
                "type":"string"
            },
            "encryptionAlgorithm":{
                "description":"Encryption algorithm for SSE",
                "default":"AES256",
                "type":"string",
                "pattern": "[a-zA-Z]*[1-9]"
            }
        },
        "required":[
        ],
        "additionalProperties":false
    },
    "handlers":{
        "preCreate":{
            "targetNames":[
                "STACK",
                "CHANGE_SET"
            ],
            "permissions":[
            ]
        },
        "preUpdate":{
            "targetNames":[
                "STACK"
            ],
            "permissions":[
            ]
        },
        "preDelete":{
            "targetNames":[
                "STACK"
            ],
            "permissions":[

            ]
        }
    },
    "additionalProperties":false
```

```
}
```

AWS CloudFormation フックの無効化と有効化

このトピックでは、フックを無効にしてから再度有効にして、アカウントで一時的にアクティブにならないようにする方法について説明します。フックを無効にすると、フックの干渉なしに問題を調査する必要がある場合に便利です。

アカウントでフックを無効化および有効化する (コンソール)

アカウントのフックを無効にするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。
2. 画面上部のナビゲーションバーで、フック AWS リージョン がある を選択します。
3. ナビゲーションペインからフックを選択します。
4. 無効にするフックの名前を選択します。
5. フックの詳細ページのフックの名前の右側で、無効化ボタンを選択します。
6. 確認を求められたら、フックを無効にするを選択します。

以前に無効にしたフックを再度有効にするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。
2. 画面上部のナビゲーションバーで、フック AWS リージョン がある を選択します。
3. ナビゲーションペインからフックを選択します。
4. 有効にするフックの名前を選択します。
5. フックの詳細ページのフックの名前の右側で、有効化ボタンを選択します。
6. 確認を求められたら、フックを有効にするを選択します。

アカウントでフックを無効化および有効化する (AWS CLI)

> **Important**
>
> フックを無効化および有効化するための AWS CLI コマンドは、フック設定全体を `--configuration`オプションで指定された値に置き換えます。意図しない変更を回避するに

は、これらのコマンドを実行するときに保持する既存の設定をすべて含める必要があります。現在の設定データを表示するには、 describe-type コマンドを使用します。

フックを無効にするには

フックを無効にするDISABLEDには、次のset-type-configurationコマンドを使用して HookInvocationStatusとして を指定します。プレースホルダーを特定の値に置き換えます。

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

以前に無効にしたフックを再度有効にするには

フックを再度有効にENABLEDするには、次のset-type-configurationコマンドを使用して を HookInvocationStatusとして指定します。プレースホルダーを特定の値に置き換えます。

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

詳細については、「フック設定スキーマ構文リファレンス」を参照してください。

フック設定スキーマ構文リファレンス

このセクションでは、フックの設定に使用されるスキーマ構文の概要を説明します。CloudFormation は、 でフックを呼び出すときに、実行時にこの設定スキーマを使用します AWS アカウント。

フックがスタックの設定をプロアクティブに検査できるようにするには、フックがアカウントに登録されてアクティブ化されたHookInvocationStatusENABLED後、 を に設定します。

トピック

> **Note**
>
> フックの設定で保存できるデータの最大量は 300 KB です。これは、SetTypeConfigurationオペレーションのConfigurationリクエストパラメータに適用されるすべての制約に追加されます。

フック設定スキーマのプロパティ

次のスキーマは、フック設定スキーマの構造です。

```
{
    "CloudFormationConfiguration": {
        "HookConfiguration": {
            "HookInvocationStatus": "ENABLED",
            "TargetOperations": ["STACK"],
            "FailureMode": "FAIL",
            "Properties": {
                ...
            }
        }
```

```
    }
}
```

HookConfiguration

フック設定では、スタックレベルでのフックの有効化または無効化、障害モード、フックプロパティ値をサポートしています。

フック設定では、次のプロパティがサポートされています。

HookInvocationStatus

フックが ENABLEDか かを指定しますDISABLED。

有効な値: ENABLED | DISABLED

TargetOperations

フックが実行されるオペレーションのリストを指定します。詳細については、「フックターゲット」を参照してください。

有効な値: STACK | RESOURCE | CHANGE_SET | CLOUD_CONTROL

TargetStacks

下位互換性のために使用できます。*HookInvocationStatus*代わりに を使用します。

モードが に設定されている場合ALL、フックは CREATE、、UPDATEまたは DELETEリソースオペレーション中にアカウント内のすべてのスタックに適用されます。

モードが に設定されている場合NONE、フックはアカウントのスタックには適用されません。

有効な値: ALL | NONE

FailureMode

このフィールドは、フック障害の処理方法をサービスに指示します。

- モードが に設定されていてFAIL、フックが失敗した場合、失敗設定はリソースのプロビジョニングを停止し、スタックをロールバックします。
- モードが に設定WARNされていて、フックが失敗した場合、警告設定によりプロビジョニングを警告メッセージで続行できます。

有効な値: FAIL | WARN

`Properties`

フックランタイムプロパティを指定します。これらは、フックスキーマでサポートされているプロパティの形状と一致する必要があります。

フック設定の例

からフックを設定する例については AWS CLI、以下のセクションを参照してください。

- ガードフックをアクティブ化する (AWS CLI)
- Lambda フックをアクティブ化する (AWS CLI)

AWS CloudFormation スタックレベルのフィルターをフックします

CloudFormation フックにスタックレベルのフィルターを追加して、スタック名とロールに基づいて特定のスタックをターゲットにできます。これは、同じリソースタイプを持つ複数のスタックがあるが、フックが特定のスタックを対象としている場合に便利です。

このセクションでは、これらのフィルターの仕組みと、従うことができる例について説明します。

スタックレベルのフィルタリングを使用しないフック設定の基本構造は次のようになります。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "TargetFilters": {
        "Actions": [
          "CREATE",
          "UPDATE",
          "DELETE"
        ]
      }
    }
  }
```

```
}
```

HookConfiguration 構文の詳細については、「」を参照してくださいフック設定スキーマ構文リファレンス。

スタックレベルのフィルターを使用するには、 の下にStackFiltersキーを追加しますHookConfiguration。

StackFilters キーには 1 つの必須メンバーと 2 つのオプションメンバーがあります。

- FilteringCriteria (必須)
- StackNames (オプション)
- StackRoles (オプション)

StackNames または StackRolesプロパティはオプションです。ただし、これらのプロパティのうち、少なくとも 1 つを指定する必要があります。

Cloud Control API オペレーションをターゲットとするフックを作成すると、すべてのスタックレベルのフィルターは無視されます。

FilteringCriteria

FilteringCriteria は、フィルタリング動作を指定する必須パラメータです。ALL または のいずれかに設定できますANY。

- ALL すべてのフィルターが一致すると、 はフックを呼び出します。
- ANY いずれかのフィルターが一致すると、 はフックを呼び出します。

StackNames

フック設定で 1 つ以上のスタック名をフィルターとして指定するには、次の JSON 構造を使用します。

```
"StackNames": {
  "Include": [
    "string"
  ],
  "Exclude": [
    "string"
```

```
  ]
}
```

いずれかを指定する必要があります。

- Include: 含めるスタック名のリスト。このリストで指定されたスタックのみがフックを呼び出します。
  - 型: 文字列の配列
  - 最大項目数: 50
  - 最小項目: 1
- Exclude: 除外するスタック名のリスト。ここにリストされているスタックを除くすべてのスタックは、 フックを呼び出します。
  - 型: 文字列の配列
  - 最大項目数: 50
  - 最小項目: 1

Include および Exclude配列の各スタック名は、次のパターンと長さの要件に従う必要があります。

- パターン: ^[a-zA-Z][-a-zA-Z0-9]*$
- 最大長: 128

StackNames は、具体的なスタック名と完全なワイルドカードマッチングをサポートします。ワイルドカードを使用する例については、「」を参照してくださいフックターゲット名でのワイルドカードの使用。

StackRoles

フック設定で 1 つ以上の IAM ロールをフィルターとして指定するには、次の JSON 構造を使用します。

```
"StackRoles": {
  "Include": [
    "string"
  ],
  "Exclude": [
    "string"
```

```
  ]
}
```

いずれかを指定する必要があります。

- Include: これらのロールに関連付けられたターゲットスタックの IAM ロール ARNs のリスト。これらのロールによって開始されたスタックオペレーションのみがフックを呼び出します。
  - 型: 文字列の配列
  - 最大項目数: 50
  - 最小項目: 1
- Exclude: 除外するスタックの IAM ロール ARNs のリスト。フックは、指定されたロールによって開始されたスタックを除くすべてのスタックで呼び出されます。
  - 型: 文字列の配列
  - 最大項目数: 50
  - 最小項目: 1

Include および Exclude配列の各スタックロールは、次のパターンと長さの要件に従う必要があります。

- パターン: arn:.+:iam::[0-9]{12}:role/.+
- 最大長: 256

StackRoles では、次の ARN 構文セクションでワイルドカード文字を使用できます。

- partition
- account-id
- resource-id

ARN 構文セクションでワイルドカードを使用する例については、「」を参照してくださいフックターゲット名でのワイルドカードの使用。

Include および Exclude

各フィルター (StackNames と StackRoles) にはIncludeリストとExcludeリストがあります。たとえばStackNames、 を使用すると、フックはIncludeリストで指定されたスタックでのみ呼び

出されます。スタック名がExcludeリストでのみ指定されている場合、フックはExcludeリストに含まれていないスタックでのみ呼び出されます。Include と の両方が指定されている場合、フックExcludeはIncludeリスト内のものをターゲットにし、Excludeリスト内のものをターゲットにしません。

たとえば、A、B、C、D の 4 つのスタックがあるとします。

- "Include": ["A","B"] フックは A と B で呼び出されます。
- "Exclude": ["B"] フックは A、C、D で呼び出されます。
- "Include": ["A","B","C"], "Exclude": ["A","D"] フックは B と C で呼び出されます。
- "Include": ["A","B","C"], "Exclude": ["A”,"B","C"] フックはどのスタックでも呼び出されません。

スタックレベルフィルターの例

このセクションでは、 AWS CloudFormation フックのスタックレベルフィルターを作成するために使用できる例を示します。

例 1: 特定のスタックを含める

次の例では、 Includeリストを指定します。フックは、stack-test-1、、stack-test-2およびという名前のスタックでのみ呼び出されますstack-test-3。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
```

```
            "stack-test-3"
          ]
        }
      }
    }
  }
}
```

例 2: 特定のスタックを除外する

代わりにスタック名がExcludeリストに追加されると、フックは stack-test-1、、stack-test-2または という名前のスタックで呼び出されますstack-test-3。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Exclude": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        }
      }
    }
  }
}
```

例 3: include と exclude の組み合わせ

Include および Excludeリストが指定されていない場合、フックはExcludeリストに含まれIncludeていない のスタックでのみ呼び出されます。次の例では、フックは でのみ呼び出されますstack-test-3。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ],
          "Exclude": [
            "stack-test-1",
            "stack-test-2"
          ]
        }
      }
    }
  }
}
```

例 4: スタック名とロールを**ALL**条件と組み合わせる

次のフックには、3 つのスタック名と 1 つのスタックロールが含まれています。FilteringCriteria は として指定されているためALL、フックは、一致するスタック名と一致するスタックロールの両方を持つスタックに対してのみ呼び出されます。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
```

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

例 5: スタック名とロールを**ANY**条件と組み合わせる

次のフックには、3 つのスタック名と 1 つのスタックロールが含まれています。FilteringCriteria は として指定されているためANY、フックは、一致するスタック名または一致するスタックロールを持つスタックに対して呼び出されます。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ANY",
        "StackNames": {
          "Include": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
```

```
        },
        "StackRoles": {
            "Include": ["arn:aws:iam::123456789012:role/hook-role"]
        }
      }
    }
  }
}
```

AWS CloudFormation フックターゲットフィルター

このトピックでは、 AWS CloudFormation フックのターゲットフィルターの設定に関するガイダンスを提供します。ターゲットフィルターを使用して、フックが呼び出されるタイミングとリソースをより詳細に制御できます。シンプルなリソースタイプのターゲティングから、リソースタイプ、アクション、呼び出しポイントのより複雑な組み合わせまで、フィルターを設定できます。

フック設定で 1 つ以上のスタック名をフィルターとして指定するには、 の下にTargetFiltersキーを追加しますHookConfiguration。

TargetFilters では、次のプロパティがサポートされています。

Actions

ターゲットにするアクションを指定する文字列配列。例については、例 1: 基本的なターゲットフィルターを参照してください。

有効な値: CREATE | UPDATE | DELETE

> **Note**
>
> RESOURCE、STACK、および CLOUD_CONTROLターゲットの場合、すべてのターゲットアクションが適用されます。CHANGE_SET ターゲットの場合、 CREATEアクションのみが適用されます。詳細については、「フックターゲット」を参照してください。

InvocationPoints

ターゲットとする呼び出しポイントを指定する文字列配列。

有効な値: PRE_PROVISION

`TargetNames`

ターゲットとするリソースタイプ名を指定する文字列配列。例: `AWS::S3::Bucket`。

ターゲット名は、具体的なターゲット名と完全なワイルドカードマッチングをサポートします。詳細については、「フックターゲット名でのワイルドカードの使用」を参照してください。

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

最大: `50`

`Targets`

ターゲットフィルタリングに使用するターゲットのリストを指定するオブジェクト配列。

ターゲット配列の各ターゲットには、次のプロパティがあります。

`Actions`

指定されたターゲットのアクション。

有効な値: `CREATE | UPDATE | DELETE`

`InvocationPoints`

指定されたターゲットの呼び出しポイント。

有効な値: `PRE_PROVISION`

`TargetNames`

ターゲットにするリソースタイプ名。

> **Note**
>
> `Targets` オブジェクト配列と `TargetNames`、、`Actions`または `InvocationPoints`配列の両方を同時に含めることはできません。これらの 3 つの項目と を使用する場合は`Targets`、それらを`Targets`オブジェクト配列に含める必要があります。例については、例 2: `Targets` オブジェクト配列の使用を参照してください。

ターゲットフィルターの例

このセクションでは、 AWS CloudFormation フックのターゲットフィルターを作成するための例を示します。

例 1: 基本的なターゲットフィルター

特定のリソースタイプに焦点を当てた基本的なターゲットフィルターを作成するには、 Actions配列で TargetFilters オブジェクトを使用します。次のターゲットフィルター設定はCreate、指定されたターゲットオペレーション (この場合は および STACKオペレーションの両方) のすべての Update、、RESOURCEおよび Deleteアクションでフックを呼び出します。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "TargetFilters": {
        "Actions": [
           "Create",
           "Update",
           "Delete"
        ]
      }
    }
  }
}
```

例 2: **Targets** オブジェクト配列の使用

より高度なフィルターについては、 Targets オブジェクト配列を使用して、特定のターゲット、アクション、呼び出しポイントの組み合わせを一覧表示できます。次のターゲットフィルター設定は、S3 バケットCREATEと DynamoDB テーブルの および UPDATEアクションの前にフックを呼び出します。これは、 オペレーションSTACKと RESOURCEオペレーションの両方に適用されます。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
```

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

フックターゲット名でのワイルドカードの使用

ターゲット名の一部としてワイルドカードを使用できます。フックターゲット名にはワイルドカード文字 (* と ?) を使用できます。アスタリスク (*) は、任意の文字の組み合わせを表します。疑問符 (?) は任意の 1 文字を表します。ターゲット名には複数の *および ?文字を使用できます。

Example : フックスキーマのターゲット名のワイルドカードの例

次の例は、Amazon S3 でサポートされているすべてのリソースタイプを対象としています。

```
{
```

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

次の例では、名前Bucketに「」が含まれているすべてのリソースタイプを照合します。

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

は、次のいずれかの具体的なリソースタイプに解決AWS::*::Bucket*される場合があります。

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

Example : フック設定スキーマのターゲット名のワイルドカードの例

次の設定例では、すべての Amazon S3 リソースタイプに対する CREATE オペレーションと、AWS::DynamobDB::Tableや など、すべての名前付きテーブルリソースタイプに対する UPDATEオペレーションに対してフックを呼び出します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"
                    }
                ]
            }
        }
   }
}
```

次の設定例では、すべての Amazon S3 リソースタイプで CREATEおよび UPDATEオペレーションのフックを呼び出します。また、 AWS::DynamobDB::Tableや など、すべての名前付きテーブルリソースタイプで CREATEおよび UPDATEオペレーションのフックを呼び出しますAWS::Glue::Table。

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

```
                    "UPDATE"
                ],
                "InvocationPoints": [
                    "PRE_PROVISION"
                ]
            }
        }
    }
}
```

Example : **Include**特定のスタック

次の例では、 Includeリストを指定します。フックは、スタック名が で始まる場合にのみ呼び出されますstack-test-。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-*"
          ]
        }
      }
    }
  }
}
```

Example : **Exclude**特定のスタック

次の例では、 Excludeリストを指定します。フックは、 で始まらないスタックで呼び出されますstack-test-。

```
{
```

```
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Exclude": [
            "stack-test-*"
          ]
        }
      }
    }
  }
}
```

Example : 特定のスタック**Exclude**の **Include**と の組み合わせ

Include および Excludeリストが指定されている場合、フックはExcludeリスト内で一致しない内のInclude一致するスタックでのみ呼び出されます。次の例では、、、および という名前のスタックstack-test-を除きstack-test-1stack-test-2、 で始まるすべてのスタックでフックが呼び出されますstack-test-3。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-*"
```

```
          ],
          "Exclude": [
            "stack-test-1",
            "stack-test-2",
            "stack-test-3"
          ]
        }
      }
    }
  }
}
```

Example : **Include**特定のロール

次の例では、2 つのワイルドカードパターンを持つ Includeリストを指定します。最初のエントリは、 partitionおよび hook-roleで で始まるロールに対してフックを実行しますaccount-id。2 番目のエントリは、 partitionに属する の任意のロールに対して account-id を実行します123456789012。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackRoles": {
          "Include": [
            "arn:*:iam::*:role/hook-role*",
            "arn:*:iam::123456789012:role/*
          ]
        }
      }
    }
  }
}
```

Example : **Exclude**特定のロール

次の例では、2 つのワイルドカードパターンを持つ Excludeリストを指定します。ロールの名前exemptに と partition がある場合、最初のエントリはフック実行をスキップしますaccount-id。2 番目のエントリは、 に属するロールaccount-id123456789012がスタックオペレーションで使用される場合、フックの実行をスキップします。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackRoles": {
          "Exclude": [
            "arn:*:iam::*:role/*exempt*",
            "arn:*:iam::123456789012:role/*
          ]
        }
      }
    }
  }
}
```

Example : 特定のロール ARN パターン**Exclude**に対して **Include**と を組み合わせる

Include および Excludeリストが指定されている場合、フックは、Excludeリスト内で一致しない Includeのロールと一致するロールで使用されるスタックでのみ呼び出されます。次の例では、ロールが に属している場合を除きpartition、、account-id、および roleの名前を持つスタックオペレーションでフックaccount-idが呼び出されます123456789012。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
```

```
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackRoles": {
          "Include": [
            "arn:*:iam::*:role/*"
          ],
          "Exclude": [
            "arn:*:iam::123456789012:role/*"
          ]
        }
      }
    }
  }
}
```

Example : スタック名とロールをすべての条件と組み合わせる

次のフックには、1 つのスタック名ワイルドカードと 1 つのスタックロールワイルドカードが含まれています。FilteringCriteria は として指定されているためALL、フックは、一致するStackNameと一致する の両方を持つスタックに対してのみ呼び出されますStackRoles。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ALL",
        "StackNames": {
          "Include": [
            "stack-test-*"
          ]
        },
```

```
        "StackRoles": {
          "Include": ["arn:*:iam::*:role/hook-role*"]
        }
      }
    }
  }
}
```

Example : **StackNames**と を任意の条件**StackRoles**と組み合わせる

次のフックには、1 つのスタック名ワイルドカードと 1 つのスタックロールワイルドカードが含まれています。FilteringCriteria は として指定されているためANY、 フックは、一致するStackNamesまたは一致する を持つスタックに対して呼び出されますStackRoles。

```
{
  "CloudFormationConfiguration": {
    "HookConfiguration": {
      "HookInvocationStatus": "ENABLED",
      "TargetOperations": [
        "STACK",
        "RESOURCE"
      ],
      "FailureMode": "WARN",
      "Properties": {},
      "StackFilters": {
        "FilteringCriteria": "ANY",
        "StackNames": {
          "Include": [
            "stack-test-*"
          ]
        },
        "StackRoles": {
            "Include": ["arn:*:iam::*:role/hook-role*"]
        }
      }
    }
  }
}
```

CloudFormation テンプレートを使用してフックを作成する

このページでは、Hooks のサンプル CloudFormation テンプレートと技術リファレンストピックへのリンクを提供します。

CloudFormation テンプレートを使用してフックを作成することで、テンプレートを再利用してフックを一貫して繰り返しセットアップできます。このアプローチにより、フックを 1 回定義し、複数の AWS アカウント およびリージョンで同じフックを何度もプロビジョニングできます。

CloudFormation には、Guard および Lambda フックの作成用に以下の特殊なリソースタイプが用意されています。

タスク	ソリューション	リンク
ガードフックを作成する	`AWS::CloudFormation::GuardHook` リソースタイプを使用して、ガードフックを作成してアクティブ化します。	サンプルテンプレート テクニカルリファレンス
Lambda フックを作成する	`AWS::CloudFormation::LambdaHook` リソースタイプを使用して、Lambda フックを作成してアクティブ化します。	サンプルテンプレート テクニカルリファレンス

CloudFormation には、カスタムフック作成用のスタックテンプレートで使用できる以下のリソースタイプもあります。

タスク	ソリューション	リンク
フックを登録する	`AWS::CloudFormation::HookVersion` リソースタイプを使用して、カスタムフックの新しいバージョンまたは最初のバージョンを CloudFormation レジストリに発行します。	サンプルテンプレート テクニカルリファレンス

タスク	ソリューション	リンク
フックの設定	AWS::CloudFormation::HookTypeConfig リソースタイプを使用して、カスタムフックの設定を指定します。	サンプルテンプレート テクニカルリファレンス
フックのデフォルトバージョンを設定する	AWS::CloudFormation::HookDefaultVersion リソースタイプを使用して、カスタムフックのデフォルトバージョンを指定します。	サンプルテンプレート テクニカルリファレンス
アカウントをパブリッシャーとして登録する	AWS::CloudFormation::Publisher リソースタイプを使用して、アカウントを CloudFormation レジストリのパブリック拡張機能 (フック、モジュール、リソースタイプ) のパブリッシャーとして登録します。	テクニカルリファレンス
フックを公開する	AWS::CloudFormation::PublicTypeVersion リソースタイプを使用して、登録されたカスタムフックをパブリックなサードパーティーフックとしてテストして公開します。	テクニカルリファレンス
パブリックサードパーティーフックをアクティブ化する	AWS::CloudFormation::TypeActivation リソースタイプは、 AWS::CloudFormation::HookTypeConfig リソースタイプと連携して、 アカウントのパブリックでサードパーティーのカスタムフックをアクティブ化します。	テクニカルリファレンス

AWS CloudFormation Hooks ユーザーガイドのドキュメント履歴

次の表は、 AWS CloudFormation フックの前回のリリース以降のドキュメントの重要な変更点を示しています。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

- ドキュメントの最終更新日： 2023 年 12 月 8 日。

変更	説明	日付
スタックレベルのフック	フックがスタックレベルでサポートされるようになりました。これにより、お客様は CloudFormation フックを使用して新しいテンプレートを評価し、スタックオペレーションの進行をブロックする可能性があります。	2024 年 11 月 13 日
AWS クラウドコントロール API フック統合	フックが Cloud Control API と統合され、CloudFormation フックを使用してプロビジョニング前にリソースの設定を事前に検査できるようになりました。非準拠のリソースが見つかった場合、フックは オペレーションを失敗させてリソースのプロビジョニングを禁止するか、警告を発してプロビジョニングオペレーションを続行します。	2024 年 11 月 13 日
AWS CloudFormation Guard フック	AWS CloudFormation Guard は、policy-as-codeの作成に使用できるオープンソースお	2024 年 11 月 13 日

	よび汎用のドメイン固有の言語 (DSL) です。Guard Hooks は Cloud Control API および CloudFormation オペレーションを評価して、プロビジョニング前にリソースの設定を検査できます。非準拠のリソースが見つかった場合、フックは オペレーションを失敗させてリソースのプロビジョニングを禁止するか、警告を発してプロビジョニングオペレーションを続行します。	
AWS Lambda フック	AWS CloudFormation Lambda フックを使用すると、独自のカスタムコードに対して CloudFormation および Cloud Control API オペレーションを評価できます。フックは、操作の進行をブロックしたり、発信者に警告を発行して操作の進行を許可したりできます。	2024 年 11 月 13 日
Hooks ユーザーガイド	AWS CloudFormation Hooks ユーザーガイドの初版。更新には、新しい概要、入門チュートリアル、概念と用語、スタックレベルのフィルタリング、前提条件、セットアップ、CloudFormation フック開発に関するトピックの更新が含まれます。	2023 年 12 月 8 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。