aws

開発者ガイド

AWS Serverless Application Model

AWS Serverless Application Model: 開発者ガイド

Table of Contents

AWS Serverless Application Model (AWS SAM) とは

AWS Serverless Application Model (AWS SAM) は、Infrastructure as Code (IaC) を使用した、サーバーレスアプリケーション構築のためのオープンソースのフレームワークです。AWS SAM の省略構文を使用して、デベロッパーは、デプロイ中にインフラストラクチャに変換される AWS CloudFormation リソースおよび特殊なサーバーレスリソースを宣言します。このフレームワークには、AWS SAM CLI および AWS SAM プロジェクトの 2 つの主要なコンポーネントが含まれています。AWS SAM プロジェクトは、sam init の実行時に作成されるアプリケーションプロジェクトディレクトリです。AWS SAM プロジェクトには、テンプレート仕様 (リソースの宣言に使用する省略構文) を含む AWS SAM テンプレートなどのファイルが含まれます。

主な特徴

AWS SAM により、以下を可能にすることでデベロッパーエクスペリエンスを向上させるさまざまな利点が提供されます。

より少ないコードを使用して、アプリケーションインフラストラクチャコードを迅速に定義する

AWS SAM テンプレートを作成してサーバーレスアプリケーションインフラストラクチャコードを定義します。テンプレートを直接 AWS CloudFormation にデプロイして、リソースをプロビジョニングします。

開発ライフサイクル全体を通じてサーバーレスアプリケーションを管理する

AWS SAM CLI を使用して、開発ライフサイクルの作成、構築、デプロイ、テスト、モニタリングの各フェーズを通じてサーバーレスアプリケーションを管理します。詳細については、「AWS SAMCLI」を参照してください。

AWS SAM コネクタを使用してリソース間の許可を迅速にプロビジョニングする

AWS SAM テンプレートで AWS SAM コネクタを使用して、AWS リソース間の許可を定義します。AWS SAM は、コードを変換して、ユーザーの意図を円滑に実現するために必要な IAM 許可にします。詳細については、「AWS SAM コネクタによるリソースに対するアクセス許可の管理」を参照してください。

開発中にローカルの変更をクラウドに継続的に同期する

AWS SAM CLI sam sync コマンドを使用すると、ローカルの変更がクラウドに自動的に同期され、開発およびクラウドテストのワークフローが高速化されます。詳細については、「の使用の概要 sam sync 同期する AWS クラウド」を参照してください。

Terraform サーバーレスアプリケーションを管理する

AWS SAM CLI を使用して、ローカルで Lambda 関数とレイヤーのデバッグやテストを実行します。詳細については、「AWS SAM CLI Terraform のサポート」を参照してください。

関連情報

- AWS SAM の仕組みについては、「AWS SAM の働き」を参照してください。
- AWS SAM の使用を開始するには、「AWS SAM の開始方法」を参照してください。
- AWS SAM を使用してサーバーレスアプリケーションを作成する方法の概要については、「AWS SAM を使用する方法」を参照してください。

AWS SAM の働き

AWS SAM は、サーバーレスアプリケーションの作成に使用する 2 つのプライマリコンポーネントで構成されます。

1. AWS SAM プロジェクト — sam init コマンドの実行時に作成されるフォルダとファイルです。このディレクトリには、AWS リソースを定義する重要なファイルである AWS SAM テンプレートが含まれています。このテンプレートには、AWS SAM テンプレート仕様が含まれています。このオープンソースフレームワークには、サーバーレスアプリケーションの関数、イベント、API、設定、アクセス許可を定義するために使用する簡略化された省略構文があります。
2. AWS SAMCLI — AWS SAM プロジェクトやサポートされているサードパーティーの統合と併用することで、サーバーレスアプリケーションを構築し、実行できるコマンドラインツールです。AWS SAM CLI は、AWS SAM プロジェクトでコマンドを実行し、最終的にサーバーレスアプリケーションに変換するために使用するツールです。

サーバーレスアプリケーションを定義するリソース、イベントソースマッピング、およびその他のプロパティを表すには、AWS SAM テンプレートや AWS SAM プロジェクト内のその他のファイルでリソースを定義し、アプリケーションを開発します。AWS SAM CLI を使用して AWS SAM プロジェクトでコマンドを実行します。これが、サーバーレスアプリケーションを初期化、構築、テスト、デプロイする方法です。

サーバーレスは初めてですか?

「AWS Serverless Application Model 向けサーバーレスの概念」を確認することをお勧めします。

AWS SAM テンプレート仕様とは

AWS SAM テンプレート仕様とは、サーバーレスアプリケーションインフラストラクチャコードを定義および管理するために使用できるオープンソースフレームワークです。AWS SAM テンプレート仕様は次のとおりです。

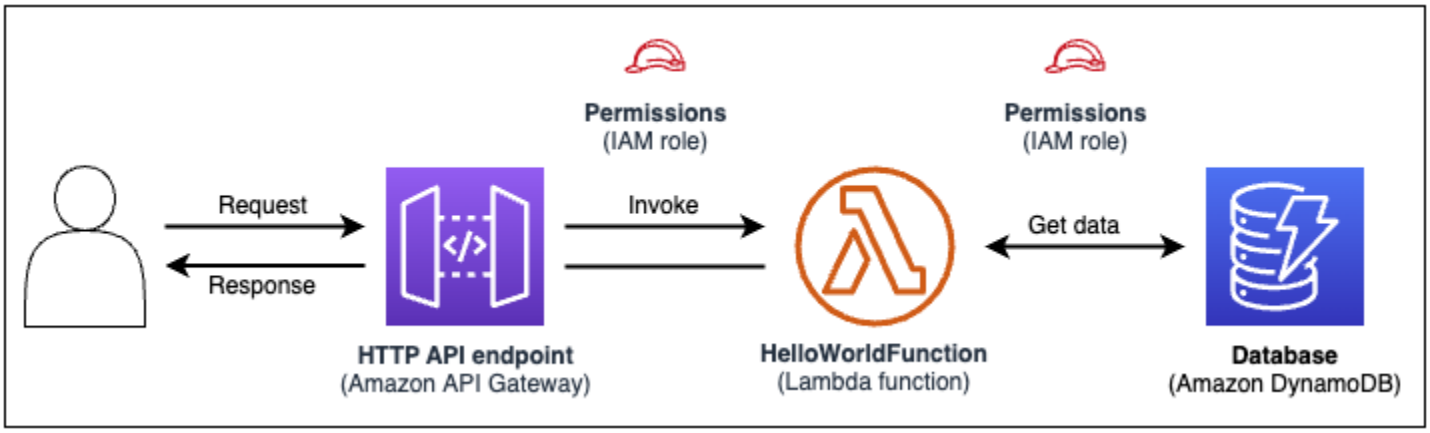
- AWS CloudFormation での構築 — リソースおよびプロパティ設定の広範なサポートを利用して、AWS SAM テンプレート内で AWS CloudFormation 構文を直接使用します。既に AWS CloudFormation に精通している場合は、アプリケーションインフラストラクチャコードを管理するために新しいサービスを学ぶ必要はありません。
- AWS CloudFormation の拡張機能 – AWS SAM は、特にサーバーレス開発の高速化に焦点を当てた独自の独自の構文を提供します。同じテンプレート内で AWS CloudFormation および AWS SAM の両方の構文を使用できます。
- 抽象的で簡潔な構文 – AWS SAM 構文を使用すると、より少ないコード行で、エラーの可能性をより低く抑えながら、インフラストラクチャを迅速に定義できます。その構文は、サーバーレスアプリケーションインフラストラクチャを定義する際の複雑さを抽象化して取り除くために特に精選されています。
- 変換 – AWS SAM は、テンプレートを変換して、AWS CloudFormation を通じてインフラストラクチャをプロビジョニングするために必要なコードにするという複雑な作業を実行します。

AWS SAM プロジェクトおよび AWS SAM テンプレートとは

AWS SAM プロジェクトには、AWS SAM テンプレート仕様を含む AWS SAM テンプレートが含まれます。この仕様は、サーバーレスアプリケーションインフラストラクチャを AWS で定義するために使用するオープンソースフレームワークで、操作を容易にする追加のコンポーネントがあります。その意味で、AWS SAM テンプレートは AWS CloudFormation テンプレートの拡張です。

基本的なサーバーレスアプリケーションの例を次に示します。このアプリケーションは、HTTP リクエストを通じてデータベースからすべての項目を取得するリクエストを処理します。これは次の部分で構成されます。

1. リクエストを処理するロジックを含む関数。
2. クライアント (リクエスタ) とアプリケーション間の通信として機能する HTTP API。
3. 項目を保存するデータベース。
4. アプリケーションを安全に実行するための許可。



このアプリケーションのインフラストラクチャコードは、次の AWS SAM テンプレートで定義できます。

```
AWSTemplateFormatVersion: 2010-09-09
Transform: AWS::Serverless-2016-10-31
Resources:
  getAllItemsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: src/get-all-items.getAllItemsHandler
      Runtime: nodejs20.x
      Events:
        Api:
          Type: HttpApi
          Properties:
            Path: /
            Method: GET
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: SampleTable
          Permissions:
            - Read
```

```
  SampleTable:
    Type: AWS::Serverless::SimpleTable
```

23 行のコードで、次のインフラストラクチャが定義されます。

- AWS Lambda サービスを利用した関数。
- Amazon API Gateway サービスを使用した HTTP API。
- Amazon DynamoDB サービスを使用するデータベース。
- これらのサービスが相互にインタラクションするために必要な AWS Identity and Access Management (IAM) 許可。

このインフラストラクチャをプロビジョニングするには、テンプレートを AWS CloudFormation にデプロイします。デプロイ中に、AWS SAM は 23 行のコードを変換して、AWS でこれらのリソースを生成するために必要な AWS CloudFormation 構文にします。変換された AWS CloudFormation テンプレートには 200 行を超えるコードが含まれています。

変換された AWS CloudFormation テンプレート

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Resources": {
    "getAllItemsFunction": {
      "Type": "AWS::Lambda::Function",
      "Metadata": {
        "SamResourceId": "getAllItemsFunction"
      },
      "Properties": {
        "Code": {
          "S3Bucket": "aws-sam-cli-managed-default-samclisourcebucket-1a4x26zbcdkqr",
          "S3Key": "what-is-app/a6f856abf1b2c4f7488c09b367540b5b"
        },
        "Handler": "src/get-all-items.getAllItemsHandler",
        "Role": {
          "Fn::GetAtt": [
            "getAllItemsFunctionRole",
            "Arn"
          ]
        },
        "Runtime": "nodejs12.x",
        "Tags": [
          {
```

```
            "Key": "lambda:createdBy",
            "Value": "SAM"
          }
        ]
      }
    },
    "getAllItemsFunctionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Action": [
                "sts:AssumeRole"
              ],
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "lambda.amazonaws.com"
                ]
              }
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
        ],
        "Tags": [
          {
            "Key": "lambda:createdBy",
            "Value": "SAM"
          }
        ]
      }
    },
    "getAllItemsFunctionApiPermission": {
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:InvokeFunction",
        "FunctionName": {
          "Ref": "getAllItemsFunction"
        },
        "Principal": "apigateway.amazonaws.com",
```

```
        "SourceArn": {
          "Fn::Sub": [
            "arn:${AWS::Partition}:execute-api:${AWS::Region}:${AWS::AccountId}:
${__ApiId__}/${__Stage__}/GET/",
            {
              "__ApiId__": {
                "Ref": "ServerlessHttpApi"
              },
              "__Stage__": "*"
            }
          ]
        }
      }
    },
    "ServerlessHttpApi": {
      "Type": "AWS::ApiGatewayV2::Api",
      "Properties": {
        "Body": {
          "info": {
            "version": "1.0",
            "title": {
              "Ref": "AWS::StackName"
            }
          },
          "paths": {
            "/": {
              "get": {
                "x-amazon-apigateway-integration": {
                  "httpMethod": "POST",
                  "type": "aws_proxy",
                  "uri": {
                    "Fn::Sub": "arn:${AWS::Partition}:apigateway:
${AWS::Region}:lambda:path/2015-03-31/functions/${getAllItemsFunction.Arn}/invocations"
                  },
                  "payloadFormatVersion": "2.0"
                },
                "responses": {}
              }
            }
          },
          "openapi": "3.0.1",
          "tags": [
            {
              "name": "httpapi:createdBy",
```

```
            "x-amazon-apigateway-tag-value": "SAM"
          }
        ]
      }
    }
  },
  "ServerlessHttpApiApiGatewayDefaultStage": {
    "Type": "AWS::ApiGatewayV2::Stage",
    "Properties": {
      "ApiId": {
        "Ref": "ServerlessHttpApi"
      },
      "StageName": "$default",
      "Tags": {
        "httpapi:createdBy": "SAM"
      },
      "AutoDeploy": true
    }
  },
  "SampleTable": {
    "Type": "AWS::DynamoDB::Table",
    "Metadata": {
      "SamResourceId": "SampleTable"
    },
    "Properties": {
      "AttributeDefinitions": [
        {
          "AttributeName": "id",
          "AttributeType": "S"
        }
      ],
      "KeySchema": [
        {
          "AttributeName": "id",
          "KeyType": "HASH"
        }
      ],
      "BillingMode": "PAY_PER_REQUEST"
    }
  },
  "getAllItemsFunctionMyConnPolicy": {
    "Type": "AWS::IAM::ManagedPolicy",
    "Metadata": {
      "aws:sam:connectors": {
```

```
          "getAllItemsFunctionMyConn": {
            "Source": {
              "Type": "AWS::Serverless::Function"
            },
            "Destination": {
              "Type": "AWS::Serverless::SimpleTable"
            }
          }
        }
      },
      "Properties": {
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": [
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PartiQLSelect"
              ],
              "Resource": [
                {
                  "Fn::GetAtt": [
                    "SampleTable",
                    "Arn"
                  ]
                },
                {
                  "Fn::Sub": [
                    "${DestinationArn}/index/*",
                    {
                      "DestinationArn": {
                        "Fn::GetAtt": [
                          "SampleTable",
                          "Arn"
                        ]
                      }
                    }
                  ]
                }
```

```
              ]
            }
          ]
        },
        "Roles": [
          {
            "Ref": "getAllItemsFunctionRole"
          }
        ]
      }
    }
  }
}
```

AWS SAM を使用して、23 行のインフラストラクチャコードを定義します。AWS SAM は、コードを変換して、アプリケーションのプロビジョニングに必要な 200 行以上の AWS CloudFormation コードに変換します。

AWS SAMCLI とは？

AWS SAM CLI は、AWS SAM テンプレートやサポートされているサードパーティーの統合と併用することで、サーバーレスアプリケーションを構築し、実行できるコマンドラインツールです。AWS SAM CLI を使用します。

- 新しいアプリケーションプロジェクトを迅速に初期化します。
- デプロイ用にアプリケーションを構築します。
- ローカルでのデバッグとテストを実行します。
- アプリケーションをデプロイします。
- CI/CD デプロイパイプラインを設定します。
- クラウド内のアプリケーションをモニタリングおよびトラブルシューティングします。
- 開発中にローカルの変更をクラウドに同期します。
- その他にも多くのことを実行できます。

AWS SAM CLI は、AWS SAM および AWS CloudFormation テンプレートとともに使用すると最も効果的に利用できます。Terraform などのサードパーティー製品とも連携します。

新しいプロジェクトを初期化する

スターターテンプレートから選択するか、カスタムテンプレートの場所を選択して、新しいプロジェクトを開始します。

ここでは、sam init コマンドを使用して新しいアプリケーションプロジェクトを初期化します。まず、Hello World サンプルプロジェクトを選択します。AWS SAM CLI はスターターテンプレートをダウンロードし、プロジェクトフォルダのディレクトリ構造を作成します。

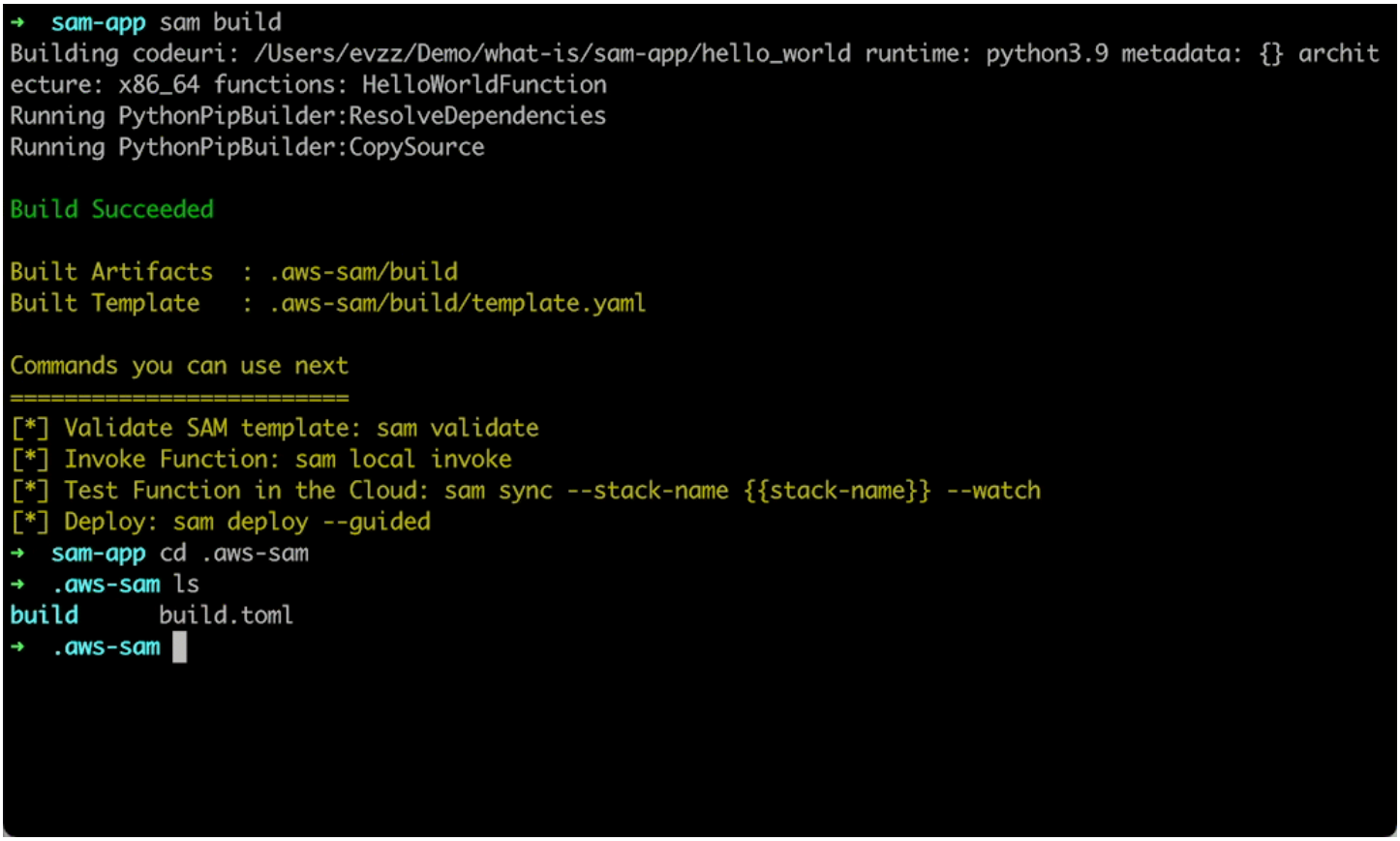
```
→  what-is sam init

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

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

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

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

詳細については、「AWS SAM でアプリケーションを作成する」を参照してください。

デプロイ用にアプリケーションを構築する

関数の依存関係をパッケージ化し、プロジェクトコードとフォルダ構造を整理して、デプロイの準備をします。

ここでは、sam build コマンドを使用してアプリケーションのデプロイを準備します。AWS SAM CLI は `.aws-sam` ディレクトリを作成し、そこにアプリケーションの依存関係とファイルをデプロイ用に整理します。

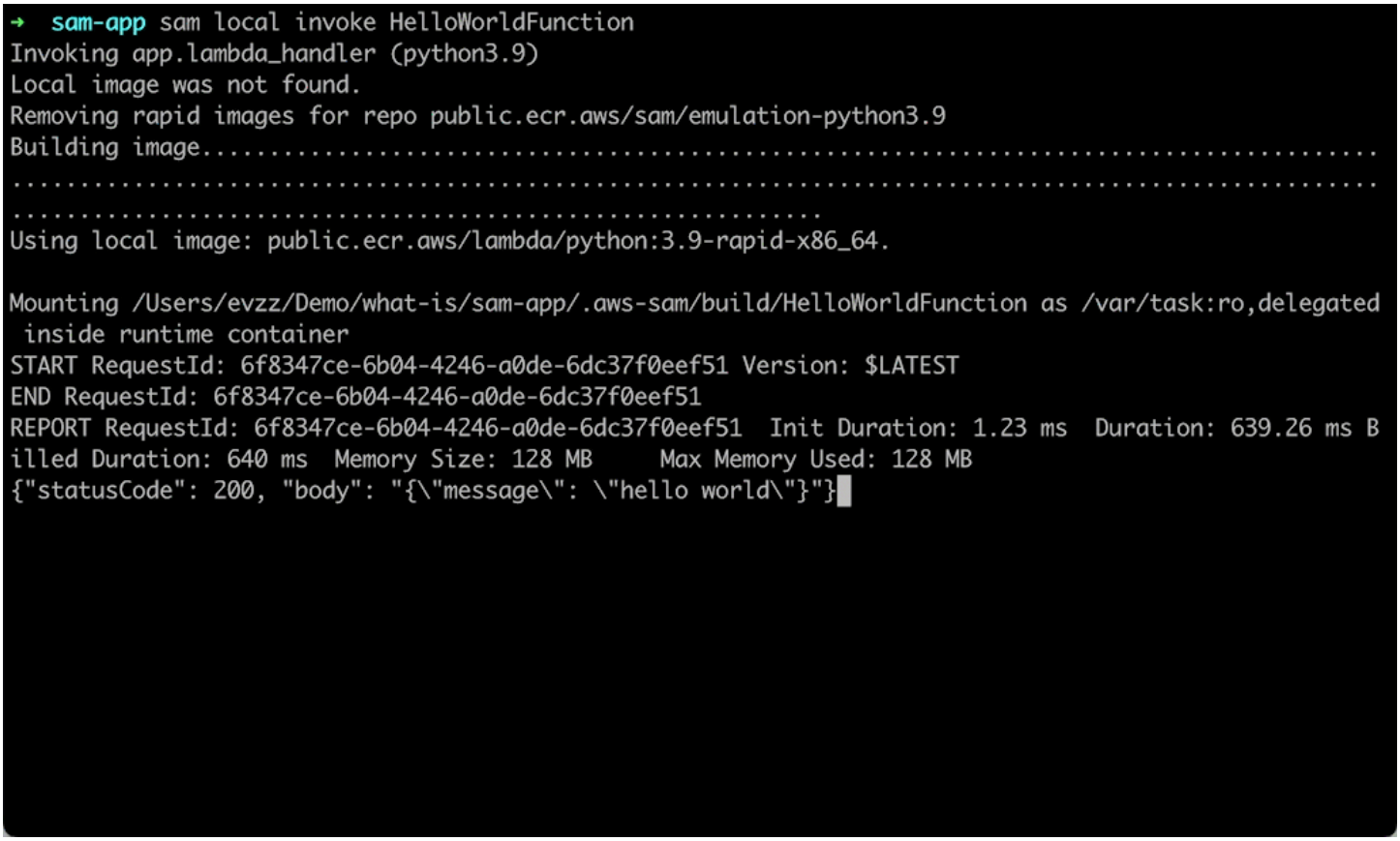


詳細については、「アプリケーションを構築する」を参照してください。

ローカルでのデバッグとテストを実行する

ローカルマシン上で、イベントのシミュレート、API のテスト、関数の呼び出しなどを実行して、アプリケーションをデバッグおよびテストします。

ここでは、sam local invoke コマンドを使用してローカルで `HelloWorldFunction` を呼び出します。これを実現するために、AWS SAM CLI はローカルコンテナを作成し、関数を構築して呼び出し、結果を出力します。Docker などのアプリケーションを使用して、マシンでコンテナを実行できます。



詳細については、「アプリケーションをテストする」および「アプリケーションをデバッグする」を参照してください。

アプリケーションをデプロイします

アプリケーションのデプロイ設定を構成し、AWS クラウドにデプロイしてリソースをプロビジョニングします。

ここでは、sam deploy --guided コマンドを使用して、インタラクティブフローを通じてアプリケーションをデプロイします。AWS SAM CLI は、アプリケーションのデプロイ設定を通じてユーザーにガイドを提供し、テンプレートを AWS CloudFormation に変換し、AWS CloudFormation にデプロイしてリソースを作成します。

```
➜  sam-app sam deploy --guided

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

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

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

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

詳細については、「アプリケーションとリソースをデプロイする」を参照してください。

CI/CD デプロイパイプラインを設定する

サポートされている CI/CD システムを使用して、安全な継続的インテグレーションおよびデリバリー (CI/CD) パイプラインを作成します。

ここでは、sam pipeline init --bootstrap コマンドを使用してアプリケーションの CI/CD デプロイパイプラインを設定します。AWS SAM CLI はオプションを通じてユーザーにガイドを提供し、CI/CD システムで使用する AWS リソースと設定ファイルを生成します。

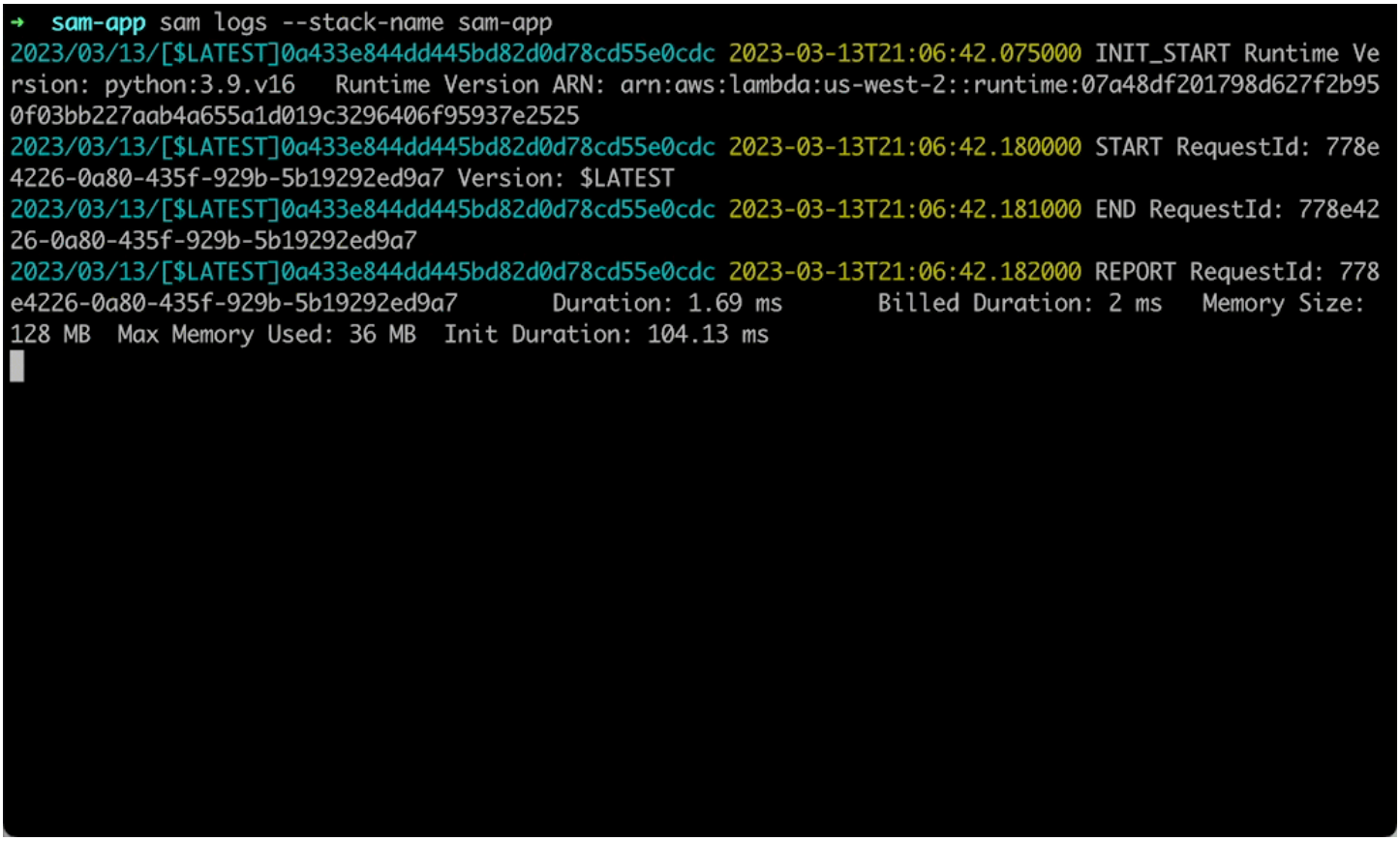
```
[3] Reference application build resources
Enter the pipeline execution role ARN if you have previously created one, or we will create one for y
ou []:
Enter the CloudFormation execution role ARN if you have previously created one, or we will create one
 for you []:
Please enter the artifact bucket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

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

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

詳細については、「CI/CD システムとパイプラインを使用したデプロイ」を参照してください。

クラウド内のアプリケーションをモニタリングおよびトラブルシューティングする

デプロイされたリソースに関する重要な情報を表示し、ログを収集し、AWS X-Ray などの組み込みモニタリングツールを利用します。

ここでは、sam list コマンドを使用してデプロイされたリソースを表示します。API エンドポイントを取得して呼び出し、関数をトリガーします。その後、sam logs を使用して関数のログを表示します。

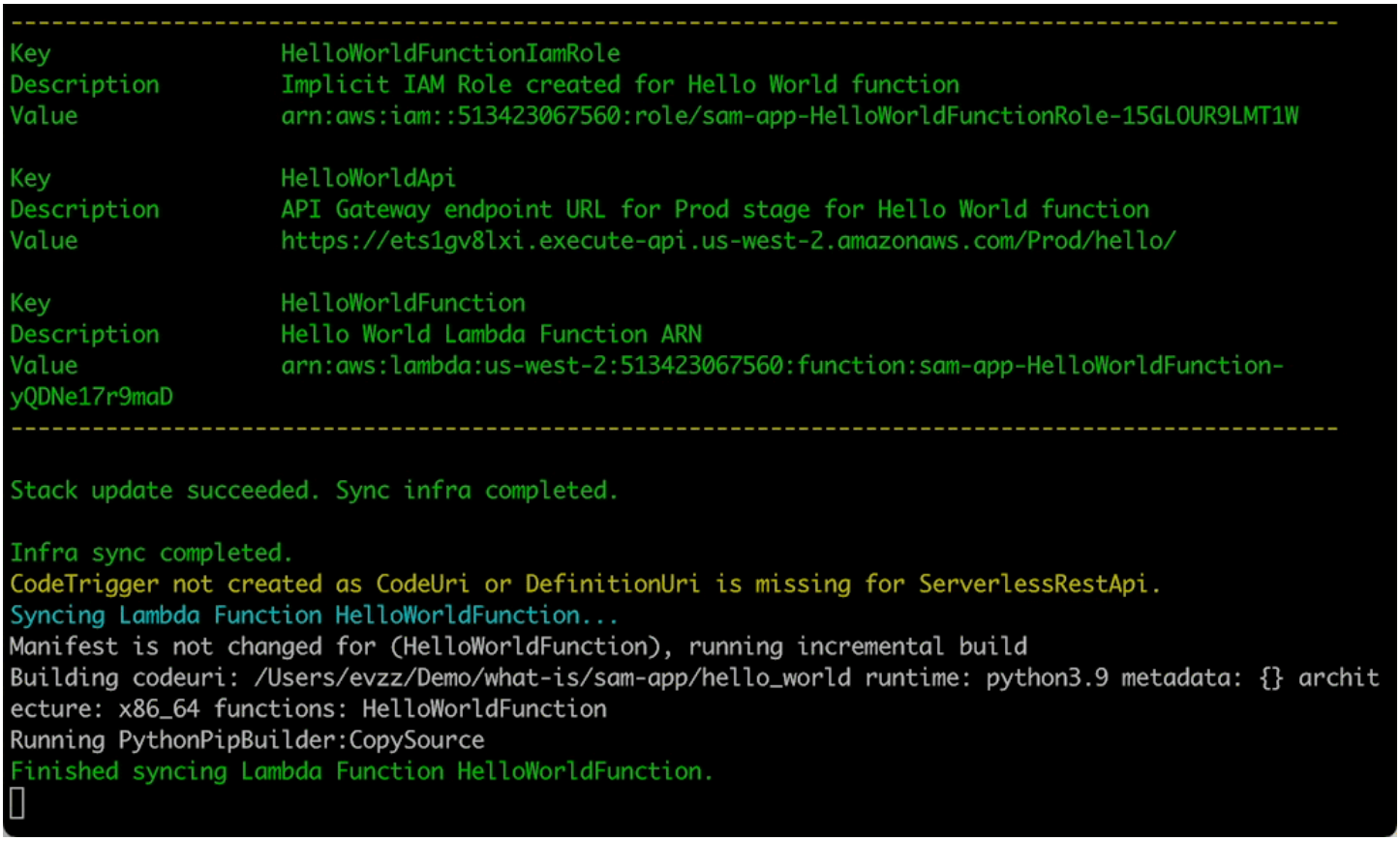


詳細については、「アプリケーションをモニタリングする」を参照してください。

開発中にローカルの変更をクラウドに同期する

ローカルマシンで開発すると、変更がクラウドに自動的に同期されます。変更をすばやく確認し、クラウドでテストと検証を実行します。

ここでは、sam sync --watch コマンドを使用して、AWS SAM CLI がローカルの変更を監視するようにします。`HelloWorldFunction` コードを変更すると、AWS SAM CLI が自動的に変更を検出し、更新をクラウドにデプロイします。



サポートされているリソースをクラウドでテストする

クラウド内のサポート対象リソースを呼び出して、イベントを渡します。

ここでは、クラウドにデプロイされた Lambda 関数をテストするために sam remote invoke コマンドを使用します。Lambda 関数を呼び出して、そのログとレスポンスを受け取ります。Lambda 関数はレスポンスをストリーミングするように設定されているので、AWS SAM CLI はそのレスポンスをリアルタイムでストリーミングして返します。

詳細はこちら

AWS SAM の詳細については、次のリソースを参照してください。

- AWS SAM コンプリートワークショップ — AWS SAM の主な機能を学ぶためのワークショップ。
- SAM を使用したセッション – AWS サーバーレスデベロッパーアドボケイトチームが作成した、AWS SAM の使用に関する動画シリーズ。
- Serverless Land – AWS サーバーレスに関する最新情報、ブログ、動画、コード、学習リソースをまとめたサイト。

次のステップ

AWS SAM を初めて使用する場合は、「AWS SAM の開始方法」を参照してください。

AWS Serverless Application Model 向けサーバーレスの概念

AWS Serverless Application Model (AWS SAM) を使用する前に、サーバーレスに関する基本的な概念をご覧ください。

サーバーレスの概念

イベント駆動型アーキテクチャ

サーバーレスアプリケーションは、コンピューティングのための AWS Lambda やデータベース管理のための Amazon DynamoDB など、それぞれが特殊な役割を果たす個別の AWS のサービスで構成されます。これらのサービスは、イベント駆動型のアーキテクチャを通じて相互に緩やかに統合されます。イベント駆動型アーキテクチャの詳細については、「イベント駆動型アーキテクチャとは」を参照してください。

Infrastructure as Code (IaC)

Infrastructure as Code (IaC) は、デベロッパーがコードを扱うのと同じ方法でインフラストラクチャを扱う方法であり、アプリケーションコード開発と同じ厳密さをインフラストラクチャのプロビジョニングに適用します。テンプレートファイルでインフラストラクチャを定義し、それを AWS にデプロイすると、AWS がリソースを作成します。IaC では、プロビジョニングする AWS をコードで定義します。詳細については、「AWS での DevOps の概要」の AWS ホワイトペーパーの「Infrastructure as Code」を参照してください。

サーバーレステクノロジー

AWS サーバーレステクノロジーを使用すると、独自のサーバーを管理することなく、アプリケーションを構築して実行できます。すべてのサーバー管理は AWS によって行われるため、自動スケーリングや組み込みの高可用性などの多くのメリットを活用でき、アイデアを迅速に本番環境で具現化できます。サーバーレステクノロジーを使用すると、サーバーの管理や運用について心配することなく、製品の中核に注力できます。サーバーレスの詳細については、次を参照してください。

- AWS でのサーバーレス
- サーバーレスデベロッパーガイド – AWS クラウドでのサーバーレス開発の概念的な概要を提供します。

中核的な AWS サーバーレスサービスの基本的な概要については、Serverless Land の「サーバーレスの基礎: サーバーレスサービスについて」を参照してください。

サーバーレスアプリケーション

AWS SAM を使用する場合、アプリケーション内で AWS SAM プロジェクトとテンプレートで構成されている関連リソースを管理します。アプリケーション内のすべてのリソースは、AWS SAM テンプレートで定義または参照されます。AWS SAM がテンプレートを処理すると、AWS CloudFormation リソースも作成されます。AWS CloudFormation では、リソースはスタックと呼ばれる単一のユニットで管理され、スタック内のすべてのリソースは、そのスタックの AWS CloudFormation テンプレートによって定義されます。

AWS SAM の開始方法

このセクションのトピックを確認して完了し、AWS SAM の使用を開始します。AWS SAM 前提条件 では、AWS アカウントの設定、IAM ユーザーの作成、キーアクセスの作成、AWS SAM CLI のインストールと設定に関する詳細な手順について説明されています。前提条件を完了したら、AWS SAM CLI のインストール の準備が整います。Linux、Windows、macOS オペレーティングシステムでこれを実行できます。インストールが完了したら、オプションで AWS SAM Hello World チュートリアルを順を追って実行できます。このチュートリアルでは、AWS SAM で基本的なサーバーレスアプリケーションを作成するプロセスについて説明します。チュートリアルを完了すると、AWS Serverless Application Model (AWS SAM) を使用する方法 で説明されている概念を確認する準備が整います。

トピック

- AWS SAM 前提条件
- AWS SAM CLI のインストール
- チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM

AWS SAM 前提条件

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI).

を使用するには AWS SAM CLI、以下が必要です。

- AWS アカウント、 AWS Identity and Access Management （ IAM) 認証情報、およびIAMアクセスキーペア。
- AWS 認証情報を設定する AWS Command Line Interface （ AWS CLI ）。

トピック

- ステップ 1: AWS アカウントにサインアップする
- ステップ 2: IAMユーザーアカウントを作成する
- ステップ 3: アクセスキー ID とシークレットアクセスキーを作成します
- ステップ 4: をインストールする AWS CLI
- ステップ 5: AWS CLI を使用して AWS 認証情報を設定する
- 次のステップ

ステップ 1: AWS アカウントにサインアップする

がない場合は AWS アカウント、次のステップを実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/サインアップを開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

ステップ 2: IAMユーザーアカウントを作成する

管理者ユーザーを作成するには、以下のいずれかのオプションを選択します。

管理者を管理する方法を 1 つ選択します	目的	方法	以下の操作も可能
IAM Identity Center 内 (推奨)	短期の認証情報を使用して AWSにアクセスします。 これはセキュリティのベストプラクティスと一致しています。ベストプラクティスの詳細については、「IAMユーザーガイド」	AWS IAM Identity Center ユーザーガイドの「開始方法」の手順に従います。	AWS Command Line Interface ユーザーガイドの を使用する AWS CLI ように を設定 AWS IAM Identity Centerして、プログラムによるアクセスを設定します。

管理者を管理する方法を 1 つ選択します	目的	方法	以下の操作も可能
	の「 のセキュリティのベストプラクティス IAM」を参照してください。		
IAM 内 (非推奨)	長期認証情報を使用して AWSにアクセスする。	「 ユーザーガイド」の「緊急アクセス用の IAM ユーザーを作成する」の手順に従います。 IAM	IAM 「 ユーザーガイド」のIAM「ユーザーのアクセスキーを管理する」でプログラムによるアクセスを設定します。

ステップ 3: アクセスキー ID とシークレットアクセスキーを作成します

CLI アクセスには、アクセスキー ID とシークレットアクセスキーが必要です。長期のアクセスキーの代わりに一時的な認証情報をできるだけ使用します。一時的な認証情報には、アクセスキー ID、シークレットアクセスキー、および認証情報の失効を示すセキュリティトークンが含まれています。詳細については、「 ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」を参照してください。 IAM

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 がアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、ま	使用するインターフェイス用の手引きに従ってください。

プログラマチックアクセス権を必要とするユーザー	目的	方法
（IAMIdentity Center で管理されるユーザー）	たは へのプログラムによるリクエストに署名します AWS APIs。	• については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 を使用する AWS CLI ための の設定 AWS IAM Identity Center」を参照してください。 • AWS SDKs、ツール、および については AWS APIs、AWS SDKs「 およびツールリファレンスガイド」のIAM「 Identity Center 認証」を参照してください。
IAM	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または へのプログラムによるリクエストに署名します AWS APIs。	「 ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。 IAM

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	(非推奨) 長期認証情報を使用して、、AWS SDKs AWS CLI、または AWS APIs へのプログラムによるリクエストに署名します。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」のIAM「ユーザー認証情報を使用した認証」を参照してください。 • および ツールについては AWS SDKs、「 AWS SDKs およびツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 • については AWS APIs、「 IAMユーザーガイド」のIAM「 ユーザーのアクセスキーの管理」を参照してください。

ステップ 4: をインストールする AWS CLI

AWS CLI は、コマンドラインシェルでコマンド AWS のサービス を使用して とやり取りできるオープンソースツールです。 AWS SAM CLI では、認証情報の設定などのアクティビティ AWS CLI にが必要です。の詳細については AWS CLI、 AWS Command Line Interface ユーザーガイドの「 とは AWS Command Line Interface」を参照してください。

をインストールするには AWS CLI、「 ユーザーガイド」の「 の最新バージョンのインストールまたは更新 AWS CLI」を参照してください。 AWS Command Line Interface

ステップ 5: AWS CLI を使用して AWS 認証情報を設定する

IAM Identity Center で認証情報を設定するには

- IAM Identity Center で認証情報を設定するには、「Configure your profile with the configure sso wizard AWS」を参照してください。

を使用して認証情報を設定するには AWS CLI

1. コマンドラインから `aws configure` コマンドを実行します。
2. 次を設定します。詳細については、各リンクを選択してください。

   a. アクセスキー ID

   b. シークレットアクセスキー

   c. AWS リージョン

   d. 出力形式

   次の例は、サンプル値を示しています。

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

は、この情報を `credentials`および `config` ファイル内の という名前のプロファイル (設定のコレクション) `default` に AWS CLI 保存します。これらのファイルは、ホームディレクトリ内の `.aws` ファイルにあります。デフォルトでは、このプロファイルの情報は、使用するプロファイルを明示的に指定しない AWS CLI コマンドを実行するときに使用されます。`credentials` ファイルの詳細については、「AWS Command Line Interface ユーザーガイド」の「設定ファイルと認証情報ファイルの設定」を参照してください。

既存の設定や認証情報ファイルの使用など、認証情報の設定の詳細については、「AWS Command Line Interface ユーザーガイド」の「クイック設定」を参照してください。

次のステップ

これで、 をインストールする準備ができました。 AWS SAM CLI と の使用を開始します AWS SAM。をインストールするには AWS SAM CLI「AWS SAM CLI のインストール」を参照してください。

AWS SAM CLI のインストール

サポートされているオペレーティングシステムに AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) の最新リリースをインストールします。

アップグレード、アンインストール、ナイトリービルドの管理など、現在インストールされている AWS SAM CLI のバージョンを管理する方法については、「AWS SAM CLI バージョンの管理」を参照してください。

> AWS SAM CLI CLI を初めてインストールする場合:
>
> 次に進む前に、前のセクションの前提条件をすべて満たします。これには、以下が含まれます。
>
> 1. AWS アカウントへのサインアップ。
> 2. 管理者 IAM ユーザーを作成する。
> 3. アクセスキー ID とシークレットアクセスキーを作成する。
> 4. AWS CLI をインストールする。
> 5. AWS 認証情報を設定する。

トピック

AWS SAM CLIのインストール

> **Note**
>
> 2023 年 9 月以降、AWS では AWS で管理される AWS SAM CLI (`aws/tap/aws-sam-cli`) 用 Homebrew インストーラーのメンテナンスを行いません。Homebrew で AWS SAM CLI をインストールし、管理する場合は、以下の方法を参照してください。
>
> - Homebrew を引き続き使用するには、コミュニティ管理のインストーラーを使用します。詳細については、「Homebrew で AWS SAM CLI を管理する」を参照してください。
> - このページに記載されているファーストパーティ製のインストール方法のいずれかを使用することをお勧めします。これらの方法のいずれかを使用する前に、Homebrew から移行する を参照してください。
> - さらなる詳細については、「リリースバージョン: 1.121.0 」を参照してください。

AWS SAM CLI をインストールするには、お使いのオペレーティングシステムの手順に従ってください。

Linux

x86_64 - command line installer

1. AWS SAM CLI .zip ファイルを任意のディレクトリにダウンロードします。
2. (オプション) インストール前にインストーラーの整合性を確認できます。手順については、オプション: AWS SAM CLI インストーラの整合性を検証する を参照してください。
3. インストールファイルを任意のディレクトリに解凍します。`sam-installation` サブディレクトリを使用した例を次に示します。

   > **Note**
   >
   > オペレーティングシステムに組み込み unzip コマンドがない場合は、同等のコマンドを使用します。

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

4. install 実行可能ファイルを実行して AWS SAM CLI をインストールします。この実行可能ファイルは、前のステップで使用したディレクトリにあります。sam-installation サブディレクトリを使用した例を次に示します。

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

5. インストールを確認します。

```
$ sam --version
```

表示される次のような出力で、正常にインストールされていることを確認します。角括弧で囲まれたテキストは、利用可能な最新バージョンに置き換わります。

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

arm64 - command line installer

1. AWS SAM CLI .zip ファイルを任意のディレクトリにダウンロードします。
2. (オプション) インストール前にインストーラーの整合性を確認できます。手順については、オプション: AWS SAM CLI インストーラの整合性を検証する を参照してください。
3. インストールファイルを任意のディレクトリに解凍します。sam-installation サブディレクトリを使用した例を次に示します。

> **Note**
>
> オペレーティングシステムに組み込み unzip コマンドがない場合は、同等のコマンドを使用します。

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

4. install 実行可能ファイルを実行して AWS SAM CLI をインストールします。この実行可能ファイルは、前のステップで使用したディレクトリにあります。sam-installation サブディレクトリを使用した例を次に示します。

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

5. インストールを確認します。

```
$ sam --version
```

表示される次のような出力で、インストールが成功したことを確認します。角括弧付きのテキストは、最新の SAM CLI バージョンに置き換わります。

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

macOS

インストール手順

パッケージインストーラーを使用して AWS SAM CLI をインストールします。さらに、このパッケージインストーラには、GUI とコマンドラインの 2 つのインストール方法があります。インストールの対象は、すべてのユーザーにすることも、現在のユーザーのみにすることもできます。すべてのユーザーにインストールするには、スーパーユーザー承認が必要です。

GUI - All users

パッケージインストーラをダウンロードして AWS SAM CLI をインストールするには

> **Note**
>
> 以前に Homebrew または pip を使用して AWS SAM CLI をインストールした場合は、まずその CLI をアンインストールする必要があります。手順については、AWS SAM CLI のアンインストール を参照してください。

1. macOS pkg を好みのディレクトリにダウンロードします。
   - Intel プロセッサーを実行している Mac の場合は、[x86_64] – [aws-sam-cli-macos-x86_64.pkg] を選択します。
   - Apple シリコンを実行している Mac の場合は、[arm64] – [aws-sam-cli-macos-arm64.pkg] を選択します。

> **Note**
>
> オプションで、インストール前にインストーラーの整合性を確認することができます。手順については、オプション: AWS SAM CLI インストーラの整合性を検証するを参照してください。

2. ダウンロードしたファイルを実行し、画面の指示に従って、「はじめに」、「Read me」、「ライセンス」の手順を続けてください。
3. [Destination Select] (インストール先の選択) で、[Install for all users of this computer] (このコンピュータのすべてのユーザーにインストール) を選択します。
4. [Installation Type] (インストールタイプ) で、AWS SAM CLI をインストールする場所を選択し、[Install] (インストール) を押します。推奨されるデフォルトの場所は `/usr/local/aws-sam-cli` です。

> **Note**
>
> sam コマンドで AWS SAM CLI を呼び出すために、インストーラは `/usr/local/bin/sam` と `/usr/local/aws-sam-cli/sam` または選択したインストールフォルダのいずれかとの間のシンボリックリンクを自動的に作成します。

5. AWS SAM CLI がインストールされ、「The installation was successful」(インストールは成功しました) というメッセージが表示されます。[Close] (閉じる) を押します。

正常にインストールされたことを確認するには

- AWS SAM CLI が正しくインストールされ、シンボリックリンクが設定されていることを確認するには、次のコマンドを実行します。

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

GUI - Current user

AWS SAM CLI をダウンロードしてインストールするには

> **Note**
>
> 以前に Homebrew または pip を使用して AWS SAM CLI をインストールした場合は、まずその CLI をアンインストールする必要があります。手順については、AWS SAM CLI のアンインストール を参照してください。

1. macOS pkg を好みのディレクトリにダウンロードします。

   - Intel プロセッサーを実行している Mac の場合は、[x86_64] – [aws-sam-cli-macos-x86_64.pkg] を選択します。
   - Apple シリコンを実行している Mac の場合は、[arm64] – [aws-sam-cli-macos-arm64.pkg] を選択します。

   > **Note**
   >
   > オプションで、インストール前にインストーラーの整合性を確認することができます。手順については、オプション: AWS SAM CLI インストーラの整合性を検証する を参照してください。

2. ダウンロードしたファイルを実行し、画面の指示に従って、「はじめに」、「Read me」、「ライセンス」の手順を続けてください。
3. [Destination Select] (インストール先の選択) で、[Install for me only] (現在のユーザーのみにインストール) を選択します。このオプションが表示されない場合は、次の手順に進みます。
4. [Installation Type] (インストールタイプ) で、以下を実行します。

   1. AWS SAM CLI をインストールする場所を選択します。デフォルトの場所は `/usr/local/aws-sam-cli` です。書き込み権限が付与されている場所を選択してください。インストール場所を変更するには、[local] (ローカル) を選択し、場所を選択します。完了したら、[Continue] (続行) を押します。
   2. 前のステップで [Install for me only] (現在のユーザーのみにインストール) を選択するオプションが表示されない場合は、[Change Install Location] (インストール場所を変更) >

[Install for me only] (現在のユーザーのみにインストール) を選択し、[Continue] (続行) を押します。

3. [Install] (インストール) を押します。

5. AWS SAM CLI がインストールされ、「The installation was successful」(インストールは成功しました) というメッセージが表示されます。[Close] (閉じる) を押します。

シンボリックリンクを作成するには

- sam コマンドで AWS SAM CLI を呼び出すには、AWS SAM CLI プログラムと $PATH との間のシンボリックリンクを手動で作成する必要があります。次のコマンドを変更および実行して、シンボリックリンクを作成します。

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

  - *sudo* — ユーザーが $PATH への書き込み権限を持っている場合、sudo は必須ではありません。それ以外の場合、sudo が必要です。
  - *path-to* — AWS SAM CLI プログラムをインストールした場所へのパス。例えば、`/Users/myUser/Desktop` と指定します。
  - *path-to-symlink-directory* — $PATH の環境変数。デフォルトの場所は `/usr/local/bin` です。

正常にインストールされたことを確認するには

- AWS SAM CLI が正しくインストールされ、シンボリックリンクが設定されていることを確認するには、次のコマンドを実行します。

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

Command line - All users

AWS SAM CLI をダウンロードしてインストールするには

> **Note**
>
> 以前に Homebrew または pip を使用して AWS SAM CLI をインストールした場合は、まずその CLI をアンインストールする必要があります。手順については、AWS SAM CLI のアンインストール を参照してください。

1. macOS pkg を好みのディレクトリにダウンロードします。

   - Intel プロセッサーを実行している Mac の場合は、[x86_64] – [aws-sam-cli-macos-x86_64.pkg] を選択します。
   - Apple シリコンを実行している Mac の場合は、[arm64] – [aws-sam-cli-macos-arm64.pkg] を選択します。

   > **Note**
   >
   > オプションで、インストール前にインストーラーの整合性を確認することができます。手順については、オプション: AWS SAM CLI インストーラの整合性を検証する を参照してください。

2. インストールスクリプトを変更および実行します。

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

   > **Note**
   >
   > sam コマンドで AWS SAM CLI を呼び出すため、インストーラは、`/usr/local/bin/sam` と `/usr/local/aws-sam-cli/sam` の間のシンボリックリンクを自動的に作成します。

正常にインストールされたことを確認するには

- AWS SAM CLI が正しくインストールされ、シンボリックリンクが設定されていることを確認するには、次のコマンドを実行します。

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

Command line - Current user

AWS SAM CLI をダウンロードしてインストールするには

> **Note**
>
> 以前に Homebrew または pip を使用して AWS SAM CLI をインストールした場合は、まずその CLI をアンインストールする必要があります。手順については、AWS SAM CLI のアンインストール を参照してください。

1. macOS pkg を好みのディレクトリにダウンロードします。
   - Intel プロセッサーを実行している Mac の場合は、[x86_64] – [aws-sam-cli-macos-x86_64.pkg] を選択します。
   - Apple シリコンを実行している Mac の場合は、[arm64] – [aws-sam-cli-macos-arm64.pkg] を選択します。

   > **Note**
   >
   > オプションで、インストール前にインストーラーの整合性を確認することができます。手順については、オプション: AWS SAM CLI インストーラの整合性を検証する を参照してください。

2. 書き込み権限のあるインストールディレクトリを決定します。次に、テンプレートを使用して `xml` ファイルを作成し、インストールディレクトリを反映するようにファイルを変更します。ディレクトリは既に存在している必要があります。

例えば、*path-to-my-directory* を /Users/myUser/Desktop に置き換えると、aws-sam-cli プログラムフォルダがそこにインストールされます。

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

3. xml ファイルを保存し、以下を実行して有効であることを確認します。

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

出力には、 AWS SAM CLI プログラムに適用される設定が表示されるはずです。

4. 次のコマンドを実行して AWS SAM CLI をインストールします。

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

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

シンボリックリンクを作成するには

- sam コマンドで AWS SAM CLI を呼び出すには、AWS SAM CLI プログラムと $PATH との間のシンボリックリンクを手動で作成する必要があります。次のコマンドを変更および実行して、シンボリックリンクを作成します。

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

  - *sudo* — ユーザーが $PATH への書き込み権限を持っている場合、sudo は必須ではありません。それ以外の場合、sudo が必要です。
  - *path-to* — AWS SAM CLI プログラムをインストールした場所へのパス。例えば、/Users/myUser/Desktop と指定します。
  - *path-to-symlink-directory* — $PATH の環境変数。デフォルトの場所は /usr/local/bin です。

正常にインストールされたことを確認するには

- AWS SAM CLI が正しくインストールされ、シンボリックリンクが設定されていることを確認するには、次のコマンドを実行します。

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

Windows

Windows インストーラ (MSI) ファイルは、Windows オペレーティングシステムのパッケージインストーラファイルです。

これらの手順に従って、MSI ファイルを使用して AWS SAM CLI をインストールします。

1. AWS SAM CLI 64 ビットをダウンロードします。
2. (オプション) インストール前にインストーラーの整合性を確認できます。手順については、オプション: AWS SAM CLI インストーラの整合性を検証する を参照してください。
3. インストールを確認します。

インストールが完了したら、新しいコマンドプロンプトまたは PowerShell プロンプトを開いて確認します。コマンドラインから sam を呼び出すことができるはずです。

```
sam --version
```

AWS SAM CLI が正常にインストールされると、以下のような出力が表示されます。

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

4. ロングパスを有効にします (Windows 10 以降のみ)。

> **Important**
>
> AWS SAM CLI が、Windows の最大パス制限を超えるファイルパスを操作している場合があります。これにより、Windows 10 MAX_PATH の制限が原因で、sam init の実行時にエラーが発生することがあります。この問題を解決するには、新しい長いパスの動作を設定する必要があります。

長いパスを有効にするには、「Windows アプリの開発に関するドキュメント」の「Windows 10、バージョン 1607 以降で長いパスを有効にする」を参照してください。

5. Git をインストールする。

sam init コマンドを使用してサンプルアプリケーションをダウンロードするには、Git をインストールする必要があります。手順については、「Git のインストール」を参照してください。

インストールエラーのトラブルシューティング

Linux

Docker エラー: 「Cannot connect to the Docker daemon。Is the docker daemon running on this host?」

ec2-user が Docker デーモンにアクセスするための許可の提供には、インスタンスの再起動が必要になる場合があります。このエラーが表示された場合は、インスタンスを再起動してみてください。

シェルエラー:「command not found」

このエラーが表示された場合は、シェルがパス内の AWS SAM CLI 実行可能ファイルを見つけられません。AWS SAM CLI 実行可能ファイルをインストールしたディレクトリの場所を確認してから、ディレクトリがパス上にあることを確認します。

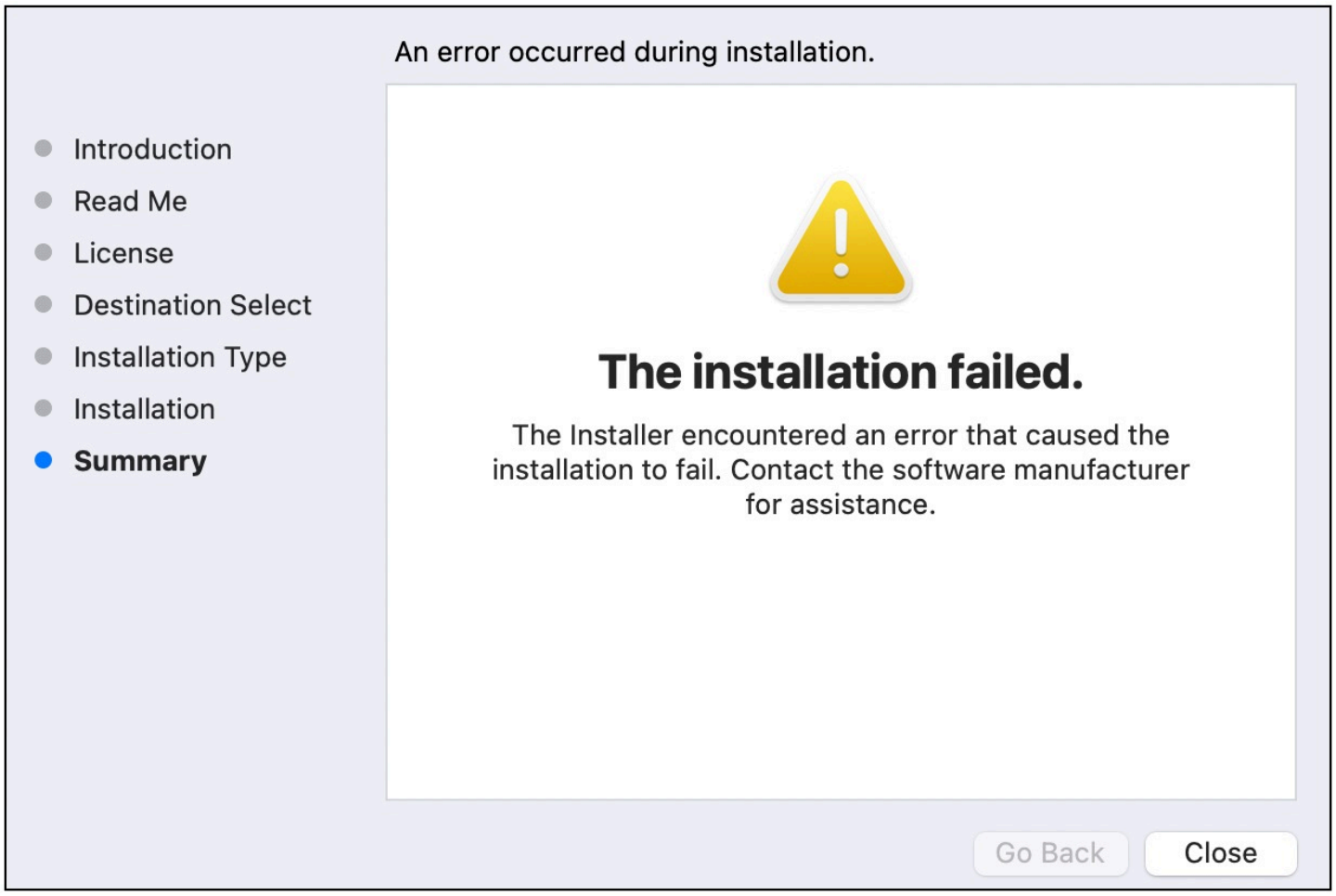
AWS SAM CLI エラー:「/lib64/libc.so.6: version `GLIBC_2.14' not found (required by /usr/local/aws-sam-cli/dist/libz.so.1)」

このエラーが表示された場合は、サポートされていないバージョンの Linux を使用しており、組み込みの glibc バージョンが古くなっています。以下のいずれかを実行してみてください。

- Linux ホストを CentOS、Fedora、Ubuntu、または Amazon Linux 2 の最新ディストリビューションの 64 ビット版にアップグレードする。
- 「AWS SAM CLI のインストール」の手順を実行する。

macOS

インストールが失敗しました



ユーザー用に AWS SAM CLI をインストールするときに、書き込み権限のないインストールディレクトリを選択した場合、このエラーが発生する可能性があります。以下のいずれかを実行してみてください。

1. 書き込み権限のある別のインストールディレクトリを選択してください。
2. インストーラを削除します。次に、再度インストーラをダウンロードして実行します。

次のステップ

AWS SAM CLIと、独自のサーバーレスアプリケーションの構築に関する詳細については、以下を参照してください。

- チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM - 基本的なサーバーレスアプリケーションをダウンロード、構築、およびデプロイするためのステップバイステップの手順です。
- AWS SAM コンプリートワークショップ — AWS SAM の主な機能を学ぶためのワークショップ。
- AWS SAM アプリケーションとパターンの例 – さらに実験できるコミュニティ作成者からのサンプルアプリケーションとパターン。

オプション: AWS SAM CLI インストーラの整合性を検証する

パッケージインストーラを使用して AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) をインストールする場合、インストール前にその整合性を確認できます。このステップはオプションですが強く推奨されます。

確認には次の 2 つのオプションがあります。

- パッケージインストーラーの署名ファイルを確認します。
- パッケージインストーラーのハッシュ値を確認します。

プラットフォームで両方のオプションで確認できる場合、署名ファイルを確認するオプションの方をお勧めします。このオプションでは、キー値がここで公開され、GitHub リポジトリとは別に管理されるため、セキュリティがさらに強化されます。

トピック

- インストーラの署名ファイルを確認
- ハッシュ値を確認

インストーラの署名ファイルを確認

Linux

arm64 - コマンドラインインストーラー

AWS SAM は GnuPG を使用して AWS SAM CLI インストーラに署名します。以下の手順で確認が実行されます。

1. プライマリパブリックキーを使用して、署名者のパブリックキーを確認します。

2. 署名者パブリックキーを使用して AWS SAM CLI パッケージインストーラを確認します。

署名者のパブリックキーの整合性を確認するには

1. プライマリパブリックキーをコピーし、ローカルマシンに .txt ファイルとして保存します。例えば、*primary-public-key.txt* と指定します。

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

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

2. プライマリパブリックキーをキーリングにインポートします。

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

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

3. 署名者のパブリックキーをコピーし、ローカルマシンに .txt ファイルとして保存します。例えば、*signer-public-key.txt* と指定します。

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

mQINBGRtS20BEAC7GjaAwverrB1zNEu2q3EGI6HC37WzwL5dy30f4LirZOWS3piK
oKfTqPjXPrLCf1GL2mMqUSgSnpEbPNXuvWTW1CfSnnjwuH8ZqbvvUQyHJwQyYpKm
KMwb+8V0bzzQkMzDVqolYQCi5XyGpAuo3wroxXSzG6r/mIhbiq3aRnL+2lo4XOYk
r7q9bhBqbJhzjkm7N62PhPWmi/+/EGdEBakA1pReE+cKjP2UAp5L6CPShQl2fRKL
9BumitNfFHHs1JZgZSCCruiWny3XkUaXUEMfyoE9nNbfqNvuqV2KjWguZCXASgz2
ZSPF4DTVIBMfP+xrZGQSWdGU/67QdysDQW81TbFOjK9ZsRwwGC4kbg/K98IsCNHT
ril5RZbyr8pw3fw7jYjjI2ElAacRWp53iRzvutm5AruPpLfoKDQ/tKzBUYItBwlu
Z/diKgcqtw7xDlyqNyTN8xFPFqMO2I8IsZ2Pdl131htdFiZMiin1RQG9pV9p2vHS
eQVY2uKCnvnA6vFCQYKXP7p0IwReuPNzDvECUsidw8VTakTqZsANT/bU17e4KuKn
+JgbNrKOasJX37sDb/9ruysozLvy78ozYKJDLmC3yoRQ8DhEjviT4cnjORgNmvnZ
0a5AA/DJPQW4buRrXdxu+fITzBxQn2+GO/iDNCxtJaq5SYVBKjTmTWPUJwARAQAB
tDBBV1MgU0FNIENMSSBUZWFtIDxhd3Mtc2FtLWNsaS1zaWduZXJAYW1hem9uLmNv
bT6JAj8EEwEJACkFAmRtS20CGy8FCQPCZwAHCwkIBwMCAQYVCAIJCgsEFgIDAQIe
AQIXgAAKCRDHoF9D/grd+lE4D/4kJW65He2LNsbLTta7lcGfsEXCf4zgIvkytS7U
3R36zMD8IEyWJjlZ+aPkIP8/jFJrFl4pVHbU7vX85Iut1vV7m+8BgWt25mJhnoJ9
KPjXGra9mYP+Cj8zFAcjvtl3NBAPodyfcfCTWsU3umF9ArOFICcrGCzHX2SS7wX5
h9n0vYRZxk5Qj5FsgskKAQLq33CKFAMlaqZnL5gWRvTeycSIxsyus+stX+8YBPCO
J64f7+y+MPIP1+m2njlVXg1xLEMMVa08oWccOMiakgzDev3LCrPy+wdwdn7Ut7oA
pna3DNy9aYNd2lh6vUCJeJ+Yi1Bl2jYpzLcCLKrHUmln9/rRSz7Orbg8P181kfPu
G/M7CD5FwhxP3p4+0XoGwxQefrV2jqpSnbLae7xbYJiJAhbpjWDQhuNGUbPcDmqk
aH0Q3XU8AonJ8YqaQ/q3VZ3JBiH3TbBrOXsvd59cwxYyf83aJ/WLCb2P8y75zDad
lnOP713ThF5J/Afj9HjO9waFV0Z2W2ZZe4rU2OJTAiXEtM8xsFMrc7TCUacJtJGs
u4kdBmXREcVpSz65h9ImSy2ner9qktnVVCW4mZPj63IhB37YtoLAMyz3a3R2RFNk
viEX8foOTUg1FmwHoftxZ9P91QwLoTajkDrh26ueIe45sG6Uxua2AP4Vo37cFfCj
ryV8OokCHAQQAQkABgUCZG5MWAAKCRBC/V96c62IWmglD/9idU43kW8Zy8Af1j8l
```

```
Am3lI4d9ksOleeKRZqxo/SZ5rovF32DO2nw7XRXq1+EbhgJaI3QwwOi0U0pfAMVT
4b9TdxdH+n+tqzCHh3jZqmo9sw+c9WFXYJN1hU9bLzcHXS8hOTbyoE2EuXx56ds9
L/BWCcd+LIvapw0lggFfavVx/QF4C7nBKjnJ66+xxwfgVIKR7oGlqDiHMfp9ZWh5
HhEqZo/nrNhdY0h3sczEdqC2N6eIa8mgHffHZdKudDMXIXHbgdhW9pcZXDIktVf7
j9wehsWOyYXiRgR0dz7DI26AUG4JLh5FTtx9XuSBdEsI69Jd4dJuibmgtImzbZjn
7un8DJWIyqi7Ckk96Tr4oXB9mYAXaWlR4C9j5XJhMNZgkOycuY2DADnbGmSb+1kA
ju77H4ff84+vMDwUzUt2Wwb+GjzXu2g6Wh+bWhGSirYlel+6xYrI6beu1BDCFLq+
VZFE8WggjJHpwcL7CiqadfVIQaw4HY0jQFTSdwzPWhJvYjXFOhMkyCcjssbtmB+z
/otfgySyQqThrD48RWS5GuyqCA+pK3UNmEJ11c1AXMdTn2VWInR1NOJNALQ2du3y
q8t1vMsErVOJ7pkZ50F4ef17PE6DKrXX8ilwGFyVuX5ddyt/t9J5pC3sRwHWXVZx
GXwoX75FwIEHA3n5Q7rZ69Ea6Q==
=ZIO7
-----END PGP PUBLIC KEY BLOCK-----
```

4. 署名者のパブリックキーをキーリングにインポートします。

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

gpg: key FE0ADDFA: public key "AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>"
 imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
gpg: no ultimately trusted keys found
```

出力のキー値を書き留めておきます。例えば、*FE0ADDFA* と指定します。

5. キー値を使用して、署名者のパブリックキーフィンガープリントを取得して確認します。

```
$ gpg --fingerprint FE0ADDFA

pub   4096R/FE0ADDFA 2023-05-23 [expires: 2025-05-22]
      Key fingerprint = 37D8 BE16 0355 2DA7 BD6A  04D8 C7A0 5F43 FE0A DDFA
uid                  AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>
```

フィンガープリントは、次のものと一致する必要があります。

```
37D8 BE16 0355 2DA7 BD6A  04D8 C7A0 5F43 FE0A DDFA
```

フィンガープリント文字列が一致しない場合は、AWS SAM CLI インストーラを使用しないでください。aws-sam-cli GitHub リポジトリに問題を作成して、AWS SAM チームにエスカレーションしてください。

6. 署名者のパブリックキーの署名を確認してください。

```
$ gpg --check-sigs FE0ADDFA

pub   4096R/FE0ADDFA 2023-05-23 [expires: 2025-05-22]
uid                  AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>
sig!3        FE0ADDFA 2023-05-23  AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>
sig!         73AD885A 2023-05-24  AWS SAM CLI Primary <aws-sam-cli-
primary@amazon.com>
```

   1 signature not checked due to a missing key が表示された場合は、前の手順を繰り返して、プライマリパブリックキーと署名者パブリックキーをキーリングにインポートします。

   プライマリパブリックキーと署名者パブリックキーの両方のキー値が一覧表示されます。

署名者パブリックキーの整合性が確認できたので、署名者パブリックキーを使用して AWS SAM CLI パッケージインストーラを確認できます。

AWS SAM CLI パッケージインストーラの整合性を確認するには

1. AWS SAM CLI パッケージ署名ファイルを取得する — 次のコマンドを使用して AWS SAM CLI パッケージインストーラの署名ファイルをダウンロードします。

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

2. 署名ファイルを確認 – ダウンロードした .sig と .zip の両方のファイルをパラメータとして gpg コマンドに渡します。以下に例を示します。

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

   出力は次の例に類似したものになります:

```
gpg: Signature made Tue 30 May 2023 10:03:57 AM UTC using RSA key ID FE0ADDFA
gpg: Good signature from "AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 37D8 BE16 0355 2DA7 BD6A  04D8 C7A0 5F43 FE0A DDFA
```

- この `WARNING: This key is not certified with a trusted signature!` メッセージは無視できます。これは、個人用の PGP キー (持っている場合) と AWS SAM CLI PGP キーの間に信頼チェーンがないために発生します。詳細については、「信用の輪 (Web of Trust)」を参照してください。
- 出力に「`BAD signature`」という句が含まれる場合、手順が正しいことを確認してください。この応答が続く場合は、aws-sam-cli GitHub リポジトリで問題を作成して AWS SAM にエスカレーションし、ダウンロードしたファイルは使用しないでください。

この `Good signature from "AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>"` メッセージは、署名が検証され、インストールを続行できることを意味します。

x86_64 - コマンドラインインストーラ

AWS SAM は GnuPG を使用して AWS SAM CLI インストーラに署名します。以下の手順で確認が実行されます。

1. プライマリパブリックキーを使用して、署名者のパブリックキーを確認します。
2. 署名者パブリックキーを使用して AWS SAM CLI パッケージインストーラを確認します。

署名者のパブリックキーの整合性を確認するには

1. プライマリパブリックキーをコピーし、ローカルマシンに `.txt` ファイルとして保存します。例えば、*`primary-public-key.txt`* と指定します。

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

mQINBGRuSzMBEADsqiwOy78w7F4+sshaMFRIwRGNRm94p5Qey2KMZBxekFtoryVD
D9jEOnvupx4tvhfBHz5EcUHCEOdl4MTqdBy6vVAshozgxVb9RE8JpECn5lw7XC69
4Y7Gy1TKKQMEWtDXElkGxIFdUWvWjSnPlzfnoXwQYGeE93CUS3h5dImP22Yk1Ct6
eGGhlcbg1X4L8EpFMj7GvcsU8f7ziVI/PyC1Xwy39Q8/I67ip5eU5ddxO/xHqrbL
YC7+8pJPbRMej2twT2LrcpWWYAbprMtRoa6WfE0/thoo3xhHpIMHdPfAA86ZNGIN
kRLjGUg7jnPTRW4Oin3pCc8nT4Tfc1QERkHm641gTC/jUvpmQsM6h/FUVP2i5iE/
JHpJcMuL2Mg6zDo3x+3gTCf+Wqz3rZzxB+wQT3yryZs6efcQy7nROiRxYBxCSXX0
2cNYzsYLb/bYaW8yqWIHD5IqKhw269gp2E5Khs60zgS3CorMb5/xHgXjUCVgcu8a
a8ncdf9fjl3WS5p0ohetPbO2ZjWv+MaqrZOmUIgKbA4RpWZ/fU97P5BW9ylwmIDB
sWy0cMxg8MlvSdLytPieogaM0qMg3u5qXRGBr6Wmevkty0qgnmpGGc5zPiUbtOE8
```

```
CnFFqyxBpj5IOnG0KZGVihvn+iRxrv6GO7WWO92+Dc6m94U0EEiBR7QiOwARAQAB
tDRBV1MgU0FNIENMSSBQcmltYXJ5IDxhd3Mtc2FtLWNsaS1wcmltYXJ5QGFtYXpv
bi5jb20+iQI/BBMBCQApBQJkbkszAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYC
AwECHgECF4AACgkQQv1fenOtiFqTuhAAzi5+ju5UVOWqHKevOJSO08T4QB8HcqAE
SVO3mY6/j29knkcL8ubZP/DbpV7QpHPI2PB5qSXsiDTP3IYPbeY78zHSDjljaIK3
njJLMScFeGPyfPpwMsuY4nzrRIgAtXShPA8N/k4ZJcafnpNqKj7QnPxiC1KaIQWm
pOtvb8msUF3/s0UTa5Ys/lNRhVC0eGg32ogXGdojZA2kHZWdm9udLo4CDrDcrQT7
NtDcJASapXSQL63XfAS3snEc4e1941YxcjfYZ33rel8K9juyDZfi1slWR/L3AviI
QFIaqSHzyOtP1oinUkoVwL8ThevKD3Ag9CZflZLzNCV7yqlF8RlhEZ4zcE/3s9El
WzCFsozb5HfE1AZonmrDh3SyOEIBMcS6vG5dWnvJrAuSYv2rX38++K5Pr/MIAfOX
DOI1rtA+XDsHNv9lSwSy0lt+iClawZANO9IXCiN1rOYcVQlwzDFwCNWDgkwdOqS0
gOA2f8NF9lE5nBbeEuYquoOl1Vy8+ICbgOFs9LoWZlnVh7/RyY6ssowiU9vGUnHI
L8f9jqRspIz/Fm3JD86ntZxLVGkeZUz62FqErdohYfkFIVcv7GONTEyrz5HLlnpv
FJ0MR0HjrMrZrnOVZnwBKhpbLocTsH+3t5It4ReYEX0f1DIOL/KRwPvjMvBVkXY5
hblRVDQoOWc=
=d9oG
-----END PGP PUBLIC KEY BLOCK-----
```

2. プライマリパブリックキーをキーリングにインポートします。

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

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

3. 署名者のパブリックキーをコピーし、ローカルマシンに .txt ファイルとして保存します。例えば、*signer-public-key.txt* と指定します。

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

mQINBGRtS20BEAC7GjaAwverrB1zNEu2q3EGI6HC37WzwL5dy30f4LirZOWS3piK
oKfTqPjXPrLCf1GL2mMqUSgSnpEbPNXuvWTW1CfSnnjwuH8ZqbvvUQyHJwQyYpKm
KMwb+8V0bzzQkMzDVqolYQCi5XyGpAuo3wroxXSzG6r/mIhbiq3aRnL+2lo4XOYk
```

```
r7q9bhBqbJhzjkm7N62PhPWmi/+/EGdEBakA1pReE+cKjP2UAp5L6CPShQl2fRKL
9BumitNfFHHs1JZgZSCCruiWny3XkUaXUEMfyoE9nNbfqNvuqV2KjWguZCXASgz2
ZSPF4DTVIBMfP+xrZGQSWdGU/67QdysDQW81TbFOjK9ZsRwwGC4kbg/K98IsCNHT
ril5RZbyr8pw3fw7jYjjI2ElAacRWp53iRzvutm5AruPpLfoKDQ/tKzBUYItBwlu
Z/diKgcqtw7xDlyqNyTN8xFPFqMO2I8IsZ2Pdl131htdFiZMiin1RQG9pV9p2vHS
eQVY2uKCnvnA6vFCQYKXP7p0IwReuPNzDvECUsidw8VTakTqZsANT/bU17e4KuKn
+JgbNrKOasJX37sDb/9ruysozLvy78ozYKJDLmC3yoRQ8DhEjviT4cnjORgNmvnZ
0a5AA/DJPQW4buRrXdxu+fITzBxQn2+GO/iDNCxtJaq5SYVBKjTmTWPUJwARAQAB
tDBBV1MgU0FNIENMSSBUZWFtIDxhd3Mtc2FtLWNsaS1zaWduZXJAYW1hem9uLmNv
bT6JAj8EEwEJACkFAmRtS20CGy8FCQPCZwAHCwkIBwMCAQYVCAIJCgsEFgIDAQIe
AQIXgAAKCRDHoF9D/grd+lE4D/4kJW65He2LNsbLTta7lcGfsEXCf4zgIvkytS7U
3R36zMD8IEyWJjlZ+aPkIP8/jFJrFl4pVHbU7vX85Iut1vV7m+8BgWt25mJhnoJ9
KPjXGra9mYP+Cj8zFAcjvtl3NBAPodyfcfCTWsU3umF9ArOFICcrGCzHX2SS7wX5
h9n0vYRZxk5Qj5FsgskKAQLq33CKFAMlaqZnL5gWRvTeycSIxsyus+stX+8YBPCO
J64f7+y+MPIP1+m2njlVXg1xLEMMVa08oWccOMiakgzDev3LCrPy+wdwdn7Ut7oA
pna3DNy9aYNd2lh6vUCJeJ+Yi1Bl2jYpzLcCLKrHUmln9/rRSz7Orbg8P181kfPu
G/M7CD5FwhxP3p4+0XoGwxQefrV2jqpSnbLae7xbYJiJAhbpjWDQhuNGUbPcDmqk
aH0Q3XU8AonJ8YqaQ/q3VZ3JBiH3TbBrOXsvd59cwxYyf83aJ/WLCb2P8y75zDad
lnOP713ThF5J/Afj9HjO9waFV0Z2W2ZZe4rU2OJTAiXEtM8xsFMrc7TCUacJtJGs
u4kdBmXREcVpSz65h9ImSy2ner9qktnVVCW4mZPj63IhB37YtoLAMyz3a3R2RFNk
viEX8foOTUg1FmwHoftxZ9P91QwLoTajkDrh26ueIe45sG6Uxua2AP4Vo37cFfCj
ryV8OokCHAQQAQkABgUCZG5MWAAKCRBC/V96c62IWmglD/9idU43kW8Zy8Af1j8l
Am3lI4d9ksOleeKRZqxo/SZ5rovF32DO2nw7XRXq1+EbhgJaI3QwwOi0U0pfAMVT
4b9TdxdH+n+tqzCHh3jZqmo9sw+c9WFXYJN1hU9bLzcHXS8hOTbyoE2EuXx56ds9
L/BWCcd+LIvapw0lggFfavVx/QF4C7nBKjnJ66+xxwfgVIKR7oGlqDiHMfp9ZWh5
HhEqZo/nrNhdY0h3sczEdqC2N6eIa8mgHffHZdKudDMXIXHbgdhW9pcZXDIktVf7
j9wehsWOyYXiRgR0dz7DI26AUG4JLh5FTtx9XuSBdEsI69Jd4dJuibmgtImzbZjn
7un8DJWIyqi7Ckk96Tr4oXB9mYAXaWlR4C9j5XJhMNZgkOycuY2DADnbGmSb+1kA
ju77H4ff84+vMDwUzUt2Wwb+GjzXu2g6Wh+bWhGSirYlel+6xYrI6beu1BDCFLq+
VZFE8WggjJHpwcL7CiqadfVIQaw4HY0jQFTSdwzPWhJvYjXFOhMkyCcjssbtmB+z
/otfgySyQqThrD48RWS5GuyqCA+pK3UNmEJ11c1AXMdTn2VWInR1NOJNALQ2du3y
q8t1vMsErVOJ7pkZ50F4ef17PE6DKrXX8ilwGFyVuX5ddyt/t9J5pC3sRwHWXVZx
GXwoX75FwIEHA3n5Q7rZ69Ea6Q==
=ZIO7
-----END PGP PUBLIC KEY BLOCK-----
```

4. 署名者のパブリックキーをキーリングにインポートします。

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

gpg: key FE0ADDFA: public key "AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>"
 imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

```
gpg: no ultimately trusted keys found
```

出力のキー値を書き留めておきます。例えば、*FE0ADDFA* と指定します。

5. キー値を使用して、署名者のパブリックキーフィンガープリントを取得して確認します。

```
$ gpg --fingerprint FE0ADDFA

pub   4096R/FE0ADDFA 2023-05-23 [expires: 2025-05-22]
      Key fingerprint = 37D8 BE16 0355 2DA7 BD6A  04D8 C7A0 5F43 FE0A DDFA
uid                  AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>
```

フィンガープリントは、次のものと一致する必要があります。

```
37D8 BE16 0355 2DA7 BD6A  04D8 C7A0 5F43 FE0A DDFA
```

フィンガープリント文字列が一致しない場合は、AWS SAM CLI インストーラを使用しないでください。aws-sam-cli GitHub リポジトリに問題を作成して、AWS SAM チームにエスカレーションしてください。

6. 署名者のパブリックキーの署名を確認してください。

```
$ gpg --check-sigs FE0ADDFA

pub   4096R/FE0ADDFA 2023-05-23 [expires: 2025-05-22]
uid                  AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>
sig!3        FE0ADDFA 2023-05-23  AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>
sig!         73AD885A 2023-05-24  AWS SAM CLI Primary <aws-sam-cli-
primary@amazon.com>
```

`1 signature not checked due to a missing key` が表示された場合は、前の手順を繰り返して、プライマリパブリックキーと署名者パブリックキーをキーリングにインポートします。

プライマリパブリックキーと署名者パブリックキーの両方のキー値が一覧表示されます。

署名者パブリックキーの整合性が確認できたので、署名者パブリックキーを使用して AWS SAM CLI パッケージインストーラを確認できます。

AWS SAM CLI パッケージインストーラの整合性を確認するには

1. AWS SAM CLI パッケージ署名ファイルを取得する — 次のコマンドを使用して AWS SAM CLI パッケージインストーラの署名ファイルをダウンロードします。

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

2. 署名ファイルを確認 – ダウンロードした .sig と .zip の両方のファイルをパラメータとして gpg コマンドに渡します。以下に例を示します。

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

出力は次の例に類似したものになります:

```
gpg: Signature made Tue 30 May 2023 10:03:57 AM UTC using RSA key ID FE0ADDFA
gpg: Good signature from "AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 37D8 BE16 0355 2DA7 BD6A  04D8 C7A0 5F43 FE0A DDFA
```

- この WARNING: This key is not certified with a trusted signature! メッセージは無視できます。これは、個人用の PGP キー (持っている場合) と AWS SAM CLI PGP キーの間に信頼チェーンがないために発生します。詳細については、「信用の輪 (Web of Trust)」を参照してください。
- 出力に「BAD signature」という句が含まれる場合、手順が正しいことを確認してください。この応答が続く場合は、aws-sam-cli GitHub リポジトリで問題を作成して AWS SAM にエスカレーションし、ダウンロードしたファイルは使用しないでください。

この Good signature from "AWS SAM CLI Team <aws-sam-cli-signer@amazon.com>" メッセージは、署名が検証され、インストールを続行できることを意味します。

macOS

GUI とコマンドラインインストーラー

AWS SAM CLI パッケージインストーラの署名ファイルの整合性は、pkgutil ツールを使用するか手動で確認できます。

pkgutil 使用して確認するには

1. ローカルマシンに次のコマンドを実行します。

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

   以下に例を示します。

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

2. 出力から Developer ID Installer: AMZN Mobile LLC の SHA256 fingerprint を見つけます。以下に例を示します。

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

3. Developer ID Installer: AMZN Mobile LLC SHA256 fingerprint は次の値と一致する必要があります。

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

フィンガープリント文字列が一致しない場合は、AWS SAM CLI インストーラを使用しないでください。aws-sam-cli GitHub リポジトリに問題を作成して、AWS SAM チームにエスカレーションしてください。フィンガープリントの文字列が一致する場合は、パッケージインストーラーを使用して先に進むことができます。

パッケージインストーラを手動で確認するには

- Apple サポートサイトで、「手動でダウンロードした Apple のソフトウェアアップデートの信頼性を検証する方法」をご覧ください。

Windows

AWS SAM CLI インストーラは、Windows オペレーティングシステム用の MSI ファイルとしてパッケージ化されています。

インストーラが整合性を確認するには

1. インストーラーを右クリックして、[プロパティ] ウィンドウを開きます。
2. デジタル署名タブを選択します。
3. 署名リストで Amazon Web Servicesを選択し、削除をクリックします。
4. すでに選択していない場合は 一般タブにアクセスし、証明書の表示 を選びます。
5. 詳細 タブを選択し、まだの場合は すべてを表示 のドロップダウンリストで選択します。
6. 拇印 フィールドが表示されるまでスクロールして、拇印 を選択します。下のウィンドウにサムプリントの値全体が表示されます。
7. サムプリントの値を次の値と一致させます。値が一致したら、インストールを続行します。そうでない場合は、 aws-sam-cli GitHub リポジトリに問題を作成して、AWS SAM チームにエスカレーションしてください。

```
d52eb68bffe6ae165b3b05c3e1f9cc66da7eeac0
```

ハッシュ値を確認

Linux

x86_64 - コマンドラインインストーラ

以下のコマンドを使用してハッシュ値を生成することによって、ダウンロードしたインストーラファイルの整合性と信頼性を検証します。

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

出力は以下のようになる必要があります。

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

SHA-256 の 64 桁のハッシュ値を、GitHub の「AWS SAM CLI リリースノート」の目的の AWS SAM CLI バージョンの値と比較します。

macOS

GUI とコマンドラインインストーラー

以下のコマンドを使用してハッシュ値を生成することによって、ダウンロードしたインストーラの整合性と信頼性を検証します。

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

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

64-character SHA-256 のハッシュ値を、「AWS SAM CLI リリースノート」の GitHub リポジトリ内の対応する値と比較します。

チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM

このチュートリアルでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) を使用して、以下を完了します。

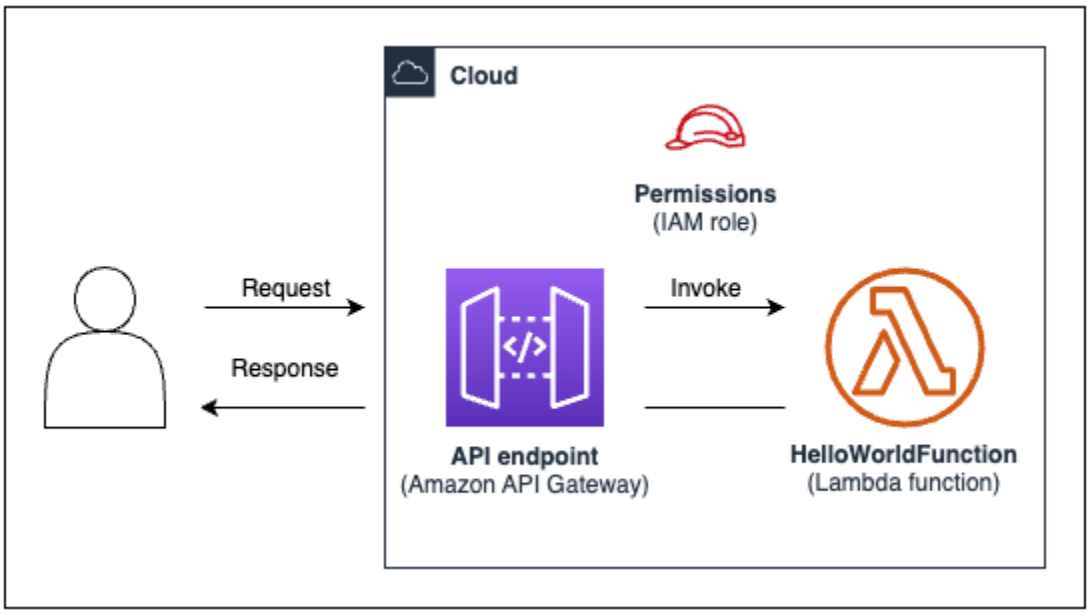
- サンプルの Hello World アプリケーションを初期化、構築、デプロイします。

- ローカルで変更を行い、 に同期します AWS CloudFormation。
- 開発ホストでローカルテストを実行します。
- AWS クラウドからサンプルアプリケーションを削除します。

サンプル Hello World アプリケーションは、基本的なAPIバックエンドを実装します。これは次のリソースで構成されます。

- Amazon API Gateway – 関数の呼び出しに使用するAPIエンドポイント。
- AWS Lambda - HTTPAPIGETリクエストを処理し、`hello world`メッセージ返す関数。
- AWS Identity and Access Management （ IAM) ロール – サービスが安全にやり取りするためのアクセス許可をプロビジョニングします。

以下の図は、このアプリケーションのコンポーネントを示しています。



トピック

- ステップ 6: アプリケーションを変更して に同期する AWS クラウド
- ステップ 7: (オプション) アプリケーションをローカルでテストする
- ステップ 8: からアプリケーションを削除する AWS クラウド
- トラブルシューティング
- 詳細

前提条件

次を完了していることを確認します。

- AWS SAM 前提条件
- AWS SAM CLI のインストール

ステップ 1: サンプルの Hello World アプリケーションを初期化する

このステップでは、 AWS SAM CLI ローカルマシンにサンプル Hello World アプリケーションプロジェクトを作成します。

サンプルの Hello World アプリケーションを初期化するには

1. コマンドラインで、選択した開始ディレクトリから次を実行します。

```
$ sam init
```

> **Note**
>
> このコマンドは、サーバーレスアプリケーションを初期化し、プロジェクトディレクトリを作成します。このディレクトリには、ファイルとフォルダがいくつか含まれています。最も重要なファイルは `template.yaml` です。これは AWS SAM テンプレートです。ご使用の Python のバージョンは、sam init コマンドで作成された `template.yaml` ファイルにリストされている Python のものと、一致する必要があります。

2. AWS SAM CLI は、新しいアプリケーションの初期化をガイドします。次を設定します。

   1. [AWS クイックスタートテンプレート] を選択して、開始テンプレートを選択します。
   2. [Hello World Example] テンプレートを選択してダウンロードします。

3. を使用する Python ランタイムとzipパッケージタイプ。
4. このチュートリアルでは、 AWS X-Ray トレースをオプトアウトします。詳細については、「 AWS X-Ray デベロッパーガイド」の「 AWS X-Rayとは」を参照してください。
5. このチュートリアルでは、Amazon CloudWatch Application Insights によるモニタリングをオプトアウトします。詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch Application Insights」を参照してください。 CloudWatch
6. このチュートリアルでは、Lambda 関数で構造化ログ記録を JSON形式で設定することをオプトアウトします。
7. アプリケーションに sam-app という名前を付けます。

を使用するには AWS SAM CLI インタラクティブフロー :

- 括弧 ([ ]) はデフォルト値を示します。回答を空白のままにすると、デフォルト値が選択されます。
- はいの場合は **y** と入力し、いいえの場合は **n** と入力します。

`sam init` インタラクティブフローの例を次に示します。

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

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

```
    13 - Machine Learning
Template: 1

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

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

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

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

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

3. AWS SAM CLI は開始テンプレートをダウンロードし、ローカルマシンにアプリケーションプロジェクトディレクトリ構造を作成します。以下は、 の例です。 AWS SAM CLI 出力：

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

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

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


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

4. コマンドラインから、新しく作成した sam-app ディレクトリに移動します。以下は、 AWS SAM CLI が作成しました。

```
$ cd sam-app

$ tree

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

6 directories, 14 files
```

強調すべきいくつかの重要なファイル:

- hello_world/app.py – Lambda 関数コードが含まれます。
- hello_world/requirements.txt – すべての Python Lambda 関数に必要な 依存関係。
- samconfig.toml – で使用されるデフォルトのパラメータを保存するアプリケーションの設定ファイル AWS SAM CLI.
- template.yaml – アプリケーションインフラストラクチャコードを含む AWS SAM テンプレート。

これで、サーバーレスアプリケーションがローカルマシン上に完成しました。

ステップ 2: アプリケーションを構築する

このステップでは、 AWS SAM CLI アプリケーションを構築し、デプロイの準備をします。構築すると、 AWS SAM CLI は.aws-samディレクトリを作成し、そこに関数の依存関係、プロジェクトコード、プロジェクトファイルを整理します。

アプリケーションを構築するには

- コマンドラインで、sam-app プロジェクトディレクトリから次を実行します。

```
$ sam build
```

> **Note**
>
> がない場合は Python ローカルマシンで、代わりに sam build --use-container コマンドを使用します。 AWS SAM CLI は を作成します。Docker 関数のランタイムと依存関係を含む コンテナ。このコマンドには が必要です Docker ローカルマシンの 。をインストールするには Docker「Docker のインストール」を参照してください。

以下は、 の例です。 AWS SAM CLI 出力：

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

Build Succeeded

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

Commands you can use next
```

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

以下は、 によって AWS SAM 作成された `.aws-sam` ディレクトリの短縮された例ですCLI。

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

強調すべきいくつかの重要なファイル:

- `build/HelloWorldFunction` – Lambda 関数のコードと依存関係が含まれます。 AWS SAM CLI は、アプリケーション内の各関数のディレクトリを作成します。
- `build/template.yaml` – デプロイ AWS CloudFormation 時に によって参照される AWS SAM テンプレートのコピーが含まれます。
- `build.toml` – によって参照されるデフォルトのパラメータ値を格納する設定ファイル AWS SAM CLI アプリケーションをビルドおよびデプロイするとき。

これで、アプリケーションを AWS クラウドにデプロイする準備が整いました。

ステップ 3: アプリケーションを にデプロイする AWS クラウド

> **Note**
>
> このステップでは、 AWS 認証情報の設定が必要です。詳細については、「AWS SAM 前提条件」の ステップ 5: AWS CLI を使用して AWS 認証情報を設定する を参照してください。

このステップでは、 AWS SAM CLI アプリケーションを にデプロイします AWS クラウド。 AWS SAM CLI は以下を実行します。

- デプロイのためのアプリケーション設定を通じてユーザーをガイドします。
- アプリケーションファイルを Amazon Simple Storage Service (Amazon S3) にアップロードします。
- AWS SAM テンプレートを AWS CloudFormation テンプレートに変換します。次に、テンプレートを AWS CloudFormation サービスにアップロードしてリソースをプロビジョニングします AWS 。

アプリケーションをデプロイするには

1. コマンドラインで、sam-app プロジェクトディレクトリから次を実行します。

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

2. に従う AWS SAM CLI インタラクティブフローを使用してアプリケーション設定を構成します。次を設定します。

   1. AWS CloudFormation スタック名 – スタックは、単一のユニットとして管理できる AWS リソースのコレクションです。詳細については、「AWS CloudFormation ユーザーガイド」の「スタックの操作」を参照してください。
   2. AWS CloudFormation スタックをデプロイAWS リージョンする 。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation エンドポイント」を参照してください。
   3. このチュートリアルでは、デプロイ前の変更の確認をオプトアウトします。
   4. IAM ロールの作成を許可する – これにより、 は API Gateway リソースと Lambda 関数リソースがやり取りするために必要なIAMロール AWS SAM を作成できます。
   5. このチュートリアルでは、ロールバックの無効化をオプトアウトします。
   6. HelloWorldFunction 承認を定義せずに許可する – APIゲートウェイエンドポイントが承認なしでパブリックアクセス可能に設定されたため、このメッセージが表示されます。これは Hello World アプリケーションの意図した設定であるため、 AWS SAM CLI を続行します。認可の設定の詳細については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。
   7. 引数を設定ファイルに保存 – これにより、アプリケーションの samconfig.toml ファイルがデプロイ設定で更新されます。
   8. デフォルトの設定ファイル名を選択します。
   9. デフォルトの設定環境を選択します。

sam deploy --guided インタラクティブフローの出力例を次に示します。

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

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

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

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

3. AWS SAM CLI は、次の操作を実行してアプリケーションをデプロイします。

   - AWS SAM CLI は Amazon S3 バケットを作成し、.aws-samディレクトリをアップロードします。
   - AWS SAM CLI は AWS SAM テンプレートを に変換 AWS CloudFormation し、 AWS CloudFormation サービスにアップロードします。
   - AWS CloudFormation は リソースをプロビジョニングします。

   デプロイ中、 AWS SAM CLI に進行状況が表示されます。以下は、その出力例です。

```
Looking for resources needed for deployment:
```

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

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

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

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

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

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

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


Waiting for changeset to be created..

CloudFormation stack changeset
-----------------------------------------------------------------------------------------
```

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


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


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

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

CloudFormation outputs from deployed stack
-------------------------------------------------------------------------------------
```

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

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

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


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

これで、アプリケーションがデプロイされ、 で実行されます AWS クラウド。

ステップ 4: アプリケーションを実行する

このステップでは、APIエンドポイントにGETリクエストを送信し、Lambda 関数の出力を確認します。

API エンドポイント値を取得するには

1. に表示される情報から AWS SAM CLI 前のステップで、 Outputsセクションを見つけます。このセクションでは、HelloWorldApiリソースを見つけてHTTPエンドポイント値を見つけます。以下は、その出力例です。

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

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

2. あるいは、sam list endpoints --output json コマンドを使用してこの情報を取得することもできます。出力例を次に示します。

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

関数を呼び出すには

- ブラウザまたはコマンドラインを使用して、APIエンドポイントにGETリクエストを送信します。curl コマンドを使用した例を次に示します。

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

ステップ 5: で 関数を操作する AWS クラウド

このステップでは、 AWS SAM CLI で Lambda 関数を呼び出す AWS クラウド。

クラウド内の Lambda 関数を呼び出す

1. 前のステップから、関数の LogicalResourceId を書き留めておきます。これは、HelloWorldFunction になります。
2. コマンドラインで、sam-app プロジェクトディレクトリから次を実行します。

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

3. AWS SAM CLI はクラウドで関数を呼び出し、レスポンスを返します。以下は、その出力例です。

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

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

ステップ 6: アプリケーションを変更して に同期する AWS クラウド

このステップでは、 AWS SAM CLI sam sync --watch ローカルの変更を に同期する コマンド AWS クラウド。

sam sync を使用するには

1. コマンドラインで、sam-app プロジェクトディレクトリから次を実行します。

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

2. AWS SAM CLI は、開発スタックを同期していることを確認するよう求めます。sam sync --watch コマンドはローカルの変更を AWS クラウド リアルタイムで に自動的にデプロイするため、開発環境にのみ推奨されます。

AWS SAM CLI は、ローカルの変更のモニタリングを開始する前に、初期デプロイを実行します。以下は、その出力例です。

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

Confirm that you are synchronizing a development stack.

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

Build Succeeded

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


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

```
Initiating deployment
=====================


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

CloudFormation events from stack operations (refresh every 0.5 seconds)
---------------------------------------------------------------------------------------------
ResourceStatus                      ResourceType
 LogicalResourceId                   ResourceStatusReason
---------------------------------------------------------------------------------------------
UPDATE_IN_PROGRESS                  AWS::CloudFormation::Stack          sam-app
                        Transformation succeeded
CREATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                        ack
CREATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   Resource creation Initiated
                                                                        ack
CREATE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                        ack
UPDATE_IN_PROGRESS                  AWS::Lambda::Function
 HelloWorldFunction                  -
UPDATE_COMPLETE                     AWS::Lambda::Function
 HelloWorldFunction                  -
UPDATE_COMPLETE_CLEANUP_IN_PROGRE   AWS::CloudFormation::Stack          sam-app
                          -
SS
UPDATE_COMPLETE                     AWS::CloudFormation::Stack          sam-app
                          -
---------------------------------------------------------------------------------------------

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

Key                 HelloWorldApi
```

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

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


Stack update succeeded. Sync infra completed.

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

次に、Lambda 関数のコードを変更します。 AWS SAM CLI は、この変更を自動的に検出し、アプリケーションを に同期します AWS クラウド。

アプリケーションを変更して同期するには

1. IDE 選択した で、 sam-app/hello_world/app.py ファイルを開きます。
2. message を変更してファイルを保存します。以下に例を示します。

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

3. AWS SAM CLI は変更を検出し、アプリケーションを に同期します AWS クラウド。以下は、その出力例です。

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

4. 変更を確認するには、APIエンドポイントGETにリクエストを再度送信します。

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

ステップ 7: (オプション) アプリケーションをローカルでテストする

> **Note**
>
> この手順は省略可能です。

> **Important**
>
> このステップでは、 が必要です。Docker ローカルマシンの 。必要なもの Docker をインストールし、 を使用するように設定 AWS SAM CLI ローカルテスト用。詳細については、「Docker のインストール」を参照してください。

このステップでは、 AWS SAM CLI sam local アプリケーションをローカルでテストするための コマンド。これを実現するには、 AWS SAM CLI は、 を使用してローカル環境を作成します。Docker。このローカル環境は、Lambda 関数のクラウドベースの実行環境をエミュレートします。

以下の作業を実行します。

1. Lambda 関数のローカル環境を作成し、呼び出します。
2. HTTP API エンドポイントをローカルでホストし、それを使用して Lambda 関数を呼び出します。

Lambda 関数をローカルで呼び出すには

1. コマンドラインで、sam-app プロジェクトディレクトリから次を実行します。

```
$ sam local invoke
```

2. AWS SAM CLI はローカル を作成します。Docker コンテナと が関数を呼び出します。以下は、その出力例です。

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

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

をAPIローカルでホストするには

1. コマンドラインで、sam-app プロジェクトディレクトリから次を実行します。

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

2. AWS SAM CLI はローカル を作成します。Docker Lambda 関数の コンテナと は、APIエンドポイントをシミュレートするローカルHTTPサーバーを作成します。以下は、その出力例です。

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

Mounting /Users/.../Demo/sam-tutorial1/sam-app/.aws-sam/build/HelloWorldFunction
 as /var/task:ro,delegated inside runtime container
```

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

3. ブラウザまたはコマンドラインを使用して、ローカルAPIエンドポイントにGETリクエストを送信します。curl コマンドを使用した例を次に示します。

```
$ curl http://127.0.0.1:3000/hello
{"message": "hello world"}
```

ステップ 8: からアプリケーションを削除する AWS クラウド

このステップでは、 AWS SAM CLI sam delete からアプリケーションを削除する コマンド AWS クラウド。

からアプリケーションを削除するには AWS クラウド

1. コマンドラインで、sam-app プロジェクトディレクトリから次を実行します。

```
$ sam delete
```

2. AWS SAM CLI から確認を求められます。次に、アプリケーションの Amazon S3 バケットと AWS CloudFormation スタックが削除されます。出力例を次に示します。

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

```
    - Deleting S3 object with key sam-app/da3c598813f1c2151579b73ad788cac8
    - Deleting S3 object with key sam-app/f798cdd93cee188a71d120f14a035b11
    - Deleting Cloudformation stack sam-app

Deleted successfully
```

トラブルシューティング

のトラブルシューティングを行うには AWS SAM CLI「AWS SAMCLI トラブルシューティング」を参照してください。

詳細

詳細については AWS SAM、以下のリソースを参照してください。

- AWS SAM コンプリートワークショップ — AWS SAM の主な機能を学ぶためのワークショップ。
- を使用したセッション SAM – AWS サーバーレス開発者アドボケートチームが作成した動画シリーズ AWS SAM。
- Serverless Land – AWS サーバーレスに関する最新情報、ブログ、動画、コード、学習リソースをまとめたサイト。

AWS Serverless Application Model (AWS SAM) を使用する方法

アプリケーションの開発に使用する主なツールは、AWS SAM CLI および AWS SAM テンプレートと AWS SAM プロジェクト (アプリケーションプロジェクトディレクトリ) です。これらのツールは、次のために使用します。

1. アプリケーションを開発する (これには、アプリケーションの初期化、リソースの定義、アプリケーションの構築が含まれます)。
2. アプリケーションをテストする.
3. アプリケーションをデバッグする.
4. アプリケーションとリソースをデプロイする.
5. アプリケーションをモニタリングする.

sam init コマンドを実行して後続のワークフローを完了すると、AWS SAM により AWS SAM プロジェクトが作成されます。サーバーレスアプリケーションを定義するには、AWS SAM プロジェクトにコードを追加します。AWS SAM プロジェクトは一連のファイルとフォルダで構成されますが、その中で最も重要なファイルは AWS SAM テンプレート (名前は `template.yaml`) です。このテンプレートでは、リソース、イベントソースマッピング、およびサーバーレスアプリケーションを定義するその他のプロパティを表すコードを記述します。

AWS SAM CLI には、AWS SAM プロジェクトで使用するコマンドのリポジトリが含まれています。より具体的に AWS SAM CLI は、AWS SAM プロジェクトの構築、変換、デプロイ、デバッグ、パッケージ化、初期化、同期に使用するものです。つまり、AWS SAM プロジェクトをサーバーレスアプリケーションに変換するために使用するものです。

トピック

- AWS SAMCLI
- AWS SAM プロジェクトと AWS SAM テンプレート

AWS SAMCLI

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) は、AWS SAM アプリケーションのプロジェクトディレクトリでコマンドを実行し、最終的にサーバーレス

アプリケーションに変換するために使用するツールです。具体的には、AWS SAM CLI を使用すると、AWS SAM アプリケーションのプロジェクトディレクトリを構築、変換、デプロイ、デバッグ、パッケージ化、初期化、同期することが可能です。

AWS SAM CLI と AWS SAM テンプレートには、サーバーレスアプリケーションを構築および実行するための、サポートされたサードパーティー統合が同梱されています。

トピック

AWS SAM CLI コマンドを文書化する方法

AWS SAM CLI コマンドは、次の形式で文書化されています。

- プロンプト – Linux プロンプトはデフォルトで文書化され、($) として表示されます。Windows 固有のコマンドの場合、(>) がプロンプトとして使用されます。コマンドを入力した場合はプロンプトを含めないでください。
- ディレクトリ – 特定のディレクトリからコマンドを実行する必要がある場合は、プロンプト記号の前にディレクトリ名が表示されます。
- ユーザー入力 – コマンドラインに入力するコマンドテキストは、**`user input`** としてフォーマットされます。
- 置換可能なテキスト – ファイル名やパラメータなどの可変テキストは、*`replaceable text`* としてフォーマットされます。複数行のコマンドまたは特定のキーボード入力が必要なコマンドの場合、キーボード入力も置き換え可能なテキストとして表示できます。例えば *`ENTER`* のようになります。
- 出力 – コマンドに対するレスポンスとして返される出力は、`computer output` としてフォーマットされます。

次の `sam deploy` コマンドと出力は例です。

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

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

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

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

1. `sam deploy --guided --template template.yaml` は、コマンドラインで入力するコマンドです。
2. **`sam deploy --guided --template`** はそのまま指定する必要があります。
3. *`template.yaml`* は、特定のファイル名に置き換えることができます。
4. 出力は `Configuring SAM deploy` から始まります。
5. 出力では、*`ENTER`* と *`y`* は、指定した置換可能な値を示します。

AWS SAM CLI の設定

AWS SAM の利点の 1 つは、反復タスクを削除することで開発者の時間を最適化することです。AWS SAM CLI には、この目的のために `samconfig` という名前の設定ファイルが含まれています。デフォルトでは AWS SAM CLI に対する設定は必要ありませんが、ご自分の設定ファイルを更新し、その中でカスタマイズしたパラメータを AWS SAM が参照できるようにすることで、より少ないパラメータでのコマンドの実行が可能になります。次の表にある例では、コマンドを最適化する方法を示しています。

元	**`samconfig`** を使用した最適化
sam build --cached --parallel --use-containers	sam build

元	**samconfig** を使用した最適化
sam local invoke --env-vars locals.json	sam local invoke
sam local start-api --env-vars locals.json --warm-containers EAGER	sam local start-api

AWS SAM CLI には、デベロッパーがサーバーレスアプリケーションを作成、開発、デプロイするのに役立つ、一連のコマンドが用意されています。これらの各コマンドには、オプションのフラグを使用して、アプリケーションとデベロッパーの好みに基づいた設定が可能です。詳細については、GitHub の AWS SAM CLI のコンテンツを参照してください。

このセクションのトピックでは、作成した AWS SAM CLI 設定ファイル のデフォルト設定をカスタマイズして、サーバーレスアプリケーションの開発時間を最適化する方法について説明しています。

トピック

- 設定ファイルを作成する方法 (samconfig ファイル)
- プロジェクト設定を構成する
- 認証情報と基本設定を構成する

設定ファイルを作成する方法 (**samconfig** ファイル)

AWS SAM CLI 設定ファイル (ファイル名 `samconfig`) はテキストファイルであり、通常は TOML 構造を使用しますが YAML でも使用できます。AWS クイックスタートテンプレートを使用する場合、sam init コマンドの実行時にこのファイルが作成されます。sam deploy -\-guided コマンドを使用してアプリケーションをデプロイする際、このファイルを更新することができます。

デプロイの完了後、デフォルト値を使用しているのであれば、`samconfig` ファイルには `default` という名前のプロファイルが含まれています。deploy コマンドを再実行すると、AWS SAM はこのプロファイルから、保存された設定を適用します。

`samconfig` ファイルの利点は、デプロイコマンドに加えて使用可能な他のコマンドの設定を、AWS SAM が保存することです。新しいデプロイで作成されたこれらの値以外にも、ユーザーが設定できるいくつかの属性が `samconfig` ファイル内にあり、これにより AWS SAM CLI を使用して開発者ワークフローの他の側面を簡素化できます。

プロジェクト設定を構成する

AWS SAM CLI で使用する設定ファイルで、AWS SAM CLI コマンドのパラメータ値など、プロジェクト固有の設定を指定できます。この設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

設定ファイルの使用

設定ファイルは、環境、コマンド、およびパラメータ値で構成されます。詳細については、「設定ファイルの基本」を参照してください。

新しい環境を設定するには

1. 設定ファイルに新しい環境を指定します。

   以下に、新しい prod 環境を指定する例を示します。

   TOML

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

   YAML

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

2. 設定ファイルの parameters セクションで、パラメーター値をキーと値のペアとして指定します。

   以下に、prod 環境のアプリケーションのスタック名を指定する例を示します。

   TOML

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

   YAML

   ```
   prod:
     global:
   ```

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

3. --config-env オプションを使用して、使用する環境を指定します。

   以下に例を示します。

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

パラメータ値を設定するには

1. パラメータ値を設定したい AWS SAM CLI コマンドを指定します。すべての AWS SAM CLI コマンドのパラメータ値を設定するには、global 識別子を使用します。

   以下に、default 環境の sam deploy コマンドにパラメーター値を指定する例を示します。

   TOML

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

   YAML

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

   以下に、default 環境のすべての AWS SAM CLI コマンドにパラメータ値を指定する例を示します。

   TOML

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

   YAML

```
default:
```

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

2. AWS SAM CLI のインタラクティブフローを使用して、パラメータ値を指定することや、設定ファイルを変更することもできます。

   `sam deploy --guided` インタラクティブフローの例を次に示します。

```
$ sam deploy --guided

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

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

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

詳細については、「設定ファイルの作成と変更」を参照してください。

例

基本的な TOML の例

以下は `samconfig.toml` 設定ファイルの例です。

```
...
version = 0.1

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

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

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

[default.sync.parameters]
watch = true

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

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

基本的な YAML の例

以下は samconfig.yaml 設定ファイルの例です。

```
version 0.1
default:
  global:
    parameters:
      stack_name: sam-app
  build:
    parameters:
      cached: true
      parallel: true
  deploy:
    parameters:
```

```
      capabilities: CAPABILITY_IAM
      confirm_changeset: true
      resolve_s3: true
  sync:
    parameters:
      watch: true
  local_start_api:
    parameters:
      warm_containers: EAGER
prod:
  sync:
    parameters:
      watch: false
```

認証情報と基本設定を構成する

AWS Command Line Interface (AWS CLI) を使用して、AWS 認証情報、デフォルトのリージョン名、デフォルトの出力形式などの基本設定を構成します。構成が完了すると、これらの設定を AWS SAM CLI で使用できるようになります。詳細については、「AWS Command Line Interface ユーザーガイド」の次を参照してください:

- 設定の基本
- 設定ファイルと認証情報ファイルの設定
- AWS CLI の名前付きプロファイル
- IAM Identity Center が有効な名前付きプロファイルの使用

簡単な設定手順については、「ステップ 5: AWS CLI を使用して AWS 認証情報を設定する」を参照してください。

AWS SAM CLI コアコマンド

AWS SAM CLI には、サーバーレスアプリケーションの作成、構築、テスト、デプロイ、同期に使用する、いくつかの基本的なコマンドが備わっています。次の表に、これらのコマンドの一覧と、それぞれの詳細情報のリンク先を示します。

AWS SAM CLI コマンドの完全なリストについては、「AWS SAM CLI コマンドリファレンス」を参照してください。

Command	その内容	関連トピック
sam build	ローカルテストや AWS クラウドへのデプロイなど、開発者ワークフローの後続のステップに備えてアプリケーションを準備します。	• AWS SAM を使用した構築の概要 • sam build
sam deploy	AWS CloudFormation を使用してアプリケーションを AWS クラウドにデプロイします。	• でのデプロイの概要 AWS SAM • sam deploy
sam init	新しいサーバーレス アプリケーションを作成および初期化するためのオプションを提供します。	• AWS SAM でアプリケーションを作成する • sam init
sam local	サーバーレスアプリケーションをローカルでテストするためのサブコマンドを提供します。	• sam local コマンドを使用したテストの概要 • sam local generate-event • sam local invoke • sam local start-api • sam local start-lambda
sam remote invoke	サポートされている AWS リソースを、AWS クラウド内で操作するための方法を提供します。	• を使用したクラウドでのテストの概要 sam remote invoke • sam remote invoke
sam remote test-event	AWS Lambda 関数の共有可能なテストイベントにアクセスし管理するための方法を提供します。	• sam remote test-event を使用したクラウドテストの概要 • sam remote test-event
sam sync	ローカルアプリケーションの変更を AWS クラウドにすば	• の使用の概要 sam sync 同期する AWS クラウド

Command	その内容	関連トピック
	やく同期するためのオプションを提供します。	• sam sync

AWS SAM プロジェクトと AWS SAM テンプレート

sam init コマンドを実行して後続のワークフローを完了すると、 は、 AWS SAM プロジェクトであるアプリケーションプロジェクトディレクトリ AWS SAM を作成します。サーバーレスアプリケーションを定義するには、 AWS SAM プロジェクトにコードを追加します。 AWS SAM プロジェクトは一連のファイルとフォルダで構成されていますが、主に使用するファイルは AWS SAM テンプレート (という名前) ですtemplate.yaml。このテンプレートでは、リソース、イベントソースマッピング、およびサーバーレスアプリケーションを定義するその他のプロパティを表すコードを記述します。

> **Note**
>
> AWS SAM テンプレートの重要な要素は、 AWS SAM テンプレート仕様です。この仕様では、 と比較して AWS CloudFormation、使用するコード行の数を減らして、サーバーレスアプリケーションのリソース、イベントソースマッピング、アクセス許可、APIs、およびその他のプロパティを定義できる簡単な構文を提供します。

このセクションでは、 AWS SAM テンプレートのセクションを使用して、リソースタイプ、リソースプロパティ、データタイプ、リソース属性、組み込み関数、APIゲートウェイ拡張機能を定義する方法について説明します。

AWS SAM テンプレートは AWS CloudFormation テンプレートの拡張機能であり、 よりもコード行数が少ない短い構文を使用する一意の構文タイプがあります AWS CloudFormation。これにより、サーバーレスアプリケーションを構築する際に開発をスピードアップできます。詳細については、「AWS SAM リソースとプロパティ」を参照してください。 AWS CloudFormation テンプレートの完全なリファレンスについては、「 ユーザーガイド」のAWS CloudFormation 「テンプレートリファレンス」を参照してください。 AWS CloudFormation

開発時には、アプリケーションコードを個別のファイルに分割すると、アプリケーションをより適切に整理および管理できる場合がよくあります。この基本的な例は、 AWS SAM テンプレートにこのコードを含めるのではなく、 AWS Lambda 関数コードに別のファイルを使用することです。こ

れを行うには、Lambda 関数コードをプロジェクトのサブディレクトリに整理し、 AWS Serverless Application Model（ AWS SAM) テンプレート内のローカルパスを参照します。

トピック

AWS SAM テンプレートの構造分析

AWS SAM テンプレートファイルは、AWS CloudFormation ユーザーガイドの「Template anatomy」で説明されている AWS CloudFormation テンプレートファイルの形式に追随しています。以下は、AWS SAM テンプレートファイルと AWS CloudFormation テンプレートファイルの主な違いです。

- 変換の宣言。`Transform: AWS::Serverless-2016-10-31` 宣言は AWS SAM テンプレートファイルに必須です。この宣言は、AWS CloudFormation テンプレートファイルを AWS SAM テンプレートファイルとして識別します。Transform の詳細については、AWS CloudFormation ユーザーガイドの「Transform」を参照してください。
- Globals セクション。`Globals` セクションは、AWS SAM に固有のセクションです。これは、すべてのサーバーレス関数と API に共通するプロパティを定義します。`Globals` セクションで定義されているプロパティは、すべての `AWS::Serverless::Function`、`AWS::Serverless::Api`、および `AWS::Serverless::SimpleTable` リソースによって継承されます。このセクションの詳細については、「AWS SAM テンプレートの Globals セクション」を参照してください。
- Resources セクション。AWS SAM テンプレートの `Resources` セクションには、AWS CloudFormation リソースと AWS SAM リソースの組み合わせを含めることができます。AWS CloudFormation リソースの詳細については、AWS CloudFormation ユーザーガイドの「AWS resource and property types reference」を参照してください。AWS SAM リソースの詳細については、「AWS SAM リソースとプロパティ」を参照してください。

AWS SAM テンプレートファイルにあるその他すべてのセクションは、同じ名前の AWS CloudFormation テンプレートファイルのセクションに対応しています。

YAML

以下の例は、YAML 形式のテンプレートフラグメントを示しています。

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

Globals:
  set of globals

Description:
  String

Metadata:
  template metadata

Parameters:
  set of parameters

Mappings:
  set of mappings

Conditions:
  set of conditions

Resources:
  set of resources

Outputs:
  set of outputs
```

テンプレートセクション

AWS SAM テンプレートには、いくつかの重要なセクションが含まれる場合があります。必須のセクションは、`Transform` と `Resources` のみです。

テンプレートセクションは任意の順序で含めることができますが、ただし次の例に示すように、言語拡張機能を使用する場合には、サーバーレス変換の前 (`AWS::Serverless-2016-10-31` の前) に `AWS::LanguageExtensions` を追加する必要があります。

```
Transform:
```

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

テンプレートを作成する際には、以下の一覧が示す論理的な順序の使用が役に立つ場合があります。これは、1 つのセクションの値が前のセクションの値を参照している可能性があるためです。

変換 (必須)

AWS SAM テンプレートの場合は、`AWS::Serverless-2016-10-31` の値を使用してこのセクションを含める必要があります。

追加の transform はオプションです。Transform の詳細については、AWS CloudFormation ユーザーガイドの「Transform」を参照してください。

グローバル (オプション)

すべてのサーバーレス関数、API、および単純テーブルに共通のプロパティです。`Globals` セクションで定義されているプロパティは、すべての `AWS::Serverless::Function`、`AWS::Serverless::Api`、および `AWS::Serverless::SimpleTable` リソースによって継承されます。

このセクションは AWS SAM に固有です。AWS CloudFormation テンプレートに対応するセクションはありません。

Description (オプション)

テンプレートを説明するテキスト文字列です。

このセクションは、AWS CloudFormation テンプレートの `Description` セクションに直接対応します。

メタデータ (オプション)

テンプレートに関する追加情報を提供するオブジェクトです。

このセクションは、AWS CloudFormation テンプレートの `Metadata` セクションに直接対応します。

パラメータ (任意)

実行時 (スタックを作成または更新するとき) にテンプレートに渡す値です。テンプレートの `Resources` および `Outputs` セクションからのパラメータを参照できます。`Parameters` セクションで宣言されるオブジェクトによって、sam deploy --guided コマンドがユーザーに追加のプロンプトを表示するようになります。

`sam deploy` コマンドの `--parameter-overrides` パラメータ (および設定ファイルのエントリ) を使用することで渡される値で、AWS SAM テンプレートファイルよりも優先されます。`sam deploy` コマンドの詳細については、AWS SAM CLI コマンドリファレンスの「sam deploy」を参照してください。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

マッピング (任意)

キーと関連する値のマッピングで、条件パラメータ値の指定に使用でき、ルックアップテーブルに似ています。`Resources` セクションと `Outputs` セクションで `Fn::FindInMap` 組み込み関数を使用することによって、キーを対応する値と一致させることができます。

このセクションは、AWS CloudFormation テンプレートの `Mappings` セクションに直接対応します。

条件 (オプション)

スタックの作成中または更新中に、特定のリソースが作成されるかどうか、または特定のリソースプロパティに値が割り当てられるかどうかを制御する条件です。例えば、スタックが実稼働用であるかテスト環境用であるかに依存するリソースを、条件付きで作成できます。

このセクションは、AWS CloudFormation テンプレートの `Conditions` セクションに直接対応します。

リソース (必須)

Amazon Elastic Compute Cloud (Amazon EC2) インスタンスや Amazon Simple Storage Service (Amazon S3) バケットなどのスタックリソースとそれらのプロパティです。テンプレートの `Resources` と `Outputs` セクションのリソースを参照できます。

このセクションは、AWS CloudFormation テンプレートの `Resources` セクションに似ています。AWS SAM テンプレートのこのセクションには、AWS SAM リソースに加えて AWS CloudFormation リソースを含めることができます。

出力 (任意)

スタックのプロパティを表示するたびに返される値です。例えば、S3 バケット名の出力を宣言してから、`aws cloudformation describe-stacks` AWS Command Line Interface (AWS CLI) コマンドを呼び出して名前を表示することができます。

このセクションは、AWS CloudFormation テンプレートの `Outputs` セクションに直接対応します。

次のステップ

AWS SAM テンプレートファイルが含まれるサンプルサーバーレスアプリケーションをダウンロードしてデプロイするには、「AWS SAM の開始方法」を参照して、「チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM」の手順に従ってください。

AWS SAM テンプレートの Globals セクション

AWS SAM テンプレートで宣言するリソースに共通の設定がある場合があります。例えば、アプリケーションに 同一の Runtime、Memory、VPCConfig、Environment、および Cors 設定を持つ複数の AWS::Serverless::Function がある場合があります。すべてのリソースにこの情報を複製する代わりに、Globals セクションでそれらを一度だけ宣言し、リソースに継承させることができます。

Globals セクションでは、次の AWS SAM リソースタイプがサポートされています。

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

例:

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

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

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

この例では、 HelloWorldFunctionと の両方が に「nodejs12.xRuntime」、 に「180」秒Timeout、 に「index.handlerThumbnailFunction」を使用しますHandler。 は、継承された TABLE_ に加えて MESSAGE環境変数HelloWorldFunctionを追加しますNAME。 はすべてのGlobalsプロパティをThumbnailFunction継承し、APIイベントソースを追加します。

サポートされるリソースとプロパティ

AWS SAM では、以下のリソースとプロパティがサポートされています。

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

  Function:
    Architectures:
```

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

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

  SimpleTable:
    SSESpecification:

  StateMachine:
```

```
    PropagateTags:
```

> **Note**
>
> 上記のリストに含まれていないリソースとプロパティはサポートされません。それらをサポートしない理由には、1) セキュリティ問題を発生させる可能性がある、または 2) テンプレートを理解しにくくするなどがあります。

暗黙的 APIs

AWS SAM `Events`セクションAPIで を宣言すると、 は暗黙APIs的を作成します。を使用してGlobals、暗黙的な のすべてのプロパティを上書きできますAPIs。

上書き可能なプロパティ

リソースは、`Globals` セクションで宣言するプロパティを上書きできます。例えば、環境変数マップに新しい変数を追加する、またはグローバルに宣言された変数を上書きすることができます。ただし、リソースは `Globals` セクションで指定されたプロパティを削除できません。

一般に、`Globals` セクションはすべてのリソースが共有するプロパティを宣言します。リソースには、グローバルに宣言されたプロパティに新しい値を提供できるものもありますが、それらを完全に削除することはできません。一部のリソースがあるプロパティを使用するが、他のリソースは使用しないという場合は、それを `Globals` セクションで宣言しない必要があります。

以下のセクションでは、さまざまなデータ型に対して上書きがどのように機能するかについて説明します。

プリミティブデータ型が置き換えられる

プリミティブデータ型には、文字列、数値、ブール値などがあります。

`Globals` セクションの値が `Resources` セクションで指定された値に置き換えられます。

例:

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

Resources:
```

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

MyFunction の Runtime は python3.9 に設定されます。

マップが統合される

マップは、ディクショナリまたはキーバリューペアのコレクションとしても知られています。

Resources セクションのマップエントリは、グローバルマップエントリと統合されます。重複がある場合は、Globals セクションのエントリが Resource セクションのエントリで上書きされます。

例:

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

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

MyFunction の環境変数は、以下のように設定されます。

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

リストが付加される

リストは配列としても知られています。

Globals セクションのリストエントリは、Resources セクションのリストの先頭に付加されます。

例:

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

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

MyFunction の VpcConfig の SecurityGroupIds は、以下のように設定されます。

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

AWS SAM リソースとプロパティ

このセクションでは、AWS SAM に固有のリソースタイプとプロパティタイプについて説明します。これらのリソースとプロパティは、AWS SAM の省略構文を使用して定義します。AWS SAM は、AWS CloudFormation のリソースとプロパティタイプもサポートします。AWS CloudFormation および AWS SAM によってサポートされるすべての AWS リソースとプロパティの参照情報については、AWS CloudFormation ユーザーガイドの「AWS リソースおよびプロパティタイプのリファレンス」を参照してください。

トピック

- AWS::Serverless::Api
- AWS::Serverless::Application
- AWS::Serverless::Connector
- AWS::Serverless::Function

- AWS::Serverless::GraphQLApi
- AWS::Serverless::HttpApi
- AWS::Serverless::LayerVersion
- AWS::Serverless::SimpleTable
- AWS::Serverless::StateMachine

AWS::Serverless::Api

HTTPS エンドポイント経由で呼び出すことができる Amazon API Gateway リソースとメソッドのコレクションを作成します。

AWS::Serverless::Api リソースを AWS サーバーレスアプリケーション定義テンプレートに明示的に追加する必要はありません。このタイプのリソースは、AWS::Serverless::Function リソースを参照しないテンプレートで定義された AWS::Serverless::Api リソースに定義される Api イベントの和集合から暗黙的に作成されます。

AWS::Serverless::Api リソースは、OpenApi を使用して API を定義および文書化するために使用する必要があります。これは、基盤となる Amazon API Gateway リソースを設定する機能をより多く提供します。

API Gateway リソースへのアクセスを制御するオーソライザーがアタッチされていることを確認するため、AWS CloudFormation フックまたは IAM ポリシーを使用することをお勧めします。

AWS CloudFormation フックの使用の詳細については、「AWS CloudFormation CLI ユーザーガイド」の「Registering hooks」(フックの登録) および GitHub リポジトリの apigw-enforce-authorizer を参照してください。

IAM ポリシーの使用の詳細については、「API ゲートウェイデベロッパーガイド」の「API ルートに認可を要求する」を参照してください。

> **Note**
>
> AWS CloudFormation にデプロイすると、AWS SAM は、AWS SAM リソースを AWS CloudFormation リソースに変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Type: AWS::Serverless::Api
Properties:
  AccessLogSetting: AccessLogSetting
  AlwaysDeploy: Boolean
  ApiKeySourceType: String
  Auth: ApiAuth
  BinaryMediaTypes: List
  CacheClusterEnabled: Boolean
  CacheClusterSize: String
  CanarySetting: CanarySetting
  Cors: String | CorsConfiguration
  DefinitionBody: JSON
  DefinitionUri: String | ApiDefinition
  Description: String
  DisableExecuteApiEndpoint: Boolean
  Domain: DomainConfiguration
  EndpointConfiguration: EndpointConfiguration
  FailOnWarnings: Boolean
  GatewayResponses: Map
  MergeDefinitions: Boolean
  MethodSettings: MethodSettings
  MinimumCompressionSize: Integer
  Mode: String
  Models: Map
  Name: String
  OpenApiVersion: String
  PropagateTags: Boolean
  StageName: String
  Tags: Map
  TracingEnabled: Boolean
  Variables: Map
```

プロパティ

`AccessLogSetting`

ステージのアクセスログ設定を行います。

タイプ: AccessLogSetting

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::Stage` リソースの `AccessLogSetting` プロパティに直接渡されます。

`AlwaysDeploy`

API への変更が検出されない場合でも、常に API をデプロイします。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ApiKeySourceType`

使用量プランに沿ってリクエストを計測するための API キーのソース。有効な値は、`HEADER` および `AUTHORIZER` です。

タイプ : 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::RestApi` リソースの `ApiKeySourceType` プロパティに直接渡されます。

`Auth`

API Gateway API へのアクセスを制御するための認可を設定します。

AWS SAM を使用したアクセス権の設定に関する詳細については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。

タイプ: ApiAuth

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`BinaryMediaTypes`

API が返すことができる MIME タイプのリストです。これは、API のバイナリサポートを有効化するために使用します。MIME タイプでは「/」の代わりに「~1」を使用してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::RestApi` リソースの `BinaryMediaTypes` プロパティに似ています。BinaryMediaTypes のリストは、AWS CloudFormation リソースと OpenAPI ドキュメントの両方に追加されます。

`CacheClusterEnabled`

ステージでキャッシュが有効化されているかどうかを示します。レスポンスをキャッシュするには、`MethodSettings` で `CachingEnabled` を `true` に設定することも必要です。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::Stage` リソースの `CacheClusterEnabled` プロパティに直接渡されます。

`CacheClusterSize`

ステージのキャッシュクラスターサイズです。

タイプ : 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::Stage` リソースの `CacheClusterSize` プロパティに直接渡されます。

`CanarySetting`

通常のデプロイの段階に Canary 設定を設定します。

タイプ: CanarySetting

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::Stage リソースの CanarySetting プロパティに直接渡されます。

Cors

すべての API Gateway API のクロスオリジンリソース共有 (CORS) を管理します。許可するドメインを文字列として指定するか、追加の CORS 設定でディクショナリを指定します。

> **Note**
>
> CORS では、OpenAPI 定義の変更に AWS SAM が必要です。CORS を有効にするには、DefinitionBody の中でインライン OpenAPI 定義を作成します。

CORS の詳細については、Amazon API Gateway デベロッパーガイドの「REST API リソースの CORS を有効にする」を参照してください。

タイプ: 文字列 | CorsConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

DefinitionBody

API を説明する OpenAPI 仕様です。DefinitionUri と DefinitionBody のどちらも指定されていない場合、SAM はテンプレート設定に基づいて DefinitionBody を生成します。

API を定義するローカルの OpenAPI ファイルを参照するには、AWS::Include 変換を使用してください。詳細については、「がローカルファイル AWS SAM をアップロードする方法」を参照してください。

Type: JSON

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::RestApi リソースの Body プロパティに似ています。特定のプロパティが提供されている場合、コンテンツは、CloudFormation に渡される前に DefinitionBody に挿入または変更される可能性があります。プロパティには Auth、BinaryMediaTypes、Cors、GatewayResponses、Models、およ

び、対応する AWS::Serverless::Function 向けの EventSource タイプの API が含まれます。

DefinitionUri

Amazon S3 URI、ローカルファイルパス、または API を定義する OpenAPI ドキュメントのロケーションオブジェクトです。このプロパティが参照する Amazon S3 オブジェクトは、有効な OpenAPI ファイルである必要があります。DefinitionUri と DefinitionBody のどちらも指定されていない場合、SAM はテンプレート設定に基づいて DefinitionBody を生成します。

ローカルファイルパスを指定する場合は、定義が適切に変換されるようにするために、テンプレートが sam deploy または sam package コマンドを含むワークフローを実行する必要があります。

組み込み関数は、DefinitionUri で参照される外部 OpenApi ファイルではサポートされていません。その代わりに、Include Transform で DefinitionBody プロパティを使用して、OpenApi 定義をテンプレートにインポートします。

タイプ: 文字列 | ApiDefinition

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::RestApi リソースの BodyS3Location プロパティに似ています。ネストされた Amazon S3 プロパティには異なる名前が付けられています。

Description

API リソースの説明です。

タイプ : 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::RestApi リソースの Description プロパティに直接渡されます。

DisableExecuteApiEndpoint

クライアントがデフォルトの execute-api エンドポイントを使用して API を呼び出すことができるかどうかを指定します。デフォルトでは、クライアントはデフォルトの https://{api_id}.execute-api.{region}.amazonaws.com を使用して API を呼び出すことができます。クライアントがカスタムドメイン名を使用して API を呼び出すように要求するには、True を指定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::RestApi リソースの DisableExecuteApiEndpoint プロパティに似ています。これは x-amazon-apigateway-endpoint-configuration 拡張機能の disableExecuteApiEndpoint プロパティに直接渡され、AWS::ApiGateway::RestApi リソースの Body プロパティに追加されます。

`Domain`

この API Gateway API のカスタムドメインを設定します。

タイプ: DomainConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`EndpointConfiguration`

REST API のエンドポイントタイプです。

タイプ: EndpointConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::RestApi リソースの EndpointConfiguration プロパティに似ています。ネストされた設定プロパティには異なる名前が付けられています。

`FailOnWarnings`

警告が発生したときに、API 作成をロールバックするか (true)、しないか (false) を指定します。デフォルト値は false です。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::RestApi リソースの FailOnWarnings プロパティに直接渡されます。

`GatewayResponses`

API のゲートウェイレスポンスを設定します。ゲートウェイレスポンスは、直接、または Lambda オーソライザーを使用して返される API Gateway からのレスポンスです。詳細については、ゲートウェイレスポンス用の API Gateway OpenApi 拡張機能に関するドキュメントを参照してください。

タイプ: マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`MergeDefinitions`

AWS SAM は API イベントソースから OpenAPI の仕様を生成します。AWS SAM が `AWS::Serverless::Api` リソースで定義されたインライン OpenAPI 仕様にこれをマージするようにするには `true` を指定します。マージしない場合は `false` を指定します。

`MergeDefinitions` では、`AWS::Serverless::Api` の `DefinitionBody` プロパティを定義する必要があります。`MergeDefinitions` は `AWS::Serverless::Api` の `DefinitionUri` プロパティと互換性がありません。

デフォルト値: `false`

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`MethodSettings`

ロギング、メトリクス、CacheTTL、スロットリングなどの API ステージのすべての設定を行います。

タイプ: MethodSetting のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::Stage` リソースの `MethodSettings` プロパティに直接渡されます。

`MinimumCompressionSize`

クライアントの Accept-Encoding ヘッダーに基づくレスポンス本文の圧縮を許可します。圧縮は、レスポンス本文のサイズが設定したしきい値以上の場合にトリガーされます。本文サイズの最大しきい値は 10 MB (10,485,760 バイト) です。gzip、deflate、および identity の圧縮タイプがサポートされます。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::RestApi` リソースの `MinimumCompressionSize` プロパティに直接渡されます。

`Mode`

このプロパティは、OpenAPI を使用して REST API を定義するときにのみ適用されます。`Mode` は、API Gateway がリソース更新を処理する方法を決定します。詳細については、AWS::ApiGateway::RestApi リソースタイプのModeプロパティを参照してください。

有効な値: `overwrite` または `merge`

タイプ : 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::RestApi` リソースの `Mode` プロパティに直接渡されます。

`Models`

API メソッドで使用されるスキーマです。これらのスキーマは、JSON または YAML を使用して記述できます。サンプルモデルについては、このページの下部にある「例」セクションを参照してください。

タイプ: マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Name`

API Gateway RestApi リソースの名前です。

タイプ : 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::RestApi` リソースの `Name` プロパティに直接渡されます。

`OpenApiVersion`

使用する OpenApi のバージョンです。これは、Swagger 仕様の `2.0`、または `3.0.1` のような OpenApi 3.0 バージョンの 1 つにすることができます。OpenAPI の詳細については、「OpenAPI Specification」を参照してください。

> **Note**
>
> AWS SAM は、`Stage` と呼ばれるステージをデフォルトで作成します。このプロパティに有効な値を設定すると、ステージ `Stage` が作成されなくなります。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`PropagateTags`

AWS::Serverless::Api が生成したリソースに `Tags` プロパティからのタグを渡すかどうかを指定します。`True` を指定して、生成されたリソースにタグを伝播します。

タイプ: ブール

必須: いいえ

デフォルト: `False`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`StageName`

API Gateway が invoke Uniform Resource Identifier (URI) の最初のパスセグメントとして使用するステージの名前です。

ステージリソースを参照するには、*<api-logical-id>*.Stage を使用します。AWS::Serverless::Api リソースの指定時に生成されるリソースの参照に関する詳細については、「AWS::Serverless::Api が指定された場合、生成される AWS CloudFormation リソース」を参照してください。生成された AWS CloudFormation リソースの一般情報については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

タイプ : 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::Stage リソースの StageName プロパティに似ています。SAM では必須ですが、API Gateway では必須ではありません。

その他の注意点: 暗黙的な API には「prod」という名前のステージがあります。

Tags

この API Gateway ステージに追加されるタグを指定するマップ (文字列対文字列) です。タグの有効なキーと値の詳細については、AWS CloudFormation ユーザーガイドのリソースタグを参照してください。

タイプ: マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::Stage リソースの Tags プロパティに似ています。SAM の Tags プロパティは、キーバリューペアで構成されています。CloudFormation では、タグオブジェクトのリストで構成されています。

TracingEnabled

このステージに X-Ray を使用したアクティブトレーシングが有効化されているかどうかを示します。X-Ray の詳細については、API Gateway デベロッパーガイドの「X-Ray を使用した REST API へのユーザーリクエストのトレース」を参照してください。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::Stage リソースの TracingEnabled プロパティに直接渡されます。

`Variables`

ステージ変数を定義するマップ (文字列対文字列) で、変数名はキー、変数値は値です。変数名に使用できるのは英数字のみです。値は次の正規表現に一致する必要があります: `[A-Za-z0-9._~:/?#&=,-]+`。

タイプ: マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::Stage` リソースの `Variables` プロパティに直接渡されます。

戻り値

参照番号

このリソースの論理 ID が `Ref` 組み込み関数に提供されると、基盤となる API Gateway API の ID が返されます。

`Ref` 関数の使用方法の詳細については、AWS CloudFormation ユーザーガイドの「`Ref`」を参照してください。

Fn::GetAtt

`Fn::GetAtt` は、このタイプの指定された属性の値を返します。利用可能な属性とサンプル戻り値は以下のとおりです。

`Fn::GetAtt` の使用の詳細については、AWS CloudFormation ユーザーガイドの「`Fn::GetAtt`」を参照してください。

`RootResourceId`

`RestApi` リソースのルートソース ID (`a0bc123d4e` など) です。

例

SimpleApiExample

API エンドポイントを持つ Lambda 関数が含まれた Hello World AWS SAM テンプレートです。これは、正常に動作するサーバーレスアプリケーションの完全な AWS SAM テンプレートファイルです。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: AWS SAM template with a simple API definition
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: prod
  ApiFunction: # Adds a GET method at the root resource via an Api event
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: Api
          Properties:
            Path: /
            Method: get
            RestApiId:
              Ref: ApiGatewayApi
      Runtime: python3.10
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
            return {'body': 'Hello World!', 'statusCode': 200}
```

ApiCorsExample

Lambda 統合および CORS 設定と共に、外部 Swagger ファイルに定義された API が含まれる AWS SAM テンプレートのスニペットです。これは、AWS::Serverless::Api 定義を示す AWS SAM テンプレートファイルの一部分です。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      # Allows www.example.com to call these APIs
      # SAM will automatically add AllowMethods with a list of methods for this API
      Cors: "'www.example.com'"
```

```
      DefinitionBody: # Pull in an OpenApi definition from S3
        'Fn::Transform':
          Name: 'AWS::Include'
          # Replace "bucket" with your bucket name
          Parameters:
            Location: s3://bucket/swagger.yaml
```

ApiCognitoAuthExample

Amazon Cognito を使用して API に対するリクエストを承認する API を使用する AWS SAM テンプレートのスニペットです。これは、AWS::Serverless::Api 定義を示す AWS SAM テンプレートファイルの一部分です。

YAML

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

ApiModelsExample

Models スキーマが含まれた API を使用する AWS SAM テンプレートのスニペットです。これは、2 つのモデルスキーマを持つ AWS::Serverless::Api 定義を示す AWS SAM テンプレートファイルの一部分です。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
```

```
      Models:
        User:
          type: object
          required:
            - username
            - employee_id
          properties:
            username:
              type: string
            employee_id:
              type: integer
            department:
              type: string
        Item:
          type: object
          properties:
            count:
              type: integer
            category:
              type: string
            price:
              type: integer
```

キャッシュの例

API エンドポイントを持つ Lambda 関数が含まれた Hello World AWS SAM テンプレートです。API では、1 つのリソースとメソッドに対してキャッシュが有効になっています。キャッシュの詳細については、「API Gateway デベロッパーガイド」の「API キャッシュを有効にして応答性を強化する」を参照してください。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: AWS SAM template with a simple API definition with caching turned on
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: prod
      CacheClusterEnabled: true
      CacheClusterSize: '0.5'
      MethodSettings:
```

```
        - ResourcePath: /
          HttpMethod: GET
          CachingEnabled: true
          CacheTtlInSeconds: 300
      Tags:
        CacheMethods: All

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

ApiAuth

Gateway API へのアクセスを制御するための認可を設定しますAPI。

を使用してアクセスを設定する方法の詳細と例については、 AWS SAM 「」を参照してくださいAWS SAM テンプレートを使用して API アクセスを制御する。

構文

AWS Serverless Application Model （ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
  AddApiKeyRequiredToCorsPreflight: Boolean
  AddDefaultAuthorizerToCorsPreflight: Boolean
  ApiKeyRequired: Boolean
  Authorizers: CognitoAuthorizer | LambdaTokenAuthorizer | LambdaRequestAuthorizer |
 AWS_IAM
```

```
  DefaultAuthorizer: String
  InvokeRole: String
  ResourcePolicy: ResourcePolicyStatement
  UsagePlan: ApiUsagePlan
```

> **Note**
>
> Authorizers プロパティには が含まれますがAWS_IAM、 に追加の設定は必要ありませんAWS_IAM。例については、AWS IAMを参照してください。

プロパティ

`AddApiKeyRequiredToCorsPreflight`

ApiKeyRequired プロパティと Corsプロパティが設定されている場合、 を設定するとAPIキーAddApiKeyRequiredToCorsPreflightがOptionsプロパティに追加されます。

タイプ: ブール

必須: いいえ

デフォルト: True

AWS CloudFormation 互換性 : このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

`AddDefaultAuthorizerToCorsPreflight`

DefaultAuthorizer プロパティと Corsプロパティが設定されている場合、 を設定すると、デフォルトのオーソライザーAddDefaultAuthorizerToCorsPreflightが OpenAPI セクションの Optionsプロパティに追加されます。

タイプ: ブール

必須: いいえ

デフォルト: True

AWS CloudFormation 互換性 : このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

`ApiKeyRequired`

true に設定すると、すべてのAPIイベントにAPIキーが必要です。API キーの詳細については、「ゲートウェイデベロッパーガイド」のAPI「キーを使用した使用プランの作成と使用」を参照してください。 API

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性 : このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

`Authorizers`

Gateway API へのアクセスを制御するために使用されるオーソライザーAPI。

詳細については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。

タイプ： CognitoAuthorizer | LambdaTokenAuthorizer | LambdaRequestAuthorizer | AWS_IAM

必須: いいえ

デフォルト: なし

AWS CloudFormation 互換性 : このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

追加のメモ : Api OpenApi の定義にオーソライザーSAMを追加します。

`DefaultAuthorizer`

API ゲートウェイ のデフォルトのオーソライザーを指定します。これはAPI、デフォルトでAPI呼び出しの承認に使用されます。

> **Note**
>
> これに関連付けられた関数 EventSource の Api IAM APIが アクセス許可を使用するように設定されている場合、このプロパティは に設定する必要があります。設定`AWS_IAM`しないとエラーが発生します。

型: 文字列

必須: いいえ

デフォルト: なし

AWS CloudFormation 互換性： このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

`InvokeRole`

すべてのリソースとメソッドの統合認証情報をこの値に設定します。

`CALLER_CREDENTIALS` が `arn:aws:iam::*:user/*` にマップされます。これは、発信者の認証情報を使用してエンドポイントを呼び出します。

有効な値: `CALLER_CREDENTIALS`、`NONE`、`IAMRoleArn`

型: 文字列

必須: いいえ

デフォルト: `CALLER_CREDENTIALS`

AWS CloudFormation 互換性： このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

`ResourcePolicy`

上のすべてのメソッドとパスのリソースポリシーを設定しますAPI。

タイプ : ResourcePolicyStatement

必須: いいえ

AWS CloudFormation 互換性 : このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

その他の注意点: この設定は、`AWS::Serverless::Function` を使用して個々の ApiFunctionAuth で定義することも可能です。これは、 APIsで に必要ですEndpointConfiguration: PRIVATE。

`UsagePlan`

この に関連付けられた使用プランを設定しますAPI。使用プランの詳細については、「ゲートウェイデベロッパーガイド」のAPI「キーを使用した使用プランの作成と使用」を参照してください。 API

この AWS SAM プロパティは、このプロパティが設定されると、次の 3 つの追加 AWS CloudFormation リソースを生成します。 AWS::ApiGateway::UsagePlan、AWS::ApiGateway::UsagePlanKey、 および AWS::ApiGateway::ApiKey。 このシナリオの詳細については、「」を参照してくださいUsagePlan プロパティが指定されている。生成された AWS CloudFormation リソースの一般的な情報については、「」を参照してくださいAWS SAM 向けに生成された AWS CloudFormation リソース。

タイプ： ApiUsagePlan

必須: いいえ

AWS CloudFormation 互換性： このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

例

CognitoAuth

Cognito Auth の例

YAML

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

```
              "aws:SourceIp": "1.2.3.4"
            }
          }
      }]
      IpRangeDenylist:
        - "10.20.30.40"
```

AWS IAM

AWS IAM 例

YAML

```
Auth:
  Authorizers: AWS_IAM
```

ApiUsagePlan

API Gateway API の使用量プランを設定します。使用量プランの詳細については、API Gateway デベロッパーガイドの「API キーを使用した使用量プランの作成と使用」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`CreateUsagePlan`

この使用量プランの設定方法を決定します。有効な値は、`PER_API`、`SHARED`、`NONE` です。

`PER_API` は、この API に固有の AWS::ApiGateway::UsagePlan、AWS::ApiGateway::ApiKey、および AWS::ApiGateway::UsagePlanKey リソースを作成します。これらのリソースには、

それぞれ *<api-logical-id>*UsagePlan、*<api-logical-id>*ApiKey、および *<api-logical-id>*UsagePlanKey の論理 ID があります。

SHARED は、同じ AWS SAM テンプレートに CreateUsagePlan: SHARED があるすべてのAPIで共有される AWS::ApiGateway::UsagePlan、AWS::ApiGateway::ApiKey、および AWS::ApiGateway::UsagePlanKey リソースを作成します。これらのリソースには、それぞれ ServerlessUsagePlan、ServerlessApiKey、および ServerlessUsagePlanKey の論理 ID があります。このオプションを使用する場合は、競合する定義と不確実な状態を避けるために、この使用量プランの設定を 1 つの API リソースだけに追加することが推奨されます。

NONE は、この API の使用プランの作成または関連付けを無効にします。これは、AWS SAM テンプレートの Globals セクション で SHARED または PER_API が指定されている場合にのみ必要になります。

有効な値: PER_API、SHARED、NONE

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Description`

使用量プランの説明です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::UsagePlan リソースの Description プロパティに直接渡されます。

`Quota`

指定された間隔内にユーザーが実行できるリクエストの数を設定します。

タイプ: QuotaSettings

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::UsagePlan リソースの Quota プロパティに直接渡されます。

`Tags`

使用量プランに関連付ける任意のタグの配列 (キーバリューペア) です。

このプロパティは、CloudFormation のタグタイプを使用します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::UsagePlan` リソースの `Tags` プロパティに直接渡されます。

`Throttle`

全体的なリクエスト率 (1 秒あたりの平均リクエスト数) とバーストキャパシティを設定します。

タイプ: ThrottleSettings

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::UsagePlan` リソースの `Throttle` プロパティに直接渡されます。

`UsagePlanName`

使用量プランの名前です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::UsagePlan` リソースの `UsagePlanName` プロパティに直接渡されます。

例

UsagePlan

以下は、使用量プランの例です。

YAML

```
Auth:
```

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

CognitoAuthorizer

Amazon Cognito ユーザープールオーソライザーを定義します。

詳細情報と例については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`AuthorizationScopes`

このオーソライザーの認可スコープのリストです。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Identity`

このプロパティは、オーソライザーの受信リクエストに `IdentitySource` を指定するために使用できます。

タイプ: CognitoAuthorizationIdentity

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`UserPoolArn`

この Cognito オーソライザーを追加するユーザープールを参照する、またはユーザープール arn を指定することができます

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

CognitoAuth

Cognito 認証の例です。

YAML

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

```
        ValidationExpression: myauthvalidationexpression
```

CognitoAuthorizationIdentity

このプロパティは、オーソライザーの受信リクエストに IdentitySource を指定するために使用できます。IdentitySource の詳細については、「ApiGateway オーソライザーの OpenApi 拡張機能」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`Header`

OpenApi 定義で認可用のヘッダー名を指定します。

型: 文字列

必須: いいえ

デフォルト: Authorization

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ReauthorizeEvery`

API Gateway がオーソライザー結果をキャッシュする時間を指定する有効期限 (TTL) (秒) です。0 より大きい値を指定する場合、API Gateway が認証レスポンスをキャッシュします。デフォルトで、API Gateway はこのプロパティを 300 に設定します。最大値は 3600、つまり 1 時間です。

タイプ: 整数

必須: いいえ

デフォルト: 300

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ValidationExpression`

受信アイデンティティを検証するための検証式を指定します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

CognitoAuthIdentity

YAML

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

LambdaRequestAuthorizer

Lambda 関数を使用して、API へのアクセスを制御するように Lambda オーソライザーを設定します。

詳細な説明と例については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DisableFunctionDefaultPermissions: Boolean
```

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

プロパティ

`DisableFunctionDefaultPermissions`

`AWS::Serverless::Api` リソースとオーソライザー Lambda 関数の間の許可をプロビジョニングするための `AWS::Lambda::Permissions` リソースを AWS SAM が自動的に作成しないように `true` を指定します。

デフォルト値: `false`

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionArn`

API に対する認可を提供する Lambda 関数の関数 ARN を指定します。

> **Note**
>
> `AWS::Serverless::Api` 用に `FunctionArn` が指定されると、AWS SAM は `AWS::Lambda::Permissions` リソースを自動的に作成します。`AWS::Lambda::Permissions` リソースは、API とオーソライザー Lambda 関数の間の許可をプロビジョニングします。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionInvokeRole`

オーソライザーの認証情報を Lambda オーソライザーの OpenApi 定義に追加します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionPayloadType`

このプロパティは、API の Lambda オーソライザーのタイプを定義するために使用できます。

有効な値: `TOKEN` または `REQUEST`

タイプ : 文字列

必須: いいえ

デフォルト: `TOKEN`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Identity`

このプロパティは、オーソライザーの受信リクエストに `IdentitySource` を指定するために使用できます。このプロパティが必要になるのは、`FunctionPayloadType` プロパティが `REQUEST` に設定されている場合のみです。

タイプ: LambdaRequestAuthorizationIdentity

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

LambdaRequestAuth

YAML

```
Authorizers:
  MyLambdaRequestAuth:
    FunctionPayloadType: REQUEST
```

```
    FunctionArn:
      Fn::GetAtt:
        - MyAuthFunction
        - Arn
    FunctionInvokeRole:
      Fn::GetAtt:
        - LambdaAuthInvokeRole
        - Arn
    Identity:
      Headers:
        - Authorization1
```

LambdaRequestAuthorizationIdentity

このプロパティは、オーソライザーの受信リクエストに IdentitySource を指定するために使用できます。IdentitySource の詳細については、「ApiGateway オーソライザーの OpenApi 拡張機能」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`Context`

所定のコンテキスト文字列を `context.contextString` 形式のマッピング式に変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Headers`

ヘッダーを、`method.request.header.name` 形式のマッピング式のカンマ区切り文字列に変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`QueryStrings`

所定のクエリ文字列を、`method.request.querystring.queryString` 形式のマッピング式のカンマ区切り文字列に変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ReauthorizeEvery`

API Gateway がオーソライザー結果をキャッシュする時間を指定する有効期限 (TTL) (秒) です。0 より大きい値を指定する場合、API Gateway が認証レスポンスをキャッシュします。デフォルトで、API Gateway はこのプロパティを 300 に設定します。最大値は 3600、つまり 1 時間です。

タイプ: 整数

必須: いいえ

デフォルト: 300

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`StageVariables`

所定のステージ変数を、`stageVariables.stageVariable` 形式のマッピング式のカンマ区切り文字列に変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

LambdaRequestIdentity

YAML

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

LambdaTokenAuthorizer

Lambda 関数を使用して、API へのアクセスを制御するように Lambda オーソライザーを設定します。

詳細な説明と例については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DisableFunctionDefaultPermissions: Boolean
FunctionArn: String
FunctionInvokeRole: String
```

```
FunctionPayloadType: String
Identity: LambdaTokenAuthorizationIdentity
```

プロパティ

`DisableFunctionDefaultPermissions`

`AWS::Serverless::Api` リソースとオーソライザー Lambda 関数の間の許可をプロビジョニングするための `AWS::Lambda::Permissions` リソースを AWS SAM が自動的に作成しないように `true` を指定します。

デフォルト値: `false`

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionArn`

API に対する認可を提供する Lambda 関数の関数 ARN を指定します。

> **Note**
>
> `AWS::Serverless::Api` 用に `FunctionArn` が指定されると、AWS SAM は `AWS::Lambda::Permissions` リソースを自動的に作成します。`AWS::Lambda::Permissions` リソースは、API とオーソライザー Lambda 関数の間の許可をプロビジョニングします。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionInvokeRole`

オーソライザーの認証情報を Lambda オーソライザーの OpenApi 定義に追加します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionPayloadType`

このプロパティは、API の Lambda オーソライザーのタイプを定義するために使用できます。

有効な値: `TOKEN` または `REQUEST`

タイプ : 文字列

必須: いいえ

デフォルト: `TOKEN`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Identity`

このプロパティは、オーソライザーの受信リクエストに `IdentitySource` を指定するために使用できます。このプロパティが必要になるのは、`FunctionPayloadType` プロパティが `REQUEST` に設定されている場合のみです。

タイプ: LambdaTokenAuthorizationIdentity

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

LambdaTokenAuth

YAML

```
Authorizers:
  MyLambdaTokenAuth:
```

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

BasicLambdaTokenAuth

YAML

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

LambdaTokenAuthorizationIdentity

このプロパティは、オーソライザーの受信リクエストに IdentitySource を指定するために使用できます。IdentitySource の詳細については、「ApiGateway オーソライザーの OpenApi 拡張機能」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`Header`

OpenApi 定義で認可用のヘッダー名を指定します。

型: 文字列

必須: いいえ

デフォルト: Authorization

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ReauthorizeEvery`

API Gateway がオーソライザー結果をキャッシュする時間を指定する有効期限 (TTL) (秒) です。0 より大きい値を指定する場合、API Gateway が認証レスポンスをキャッシュします。デフォルトで、API Gateway はこのプロパティを 300 に設定します。最大値は 3600、つまり 1 時間です。

タイプ: 整数

必須: いいえ

デフォルト: 300

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ValidationExpression`

受信アイデンティティを検証するための検証式を指定します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

LambdaTokenIdentity

YAML

```
Identity:
```

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

ResourcePolicyStatement

API 上のすべてのメソッドとパスのリソースポリシーを設定します。リソースポリシーの詳細については、API Gateway デベロッパーガイドの「API Gateway リソースポリシーを使用して API へのアクセスを制御する」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`AwsAccountBlacklist`

ブロックする AWS アカウントです。

型: 文字列のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AwsAccountWhitelist`

許可する AWS アカウントです。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

型: 文字列のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`CustomStatements`

この API に適用するカスタムリソースポリシーステートメントのリストです。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpcBlacklist`

ブロックする仮想プライベートクラウド (VPC) のリストで、各 VPC が動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpcWhitelist`

許可する VPC のリストで、各 VPC が動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpceBlacklist`

ブロックする VPC エンドポイントのリストで、各 VPC エンドポイントが動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpceWhitelist`

許可する VPC エンドポイントのリストで、各 VPC エンドポイントが動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IpRangeBlacklist`

ブロックする IP アドレスまたはアドレス範囲です。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IpRangeWhitelist`

許可する IP アドレスまたはアドレス範囲です。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceVpcBlacklist`

ブロックするソース VPC またはソース VPC エンドポイントです。ソース VPC 名は `"vpc-"` で始まり、ソース VPC エンドポイント名は `"vpce-"` で始まる必要があります。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceVpcWhitelist`

許可するソース VPC またはソース VPC エンドポイントです。ソース VPC 名は `"vpc-"` で始まり、ソース VPC エンドポイント名は `"vpce-"` で始まる必要があります。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

リソースポリシーの例

以下の例は、2 つの IP アドレスと 1 つのソース VPC をブロックし、AWS アカウントを許可します。

YAML

```
Auth:
  ResourcePolicy:
    CustomStatements: [{
                        "Effect": "Allow",
```

```
                        "Principal": "*",
                        "Action": "execute-api:Invoke",
                        "Resource": "execute-api:/Prod/GET/pets",
                        "Condition": {
                          "IpAddress": {
                            "aws:SourceIp": "1.2.3.4"
                          }
                        }
                      }]
    IpRangeBlacklist:
      - "10.20.30.40"
      - "1.2.3.4"
    SourceVpcBlacklist:
      - "vpce-1a2b3c4d"
    AwsAccountWhitelist:
      - "111122223333"
    IntrinsicVpcBlacklist:
      - "{{resolve:ssm:SomeVPCReference:1}}"
      - !Ref MyVPC
    IntrinsicVpceWhitelist:
      - "{{resolve:ssm:SomeVPCEReference:1}}"
      - !Ref MyVPCE
```

ApiDefinition

を定義する OpenAPI ドキュメントAPI。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

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

プロパティ

`Bucket`

OpenAPI ファイルが保存されている Amazon S3 バケットの名前。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、AWS::ApiGateway::RestApiS3Locationデータ型の Bucket プロパティに直接渡されます。

Key

OpenAPI ファイルの Amazon S3 キー。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、AWS::ApiGateway::RestApiS3Locationデータ型の Key プロパティに直接渡されます。

Version

バージョニングされたオブジェクトの場合、OpenAPI ファイルのバージョン。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::ApiGateway::RestApiS3Locationデータ型の Version プロパティに直接渡されます。

例

定義 URI の例

API 定義の例

YAML

```
DefinitionUri:
  Bucket: amzn-s3-demo-bucket-name
```

```
  Key: mykey-name
  Version: 121212
```

CorsConfiguration

API Gateway API のクロスオリジンリソース共有 (CORS) を管理します。許可するドメインを文字列として指定するか、追加の CORS 設定でディクショナリを指定します。

> **Note**
>
> CORS では、OpenAPI 定義の変更に AWS SAM が必要です。CORS を有効にするには、`DefinitionBody` の中でインライン OpenAPI 定義を作成します。`CorsConfiguration` が OpenAPI 定義内とプロパティレベルで同時に設定されている場合、AWS SAM はそれらをマージします。プロパティレベルは OpenAPI 定義よりも優先されます。

CORS の詳細については、Amazon API Gateway デベロッパーガイドの「REST API リソースの CORS を有効にする」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`AllowCredentials`

リクエストに認証情報を含めることができるかどうかを示すブール値です。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AllowHeaders`

許可するヘッダーの文字列です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AllowMethods`

許可する HTTP メソッドが含まれる文字列です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AllowOrigin`

許可するオリジンの文字列です。これは、文字列形式のカンマ区切りリストになります。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`MaxAge`

CORS プリフライトリクエストをキャッシュする秒数が含まれる文字列です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

CorsConfiguration

CORS 設定の例です。これは、Cors が設定された AWS::Serverless::Api 定義と AWS::Serverless::Function を示す、AWS SAM テンプレートファイルのごく一部分です。Lambda プロキシ統合または HTTP プロキシ統合を使用する場合、バックエンドが `Access-Control-Allow-Origin`、`Access-Control-Allow-Methods`、および `Access-Control-Allow-Headers`ヘッダーを返す必要があります。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Cors:
        AllowMethods: "'POST, GET'"
        AllowHeaders: "'X-Forwarded-For'"
        AllowOrigin: "'https://example.com'"
        MaxAge: "'600'"
        AllowCredentials: true
  ApiFunction: # Adds a GET method at the root resource via an Api event
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: Api
          Properties:
            Path: /
            Method: get
            RestApiId:
              Ref: ApiGatewayApi
      Runtime: python3.10
      Handler: index.handler
      InlineCode: |
```

```
        import json
        def handler(event, context):
          return {
          'statusCode': 200,
          'headers': {
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Origin': 'https://example.com',
            'Access-Control-Allow-Methods': 'POST, GET'
            },
          'body': json.dumps('Hello from Lambda!')
          }
```

DomainConfiguration

API のカスタムドメインを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
BasePath: List
NormalizeBasePath: Boolean
CertificateArn: String
DomainName: String
EndpointConfiguration: String
MutualTlsAuthentication: MutualTlsAuthentication
OwnershipVerificationCertificateArn: String
Route53: Route53Configuration
SecurityPolicy: String
```

プロパティ

`BasePath`

Amazon API Gateway ドメイン名で設定する basepaths のリストです。

タイプ: リスト

必須: いいえ

デフォルト: /

AWS CloudFormation との互換性: このプロパティは AWS::ApiGateway::BasePathMapping リソースの BasePath プロパティと似ています。AWS SAM は複数の AWS::ApiGateway::BasePathMapping リソースを作成します (このプロパティに指定された BasePath につき 1 つ)。

NormalizeBasePath

BasePath プロパティで定義されたベースパスに英数字以外の文字を使用できるかどうかを示します。True に設定すると、英数字以外の文字がベースパスから削除されます。

NormalizeBasePath を BasePath プロパティと一緒に使用します。

タイプ: ブール

必須: いいえ

デフォルト: True

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

CertificateArn

このドメイン名のエンドポイント用の AWS マネージド証明書の Amazon リソースネーム (ARN) です。サポートされるソースは AWS Certificate Manager のみです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway::DomainName リソースの CertificateArn プロパティに似ています。EndpointConfiguration が REGIONAL (デフォルト値) に設定されている場合、CertificateArn は AWS::ApiGateway::DomainName の RegionalCertificateArn にマップされます。EndpointConfiguration が EDGE に設定されている場合、CertificateArn は AWS::ApiGateway::DomainName の CertificateArn にマップされます。

その他の注意点: EDGE エンドポイントの場合は、証明書を us-east-1 AWS リージョンで作成する必要があります。

`DomainName`

API Gateway API のカスタムドメイン名です。大文字はサポートされていません。

このプロパティが設定されていると、AWS SAM は AWS::ApiGateway::DomainName リソースを生成します。このシナリオの詳細については、「DomainName プロパティが指定されている」を参照してください。生成された AWS CloudFormation リソースについては、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::DomainName` リソースの `DomainName` プロパティに直接渡されます。

`EndpointConfiguration`

カスタムドメインにマップする API Gateway エンドポイントのタイプを定義します。このプロパティの値は、`CertificateArn` プロパティが AWS CloudFormation でマップされる方法を決定します。

有効な値: `REGIONAL` または `EDGE`

タイプ : 文字列

必須: いいえ

デフォルト: `REGIONAL`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`MutualTlsAuthentication`

カスタムドメイン名の相互 Transport Layer Security (TLS) 認証設定です。

タイプ: MutualTlsAuthentication

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::DomainName` リソースの `MutualTlsAuthentication` プロパティに直接渡されます。

`OwnershipVerificationCertificateArn`

カスタムドメインの所有権を検証するために ACM によって発行されたパブリック証明書の ARN。相互 TLS を設定し、ACM にインポートされた、またはプライベート CA 証明書の ARN を `CertificateArn` に指定する場合のみ必須です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::DomainName` リソースの `OwnershipVerificationCertificateArn` プロパティに直接渡されます。

`Route53`

Amazon Route 53 設定を定義します。

タイプ: Route53Configuration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SecurityPolicy`

このドメイン名の TLS バージョンと暗号スイートです。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::DomainName` リソースの `SecurityPolicy` プロパティに直接渡されます。

例

DomainName

DomainName の例

YAML

```
Domain:
```

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

Route53Configuration

API の Route53 レコードセットを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  DistributionDomainName: String
  EvaluateTargetHealth: Boolean
  HostedZoneId: String
  HostedZoneName: String
  IpV6: Boolean
  Region: String
  SetIdentifier: String
```

プロパティ

`DistributionDomainName`

API カスタムドメイン名のカスタムディストリビューションを設定します。

型: 文字列

必須: いいえ

デフォルト: API Gateway ディストリビューションを使用します。

AWS CloudFormation との互換性: このプロパティは、`AWS::Route53::RecordSetGroup` `AliasTarget` リソースの `DNSName` プロパティに直接渡されます。

その他の注意点: CloudFront ディストリビューションのドメイン名です。

`EvaluateTargetHealth`

EvaluateTargetHealth が true の場合は、参照される AWS リソース (Elastic Load Balancing ロードバランサーやホストゾーン内の別のレコードなど) のヘルスがエイリアスレコードに継承されます。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Route53::RecordSetGroup` `AliasTarget` リソースの `EvaluateTargetHealth` プロパティに直接渡されます。

その他の注意点: エイリアスターゲットが CloudFront ディストリビューションの場合は EvaluateTargetHealth を true に設定できません。

`HostedZoneId`

レコードを作成するホストゾーンの ID です。

`HostedZoneName` または `HostedZoneId` を指定します。両方を指定することはできません。同じドメイン名のホストゾーンが複数ある場合は、`HostedZoneId` を使用してホストゾーンを指定する必要があります。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Route53::RecordSetGroup` `RecordSet` リソースの `HostedZoneId` プロパティに直接渡されます。

`HostedZoneName`

レコードを作成するホストゾーンの名前です。

`HostedZoneName` または `HostedZoneId` を指定します。両方を指定することはできません。同じドメイン名のホストゾーンが複数ある場合は、`HostedZoneId` を使用してホストゾーンを指定する必要があります。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Route53::RecordSetGroup RecordSet リソースの HostedZoneName プロパティに直接渡されます。

IpV6

このプロパティを設定すると、AWS SAM が AWS::Route53::RecordSet リソースを作成し、提供された HostedZone のタイプを AAAA に設定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

Region

レイテンシーベースのリソースレコードセットのみ: このリソースレコードセットが参照するリソースを作成した Amazon EC2 リージョン。リソースは通常、AWS リソース (EC2 インスタンスや ELB ロードバランサーなど) であり、レコードタイプに応じて IP アドレスまたは DNS ドメイン名で参照されます。

レイテンシーリソースレコードセットが作成されているドメインの名前や種類を要求する DNS クエリを受け取ると、Amazon Route 53 は、エンドユーザーとそのユーザーに関連付けられている Amazon EC2 リージョンとの間でレイテンシーが最も小さいレイテンシーリソースレコードセットを選択します。その後、Route 53 は、選択したリソースレコードセットに関連付けられている値を返します。

次の点に注意してください。

- レイテンシーリソースレコードセットごとに 1 つの ResourceRecord のみ指定できます。
- 作成できるレイテンシーリソースレコードセットは、各 Amazon EC2 リージョンにつき 1 つだけです。
- すべての Amazon EC2 リージョンに対してレイテンシーリソースレコードセットを作成する必要はありません。レイテンシーリソースレコードセットを作成したリージョンの中から、レイテンシーの最も小さいリージョンが Route 53 によって選択されます。
- レイテンシーリソースレコードセットリソースと Name および Type 要素の値が同じである非レイテンシーリソースレコードセットを作成することはできません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Route53::RecordSetGroup RecordSet データ型の Region プロパティに直接渡されます。

SetIdentifier

シンプル以外のルーティングポリシーを持つリソースレコードセット: タイプが A である acme.example.com という名前の複数の加重リソースレコードセットなど、名前とタイプの組み合わせが同じである複数のリソースレコードセットを区別する識別子。名前とタイプが同じであるリソースレコードセットのグループでは、リソースレコードセットごとに SetIdentifier の値が一意である必要があります。

ルーティングポリシーの詳細については、「Amazon Route 53 デベロッパーガイド」の「ルーティングポリシーの選択」を参照してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Route53::RecordSetGroup RecordSet データ型の SetIdentifier プロパティに直接渡されます。

例

基本的な の例

この例では、API 用にカスタムドメインと Route 53 のレコードセットを設定します。

YAML

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

```
          HostedZoneId: ABCDEFGHIJKLMNOP
```

EndpointConfiguration

REST API のエンドポイントタイプです。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`Type`

REST API のエンドポイントタイプです。

有効な値: `EDGE`、`REGIONAL`、または `PRIVATE`

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::RestApi EndpointConfiguration` データ型の `Types` プロパティに直接渡されます。

`VPCEndpointIds`

Route53 エイリアスを作成する REST API の VPC エンドポイント ID のリストです。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGateway::RestApi EndpointConfiguration` データ型の `VpcEndpointIds` プロパティに直接渡されます。

例

EndpointConfiguration

エンドポイント設定の例

YAML

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

AWS::Serverless::Application

AWS Serverless Application Repository から、または Amazon S3 バケットからのサーバーレスアプリケーションを、ネストされたアプリケーションとして埋め込みます。ネストされたアプリケーションはネストされたものとしてデプロイされます AWS::CloudFormation::Stack リソース。これには、他のリソースを含む複数の他のAWS::Serverless::Applicationリソースを含めることができます。

> **Note**
>
> にデプロイすると AWS CloudFormation、 は AWS SAM リソースを AWS CloudFormation リソース AWS SAM に変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
Type: AWS::Serverless::Application
Properties:
  Location: String | ApplicationLocationObject
  NotificationARNs: List
  Parameters: Map
```

```
  Tags: Map
  TimeoutInMinutes: Integer
```

プロパティ

`Location`

ネストされたアプリケーションのテンプレート URL、ファイルパス、またはロケーションオブジェクト。

テンプレートURLが提供される場合は、CloudFormation TemplateUrl ドキュメントで指定された形式に従い、有効な CloudFormation または SAM テンプレートが含まれている必要があります。AWS Serverless Application Repository に公開されたアプリケーションを指定するには、ApplicationLocationObject を使用することができます。

ローカルファイルパスを指定する場合は、定義が適切に変換されるようにするために、テンプレートが `sam deploy` または `sam package` コマンドを含むワークフローを実行する必要があります。

タイプ : 文字列 | ApplicationLocationObject

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::CloudFormation::Stack`リソースの `TemplateURL`プロパティに似ています。 CloudFormation バージョンではApplicationLocationObject、 を使用して からアプリケーションを取得しません AWS Serverless Application Repository。

`NotificationARNs`

スタックイベントに関する通知が送信される既存の Amazon SNSトピックのリスト。

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::CloudFormation::Stack`リソースの `NotificationARNs`プロパティに直接渡されます。

`Parameters`

アプリケーションパラメータ値です。

タイプ: マップ

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::CloudFormation::Stackリソースの Parameters プロパティに直接渡されます。

Tags

このアプリケーションに追加されるタグを指定するマップ (文字列対文字列) です。キーと値に使用できるのは英数字のみです。キーの長さは 1 ~ 127 文字の Unicode 文字で、「aws:」をプレフィックスとして使用することはできません。値の長さは 1 ~ 255 文字の Unicode 文字にすることができます。

タイプ: マップ

必須: いいえ

AWS CloudFormation 互換性 : このプロパティは、AWS::CloudFormation::Stackリソースの Tags プロパティに似ています。の Tags プロパティSAMは Key:Value ペアで構成され CloudFormation 、その中でタグオブジェクトのリストで構成されます。スタックが作成されると、 SAM はこのアプリケーションにlambda:createdBy:SAMタグを自動的に追加します。さらに、このアプリケーションが からのものである場合 AWS Serverless Application Repository、SAM は 2 つの追加タグserverlessrepo:applicationId:*ApplicationId*と も自動的に追加しますserverlessrepo:semanticVersion:*SemanticVersion*。

TimeoutInMinutes

ネストされたスタックが CREATE_COMPLETE状態になるまで AWS CloudFormation 待機する分単位の時間。デフォルトではタイムアウトが設定されていません。がネストされたスタックが CREATE_COMPLETE状態に達したことを AWS CloudFormation 検出した場合、ネストされたスタックリソースは親スタックCREATE_COMPLETEにある としてマークされ、親スタックの作成が再開されます。ネストされたスタックが に達する前にタイムアウト期間が過ぎるとCREATE_COMPLETE、 はネストされたスタックを失敗として AWS CloudFormation マークし、ネストされたスタックと親スタックの両方をロールバックします。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::CloudFormation::Stackリソースの TimeoutInMinutes プロパティに直接渡されます。

戻り値

参照番号

このリソースの論理 ID が `Ref` 組み込み関数に提供されると、基盤となる `AWS::CloudFormation::Stack` リソースのリソース名が返されます。

`Ref` 関数の使用方法の詳細については、AWS CloudFormation ユーザーガイドの「Ref」を参照してください。

Fn::GetAtt

`Fn::GetAtt` は、このタイプの指定された属性の値を返します。利用可能な属性とサンプル戻り値は以下のとおりです。

`Fn::GetAtt` の使用の詳細については、AWS CloudFormation ユーザーガイドの「Fn::GetAtt」を参照してください。

`Outputs.ApplicationOutputName`

`ApplicationOutputName` という前のスタック出力の値です。

例

SAR アプリケーション

Serverless Application Repository からのテンプレートを使用するアプリケーションです。

YAML

```
Type: AWS::Serverless::Application
Properties:
  Location:
    ApplicationId: 'arn:aws:serverlessrepo:us-east-1:012345678901:applications/my-
application'
    SemanticVersion: 1.0.0
  Parameters:
    StringParameter: parameter-value
    IntegerParameter: 2
```

通常のアプリケーション

S3 url からのアプリケーションです。

YAML

```
Type: AWS::Serverless::Application
Properties:
  Location: https://s3.amazonaws.com/amzn-s3-demo-bucket/template.yaml
```

ApplicationLocationObject

AWS Serverless Application Repository に公開されたアプリケーションです。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  ApplicationId: String
  SemanticVersion: String
```

プロパティ

`ApplicationId`

アプリケーションの Amazon リソースネーム (ARN) です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SemanticVersion`

アプリケーションのセマンティックバージョンです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

my-application

サンプルアプリケーションのロケーションオブジェクトです。

YAML

```
Location:
  ApplicationId: 'arn:aws:serverlessrepo:us-east-1:012345678901:applications/my-
application'
  SemanticVersion: 1.0.0
```

AWS::Serverless::Connector

2 つのリソース間のアクセス許可を設定します。コネクタの概要については、「AWS SAM コネクタによるリソースに対するアクセス許可の管理」を参照してください。

生成された AWS CloudFormation リソースの詳細については、「AWS::Serverless::Connector を指定したときに生成された AWS CloudFormation リソース」を参照してください。

コネクタに関するフィードバックを提供するには、GitHub の serverless-application-model AWS リポジトリで新しい問題を送信してください。

> **Note**
>
> AWS CloudFormation にデプロイすると、AWS SAM は、AWS SAM リソースを AWS CloudFormation リソースに変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、次のいずれかの構文を使用します。

> **Note**
>
> ほとんどのユースケースでは、埋め込みコネクタ構文を使用することをお勧めします。ソースリソースに埋め込まれているため、読み取りと長期にわたる維持がより容易になっています。ネストされたスタック内のリソースや共有リソースなど、同じ AWS SAM テンプレート

内にないソースリソースを参照する必要がある場合は、AWS::Serverless::Connector 構文を使用します。

埋め込みコネクタ

```
<source-resource-logical-id>:
  Connectors:
    <connector-logical-id>:
      Properties:
        Destination: ResourceReference | List of ResourceReference
        Permissions: List
        SourceReference: SourceReference
```

AWS::Serverless::Connector

```
Type: AWS::Serverless::Connector
Properties:
  Destination: ResourceReference | List of ResourceReference
  Permissions: List
  Source: ResourceReference
```

プロパティ

`Destination`

送信先リソースです。

タイプ: ResourceReference | ResourceReference のリスト

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Permissions`

送信元リソースが送信先リソースで実行できるアクセス許可タイプです。

`Read` には、リソースからデータを読み取ることができる AWS Identity and Access Management (IAM) アクションが含まれます。

`Write` には、リソースへのデータの開始と書き込みが可能な IAM アクションが含まれます。

有効な値: `Read` または `Write`

タイプ: リスト

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Source`

送信元リソースです。`AWS::Serverless::Connector` 構文を使用する場合に必要です。

タイプ: ResourceReference

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceReference`

送信元リソースです。

> **Note**
> ソースリソース用に追加のプロパティを定義するときに、埋め込みコネクタ構文とともに使用します。

タイプ: SourceReference

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

埋め込みコネクタ

次の例では、埋め込みコネクタを使用して、AWS Lambda 関数と Amazon DynamoDB テーブルの間の `Write` データ接続を定義しています。

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyTable:
    Type: AWS::Serverless::SimpleTable
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Write
    ...
```

次の例では、埋め込みコネクタを使用して Read および Write 許可を定義しています。

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
    ...
```

次の例では、埋め込みコネクタを使用して、Id 以外のプロパティを持つソースリソースを定義しています。

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

```
  MyApi:
    Type: AWS::Serverless::Api
    Connectors:
      ApitoLambdaConn:
        Properties:
          SourceReference:
            Qualifier: Prod/GET/foobar
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
    ...
```

AWS::Serverless::Connector

次の例では、AWS::Serverless::Connector リソースを使用して AWS Lambda 関数に Amazon DynamoDB テーブルへの読み書きを実行させます。

```
MyConnector:
  Type: AWS::Serverless::Connector
  Properties:
    Source:
      Id: MyFunction
    Destination:
      Id: MyTable
    Permissions:
      - Read
      - Write
```

次の例では、AWS::Serverless::Connector リソースを使用して Lambda 関数に Amazon SNS トピックへの書き込みをさせ、両方のリソースを同じテンプレートに配置します。

```
MyConnector:
  Type: AWS::Serverless::Connector
  Properties:
    Source:
      Id: MyLambda
    Destination:
      Id: MySNSTopic
    Permissions:
```

```
      - Write
```

次の例では、AWS::Serverless::Connector リソースを使用して Amazon SNS トピックに Lambda 関数への書き込みをさせ、次に Amazon DynamoDB テーブルへの書き込みをさせ、すべてのリソースを同じテンプレートに配置します。

```
Transform: AWS::Serverless-2016-10-31
Resources:
  Topic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint: !GetAtt Function.Arn
          Protocol: lambda

  Function:
    Type: AWS::Serverless::Function
    Properties:
      Runtime: nodejs16.x
      Handler: index.handler
      InlineCode: |
        const AWS = require('aws-sdk');
        exports.handler = async (event, context) => {
          const docClient = new AWS.DynamoDB.DocumentClient();
          await docClient.put({
            TableName: process.env.TABLE_NAME,
            Item: {
              id: context.awsRequestId,
              event: JSON.stringify(event)
            }
          }).promise();
        };
      Environment:
        Variables:
          TABLE_NAME: !Ref Table

  Table:
    Type: AWS::Serverless::SimpleTable

  TopicToFunctionConnector:
    Type: AWS::Serverless::Connector
    Properties:
      Source:
```

```
        Id: Topic
      Destination:
        Id: Function
      Permissions:
        - Write

  FunctionToTableConnector:
    Type: AWS::Serverless::Connector
    Properties:
      Source:
        Id: Function
      Destination:
        Id: Table
      Permissions:
        - Write
```

以下は、上の例から変換された AWS CloudFormation テンプレートです。

```
"FunctionToTableConnectorPolicy": {
  "Type": "AWS::IAM::ManagedPolicy",
  "Metadata": {
    "aws:sam:connectors": {
      "FunctionToTableConnector": {
        "Source": {
          "Type": "AWS::Lambda::Function"
        },
        "Destination": {
          "Type": "AWS::DynamoDB::Table"
        }
      }
    }
  },
  "Properties": {
    "PolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem",
```

```
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate"
          ],
          "Resource": [
            {
              "Fn::GetAtt": [
                "MyTable",
                "Arn"
              ]
            },
            {
              "Fn::Sub": [
                "${DestinationArn}/index/*",
                {
                  "DestinationArn": {
                    "Fn::GetAtt": [
                      "MyTable",
                      "Arn"
                    ]
                  }
                }
              ]
            }
          ]
        }
      ]
    },
    "Roles": [
      {
        "Ref": "MyFunctionRole"
      }
    ]
  }
}
```

ResourceReference

AWS::Serverless::Connector リソースタイプが使用するリソースへの参照。

> **Note**
>
> 同じテンプレート内のリソースには、`Id` を指定します。同じテンプレートにないリソースには、他のプロパティを組み合わせて使用します。詳細については、「AWS SAM コネクタリファレンス」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Arn: String
Id: String
Name: String
Qualifier: String
QueueUrl: String
ResourceId: String
RoleName: String
Type: String
```

プロパティ

`Arn`

リソースの ARN です。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Id`

同じテンプレート内のリソースの論理 ID です。

> **Note**
>
> `Id` が指定されている場合、コネクタが AWS Identity and Access Management (IAM) ポリシーを生成すると、それらのポリシーに関連付けられた IAM ロールがリソース `Id` から推測されます。`Id` が指定されていない場合は、コネクタ用にリソースの `RoleName` を指定して、生成された IAM ポリシーを IAM ロールにアタッチします。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Name`

リソースの名前です。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Qualifier`

範囲を狭めるリソースの修飾子。`Qualifier` はリソース制限の ARN の末尾にある `*` 値を置き換えます。例については、「Lambda 関数を呼び出す API Gateway」を参照してください。

> **Note**
>
> 修飾子の定義はリソースタイプによって異なります。サポートされている送信元リソースタイプおよび送信先リソースタイプの一覧については、「AWS SAM コネクタリファレンス」を参照してください。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`QueueUrl`

Amazon SQS キューの URL です。このプロパティは Amazon SQS リソースにのみ適用されます。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ResourceId`

リソースの ID です。例: API Gateway API ID。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`RoleName`

リソースに関連付けられたロール名です。

> **Note**
>
> `Id` が指定されている場合、コネクタが IAM ポリシーを生成すると、それらのポリシーに関連付けられた IAM ロールがリソース `Id` から推測されます。`Id` が指定されていない場合は、コネクタ用にリソースの `RoleName` を指定して、生成された IAM ポリシーを IAM ロールにアタッチします。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

Type

リソースの AWS CloudFormation タイプです。詳細については、「AWS リソースおよびプロパティタイプのリファレンス」を参照してください。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

Lambda 関数を呼び出す API Gateway

次の例では、AWS::Serverless::Connector リソースを使用して Amazon API Gateway が AWS Lambda 関数を呼び出すことを許可しています。

YAML

```
Transform: AWS::Serverless-2016-10-31
Resources:
  MyRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Action: sts:AssumeRole
            Principal:
              Service: lambda.amazonaws.com
      ManagedPolicyArns:
        - !Sub arn:${AWS::Partition}:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole

  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Role: !GetAtt MyRole.Arn
      Runtime: nodejs16.x
      Handler: index.handler
      Code:
```

```
        ZipFile: |
          exports.handler = async (event) => {
            return {
              statusCode: 200,
              body: JSON.stringify({
                "message": "It works!"
              }),
            };
          };

  MyApi:
    Type: AWS::ApiGatewayV2::Api
    Properties:
      Name: MyApi
      ProtocolType: HTTP

  MyStage:
    Type: AWS::ApiGatewayV2::Stage
    Properties:
      ApiId: !Ref MyApi
      StageName: prod
      AutoDeploy: True

  MyIntegration:
    Type: AWS::ApiGatewayV2::Integration
    Properties:
      ApiId: !Ref MyApi
      IntegrationType: AWS_PROXY
      IntegrationUri: !Sub arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/
functions/${MyFunction.Arn}/invocations
      IntegrationMethod: POST
      PayloadFormatVersion: "2.0"

  MyRoute:
    Type: AWS::ApiGatewayV2::Route
    Properties:
      ApiId: !Ref MyApi
      RouteKey: GET /hello
      Target: !Sub integrations/${MyIntegration}

  MyConnector:
    Type: AWS::Serverless::Connector
    Properties:
      Source: # Use 'Id' when resource is in the same template
```

```
        Type: AWS::ApiGatewayV2::Api
        ResourceId: !Ref MyApi
        Qualifier: prod/GET/hello # Or "*" to allow all routes
      Destination: # Use 'Id' when resource is in the same template
        Type: AWS::Lambda::Function
        Arn: !GetAtt MyFunction.Arn
      Permissions:
        - Write

Outputs:
  Endpoint:
    Value: !Sub https://${MyApi}.execute-api.${AWS::Region}.${AWS::URLSuffix}/prod/
hello
```

SourceReference

AWS::Serverless::Connector リソースタイプが使用するソースリソースへの参照。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Qualifier: String
```

プロパティ

`Qualifier`

範囲を狭めるリソースの修飾子。`Qualifier` はリソース制限の ARN の末尾にある `*` 値を置き換えます。

> **Note**
>
> 修飾子の定義はリソースタイプによって異なります。サポートされている送信元リソースタイプおよび送信先リソースタイプの一覧については、「AWS SAM コネクタリファレンス」を参照してください。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

次の例では、埋め込みコネクタを使用して、**Id** 以外のプロパティを持つソースリソースを定義しています。

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Connectors:
      ApitoLambdaConn:
        Properties:
          SourceReference:
            Qualifier: Prod/GET/foobar
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
    ...
```

AWS::Serverless::Function

AWS Lambda 関数、 AWS Identity and Access Management （ IAM) 実行ロール、および関数をトリガーするイベントソースマッピングを作成します。

AWS::Serverless::Function リソースは Metadataリソース属性もサポートしているため、アプリケーションに必要なカスタムランタイムを構築する AWS SAM ように に指示できます。カスタムランタイムの構築の詳細については、「でのカスタムランタイムを使用した Lambda 関数の構築 AWS SAM」を参照してください。

> **Note**
>
> にデプロイすると AWS CloudFormation、 は AWS SAM リソースを AWS CloudFormation リソース AWS SAM に変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
Type: AWS::Serverless::Function
Properties:
  Architectures: List
  AssumeRolePolicyDocument: JSON
  AutoPublishAlias: String
  AutoPublishAliasAllProperties: Boolean
  AutoPublishCodeSha256: String
  CodeSigningConfigArn: String
  CodeUri: String | FunctionCode
  DeadLetterQueue: Map | DeadLetterQueue
  DeploymentPreference: DeploymentPreference
  Description: String
  Environment: Environment
  EphemeralStorage: EphemeralStorage
  EventInvokeConfig: EventInvokeConfiguration
  Events: EventSource
  FileSystemConfigs: List
  FunctionName: String
  FunctionUrlConfig: FunctionUrlConfig
  Handler: String
  ImageConfig: ImageConfig
  ImageUri: String
  InlineCode: String
  KmsKeyArn: String
  Layers: List
  LoggingConfig: LoggingConfig
  MemorySize: Integer
  PackageType: String
```

```
  PermissionsBoundary: String
  Policies: String | List | Map
  PropagateTags: Boolean
  ProvisionedConcurrencyConfig: ProvisionedConcurrencyConfig
  RecursiveLoop: String
  ReservedConcurrentExecutions: Integer
  Role: String
  RolePath: String
  Runtime: String
  RuntimeManagementConfig: RuntimeManagementConfig
  SnapStart: SnapStart
  SourceKMSKeyArn: String
  Tags: Map
  Timeout: Integer
  Tracing: String
  VersionDescription: String
  VpcConfig: VpcConfig
```

プロパティ

`Architectures`

関数の命令セットアーキテクチャ。

このプロパティの詳細については、AWS Lambda デベロッパーガイドの「Lambda 命令セットアーキテクチャ」を参照してください。

有効な値: `x86_64` または `arm64` のいずれか。

タイプ: リスト

必須: いいえ

デフォルト: `x86_64`

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Architectures`プロパティに直接渡されます。

`AssumeRolePolicyDocument`

この関数 AssumeRolePolicyDocument 用に作成されたデフォルトの `Role` を追加します。このプロパティが指定されていない場合、 はこの関数のデフォルトの継承ロール AWS SAM を追加します。

タイプ: JSON

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::IAM::Role` resource の `AssumeRolePolicyDocument` プロパティに似ています。 はこのプロパティを、この関数用に生成されたIAMロール AWS SAM に追加します。この関数にロールの Amazon リソースネーム (ARN) が指定されている場合、このプロパティは何もしません。

`AutoPublishAlias`

Lambda エイリアスの名前です。Lambda エイリアスの詳細については、AWS Lambda デベロッパーガイドの「Lambda 関数のエイリアス」を参照してください。このプロパティを使用する例については、「AWS SAM を使用したサーバーレスアプリケーションの段階的なデプロイ」を参照してください。

AWS SAM 生成 AWS::Lambda::Version および AWS::Lambda::Alias このプロパティが設定されている場合の リソース。このシナリオの詳細については、「AutoPublishAlias プロパティが指定されている」を参照してください。生成された AWS CloudFormation リソースの一般的な情報については、「」を参照してくださいAWS SAM 向けに生成された AWS CloudFormation リソース。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`AutoPublishAliasAllProperties`

新しい `AWS::Lambda::Version` が作成されるタイミングを指定します。`true` の場合、Lambda 関数のプロパティが変更されると、新しい Lambda バージョンが作成されます。`false` の場合、次のプロパティのいずれかが変更された場合にのみ、新しい Lambda バージョンが作成されます。

- `Environment`、`MemorySize`、`SnapStart`
- `Code` プロパティの更新を伴う変更 (`CodeDict`、`ImageUri`、`InlineCode` など)。

このプロパティでは `AutoPublishAlias` を定義する必要があります。

`AutoPublishSha256` も指定されている場合、その動作は `AutoPublishAliasAllProperties: true` よりも優先されます。

タイプ: ブール

必須: いいえ

デフォルト値: `false`

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`AutoPublishCodeSha256`

使用した場合、`CodeUri` 値とともに機能し、新しい Lambda バージョンを発行する必要があるかどうかを判断します。このプロパティは、しばしば次のようなデプロイの問題を解決するために使用されます: Amazon S3 のロケーションに保存されているデプロイパッケージが更新済みの Lambda 関数コードを使用する新しいデプロイパッケージに置き換えられたものの (新しいデプロイパッケージが新しい Amazon S3 のロケーションにアップロードされていて、`CodeUri` が新しい場所に変更されるのと反対に)、`CodeUri` プロパティがそのまま変更されていない。

この問題は、次の特性を持つ AWS SAM テンプレートによってマークされます。

- `DeploymentPreference` オブジェクトが、(AWS SAM を使用したサーバーレスアプリケーションの段階的なデプロイ で説明されているように) 段階的なデプロイ用に設定されている
- `AutoPublishAlias` プロパティが設定されていてもデプロイ間では変更されていない
- `CodeUri` プロパティが設定されていてもデプロイ間では変更されていない。

このシナリオでは、`AutoPublishCodeSha256` を更新することで新しい Lambda バージョンが正常に作成されます。ただし、Amazon S3 にデプロイされた新しい関数コードは認識されません。新しい関数コードを認識するには、Amazon S3 バケットのバージョニングを使用することを検討してください。Lambda 関数の `Version` プロパティを指定し、常に最新のデプロイパッケージを使用するようにバケットを設定します。

このシナリオで段階的なデプロイを正常にトリガーするには、`AutoPublishCodeSha256` に一意の値を提供する必要があります。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`CodeSigningConfigArn`

ARN の AWS::Lambda::CodeSigningConfig リソース。この関数のコード署名を有効にするために使用されます。コード署名の詳細については、「AWS SAM アプリケーションのコード署名を設定する」を参照してください。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `CodeSigningConfigArn`プロパティに直接渡されます。

`CodeUri`

関数のコード 以下のような値を設定できます。

- 関数の Amazon S3 URI。例えば、`s3://bucket-123456789/sam-app/1234567890abcdefg` と指定します。
- 関数へのローカルパス。例えば、`hello_world/` と指定します。
- FunctionCode オブジェクト。

> **Note**
>
> 関数の Amazon S3 URIまたは FunctionCode オブジェクトを指定する場合は、有効な Lambda デプロイパッケージを参照する必要があります。
> ローカルファイルパスを指定する場合は、 AWS SAM CLI デプロイ時にローカルファイルをアップロードします。詳細については、「デプロイ時にローカルファイル AWS SAM をアップロードする方法」を参照してください。
> `CodeUri` プロパティで組み込み関数を使用する場合、 は値を正しく解析 AWS SAM できません。代わりに AWS::LanguageExtensions transform の使用を検討してください。

タイプ: [文字列 | FunctionCode]

必須: 条件的。`PackageType` が `Zip` に設定されている場合、`CodeUri` または `InlineCode` のいずれかが必須です。

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Code`プロパティに似ています。ネストされた Amazon S3 プロパティには異なる名前が付けられています。

`DeadLetterQueue`

Lambda が処理できないイベントを送信する Amazon Simple Notification Service (Amazon SNS) トピックまたは Amazon Simple Queue Service (Amazon SQS) キューを設定します。デッドレターキュー機能の詳細については、「AWS Lambda デベロッパーガイド」の「デッドレターキュー」を参照してください。

> **Note**
>
> Lambda 関数のイベントソースが Amazon SQSキューの場合は、Lambda 関数ではなく、ソースキューのデッドレターキューを設定します。関数用に設定するデッドレターキューは、イベントソースキューではなく、関数の非同期呼び出しキューに使用されます。

タイプ: Map | DeadLetterQueue

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `DeadLetterConfig`プロパティに似ています。 AWS CloudFormation タイプは から派生しますが`TargetArn`、 では AWS SAM タイプを と一緒に渡す必要があります`TargetArn`。

`DeploymentPreference`

段階的な Lambda デプロイを有効にする設定です。

`DeploymentPreference` オブジェクトが指定されている場合、 は AWS SAM を作成します。 AWS::CodeDeploy::Application と呼ばれる `ServerlessDeploymentApplication` (スタックごとに 1 つ)、 AWS::CodeDeploy::DeploymentGroup と呼ばれる *`<function-logical-id>`*`DeploymentGroup`、および AWS::IAM::Role という名前の `CodeDeployServiceRole`。

タイプ: DeploymentPreference

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

その他の参照資料: このプロパティの詳細については、「AWS SAM を使用したサーバーレスアプリケーションの段階的なデプロイ」を参照してください。

`Description`

関数の説明です。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Description`プロパティに直接渡されます。

`Environment`

ランタイム環境の設定です。

タイプ: Environment

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Environment`プロパティに直接渡されます。

`EphemeralStorage`

`/tmp` の Lambda 関数で使用可能なディスク容量を MB 単位で指定するオブジェクト。

このプロパティの詳細については、「AWS Lambda デベロッパーガイド」の「Lambda 実行環境」を参照してください。

タイプ: EphemeralStorage

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `EphemeralStorage`プロパティに直接渡されます。

`EventInvokeConfig`

Lambda 関数でのイベントの呼び出し設定を説明するオブジェクトです。

タイプ: EventInvokeConfiguration

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`Events`

この関数をトリガーするイベントを指定します。イベントは、1 つのタイプと、そのタイプに依存する一連のプロパティで構成されます。

タイプ: EventSource

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`FileSystemConfigs`

Amazon Elastic File System (Amazon EFS) ファイルシステムの接続設定を指定するFileSystemConfigオブジェクトのリスト。

テンプレートに が含まれている場合 AWS::EFS::MountTarget リソースでは、マウントターゲットが関数の前に作成または更新されるように、`DependsOn`リソース属性も指定する必要があります。

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `FileSystemConfigs`プロパティに直接渡されます。

`FunctionName`

関数の名前です。名前を指定しない場合は、一意の名前が生成されます。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `FunctionName`プロパティに直接渡されます。

`FunctionUrlConfig`

関数 を記述するオブジェクトURL。関数URLは、関数を呼び出すために使用できるHTTPSエンドポイントです。

詳細については、「 AWS Lambda デベロッパーガイド」の「 関数URLs」を参照してください。

タイプ: FunctionUrlConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`Handler`

実行を開始するために呼び出されるコード内の関数です。このプロパティが必要になるのは、`PackageType` プロパティが `Zip` に設定されている場合のみです。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Handler`プロパティに直接渡されます。

`ImageConfig`

Lambda のコンテナイメージ設定に使用されるオブジェクトです。詳細については、AWS Lambda デベロッパーガイドの「Lambda でのコンテナイメージの使用」を参照してください。

タイプ: ImageConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `ImageConfig`プロパティに直接渡されます。

`ImageUri`

Lambda 関数URIのコンテナイメージの Amazon Elastic Container Registry (Amazon ECR) リポジトリの 。このプロパティは、`PackageType` プロパティが `Image` に設定されている場合にのみ適用され、それ以外の場合は無視されます。詳細については、AWS Lambda デベロッパーガイドの「Lambda でのコンテナイメージの使用」を参照してください。

> **Note**
>
> `PackageType` プロパティが に設定されている場合`Image`、 `ImageUri`は必須です。または、 AWS SAM テンプレートファイルに必要な`Metadata`エントリを使用してアプリ

ケーションを構築する必要があります。詳細については、「AWS SAM を使用したデフォルトのビルド」を参照してください。

必要な Metadata エントリを使用してアプリケーションを構築することは、ImageUri よりも優先されるので、両方を指定すれば ImageUri は無視されます。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Function Code データ型の ImageUriプロパティに直接渡されます。

`InlineCode`

テンプレートに直接記述された Lambda 関数コードです。このプロパティは、PackageType プロパティが Zip に設定されている場合にのみ適用され、それ以外の場合は無視されます。

> **Note**
>
> PackageType が Zip (デフォルト) に設定されていると、CodeUri または InlineCode のいずれかが必要になります。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Function Code データ型の ZipFileプロパティに直接渡されます。

`KmsKeyArn`

Lambda が関数ARNの環境変数の暗号化と復号に使用する AWS Key Management Service (AWS KMS) キーの 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Functionリソースの KmsKeyArnプロパティに直接渡されます。

`Layers`

この関数`LayerVersionARNs`が使用する のリスト。ここで指定されている順序は、Lambda 関数の実行時にそれらがインポートされる順序です。

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Layers`プロパティに直接渡されます。

`LoggingConfig`

関数の Amazon CloudWatch Logs 構成設定。

タイプ: LoggingConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `LoggingConfig`プロパティに直接渡されます。

`MemorySize`

関数の各呼び出しに割り当てられるメモリのサイズ (MB) です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `MemorySize`プロパティに直接渡されます。

`PackageType`

Lambda 関数のデプロイパッケージタイプです。詳細については、AWS Lambda デベロッパーガイドの「Lambda デプロイパッケージ」を参照してください。

注意:

1. このプロパティが `Zip` (デフォルト) に設定されている場合は、`CodeUri` または `InlineCode` が適用され、`ImageUri` は無視されます。

2. このプロパティが `Image` に設定されている場合は、`ImageUri` のみが適用され、`CodeUri` と`InlineCode` は無視されます。関数のコンテナイメージを保存するために必要な Amazon

ECRリポジトリは、 によって自動的に作成できます。 AWS SAM CLI。 詳細については、「」を参照してくださいsam deploy。

有効な値: `Zip` または `Image`

タイプ : 文字列

必須: いいえ

デフォルト: `Zip`

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `PackageType`プロパティに直接渡されます。

`PermissionsBoundary`

この関数の実行ロールに使用するアクセス許可ARNの境界の 。このプロパティは、ユーザーのためにロールが生成される場合にのみ機能します。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::IAM::Role`リソースの `PermissionsBoundary`プロパティに直接渡されます。

`Policies`

この関数の許可ポリシー。ポリシーは、関数の default AWS Identity and Access Management (IAM) 実行ロールに追加されます。

このプロパティは、単一の値または値のリストを受け入れます。使用できる値は次のとおりです。

- AWS SAMポリシーテンプレート.
- - ARN AWS 管理ポリシーまたはカスタマー管理ポリシーの 。
- 次のリストからの AWS 管理ポリシーの名前。
- でフォーマットされたインラインIAMポリシー YAML マップとしての 。

> **Note**
>
> `Role` プロパティを設定する場合、このプロパティは無視されます。

タイプ: 文字列 | リスト | マップ

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::IAM::Roleリソースの Policiesプロパティに似ています。

PropagateTags

AWS::Serverless::Function が生成したリソースに Tags プロパティからのタグを渡すかどうかを指定します。True を指定して、生成されたリソースにタグを伝播します。

タイプ: ブール

必須: いいえ

デフォルト: False

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

ProvisionedConcurrencyConfig

関数のエイリアスのプロビジョニングされた同時実行設定です。

> **Note**
>
> ProvisionedConcurrencyConfig を指定できるのは、AutoPublishAlias が設定されている場合のみです。それ以外の場合は、エラーが発生します。

タイプ: ProvisionedConcurrencyConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Aliasリソースの ProvisionedConcurrencyConfigプロパティに直接渡されます。

RecursiveLoop

関数の再帰的ループ検出設定のステータス。

この値が Allow に設定されている場合、Lambda が再帰的ループの一部として呼び出されている関数を検出しても、いずれのアクションも実行されません。

この値が `Terminate` に設定されている場合、Lambda が再帰的ループの一部として呼び出されている関数を検出すると、その関数の呼び出しが停止され通知が送信されます。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `RecursiveLoop`プロパティに直接渡されます。

`ReservedConcurrentExecutions`

関数用に予約する同時実行の最大数です。

このプロパティの詳細については、AWS Lambda デベロッパーガイドの「Lambda 関数スケーリング」を参照してください。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `ReservedConcurrentExecutions`プロパティに直接渡されます。

`Role`

この関数の実行ARNIAMロールとして使用する ロールの 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Role`プロパティに似ています。これは では必須 AWS CloudFormation ですが、 では必須ではありません AWS SAM。ロールが指定されていない場合は、*`<function-logical-id>`*`Role` の論理 ID を持つロールが作成されます。

`RolePath`

関数IAMの実行ロールへのパス。

このプロパティは、ユーザーのためにロールが生成される場合にのみ機能します。`Role` プロパティでロールが指定されている場合は使用しないでください。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation 互換性: このプロパティは、 `AWS::IAM::Role`リソースの `Path`プロパティに直接渡されます。

`Runtime`

関数のランタイムの識別子です。このプロパティが必要になるのは、PackageType プロパティが Zip に設定されている場合のみです。

> **Note**
>
> このプロパティのprovided識別子を指定する場合、Metadataリソース属性を使用して、この関数が必要とするカスタムランタイムを構築する AWS SAM ように に指示できます。カスタムランタイムの構築の詳細については、「でのカスタムランタイムを使用した Lambda 関数の構築 AWS SAM」を参照してください。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `Runtime`プロパティに直接渡されます。

`RuntimeManagementConfig`

ランタイム環境の更新、ロールバック動作、特定のランタイムバージョンの選択など、Lambda 関数のランタイム管理オプションを設定します。詳細については、「AWS Lambda デベロッパーガイド」の「Lambda ランタイム更新」を参照してください。

タイプ: RuntimeManagementConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `RuntimeManagementConfig`プロパティに直接渡されます。

`SnapStart`

新しい Lambda 関数バージョンのスナップショットを作成します。スナップショットは、すべての依存関係を含む、初期化された関数のキャッシュされた状態です。関数は一度だけ初期化され、キャッシュされた状態は将来の すべての呼び出しで再利用されるため、関数の初期化が必

要な回数が減ることでアプリケーションのパフォーマンスが向上します。詳細については、「AWS Lambda デベロッパーガイド」の「Lambda によるスタートアップパフォーマンスの向上 SnapStart」を参照してください。

タイプ: SnapStart

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Functionリソースの SnapStartプロパティに直接渡されます。

SourceKmsKeyArn

顧客のZIP関数コードを暗号化するARNために使用されるKMSキーを表します。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Function Code データ型の SourceKmsKeyArnプロパティに直接渡されます。

Tags

この関数に追加されるタグを指定するマップ (文字列対文字列) です。タグの有効なキーと値の詳細については、「AWS Lambda デベロッパーガイド」の「タグのキーと値の要件」を参照してください。

スタックが作成されると、 はこの Lambda 関数と、この関数用に生成されたデフォルトのロールにlambda:createdBy:SAMタグ AWS SAM を自動的に追加します。

タイプ: マップ

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Functionリソースの Tagsプロパティに似ています。の Tagsプロパティ AWS SAM は、キーと値のペアで構成されます (AWS CloudFormation このプロパティはTagオブジェクトのリストで構成されます) 。また、 は、この Lambda 関数と、この関数用に生成されたデフォルトのロールにlambda:createdBy:SAMタグ AWS SAM を自動的に追加します。

Timeout

関数が停止されるまでの最大実行時間 (秒) です。

タイプ: 整数

必須: いいえ

デフォルト: 3

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Functionリソースの Timeoutプロパティに直接渡されます。

Tracing

関数の X-Ray トレーシングモードを指定する文字列です。

- Active – 関数の X-Ray トレーシングを有効にします。
- Disabled – 関数の X-Ray を無効にします。
- PassThrough – 関数の X-Ray トレーシングを有効にします。サンプリングデシジョンはダウンストリームサービスに委任されます。

Active または PassThrough が指定されており、Role プロパティが設定されていない場合、AWS SAM は、ユーザー用に作成する Lambda 実行ロールに arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess ポリシーを追加します。

X-Ray の詳細については、「 AWS Lambda デベロッパーガイド」の「 AWS Lambda で を使用する AWS X-Ray」を参照してください。

有効な値: [Active|Disabled|PassThrough]

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Functionリソースの TracingConfigプロパティに似ています。

VersionDescription

新しい Lambda バージョンリソースに追加される Description フィールドを指定します。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::Versionリソースの Descriptionプロパティに直接渡されます。

`VpcConfig`

この関数が仮想プライベートクラウド () 内のプライベートリソースにアクセスできるようにする設定VPC。

タイプ: VpcConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::Function`リソースの `VpcConfig`プロパティに直接渡されます。

戻り値

参照番号

このリソースの論理 ID が `Ref` 組み込み関数に提供されると、基盤となる Lambda 関数のリソース名が返されます。

`Ref` 関数の使用方法の詳細については、「AWS CloudFormation ユーザーガイド」の「`Ref`」を参照してください。

Fn::GetAtt

`Fn::GetAtt` は、このタイプの指定された属性の値を返します。利用可能な属性とサンプル戻り値は以下のとおりです。

`Fn::GetAtt` の使用の詳細については、「AWS CloudFormation ユーザーガイド」の「`Fn::GetAtt`」を参照してください。

`Arn`

基盤となる Lambda 関数ARNの 。

例

シンプルな関数

以下は、AWS::Serverless::Function パッケージタイプ (デフォルト) の `Zip` リソースと、Amazon S3 バケット内にある関数コードの基本的な例です。

YAML

```
Type: AWS::Serverless::Function
Properties:
  Handler: index.handler
  Runtime: python3.9
  CodeUri: s3://bucket-name/key-name
```

関数プロパティの例

以下は、InlineCode、Layers、Tracing、Policies、Amazon EFS、および Api イベントソースを使用する、AWS::Serverless::Function パッケージタイプ (デフォルト) の Zip の例です。

YAML

```
Type: AWS::Serverless::Function
DependsOn: MyMountTarget        # This is needed if an AWS::EFS::MountTarget resource
 is declared for EFS
Properties:
  Handler: index.handler
  Runtime: python3.9
  InlineCode: |
    def handler(event, context):
      print("Hello, world!")
  ReservedConcurrentExecutions: 30
  Layers:
    - Ref: MyLayer
  Tracing: Active
  Timeout: 120
  FileSystemConfigs:
    - Arn: !Ref MyEfsFileSystem
      LocalMountPath: /mnt/EFS
  Policies:
    - AWSLambdaExecute
    - Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Action:
            - s3:GetObject
            - s3:GetObjectACL
          Resource: 'arn:aws:s3:::amzn-s3-demo-bucket/*'
  Events:
    ApiEvent:
```

```
      Type: Api
      Properties:
        Path: /path
        Method: get
```

ImageConfig の例

以下は、Image パッケージタイプの Lambda 関数向けの ImageConfig の例です。

YAML

```
HelloWorldFunction:
  Type: AWS::Serverless::Function
  Properties:
    PackageType: Image
    ImageUri: account-id.dkr.ecr.region.amazonaws.com/ecr-repo-name:image-name
    ImageConfig:
      Command:
        - "app.lambda_handler"
      EntryPoint:
        - "entrypoint1"
      WorkingDirectory: "workDir"
```

RuntimeManagementConfig の例

現在の動作に従ってランタイム環境を更新するように設定された Lambda 関数:

```
TestFunction
  Type: AWS::Serverless::Function
  Properties:
    ...
    Runtime: python3.9
    RuntimeManagementConfig:
      UpdateRuntimeOn: Auto
```

関数が更新されたときにランタイム環境を更新するように設定された Lambda 関数:

```
TestFunction
  Type: AWS::Serverless::Function
  Properties:
    ...
```

```
    Runtime: python3.9
    RuntimeManagementConfig:
      UpdateRuntimeOn: FunctionUpdate
```

ランタイム環境を手動で更新するように構成された Lambda 関数:

```
TestFunction
  Type: AWS::Serverless::Function
  Properties:
    ...
    Runtime: python3.9
    RuntimeManagementConfig:
      RuntimeVersionArn: arn:aws:lambda:us-
east-1::runtime:4c459dd0104ee29ec65dcad056c0b3ddbe20d6db76b265ade7eda9a066859b1e
      UpdateRuntimeOn: Manual
```

SnapStart の例

将来のバージョンで SnapStart が有効になっている Lambda 関数の例 :

```
TestFunc
  Type: AWS::Serverless::Function
  Properties:
    ...
    SnapStart:
      ApplyOn: PublishedVersions
```

DeadLetterQueue

AWS Lambda (Lambda) がイベントを処理できないときにそれらを送信する SQS キューまたは SNS トピックを指定します。デッドレターキュー機能の詳細については、「AWS Lambda デベロッパーガイド」の「デッドレターキュー」を参照してください。

SAM は、Lambda 関数実行ロールに適切なアクセス許可を自動的に追加して、Lambda サービスにリソースへのアクセスを許可します。sq:SendMessage が SQS キューに、sns:Publish が SNS トピックに追加されます。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  TargetArn: String
  Type: String
```

プロパティ

`TargetArn`

Amazon SQS キューまたは Amazon SNS トピックの Amazon リソースネーム (ARN) です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::Function DeadLetterConfig` データ型の `TargetArn` プロパティに直接渡されます。

`Type`

デッドレターキューのタイプです。

有効な値:`SNS`、`SQS`|

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

DeadLetterQueue

SNS トピックのデッドレターキューの例です。

YAML

```
DeadLetterQueue:
  Type: SNS
  TargetArn: arn:aws:sns:us-east-2:123456789012:my-topic
```

DeploymentPreference

段階的な Lambda デプロイを有効にする設定を指定します。段階的な Lambda デプロイの設定の詳細については、「AWS SAM を使用したサーバーレスアプリケーションの段階的なデプロイ」を参照してください。

> **Note**
>
> DeploymentPreference オブジェクトを使用するには、AWS::Serverless::Function で AutoPublishAlias を指定する必要があります。指定しない場合は、エラーが発生します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Alarms: List
Enabled: Boolean
Hooks: Hooks
PassthroughCondition: Boolean
Role: String
TriggerConfigurations: List
Type: String
```

プロパティ

`Alarms`

デプロイによって発生したエラーによってトリガーされる CloudWatch アラームのリストです。

このプロパティは、`Fn::If` 組み込み関数を受け入れます。`Fn::If` を使用するテンプレートの例については、このトピックの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Enabled`

このデプロイプリファレンスが有効になっているかどうかです。

タイプ: ブール

必須: いいえ

デフォルト: True

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Hooks`

トラフィックシフトの前後に実行される検証 Lambda 関数です。

タイプ: Hooks

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`PassthroughCondition`

True の場合、このデプロイメントプリファレンスが有効な場合、関数の条件が、生成された CodeDeploy リソースに渡されます。通常、これを True に設定する必要があります。そうしないと、関数の条件が False に解決された場合でも、CodeDeploy リソースが作成されます。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Role`

CodeDeploy がトラフィックの移行に使用する IAM ロール ARN です。これが提供されている場合、IAM ロールは作成されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`TriggerConfigurations`

デプロイグループに関連付けるトリガー設定のリストです。ライフサイクルイベントについて SNS トピックに通知するために使用されます。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::CodeDeploy::DeploymentGroup` リソースの `TriggerConfigurations` プロパティに直接渡されます。

`Type`

現在、デプロイタイプには Linear と Canary の 2 つのカテゴリがあります。使用可能なデプロイタイプの詳細については、「AWS SAM を使用したサーバーレスアプリケーションの段階的なデプロイ」を参照してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

pre-traffic hook と post-traffic hook を使用する DeploymentPreference です。

pre-traffic hook と post-traffic hook が含まれるデプロイプリファレンスの例です。

YAML

```
DeploymentPreference:
  Enabled: true
  Type: Canary10Percent10Minutes
  Alarms:
    - !Ref: AliasErrorMetricGreaterThanZeroAlarm
```

```
    - !Ref: LatestVersionErrorMetricGreaterThanZeroAlarm
  Hooks:
    PreTraffic:
      !Ref: PreTrafficLambdaFunction
    PostTraffic:
      !Ref: PostTrafficLambdaFunction
```

Fn::If 組み込み関数を使用する DeploymentPreference

アラームの設定に `Fn::If` を使用するデプロイプリファレンスの例です。この例では、`MyCondition` が `true` の場合は `Alarm1` が設定され、`MyCondition` が `false` の場合は `Alarm2` と `Alarm5` が設定されます。

YAML

```
DeploymentPreference:
  Enabled: true
  Type: Canary10Percent10Minutes
  Alarms:
    Fn::If:
      - MyCondition
      - - Alarm1
      - - Alarm2
        - Alarm5
```

Hooks

トラフィックシフトの前後に実行される検証 Lambda 関数です。

> **Note**
>
> このプロパティで参照される Lambda 関数は、生成される AWS::Lambda::Alias リソースの `CodeDeployLambdaAliasUpdate` オブジェクトを設定します。詳細については、AWS CloudFormation ユーザーガイドの「CodeDeployLambdaAliasUpdate Policy」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  PostTraffic: String
  PreTraffic: String
```

プロパティ

`PostTraffic`

トラフィックの移行後に実行される Lambda 関数です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`PreTraffic`

トラフィックの移行前に実行される Lambda 関数 です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

フック

フック関数の例

YAML

```
Hooks:
  PreTraffic:
    Ref: PreTrafficLambdaFunction
  PostTraffic:
```

```
    Ref: PostTrafficLambdaFunction
```

EventInvokeConfiguration

非同期の Lambda エイリアスまたはバージョン呼び出しの設定オプションです。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DestinationConfig: EventInvokeDestinationConfiguration
MaximumEventAgeInSeconds: Integer
MaximumRetryAttempts: Integer
```

プロパティ

`DestinationConfig`

Lambda がイベントを処理した後のイベントの送信先を指定する構成オブジェクト。

タイプ: EventInvokeDestinationConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventInvokeConfig` リソースの `DestinationConfig` プロパティに似ています。SAM には、CloudFormation には存在しない追加のパラメータ「Type」が必要です。

`MaximumEventAgeInSeconds`

Lambda が処理のために関数に送信するリクエストの最大存続時間です。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventInvokeConfig` リソースの `MaximumEventAgeInSeconds` プロパティに直接渡されます。

`MaximumRetryAttempts`

関数がエラーを返すまでの最大再試行回数です。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventInvokeConfig` リソースの `MaximumRetryAttempts` プロパティに直接渡されます。

例

MaximumEventAgeInSeconds

MaximumEventAgeInSeconds の例

YAML

```
EventInvokeConfig:
  MaximumEventAgeInSeconds: 60
  MaximumRetryAttempts: 2
  DestinationConfig:
    OnSuccess:
      Type: SQS
      Destination: arn:aws:sqs:us-west-2:012345678901:my-queue
    OnFailure:
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn
```

EventInvokeDestinationConfiguration

Lambda がイベントを処理した後のイベントの送信先を指定する構成オブジェクトです。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
  OnFailure: OnFailure
```

```
  OnSuccess: OnSuccess
```

プロパティ

`OnFailure`

処理が失敗したイベントの送信先です。

タイプ: OnFailure

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventInvokeConfig`リソースの `OnFailure`プロパティに似ています。追加の SAM専用プロパティ`Type`である が必要です。

`OnSuccess`

正常に処理されたイベントの送信先です。

タイプ: OnSuccess

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventInvokeConfig`リソースの `OnSuccess`プロパティに似ています。追加の SAM専用プロパティ`Type`である が必要です。

例

OnSuccess

OnSuccess 例

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SQS
      Destination: arn:aws:sqs:us-west-2:012345678901:my-queue
    OnFailure:
```

```
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn
```

OnFailure

処理が失敗したイベントの送信先です。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
  Destination: String
  Type: String
```

プロパティ

`Destination`

送信先リソースの Amazon リソースネーム (ARN) 。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventInvokeConfig`リソースの `OnFailure`プロパティに似ています。 SAMは、このプロパティで参照されるリソースにアクセスするために、この関数に関連付けられた自動生成されたIAMロールに必要なアクセス許可を追加します。

その他の注意点: タイプが Lambda/ の場合EventBridge、送信先が必要です。

`Type`

送信先で参照されるリソースのタイプです。サポートされているタイプは、`SQS`、`SNS`、`S3`、`Lambda`、および ですEventBridge。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

その他の注意点: タイプが SQS/SNS で、`Destination`プロパティが空白のままの場合、SQS/SNS リソースは によって自動的に生成されますSAM。リソースを参照するには、*`<function-logical-id>`*`.DestinationQueue` の SQSまたは *`<function-logical-id>`*`.DestinationTopic`の を使用しますSNS。タイプが Lambda/ の場合EventBridge、`Destination` は必須です。

例

EventInvoke SQSおよび Lambda 送信先を使用した設定例

この例では、SQS OnSuccess 設定に Destination SAM が指定されていないため、暗黙的にSQSキューを作成し、必要なアクセス許可を追加します。また、この例では、テンプレートファイルで宣言された Lambda リソースの送信先が OnFailure 設定で指定されているため、 はこの Lambda 関数に、送信先 Lambda 関数を呼び出すために必要なアクセス許可SAMを追加します。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SQS
    OnFailure:
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn  # Arn of a Lambda function declared
 in the template file.
```

EventInvoke SNS送信先の設定例

この例では、 OnSuccess 設定のテンプレートファイルで宣言された SNSトピックに Destination が指定されています。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SNS
```

```
      Destination:
        Ref: DestinationSNS       # Arn of an SNS topic declared in the tempate file
```

OnSuccess

正常に処理されたイベントの送信先です。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
Destination: String
Type: String
```

プロパティ

`Destination`

送信先リソースの Amazon リソースネーム (ARN) 。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventInvokeConfig`リソースの `OnSuccess`プロパティに似ています。 SAMは、このプロパティで参照されるリソースにアクセスするために必要なアクセス許可を、この関数に関連付けられた自動生成されたIAMロールに追加します。

その他の注意点: タイプが Lambda/ の場合EventBridge、送信先が必要です。

`Type`

送信先で参照されるリソースのタイプです。サポートされているタイプは、`SQS`、`SNS`、`S3`、`Lambda`、および ですEventBridge。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

その他の注意点: タイプが SQS/SNS で、`Destination`プロパティが空白のままの場合、SQS/SNS リソースは によって自動的に生成されますSAM。リソースを参照するには、*`<function-logical-id>`*`.DestinationQueue` の SQSまたは *`<function-logical-id>`*`.DestinationTopic`の を使用しますSNS。タイプが Lambda/ の場合EventBridge、`Destination` は必須です。

例

EventInvoke SQSおよび Lambda 送信先を使用した設定例

この例では、SQS OnSuccess 設定に Destination SAM が指定されていないため、暗黙的にSQSキューを作成し、必要なアクセス許可を追加します。また、この例では、テンプレートファイルで宣言された Lambda リソースの送信先が OnFailure 設定で指定されているため、 はこの Lambda 関数に、送信先 Lambda 関数を呼び出すために必要なアクセス許可SAMを追加します。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SQS
    OnFailure:
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn  # Arn of a Lambda function declared
 in the template file.
```

EventInvoke SNS送信先の設定例

この例では、 OnSuccess 設定のテンプレートファイルで宣言された SNSトピックに Destination が与えられます。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SNS
      Destination:
```

```
        Ref: DestinationSNS       # Arn of an SNS topic declared in the tempate file
```

EventSource

関数をトリガーするイベントのソースを説明するオブジェクトです。各イベントは、1 つのタイプと、そのタイプに依存する一連のプロパティで構成されます。各イベントソースのプロパティの詳細については、そのタイプに対応するトピックを参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Properties: AlexaSkill | Api | CloudWatchEvent | CloudWatchLogs | Cognito
 | DocumentDB | DynamoDB | EventBridgeRule | HttpApi | IoTRule | Kinesis | MQ | MSK
 | S3 | Schedule | ScheduleV2 | SelfManagedKafka | SNS | SQS
  Type: String
```

プロパティ

`Properties`

このイベントマッピングのプロパティを説明するオブジェクトです。プロパティのセットは、定義されたタイプに準拠する必要があります。

タイプ: AlexaSkill | Api | CloudWatchEvent | CloudWatchLogs | Cognito | DocumentDB | DynamoDB | EventBridgeRule | HttpApi | IoTRule | Kinesis | MQ | MSK | S3 | Schedule | ScheduleV2 | SelfManagedKafka | SNS | SQS

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Type`

イベントタイプです。

有効な値:
`AlexaSkill、Api、CloudWatchEvent、CloudWatchLogs、Cognito、DocumentDB、DynamoDB、`

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

APIEvent

API Event の使用例

YAML

```
ApiEvent:
  Type: Api
  Properties:
    Method: get
    Path: /group/{user}
    RestApiId:
      Ref: MyApi
```

AlexaSkill

`AlexaSkill` イベントソースタイプを説明するオブジェクトです。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  SkillId: String
```

プロパティ

`SkillId`

Alexa Skill の Alexa Skill ID です。Skill ID の詳細については、Alexa Skills Kit ドキュメントの「Lambda 関数のトリガー設定」を参照してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

AlexaSkillTrigger

Alexa Skill イベントの例

YAML

```
AlexaSkillEvent:
  Type: AlexaSkill
```

Api

Api イベントソースタイプを説明するオブジェクトです。AWS::Serverless::Api リソースが定義されている場合、パスとメソッドの値は、API の OpenApi 定義にあるオペレーションに対応している必要があります。

AWS::Serverless::Api が定義されていない場合、関数の入出力は HTTP リクエストと HTTP レスポンスの表現です。

例えば、JavaScript API を使用すると、statusCode および body キーを持つオブジェクトを返すことによって、レスポンスのステータスコードと本文を制御できます。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Auth: ApiFunctionAuth
  Method: String
  Path: String
  RequestModel: RequestModel
  RequestParameters: List of [ String | RequestParameter ]
  RestApiId: String
```

```
  TimeoutInMillis: Integer
```

プロパティ

`Auth`

この特定の Api とパスとメソッドの認証設定です。

`DefaultAuthorizer` が指定されていない場合に個々のパス上にある API の `DefaultAuthorizer` 設定の認証設定を上書きする、またはデフォルトの `ApiKeyRequired` 設定を上書きするために役立ちます。

タイプ: ApiFunctionAuth

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Method`

この関数が呼び出される HTTP メソッドです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Path`

この関数が呼び出される URI パスです。`/` で始まる必要があります。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`RequestModel`

この特定の Api とパスとメソッドに使用するリクエストモデルです。これは、AWS::Serverless::Api リソースの `Models` セクションに指定されているモデルの名前を参照する必要があります。

タイプ: RequestModel

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`RequestParameters`

この特定の API、パス、メソッドのリクエストパラメータ設定です。すべてのパラメータ名は `method.request` で始まり `method.request.header`、`method.request.querystring`、または `method.request.path` に制限する必要があります。

リストには、パラメータ名の文字列と RequestParameter オブジェクトの両方を含めることができます。文字列の場合、`Required` および `Caching` プロパティはデフォルトで `false` になります。

タイプ: [文字列 | RequestParameter] のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`RestApiId`

RestApi リソースの識別子です。これには、指定されたパスとメソッドを持つオペレーションが含まれている必要があります。通常、このテンプレートで定義される AWS::Serverless::Api リソースを参照するように設定されます。

このプロパティを定義しない場合は、AWS SAM が生成された `OpenApi` ドキュメントを使用してデフォルトの AWS::Serverless::Api リソースを作成します。そのリソースには、`RestApiId` を指定しない同じテンプレート内の `Api` イベントによって定義されるすべてのパスとメソッドの和集合が含まれます。

これは、別のテンプレートで定義された AWS::Serverless::Api リソースを参照できません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`TimeoutInMillis`

50 ~ 29,000 ミリ秒のカスタムタイムアウトです。

> **Note**
>
> このプロパティを指定すると、AWS SAM は OpenAPI の定義を変更します。OpenAPI の定義は、`DefinitionBody` プロパティを使用してインラインで指定する必要があります。

タイプ: 整数

必須: いいえ

デフォルト = 29,000 ミリ秒 (29 秒)

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

基本的な の例

YAML

```
Events:
  ApiEvent:
    Type: Api
    Properties:
      Path: /path
      Method: get
      RequestParameters:
        - method.request.header.Authorization
        - method.request.querystring.keyword:
            Required: true
            Caching: false
```

ApiFunctionAuth

特定の API、パス、およびメソッドに対して、イベントレベルで認可を設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  ApiKeyRequired: Boolean
  AuthorizationScopes: List
  Authorizer: String
  InvokeRole: String
  OverrideApiAuth: Boolean
  ResourcePolicy: ResourcePolicyStatement
```

プロパティ

`ApiKeyRequired`

この API、パス、およびメソッドの API キーが必要です。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AuthorizationScopes`

この API、パス、およびメソッドに適用する認可スコープです。

指定するスコープは、`DefaultAuthorizer` プロパティが適用するスコープ (指定されている場合) を上書きします。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Authorizer`

特定の関数用の `Authorizer` です。

AWS::Serverless::Api リソースにグローバルオーソライザーが指定されている場合は、Authorizer を NONE に設定することでオーソライザーをオーバーライドできます。例については、「Amazon API Gateway REST API のグローバルオーソライザーをオーバーライドする」を参照してください。

> **Note**
>
> AWS::Serverless::Api リソースの DefinitionBody プロパティを使用して API を記述する場合は、Authorizer で OverrideApiAuth を使用してグローバルオーソライザーをオーバーライドする必要があります。詳細については、「OverrideApiAuth」を参照してください。

有効な値: AWS_IAM、NONE、または AWS SAM テンプレートで定義されているオーソライザーの論理 ID。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

InvokeRole

AWS_IAM 認可に使用する InvokeRole を指定します。

型: 文字列

必須: いいえ

デフォルト: CALLER_CREDENTIALS

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

追加のメモ: CALLER_CREDENTIALS は arn:aws:iam::*:user/* にマップします。これは、発信者の認証情報を使用してエンドポイントを呼び出します。

OverrideApiAuth

true リソースのグローバルオーソライザー設定をオーバーライドするには、AWS::Serverless::Api を指定します。このプロパティは、グローバルオーソライザーを

指定し、AWS::Serverless::Api リソースの DefinitionBody プロパティを使用して API を記述する場合にのみ必要です。

> **Note**
>
> OverrideApiAuth を true に指定すると、AWS SAM は ApiKeyRequired、Authorizer、ResourcePolicy に指定された値でグローバルオーソライザーをオーバーライドします。したがって、OverrideApiAuth を使用するときには、これらのプロパティを少なくとも 1 つ指定する必要もあります。例については、「AWS:: Serverless:: Apiの DefinitionBody が指定されている場合にグローバルオーソライザーをオーバーライドする」を参照してください。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

ResourcePolicy

この API のパスのためのリソースポリシーを設定します。

タイプ: ResourcePolicyStatement

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

Function-Auth

以下の例は、関数レベルで認可を指定します。

YAML

```
Auth:
```

```
  ApiKeyRequired: true
  Authorizer: NONE
```

Amazon API Gateway REST API のグローバルオーソライザーをオーバーライドする

AWS::Serverless::Api リソースにグローバルオーソライザーを指定できます。以下は、デフォルトのグローバルオーソライザーの設定例です。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApiWithLambdaRequestAuth:
    Type: AWS::Serverless::Api
    Properties:
      ...
      Auth:
        Authorizers:
          MyLambdaRequestAuth:
            FunctionArn: !GetAtt MyAuthFn.Arn
        DefaultAuthorizer: MyLambdaRequestAuth
```

AWS Lambda 関数のデフォルトのオーソライザーをオーバーライドするには、Authorizer を NONE に指定します。以下に例を示します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  ...
  MyFn:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Events:
        LambdaRequest:
          Type: Api
          Properties:
            RestApiId: !Ref MyApiWithLambdaRequestAuth
            Method: GET
            Auth:
              Authorizer: NONE
```

AWS:: Serverless:: Apiの DefinitionBody が指定されている場合にグローバルオーソライザーをオーバーライドする

DefinitionBody プロパティを使用して AWS::Serverless::Api リソースを記述する場合、以前のオーバーライドメソッドは機能しません。以下は、AWS::Serverless::Api リソースの DefinitionBody プロパティを使用する例です。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApiWithLambdaRequestAuth:
    Type: AWS::Serverless::Api
    Properties:
      ...
      DefinitionBody:
        swagger: 2.0
        ...
        paths:
          /lambda-request:
            ...
      Auth:
        Authorizers:
          MyLambdaRequestAuth:
            FunctionArn: !GetAtt MyAuthFn.Arn
        DefaultAuthorizer: MyLambdaRequestAuth
```

グローバルオーソライザーをオーバーライドするには、OverrideApiAuth プロパティを使用します。以下は、OverrideApiAuth を使用してグローバルオーソライザーを Authorizer に指定された値でオーバーライドする例です。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApiWithLambdaRequestAuth:
    Type: AWS::Serverless::Api
    Properties:
      ...
      DefinitionBody:
        swagger: 2-0
        ...
```

```
        paths:
          /lambda-request:
            ...
      Auth:
        Authorizers:
          MyLambdaRequestAuth:
            FunctionArn: !GetAtt MyAuthFn.Arn
        DefaultAuthorizer: MyLambdaRequestAuth

  MyAuthFn:
    Type: AWS::Serverless::Function
    ...

  MyFn:
    Type: AWS::Serverless::Function
      Properties:
        ...
        Events:
          LambdaRequest:
            Type: Api
            Properties:
              RestApiId: !Ref MyApiWithLambdaRequestAuth
              Method: GET
              Auth:
                Authorizer: NONE
                OverrideApiAuth: true
              Path: /lambda-token
```

ResourcePolicyStatement

API 上のすべてのメソッドとパスのリソースポリシーを設定します。リソースポリシーの詳細については、API Gateway デベロッパーガイドの「API Gateway リソースポリシーを使用して API へのアクセスを制御する」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  AwsAccountBlacklist: List
  AwsAccountWhitelist: List
```

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

プロパティ

`AwsAccountBlacklist`

ブロックする AWS アカウントです。

型: 文字列のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AwsAccountWhitelist`

許可する AWS アカウントです。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

型: 文字列のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`CustomStatements`

この API に適用するカスタムリソースポリシーステートメントのリストです。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpcBlacklist`

ブロックする仮想プライベートクラウド (VPC) のリストで、各 VPC が動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpcWhitelist`

許可する VPC のリストで、各 VPC が動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpceBlacklist`

ブロックする VPC エンドポイントのリストで、各 VPC エンドポイントが動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpceWhitelist`

許可する VPC エンドポイントのリストで、各 VPC エンドポイントが動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IpRangeBlacklist`

ブロックする IP アドレスまたはアドレス範囲です。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IpRangeWhitelist`

許可する IP アドレスまたはアドレス範囲です。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceVpcBlacklist`

ブロックするソース VPC またはソース VPC エンドポイントです。ソース VPC 名は `"vpc-"` で始まり、ソース VPC エンドポイント名は `"vpce-"` で始まる必要があります。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceVpcWhitelist`

許可するソース VPC またはソース VPC エンドポイントです。ソース VPC 名は `"vpc-"` で始まり、ソース VPC エンドポイント名は `"vpce-"` で始まる必要があります。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

リソースポリシーの例

以下の例は、2 つの IP アドレスと 1 つのソース VPC をブロックし、AWS アカウントを許可します。

YAML

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

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

RequestModel

特定の API とパスとメソッドのリクエストモデルを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Model: String
Required: Boolean
ValidateBody: Boolean
ValidateParameters: Boolean
```

プロパティ

`Model`

AWS::Serverless::Api の Models プロパティで定義されたモデルの名前です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Required`

指定された API エンドポイントの OpenAPI 定義のパラメータセクションに `required` プロパティを追加します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ValidateBody`

API Gateway が `Model` を使用してリクエストボディを検証するかどうかを指定します。詳細については、API Gateway デベロッパーガイドのAPI Gateway でリクエストに対する検証を有効にするを参照してください。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ValidateParameters`

API Gateway が `Model` を使用してリクエストパスのパラメータ、クエリ文字列、ヘッダーを検証するかどうかを指定します。詳細については、API Gateway デベロッパーガイドのAPI Gateway でリクエストに対する検証を有効にするを参照してください。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

リクエストモデル

リクエストモデルの例

YAML

```
RequestModel:
  Model: User
  Required: true
  ValidateBody: true
  ValidateParameters: true
```

RequestParameter

特定の Api とパスとメソッドのリクエストパラメータを設定します。

リクエストパラメータには、`Required` または `Caching` プロパティを指定する必要があります

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Caching: Boolean
Required: Boolean
```

プロパティ

`Caching`

API Gateway OpenApi 定義に `cacheKeyParameters` セクションを追加します。

タイプ: ブール

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Required`

このフィールドは、パラメータが必須かどうかを指定します。

タイプ: ブール

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

リクエストパラメータ

リクエストパラメータの設定例

YAML

```
RequestParameters:
  - method.request.header.Authorization:
      Required: true
      Caching: true
```

CloudWatchEvent

`CloudWatchEvent` イベントソースタイプを説明するオブジェクトです。

このイベントタイプが設定されていると、AWS Serverless Application Model (AWS SAM) は AWS::Events::Rule リソースを生成します。

重要な注意点: EventBridgeRule は、`CloudWatchEvent` の代わりに使用することが推奨されるイベントソースタイプです。`EventBridgeRule` と `CloudWatchEvent` は同じ基盤となるサービス、API、および AWS CloudFormation リソースを使用しますが、AWS SAM は `EventBridgeRule` のみに新しい機能のサポートを追加します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Enabled: Boolean
  EventBusName: String
  Input: String
  InputPath: String
  Pattern: EventPattern
  State: String
```

プロパティ

`Enabled`

ルールが有効かどうかを示します。

ルールを無効にするには、このプロパティを `false` に設定します。

> **Note**
>
> `Enabled` プロパティと `State` プロパティは、両方ではなく、どちらか一方を指定してください。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `State` プロパティに似ています。このプロパティが `true` に設定されている場合は、AWS SAM が `ENABLED` を渡します。それ以外の場合は `DISABLED` を渡します。

`EventBusName`

このルールに関連付けるイベントバスです。このプロパティを省略する場合、AWS SAM はデフォルトのイベントバスを使用します。

型: 文字列

必須: いいえ

デフォルト値: デフォルトのイベントバス

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `EventBusName` プロパティに直接渡されます。

`Input`

ターゲットに渡された有効な JSON テキストです。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` リソースの `Input` プロパティに直接渡されます。

`InputPath`

一致するイベント全体をターゲットに渡したくない場合は、`InputPath` プロパティを使用して、イベントのどの部分を渡すかを説明します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` リソースの `InputPath` プロパティに直接渡されます。

`Pattern`

どのイベントが指定されたターゲットにルーティングされるかを説明します。詳細については、Amazon EventBridge ユーザーガイドの「Events andEvent Patterns in EventBridge」を参照してください。

タイプ: EventPattern

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `EventPattern` プロパティに直接渡されます。

`State`

ルールの状態。

使用できる値: `DISABLED | ENABLED`

> **Note**
>
> `Enabled` プロパティと `State` プロパティは、両方ではなく、どちらか一方を指定してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `State` プロパティに直接渡されます。

例

CloudWatchEvent

以下は、`CloudWatchEvent` イベントソースタイプの例です。

YAML

```
CWEvent:
  Type: CloudWatchEvent
  Properties:
    Enabled: false
    Input: '{"Key": "Value"}'
    Pattern:
      detail:
        state:
          - running
```

CloudWatchLogs

`CloudWatchLogs` イベントソースタイプを説明するオブジェクトです。

このイベントは、AWS::Logs::SubscriptionFilter リソースを生成し、サブスクリプションフィルターを指定して、それを特定のロググループに関連付けます。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  FilterPattern: String
  LogGroupName: String
```

プロパティ

`FilterPattern`

送信先の AWS リソースに配信される内容を制限するフィルタリング式です。フィルターパターン構文の詳細については、「フィルターとパターンの構文」を参照してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Logs::SubscriptionFilter` リソースの `FilterPattern` プロパティに直接渡されます。

LogGroupName

サブスクリプションフィルターに関連付けるロググループです。フィルターパターンがログイベントに一致する場合、このロググループにアップロードされたすべてのログイベントがフィルタリングされ、指定された AWS リソースに配信されます。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::Logs::SubscriptionFilter リソースの LogGroupName プロパティに直接渡されます。

例

CloudWatchLogs サブスクリプションフィルター

CloudWatchLogs サブスクリプションフィルターの例

YAML

```
CWLog:
  Type: CloudWatchLogs
  Properties:
    LogGroupName:
      Ref: CloudWatchLambdaLogsGroup
    FilterPattern: My pattern
```

Cognito

Cognito イベントソースタイプを説明するオブジェクトです。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Trigger: List
  UserPool: String
```

プロパティ

`Trigger`

新しいユーザープールのための Lambda トリガーの構成情報です。

タイプ: リスト

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Cognito::UserPool` リソースの `LambdaConfig` プロパティに直接渡されます。

`UserPool`

同じテンプレートで定義された UserPool へのリファレンスです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

Cognito イベント

Cognito イベントの例

YAML

```
CognitoUserPoolPreSignup:
  Type: Cognito
  Properties:
    UserPool:
      Ref: MyCognitoUserPool
    Trigger: PreSignUp
```

DocumentDB

`DocumentDB` イベントソースタイプを説明するオブジェクトです。詳細については、「AWS Lambda デベロッパーガイド」の「Amazon DocumentDB で AWS Lambda を使用する」を参照してください。

構文

AWS SAM テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
BatchSize: Integer
Cluster: String
CollectionName: String
DatabaseName: String
Enabled: Boolean
FilterCriteria: FilterCriteria
FullDocument: String
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
SecretsManagerKmsKeyId: String
SourceAccessConfigurations: List
StartingPosition: String
StartingPositionTimestamp: Double
```

プロパティ

`BatchSize`

単一のバッチで取得する項目の最大数です。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `BatchSize` プロパティに直接渡されます。

`Cluster`

Amazon DocumentDB クラスターの Amazon リソースネーム (ARN)。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `EventSourceArn` プロパティに直接渡されます。

`CollectionName`

データベース内で使用するコレクションの名前。コレクションを指定しない場合、Lambda はすべてのコレクションを使用します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` `DocumentDBEventSourceConfig` データ型の `CollectionName` プロパティに直接渡されます。

`DatabaseName`

Amazon DocumentDB クラスター内で使用するデータベースの名前。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` `DocumentDBEventSourceConfig` データ型の `DatabaseName` プロパティに直接渡されます。

`Enabled`

`true` の場合、イベントソースマッピングがアクティブになります。ポーリングと呼び出しを一時停止するには、`false` に設定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `Enabled` プロパティに直接渡されます。

`FilterCriteria`

Lambda がイベントを処理する必要があるかどうかを判断する基準を定義するオブジェクト。詳細については、「AWS Lambda デベロッパーガイド」の「Lambda イベントのフィルタリング」を参照してください。

タイプ: FilterCriteria

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの FilterCriteria プロパティに直接渡されます。

FullDocument

ドキュメントの更新オペレーション中に Amazon DocumentDB がイベントストリームに送信する内容を決定します。[UpdateLookup] に設定すると、Amazon DocumentDB は、ドキュメント全体のコピーとともに、変更について記述するデルタを送信します。それ以外の場合、Amazon DocumentDB は、変更を含む部分的なドキュメントのみを送信します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping DocumentDBEventSourceConfig データ型の FullDocument プロパティに直接渡されます。

KmsKeyArn

このイベントに関連する情報を暗号化するためのキーの Amazon リソースネーム (ARN)。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの KmsKeyArn プロパティに直接渡されます。

MaximumBatchingWindowInSeconds

関数を呼び出すまでのレコード収集の最大時間 (秒) です。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの MaximumBatchingWindowInSeconds プロパティに直接渡されます。

`SecretsManagerKmsKeyId`

AWS Secrets Manager からのカスタマーマネージドキーの AWS Key Management Service (AWS KMS) キー ID。`kms:Decrypt` 許可が含まれていない Lambda 実行ロールを使用して Secrets Manager からのカスタマーマネージドキーを使用する場合に必要です。

このプロパティの値は UUID です。例: `1abc23d4-567f-8ab9-cde0-1fab234c5d67`。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceAccessConfigurations`

認証プロトコルまたは仮想ホストの配列です。これは、SourceAccessConfigurations データ型を使用して指定します。

`DocumentDB` イベントソースタイプの場合、有効な設定タイプは `BASIC_AUTH` のみです。

- `BASIC_AUTH` – ブローカー認証情報を保存する Secrets Manager シークレットです。このタイプの場合、資格情報は `{"username": "your-username", "password": "your-password"}` 形式にする必要があります。`BASIC_AUTH` タイプのオブジェクトが 1 つだけ許可されます。

タイプ: リスト

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `SourceAccessConfigurations` プロパティに直接渡されます。

`StartingPosition`

読み取りを開始するストリームの場所です。

- `AT_TIMESTAMP` - レコードの読み取りを開始する時間を指定します。
- `LATEST` - 新しいレコードのみを読み込みます。
- `TRIM_HORIZON` - 使用可能なすべてのレコードを処理します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの StartingPosition プロパティに直接渡されます。

StartingPositionTimestamp

Unix タイム秒単位で読み取りをスタートする時間。StartingPosition が AT_TIMESTAMP として指定されている場合の StartingPositionTimestamp を定義します。

型: 倍精度

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの StartingPositionTimestamp プロパティに直接渡されます。

例

Amazon DocumentDB イベントソース

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
    ...
      Events:
        MyDDBEvent:
          Type: DocumentDB
          Properties:
            Cluster: "arn:aws:rds:us-west-2:123456789012:cluster:docdb-2023-01-01"
            BatchSize: 10
            MaximumBatchingWindowInSeconds: 5
            DatabaseName: "db1"
            CollectionName: "collection1"
            FullDocument: "UpdateLookup"
            SourceAccessConfigurations:
              - Type: BASIC_AUTH
                URI: "arn:aws:secretsmanager:us-west-2:123456789012:secret:doc-db"
```

DynamoDB

DynamoDB イベントソースタイプを説明するオブジェクトです。詳細については、「 AWS Lambda デベロッパーガイド」の「Amazon DynamoDB AWS Lambda での の使用」を参照してください。

AWS SAM は を生成します。 AWS::Lambda::EventSourceMapping このイベントタイプが設定されている場合の リソース。

構文

AWS Serverless Application Model （ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
BatchSize: Integer
BisectBatchOnFunctionError: Boolean
DestinationConfig: DestinationConfig
Enabled: Boolean
FilterCriteria: FilterCriteria
FunctionResponseTypes: List
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
MaximumRecordAgeInSeconds: Integer
MaximumRetryAttempts: Integer
MetricsConfig: MetricsConfig
ParallelizationFactor: Integer
StartingPosition: String
StartingPositionTimestamp: Double
Stream: String
TumblingWindowInSeconds: Integer
```

プロパティ

`BatchSize`

単一のバッチで取得する項目の最大数です。

タイプ: 整数

必須: いいえ

デフォルト: 100

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `BatchSize`プロパティに直接渡されます。

最小: `1`

最大: `1000`

`BisectBatchOnFunctionError`

関数がエラーを返す場合は、バッチを 2 つに分割して再試行します。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `BisectBatchOnFunctionError`プロパティに直接渡されます。

`DestinationConfig`

破棄されたレコードの Amazon Simple Queue Service (Amazon SQS) キューまたは Amazon Simple Notification Service (Amazon SNS) トピックの送信先。

タイプ: DestinationConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `DestinationConfig`プロパティに直接渡されます。

`Enabled`

ポーリングと呼び出しを一時停止するために、イベントソースマッピングを無効にします。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `Enabled`プロパティに直接渡されます。

`FilterCriteria`

Lambda がイベントを処理する必要があるかどうかを判断するための基準を定義するオブジェクト。詳細については、AWS Lambda デベロッパーガイドの AWS Lambda イベントのフィルタリングを参照してください。

タイプ: FilterCriteria

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの FilterCriteriaプロパティに直接渡されます。

FunctionResponseTypes

現在イベントソースマッピングに適用されているレスポンスタイプのリストです。詳細については、「AWS Lambda デベロッパーガイド」の「バッチアイテムの失敗をレポートする」を参照してください。

有効な値: ReportBatchItemFailures

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの FunctionResponseTypesプロパティに直接渡されます。

KmsKeyArn

このイベントに関連する情報を暗号化するためのキーの Amazon リソースネーム (ARN) 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの KmsKeyArnプロパティに直接渡されます。

MaximumBatchingWindowInSeconds

関数を呼び出すまでのレコード収集の最大時間 (秒) です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumBatchingWindowInSecondsプロパティに直接渡されます。

MaximumRecordAgeInSeconds

Lambda が処理のために関数に送信するレコードの最大存続時間です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumRecordAgeInSecondsプロパティに直接渡されます。

MaximumRetryAttempts

関数がエラーを返すときの最大再試行回数です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumRetryAttemptsプロパティに直接渡されます。

MetricsConfig

処理の各ステージをキャプチャするイベントソースマッピングの拡張メトリクスを取得するためのオプトイン設定。例については、MetricsConfig イベントを参照してください。

タイプ: MetricsConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MetricsConfigプロパティに直接渡されます。

ParallelizationFactor

各シャードから同時に処理するバッチの数です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの ParallelizationFactorプロパティに直接渡されます。

StartingPosition

読み取りを開始するストリームの場所です。

- AT_TIMESTAMP - レコードの読み取りを開始する時間を指定します。

- LATEST - 新しいレコードのみを読み込みます。
- TRIM_HORIZON - 使用可能なすべてのレコードを処理します。

有効な値: AT_TIMESTAMP | LATEST | TRIM_HORIZON

型: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの StartingPositionプロパティに直接渡されます。

StartingPositionTimestamp

Unix タイム秒単位で読み取りをスタートする時間。StartingPosition が AT_TIMESTAMP として指定されている場合の StartingPositionTimestamp を定義します。

型: 倍精度

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの StartingPositionTimestampプロパティに直接渡されます。

Stream

DynamoDB ストリームの Amazon リソースネーム (ARN) 。

型: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの EventSourceArnプロパティに直接渡されます。

TumblingWindowInSeconds

処理ウィンドウの継続時間 (秒) です。有効範囲は 1 ~ 900 (15 分) です。

詳細については、AWS Lambda デベロッパーガイドの「タンブリングウィンドウ」を参照してください。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `TumblingWindowInSeconds`プロパティに直接渡されます。

例

MetricsConfig イベント

以下は、 `MetricsConfig`プロパティを使用してイベントソースマッピングの各処理ステージをキャプチャするリソースの例です。

```
Resources:
  FilteredEventsFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/metricsConfig.zip
      Handler: index.handler
      Runtime: nodejs16.x
      Events:
        KinesisStream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt KinesisStream.Arn
            StartingPosition: LATEST
            MetricsConfig:
              Metrics:
              - EventCount
```

既存の DynamoDB テーブル用の DynamoDB イベントソース

AWS アカウントにすでに存在する DynamoDB テーブルの DynamoDB イベントソース。

YAML

```
Events:
  DDBEvent:
    Type: DynamoDB
    Properties:
      Stream: arn:aws:dynamodb:us-east-1:123456789012:table/TestTable/
stream/2016-08-11T21:21:33.291
      StartingPosition: TRIM_HORIZON
      BatchSize: 10
      Enabled: false
```

テンプレートで宣言された DynamoDB テーブルの DynamoDB イベント

同じテンプレートファイルで宣言されている DynamoDB テーブルの DynamoDB イベントです。

YAML

```
Events:
  DDBEvent:
    Type: DynamoDB
    Properties:
      Stream:
        !GetAtt MyDynamoDBTable.StreamArn   # This must be the name of a DynamoDB table
 declared in the same template file
      StartingPosition: TRIM_HORIZON
      BatchSize: 10
      Enabled: false
```

EventBridgeRule

`EventBridgeRule` イベントソースタイプを記述するオブジェクト。これにより、サーバーレス関数が Amazon EventBridge ルールのターゲットとして設定されます。詳細については、「Amazon ユーザーガイド」の「Amazon とは EventBridge」を参照してください。 EventBridge

AWS SAM は を生成します AWS::Events::Rule このイベントタイプが設定されている場合のリソース。 は、 が Lambda `EventBridgeRule`を呼び出すために必要な`AWS::Lambda::Permission`リソース AWS SAM も作成します。

構文

AWS Serverless Application Model （ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
  DeadLetterConfig: DeadLetterConfig
  EventBusName: String
  Input: String
  InputPath: String
  InputTransformer: InputTransformer
```

```
  Pattern: EventPattern
  RetryPolicy: RetryPolicy
  RuleName: String
  State: String
  Target: Target
```

プロパティ

`DeadLetterConfig`

が失敗したターゲット呼び出しの後にイベント EventBridge を送信する Amazon Simple Queue Service (Amazon SQS) キューを設定します。呼び出しは、存在しない Lambda 関数にイベントを送信する場合や、Lambda 関数を呼び出すためのアクセス許可 EventBridge が不十分な場合などに失敗する可能性があります。詳細については、「Amazon EventBridge ユーザーガイド」の「イベント再試行ポリシー」と「デッドレターキューの使用」を参照してください。

> **Note**
>
> AWS::Serverless::Function リソースタイプには `DeadLetterQueue` という同じようなデータ型があります。これは、ターゲット Lambda 関数の呼び出しが成功した後で発生する失敗を処理します。これらのタイプの失敗の例には、Lambda のスロットリングや、Lambda ターゲット関数によって返されるエラーなどがあります。関数の `DeadLetterQueue` プロパティの詳細については、「AWS Lambda デベロッパーガイド」の「デッドレターキュー」を参照してください。

タイプ : DeadLetterConfig

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Events::RuleTarget`データ型の `DeadLetterConfig` プロパティに似ています。このプロパティ AWS SAM のバージョンには、デッドレターキュー AWS SAM を作成する場合に備えて、追加のサブプロパティが含まれています。

`EventBusName`

このルールに関連付けるイベントバスです。このプロパティを省略すると、 はデフォルトのイベントバス AWS SAM を使用します。

型: 文字列

必須: いいえ

デフォルト値: デフォルトのイベントバス

AWS CloudFormation 互換性： このプロパティは、AWS::Events::Ruleリソースの EventBusNameプロパティに直接渡されます。

Input

ターゲットに渡された有効なJSONテキスト。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Events::Rule Targetリソースの Inputプロパティに直接渡されます。

InputPath

一致するイベント全体をターゲットに渡したくない場合は、InputPath プロパティを使用して、イベントのどの部分を渡すかを説明します。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Events::Rule Targetリソースの InputPathプロパティに直接渡されます。

InputTransformer

特定のイベントデータに基づいてターゲットにカスタム入力を提供できるための設定。イベントから 1 つ以上のキーと値のペアを抽出し、そのデータを使用して、カスタマイズされた入力をターゲットに送信できます。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon 入力変換 EventBridge 」を参照してください。

タイプ : InputTransformer

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Events::RuleTargetデータ型の InputTransformer プロパティに直接渡されます。

`Pattern`

どのイベントが指定されたターゲットにルーティングされるかを説明します。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon イベントとEventBridge イベントパターン」を参照してください。 EventBridge

タイプ : EventPattern

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::Events::Rule`リソースの`EventPattern`プロパティに直接渡されます。

`RetryPolicy`

再試行ポリシーの設定に関する情報が含まれた `RetryPolicy` オブジェクトです。詳細については、「Amazon EventBridge ユーザーガイド」の「イベント再試行ポリシー」と「デッドレターキューの使用」を参照してください。

タイプ : RetryPolicy

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Events::RuleTarget`データ型の`RetryPolicy` プロパティに直接渡されます。

`RuleName`

ルールの名前。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Events::Rule`リソースの `Name`プロパティに直接渡されます。

`State`

ルールの状態。

使用可能な値： `DISABLED | ENABLED | ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS`

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Events::Rule`リソースの `State` プロパティに直接渡されます。

`Target`

ルールがトリガーされたときに EventBridge 呼び出される AWS リソース。このプロパティを使用して、ターゲットの論理 ID を指定できます。このプロパティが指定されていない場合、 はターゲットの論理 ID AWS SAM を生成します。

タイプ: Target

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Events::Rule`リソースの `Targets` プロパティに似ています。 `Amazon EC2 RebootInstances API call`はターゲットプロパティの例です。このプロパティの AWS SAM バージョンでは、単一のターゲットの論理 ID しか指定できません。

例

EventBridgeRule

以下は、`EventBridgeRule` イベントソースタイプの例です。

YAML

```
EBRule:
  Type: EventBridgeRule
  Properties:
    Input: '{"Key": "Value"}'
    Pattern:
      detail:
        state:
          - terminated
    RetryPolicy:
      MaximumRetryAttempts: 5
      MaximumEventAgeInSeconds: 900
    DeadLetterConfig:
```

```
      Type: SQS
      QueueLogicalId: EBRuleDLQ
    Target:
      Id: MyTarget
```

DeadLetterConfig

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを指定するために使用されるオブジェクトです。呼び出しは、存在しない Lambda 関数にイベントを送信した場合、または Lambda 関数を呼び出すために十分な許可がない場合などに失敗します。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

> **Note**
>
> AWS::Serverless::Function リソースタイプには `DeadLetterQueue` という同じようなデータ型があります。これは、ターゲット Lambda 関数の呼び出しが成功した後で発生する失敗を処理します。このタイプの失敗の例には、Lambda のスロットリングや、Lambda ターゲット関数によって返されるエラーなどがあります。関数の `DeadLetterQueue` プロパティの詳細については、「AWS Lambda デベロッパーガイド」の「デッドレターキュー」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Arn: String
QueueLogicalId: String
Type: String
```

プロパティ

`Arn`

デッドレターキューのターゲットとして指定された Amazon SQS キューの Amazon リソースネーム (ARN) です。

> **Note**
>
> Type プロパティと Arn プロパティは、両方ではなく、どちらか一方を指定してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule DeadLetterConfig データ型の Arn プロパティに直接渡されます。

`QueueLogicalId`

Type が指定されている場合に AWS SAM が作成するデッドレターキューのカスタム名です。

> **Note**
>
> Type プロパティが設定されていない場合、このプロパティは無視されます。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Type`

キューのタイプです。このプロパティが設定されていると、AWS SAM がデッドレターキューを自動的に作成し、そのキューにイベントを送信する許可をルールリソースに付与するために必要なリソースベースのポリシーをアタッチします。

> **Note**
>
> Type プロパティと Arn プロパティは、両方ではなく、どちらか一方を指定してください。

有効な値: SQS

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

ルールがトリガーされるときに EventBridge が呼び出す AWS リソースを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Id: String
```

プロパティ

`Id`

ターゲットの論理 ID です。

`Id` の値には、英数字、ピリオド (.)、ハイフン (-)、およびアンダースコア (_) を含めることができます。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` データ型の `Id` プロパティに直接渡されます。

例

Target

YAML

```
EBRule:
  Type: EventBridgeRule
  Properties:
    Target:
      Id: MyTarget
```

HttpApi

HttpApi タイプのイベントソースを説明するオブジェクトです。

指定したパスとメソッドの OpenApi 定義が API に存在する場合、SAM は Lambda 統合とセキュリティセクション (該当する場合) を追加します。

指定したパスとメソッドの OpenApi 定義が API に存在しない場合は、SAM がこの定義を作成します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  ApiId: String
  Auth: HttpApiFunctionAuth
  Method: String
  Path: String
  PayloadFormatVersion: String
  RouteSettings: RouteSettings
```

```
  TimeoutInMillis: Integer
```

プロパティ

`ApiId`

このテンプレートで定義されている AWS::Serverless::HttpApi リソースの識別子です。

定義されていない場合、生成された OpenApi ドキュメント (AWS::Serverless::HttpApi を指定しないこのテンプレートで定義された Api イベントによって定義されるすべてのパスとメソッドの和集合が含まれるもの) を使用して、`ServerlessHttpApi` と呼ばれるデフォルトの `ApiId` リソースが作成されます。

これは、別のテンプレートで定義された AWS::Serverless::HttpApi リソースを参照できません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Auth`

この特定の Api とパスとメソッドの認証設定です。

API の `DefaultAuthorizer` を上書きする、または `DefaultAuthorizer` が指定されていない場合に個々のパス上の認証設定を設定するために役立ちます。

タイプ: HttpApiFunctionAuth

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Method`

この関数が呼び出される HTTP メソッドです。

`Path` と `Method` が指定されていない場合は、SAM がデフォルトの API パスを作成します。このパスは、別のエンドポイントにマップされないリクエストをこの Lambda 関数にルーティングします。これらのデフォルトパスは、API ごとに 1 つしか存在できません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Path`

この関数が呼び出される URI パスです。/ で始まる必要があります。

`Path` と `Method` が指定されていない場合は、SAM がデフォルトの API パスを作成します。このパスは、別のエンドポイントにマップされないリクエストをこの Lambda 関数にルーティングします。これらのデフォルトパスは、API ごとに 1 つしか存在できません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`PayloadFormatVersion`

統合に送信されるペイロードの形式を指定します。

注意: PayloadFormatVersion では OpenAPI 定義の変更に SAM が必要となるため、これが機能するのは `DefinitionBody` で Inline OpenApi が定義されている場合のみです。

型: 文字列

必須: いいえ

デフォルト: 2.0

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`RouteSettings`

この HTTP API に対するルートごとのルート設定です。ルート設定の詳細については、API Gateway デベロッパーガイドの「AWS::ApiGatewayV2::Stage RouteSettings」を参照してください。

注意: HttpApi リソースとイベントソースの両方で RouteSettings が 指定されている場合、AWS SAM はそれらを統合し、イベントソースが優先されます。

Type: RouteSettings

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::Stage` リソースの `RouteSettings` プロパティに直接渡されます。

`TimeoutInMillis`

50 ~ 29,000 ミリ秒のカスタムタイムアウトです。

注意: TimeoutInMillis では OpenAPI 定義の変更に SAM が必要となるため、これが機能するのは `DefinitionBody` で Inline OpenApi が定義されている場合のみです。

タイプ: 整数

必須: いいえ

デフォルト: 5000

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

デフォルトの HttpApi イベント

デフォルトパスを使用する HttpApi イベントです。この API 上にあるマップされていないパスとメソッドのすべてがこのエンドポイントにルーティングされます。

YAML

```
Events:
  HttpApiEvent:
    Type: HttpApi
```

HttpApi

特定のパスとメソッドを使用する HttpApi イベントです。

YAML

```
Events:
  HttpApiEvent:
    Type: HttpApi
    Properties:
      Path: /
      Method: GET
```

HTTP API の認可

オーソライザーを使用する HttpApi イベントです。

YAML

```
Events:
  HttpApiEvent:
    Type: HttpApi
    Properties:
      Path: /authenticated
      Method: GET
      Auth:
        Authorizer: OpenIdAuth
        AuthorizationScopes:
          - scope1
          - scope2
```

HttpApiFunctionAuth

イベントレベルでの認可を設定します。

特定の API とパスとメソッドの認証を設定します

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  AuthorizationScopes: List
```

```
  Authorizer: String
```

プロパティ

`AuthorizationScopes`

この API、パス、およびメソッドに適用する認可スコープです。

ここにリストされているスコープは、`DefaultAuthorizer` によって適用されたスコープを上書きします (存在する場合)。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Authorizer`

特定の関数用の `Authorizer` です。IAM 認証を使用するには、 テンプレートの `Globals` セクションで `EnableIamAuthorizer` の `AWS_IAM` および `true` を指定します。

API でグローバルオーソライザーを指定しており、特定の関数を公開する場合は、`Authorizer` を `NONE` に設定して上書きします。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

Function-Auth

関数レベルでの認可の指定

YAML

```
Auth:
```

```
  Authorizer: OpenIdAuth
  AuthorizationScopes:
    - scope1
    - scope2
```

IAM 認可

イベントレベルで IAM 認可を指定します。イベントレベルで AWS_IAM 認可を使用するには、テンプレートの Globals セクションで EnableIamAuthorizer の true を指定する必要もあります。詳細については、「AWS SAM テンプレートの Globals セクション」を参照してください。

YAML

```
Globals:
  HttpApi:
    Auth:
      EnableIamAuthorizer: true

Resources:
  HttpApiFunctionWithIamAuth:
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: HttpApi
          Properties:
            Path: /iam-auth
            Method: GET
            Auth:
              Authorizer: AWS_IAM
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
          return {'body': 'HttpApiFunctionWithIamAuth', 'statusCode': 200}
      Runtime: python3.9
```

IoTRule

IoTRule イベントソースタイプを説明するオブジェクトです。

AWS::IoT::TopicRule リソースを作成し、AWS IoT ルールを宣言します。詳細については、AWS CloudFormation ドキュメントを参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  AwsIotSqlVersion: String
  Sql: String
```

プロパティ

`AwsIotSqlVersion`

ルールを評価するときに使用する SQL ルールエンジンのバージョンです。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::IoT::TopicRule` `TopicRulePayload` リソースの `AwsIotSqlVersion` プロパティに直接渡されます。

`Sql`

トピックのクエリに使用される SQL ステートメントです。詳細については、AWS IoT デベロッパーガイドの AWS IoT SQL リファレンスを参照してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::IoT::TopicRule` `TopicRulePayload` リソースの `Sql` プロパティに直接渡されます。

例

IOT Rule

IOT Rule の例

YAML

```
IoTRule:
```

```
  Type: IoTRule
  Properties:
    Sql: SELECT * FROM 'topic/test'
```

Kinesis

Kinesis イベントソースタイプを説明するオブジェクトです。詳細については、「 AWS Lambda デベロッパーガイド」のAmazon Kinesis AWS Lambda での の使用」を参照してください。

AWS SAM は を生成します。 AWS::Lambda::EventSourceMapping このイベントタイプが設定されている場合の リソース。

構文

AWS Serverless Application Model （ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
BatchSize: Integer
BisectBatchOnFunctionError: Boolean
DestinationConfig: DestinationConfig
Enabled: Boolean
FilterCriteria: FilterCriteria
FunctionResponseTypes: List
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
MaximumRecordAgeInSeconds: Integer
MaximumRetryAttempts: Integer
MetricsConfig: MetricsConfig
ParallelizationFactor: Integer
StartingPosition: String
StartingPositionTimestamp: Double
Stream: String
TumblingWindowInSeconds: Integer
```

プロパティ

BatchSize

単一のバッチで取得する項目の最大数です。

タイプ: 整数

必須: いいえ

デフォルト: 100

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `BatchSize`プロパティに直接渡されます。

最小: 1

最大: `10000`

`BisectBatchOnFunctionError`

関数がエラーを返す場合は、バッチを 2 つに分割して再試行します。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `BisectBatchOnFunctionError`プロパティに直接渡されます。

`DestinationConfig`

破棄されたレコードの Amazon Simple Queue Service (Amazon SQS) キューまたは Amazon Simple Notification Service (Amazon SNS) トピックの送信先。

タイプ: DestinationConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `DestinationConfig`プロパティに直接渡されます。

`Enabled`

ポーリングと呼び出しを一時停止するために、イベントソースマッピングを無効にします。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `Enabled`プロパティに直接渡されます。

`FilterCriteria`

Lambda がイベントを処理する必要があるかどうかを判断するための基準を定義するオブジェクト。詳細については、AWS Lambda デベロッパーガイドの AWS Lambda イベントのフィルタリングを参照してください。

タイプ: FilterCriteria

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `FilterCriteria`プロパティに直接渡されます。

`FunctionResponseTypes`

現在イベントソースマッピングに適用されているレスポンスタイプのリストです。詳細については、「AWS Lambda デベロッパーガイド」の「バッチアイテムの失敗をレポートする」を参照してください。

有効な値: `ReportBatchItemFailures`

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `FunctionResponseTypes`プロパティに直接渡されます。

`KmsKeyArn`

このイベントに関連する情報を暗号化するためのキーの Amazon リソースネーム (ARN) 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `KmsKeyArn`プロパティに直接渡されます。

`MaximumBatchingWindowInSeconds`

関数を呼び出すまでのレコード収集の最大時間 (秒) です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumBatchingWindowInSecondsプロパティに直接渡されます。

MaximumRecordAgeInSeconds

Lambda が処理のために関数に送信するレコードの最大存続時間です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumRecordAgeInSecondsプロパティに直接渡されます。

MaximumRetryAttempts

関数がエラーを返すときの最大再試行回数です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumRetryAttemptsプロパティに直接渡されます。

MetricsConfig

処理の各ステージをキャプチャするイベントソースマッピングの拡張メトリクスを取得するためのオプトイン設定。例については、MetricsConfig イベントを参照してください。

タイプ: MetricsConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MetricsConfigプロパティに直接渡されます。

ParallelizationFactor

各シャードから同時に処理するバッチの数です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの ParallelizationFactorプロパティに直接渡されます。

StartingPosition

読み取りを開始するストリームの場所です。

- AT_TIMESTAMP - レコードの読み取りを開始する時間を指定します。
- LATEST - 新しいレコードのみを読み込みます。
- TRIM_HORIZON - 使用可能なすべてのレコードを処理します。

有効な値: AT_TIMESTAMP | LATEST | TRIM_HORIZON

型: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの StartingPositionプロパティに直接渡されます。

StartingPositionTimestamp

Unix タイム秒単位で読み取りをスタートする時間。StartingPosition が AT_TIMESTAMP として指定されている場合の StartingPositionTimestamp を定義します。

型: 倍精度

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの StartingPositionTimestampプロパティに直接渡されます。

Stream

データストリームまたはストリームコンシューマーの Amazon リソースネーム (ARN) 。

型: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの EventSourceArnプロパティに直接渡されます。

TumblingWindowInSeconds

処理ウィンドウの継続時間 (秒) です。有効範囲は 1 ~ 900 (15 分) です。

詳細については、AWS Lambda デベロッパーガイドの「タンブリングウィンドウ」を参照してください。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `TumblingWindowInSeconds`プロパティに直接渡されます。

例

MetricsConfig イベント

以下は、 `MetricsConfig`プロパティを使用してイベントソースマッピングの各処理ステージをキャプチャするリソースの例です。

```
Resources:
  FilteredEventsFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/metricsConfig.zip
      Handler: index.handler
      Runtime: nodejs16.x
      Events:
        KinesisStream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt KinesisStream.Arn
            StartingPosition: LATEST
            MetricsConfig:
              Metrics:
              - EventCount
```

Kinesis イベントソース

以下は、Kinesis イベントソースの例です。

YAML

```
Events:
```

```
  KinesisEvent:
    Type: Kinesis
    Properties:
      Stream: arn:aws:kinesis:us-east-1:123456789012:stream/my-stream
      StartingPosition: TRIM_HORIZON
      BatchSize: 10
      Enabled: false
      FilterCriteria:
        Filters:
          - Pattern: '{"key": ["val1", "val2"]}'
```

MQ

MQ イベントソースタイプを説明するオブジェクトです。詳細については、AWS Lambda デベロッパーガイドの「Amazon MQ での Lambda の使用」を参照してください。

このイベントタイプが設定されていると、AWS Serverless Application Model (AWS SAM) は AWS::Lambda::EventSourceMapping リソースを生成します。

> **Note**
>
> パブリックネットワークで Lambda 関数に接続する仮想プライベートクラウド (VPC) に Amazon MQ キューを配置するには、関数の実行ロールに次の許可が含まれている必要があります。
>
> - ec2:CreateNetworkInterface
> - ec2:DeleteNetworkInterface
> - ec2:DescribeNetworkInterfaces
> - ec2:DescribeSecurityGroups
> - ec2:DescribeSubnets
> - ec2:DescribeVpcs
>
> 詳細については、AWS Lambda デベロッパーガイドの「実行ロールのアクセス許可」を参照してください。

構文

AWS SAM テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  BatchSize: Integer
  Broker: String
  DynamicPolicyName: Boolean
  Enabled: Boolean
  FilterCriteria: FilterCriteria
  KmsKeyArn: String
  MaximumBatchingWindowInSeconds: Integer
  Queues: List
  SecretsManagerKmsKeyId: String
  SourceAccessConfigurations: List
```

プロパティ

`BatchSize`

単一のバッチで取得する項目の最大数です。

タイプ: 整数

必須: いいえ

デフォルト: 100

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `BatchSize` プロパティに直接渡されます。

最小: 1

最大: 10000

`Broker`

Amazon MQ ブローカーの Amazon リソースネーム (ARN) です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `EventSourceArn` プロパティに直接渡されます。

`DynamicPolicyName`

デフォルトでは、AWS Identity and Access Management (IAM) ポリシー名は下位互換性のために `SamAutoGeneratedAMQPolicy` となっています。IAM ポリシーのために自動生成された名前を使用するには `true` を指定します。この名前には、Amazon MQ イベントソースの論理 ID が含まれます。

> **Note**
>
> 複数の Amazon MQ イベントソースを使用する場合は、IAM ポリシー名の重複を避けるために `true` を指定します。

タイプ: ブール

必須: いいえ

デフォルト: `false`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Enabled`

`true` の場合、イベントソースマッピングがアクティブになります。ポーリングと呼び出しを一時停止するには、`false` に設定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `Enabled` プロパティに直接渡されます。

`FilterCriteria`

Lambda がイベントを処理する必要があるかどうかを判断する基準を定義するオブジェクト。詳細については、AWS Lambda デベロッパーガイドの AWS Lambda イベントのフィルタリングを参照してください。

タイプ: FilterCriteria

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの FilterCriteria プロパティに直接渡されます。

KmsKeyArn

このイベントに関連する情報を暗号化するためのキーの Amazon リソースネーム (ARN)。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの KmsKeyArn プロパティに直接渡されます。

MaximumBatchingWindowInSeconds

関数を呼び出すまでのレコード収集の最大時間 (秒) です。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの MaximumBatchingWindowInSeconds プロパティに直接渡されます。

Queues

消費する Amazon MQ ブローカーの送信先キューの名前です。

タイプ: リスト

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::Lambda::EventSourceMapping リソースの Queues プロパティに直接渡されます。

SecretsManagerKmsKeyId

AWS Secrets Manager からのカスタマーマネージドキーの AWS Key Management Service (AWS KMS) キー ID です。kms:Decrypt アクセス許可が含まれていない Lambda 実行ロールを使用して Secrets Manager からのカスタマーマネージドキーを使用する場合に必要です。

このプロパティの値は UUID です。例: 1abc23d4-567f-8ab9-cde0-1fab234c5d67。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceAccessConfigurations`

認証プロトコルまたは仮想ホストの配列です。これは、SourceAccessConfigurations データ型を使用して指定します。

MQ イベントソースタイプの場合、有効な設定タイプは BASIC_AUTH と VIRTUAL_HOST だけです。

- **BASIC_AUTH** – ブローカー認証情報を保存する Secrets Manager シークレットです。このタイプの場合、資格情報は {"username": "your-username", "password": "your-password"} 形式にする必要があります。BASIC_AUTH タイプのオブジェクトが 1 つだけ許可されます。
- **VIRTUAL_HOST** – RabbitMQ ブローカー内の仮想ホストの名前です。Lambda は、この RabbitMQ のホストをイベントソースとして使用します。VIRTUAL_HOST タイプのオブジェクトが 1 つだけ許可されます。

タイプ: リスト

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::EventSourceMapping` リソースの `SourceAccessConfigurations` プロパティに直接渡されます。

例

Amazon MQ イベントソース

以下は、Amazon MQ ブローカー向けの MQ イベントソースタイプの例です。

YAML

```
Events:
  MQEvent:
    Type: MQ
    Properties:
      Broker: arn:aws:mq:us-
east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9
```

```
      Queues: List of queues
      SourceAccessConfigurations:
        - Type: BASIC_AUTH
          URI: arn:aws:secretsmanager:us-east-1:01234567890:secret:MyBrokerSecretName
      BatchSize: 200
      Enabled: true
```

MSK

MSK イベントソースタイプを説明するオブジェクトです。詳細については、「 AWS Lambda デベロッパーガイド」の「Amazon AWS Lambda での の使用MSK」を参照してください。

AWS Serverless Application Model （ AWS SAM) は を生成します。
AWS::Lambda::EventSourceMapping このイベントタイプが設定されている場合の リソース。

構文

AWS SAM テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
ConsumerGroupId: String
DestinationConfig: DestinationConfig
FilterCriteria: FilterCriteria
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
ProvisionedPollerConfig: ProvisionedPollerConfig
SourceAccessConfigurations: SourceAccessConfigurations
StartingPosition: String
StartingPositionTimestamp: Double
Stream: String
Topics: List
```

プロパティ

ConsumerGroupId

Kafka トピックからイベントを読み取る方法を設定する文字列。

タイプ: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `AmazonManagedKafkaConfiguration`プロパティに直接渡されます。

`DestinationConfig`

Lambda がイベントを処理した後のイベントの送信先を指定する構成オブジェクト。

このプロパティを使用して、Amazon MSKイベントソースからの失敗した呼び出しの送信先を指定します。

タイプ: DestinationConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `DestinationConfig`プロパティに直接渡されます。

`FilterCriteria`

Lambda がイベントを処理する必要があるかどうかを判断する基準を定義するオブジェクト。詳細については、AWS Lambda デベロッパーガイドの AWS Lambda イベントのフィルタリングを参照してください。

タイプ: FilterCriteria

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `FilterCriteria`プロパティに直接渡されます。

`KmsKeyArn`

このイベントに関連する情報を暗号化するためのキーの Amazon リソースネーム (ARN) 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `KmsKeyArn`プロパティに直接渡されます。

`MaximumBatchingWindowInSeconds`

関数を呼び出すまでのレコード収集の最大時間 (秒) です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumBatchingWindowInSecondsプロパティに直接渡されます。

ProvisionedPollerConfig

イベントソースマッピングの計算に使用されるポーラーの量を増やすための設定。この設定では、ポーラーを 1 つ以上、ポーラーを最大 20 個にすることができます。例については、「」を参照してくださいProvisionedPollerConfig 例。

タイプ: ProvisionedPollerConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの ProvisionedPollerConfigプロパティに直接渡されます。

SourceAccessConfigurations

イベントソースを保護して定義する認証プロトコル、VPCコンポーネント、または仮想ホストの配列。

有効な値: CLIENT_CERTIFICATE_TLS_AUTH

タイプ: SourceAccessConfiguration のリスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの SourceAccessConfigurationsプロパティに直接渡されます。

StartingPosition

読み取りを開始するストリームの場所です。

- AT_TIMESTAMP - レコードの読み取りを開始する時間を指定します。
- LATEST - 新しいレコードのみを読み込みます。
- TRIM_HORIZON - 使用可能なすべてのレコードを処理します。

有効な値: AT_TIMESTAMP | LATEST | TRIM_HORIZON

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの StartingPositionプロパティに直接渡されます。

StartingPositionTimestamp

Unix タイム秒単位で読み取りをスタートする時間。StartingPosition が AT_TIMESTAMP として指定されている場合の StartingPositionTimestamp を定義します。

型: 倍精度

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの StartingPositionTimestampプロパティに直接渡されます。

Stream

データストリームまたはストリームコンシューマーの Amazon リソースネーム (ARN) 。

型: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの EventSourceArnプロパティに直接渡されます。

Topics

Kafka トピックの名前です。

タイプ: リスト

必須: はい

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの Topicsプロパティに直接渡されます。

例

ProvisionedPollerConfig 例

```
ProvisionedPollerConfig:
```

```
  MinimumPollers: 1
  MaximumPollers: 20
```

既存のクラスターの Amazon MSKの例

以下は、 に既に存在する Amazon MSKクラスターのMSKイベントソースタイプの例です AWS アカウント。

YAML

```
Events:
  MSKEvent:
    Type: MSK
    Properties:
      StartingPosition: LATEST
      Stream: arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/
abcdefab-1234-abcd-5678-cdef0123ab01-2
      Topics:
        - MyTopic
```

同じテンプレートで宣言されたクラスターの Amazon MSKの例

以下は、同じテンプレートファイルで宣言された Amazon MSKクラスターのMSKイベントソースタイプの例です。

YAML

```
Events:
  MSKEvent:
    Type: MSK
    Properties:
      StartingPosition: LATEST
      Stream:
        Ref: MyMskCluster   # This must be the name of an MSK cluster declared in the
 same template file
      Topics:
        - MyTopic
```

S3

S3 イベントソースタイプを説明するオブジェクトです。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Bucket: String
  Events: String | List
  Filter: NotificationFilter
```

プロパティ

`Bucket`

S3 バケット名です。このバケットは、同じテンプレートに存在する必要があります。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::S3::Bucket` リソースの `BucketName` プロパティに似ています。これは SAM の必須フィールドです。このフィールドは、このテンプレートで作成された S3 バケットへのリファレンスのみを受け入れます。

`Events`

Lambda 関数を呼び出す Amazon S3 バケットイベントです。有効な値のリストについては、Amazon S3 の「サポートされるイベントタイプ」を参照してください。

タイプ: 文字列 | リスト

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::S3::Bucket` `LambdaConfiguration` データ型の `Event` プロパティに直接渡されます。

`Filter`

どの Amazon S3 オブジェクトが Lambda 関数を呼び出すかを決定するフィルタリングルールです。Amazon S3 キー名によるフィルタリングの詳細については、Amazon Simple Storage Service ユーザーガイドで Amazon S3 イベント通知の設定を参照してください。

タイプ: NotificationFilter

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::S3::Bucket` `LambdaConfiguration` データ型の `Filter` プロパティに直接渡されます。

例

S3 イベント

S3 イベントの例です。

YAML

```
Events:
  S3Event:
    Type: S3
    Properties:
      Bucket:
        Ref: ImagesBucket     # This must be the name of an S3 bucket declared in the
 same template file
      Events: s3:ObjectCreated:*
      Filter:
        S3Key:
          Rules:
          - Name: prefix      # or "suffix"
            Value: value      # The value to search for in the S3 object key names
```

Schedule

`Schedule` イベントソースタイプを説明するオブジェクトです。これは、サーバーレス関数をスケジュールに従ってトリガーする Amazon EventBridge ルールのターゲットとして設定します。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge とは」を参照してください。

このイベントタイプが設定されていると、AWS Serverless Application Model (AWS SAM) は AWS::Events::Rule リソースを生成します。

> **Note**
>
> EventBridge では、Amazon EventBridge Scheduler という新しいスケジューリング機能が提供されるようになりました。Amazon EventBridge Scheduler はサーバーレススケジューラ

で、一元化されたマネージドサービスからタスクを作成、実行、管理できます。EventBridge Scheduler は高度にカスタマイズ可能で、EventBridge のスケジュールルールよりもスケーラビリティが高く、ターゲット API オペレーションと AWS のサービス の範囲が広がります。スケジュールに従ってターゲットを呼び出すには、EventBridge Scheduler を使用することをお勧めします。AWS SAM テンプレートでこのイベントソースタイプを定義するには、「ScheduleV2」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DeadLetterConfig: DeadLetterConfig
Description: String
Enabled: Boolean
Input: String
Name: String
RetryPolicy: RetryPolicy
Schedule: String
State: String
```

プロパティ

`DeadLetterConfig`

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを設定します。呼び出しは、存在しない Lambda 関数にイベントを送信した場合、または Lambda 関数を呼び出すために十分な許可が EventBridge にない場合などに失敗します。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

> **Note**
>
> AWS::Serverless::Function リソースタイプには `DeadLetterQueue` という同じようなデータ型があります。これは、ターゲット Lambda 関数の呼び出しが成功した後で発生する失敗を処理します。これらのタイプの失敗の例には、Lambda のスロットリングや、Lambda ターゲット関数によって返されるエラーなどがあります。関数の

DeadLetterQueue プロパティの詳細については、「AWS Lambda デベロッパーガイド」の「デッドレターキュー」を参照してください。

タイプ: DeadLetterConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target データ型の DeadLetterConfig プロパティに似ています。このプロパティの AWS SAM バージョンには、AWS SAM にデッドレターキューを作成させる場合のために、追加のサブプロパティが含まれています。

`Description`

ルールの説明です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの Description プロパティに直接渡されます。

`Enabled`

ルールが有効かどうかを示します。

ルールを無効にするには、このプロパティを false に設定します。

> **Note**
>
> Enabled プロパティと State プロパティは、両方ではなく、どちらか一方を指定してください。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの State プロパティに似ています。このプロパティが true に設定されている場合は、AWS SAM が ENABLED を渡します。それ以外の場合は DISABLED を渡します。

`Input`

ターゲットに渡された有効な JSON テキストです。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` リソースの `Input` プロパティに直接渡されます。

`Name`

ルールの名前です。名前を指定しない場合、AWS CloudFormation が一意の物理 ID を生成し、その ID をルール名として使用します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `Name` プロパティに直接渡されます。

`RetryPolicy`

再試行ポリシーの設定に関する情報が含まれた `RetryPolicy` オブジェクトです。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

タイプ: RetryPolicy

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` データ型の `RetryPolicy` プロパティに直接渡されます。

`Schedule`

ルールがいつ、どのくらいの頻度で実行されるかを決定するスケジューリング式です。詳細については、「Schedule Expressions for Rules」を参照してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの ScheduleExpression プロパティに直接渡されます。

State

ルールの状態。

使用できる値: DISABLED | ENABLED

> **Note**
>
> Enabled プロパティと State プロパティは、両方ではなく、どちらか一方を指定してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの State プロパティに直接渡されます。

例

CloudWatch スケジュールイベント

CloudWatch スケジュールイベントの例

YAML

```
CWSchedule:
  Type: Schedule
  Properties:
    Schedule: 'rate(1 minute)'
    Name: TestSchedule
    Description: test schedule
    Enabled: false
```

DeadLetterConfig

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを指定するために使用されるオブジェクトです。呼び出しは、存在し

ない Lambda 関数にイベントを送信した場合、または Lambda 関数を呼び出すために十分な許可がない場合などに失敗します。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

> **Note**
>
> AWS::Serverless::Function リソースタイプには `DeadLetterQueue` という同じようなデータ型があります。これは、ターゲット Lambda 関数の呼び出しが成功した後で発生する失敗を処理します。このタイプの失敗の例には、Lambda のスロットリングや、Lambda ターゲット関数によって返されるエラーなどがあります。関数の `DeadLetterQueue` プロパティの詳細については、「AWS Lambda デベロッパーガイド」の「デッドレターキュー」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Arn: String
QueueLogicalId: String
Type: String
```

プロパティ

`Arn`

デッドレターキューのターゲットとして指定された Amazon SQS キューの Amazon リソースネーム (ARN) です。

> **Note**
>
> `Type` プロパティと `Arn` プロパティは、両方ではなく、どちらか一方を指定してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` `DeadLetterConfig` データ型の `Arn` プロパティに直接渡されます。

`QueueLogicalId`

`Type` が指定されている場合に AWS SAM が作成するデッドレターキューのカスタム名です。

> **Note**
>
> `Type` プロパティが設定されていない場合、このプロパティは無視されます。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Type`

キューのタイプです。このプロパティが設定されていると、AWS SAM がデッドレターキューを自動的に作成し、そのキューにイベントを送信する許可をルールリソースに付与するために必要なリソースベースのポリシーをアタッチします。

> **Note**
>
> `Type` プロパティと `Arn` プロパティは、両方ではなく、どちらか一方を指定してください。

有効な値: `SQS`

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

ScheduleV2

ScheduleV2 イベントソースタイプを説明するオブジェクトです。これは、サーバーレス関数をスケジュールに従ってトリガーする Amazon EventBridge スケジューライベントのターゲットとして設定します。詳細については、「EventBridge スケジューラユーザーガイド」の「What is Amazon EventBridge Scheduler?」(Amazon EventBridge スケジューラとは) を参照してください。

このイベントタイプが設定されていると、AWS Serverless Application Model (AWS SAM) は AWS::Scheduler::Schedule リソースを生成します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DeadLetterConfig: DeadLetterConfig
Description: String
EndDate: String
FlexibleTimeWindow: FlexibleTimeWindow
GroupName: String
Input: String
KmsKeyArn: String
Name: String
OmitName: Boolean
PermissionsBoundary: String
RetryPolicy: RetryPolicy
RoleArn: String
ScheduleExpression: String
```

```
ScheduleExpressionTimezone: String
StartDate: String
State: String
```

プロパティ

`DeadLetterConfig`

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを設定します。呼び出しは、存在しない Lambda 関数にイベントを送信した場合、または Lambda 関数を呼び出すために十分な許可が EventBridge にない場合などに失敗します。詳細については、「EventBridge スケジューラユーザーガイド」の「Configuring a dead-letter queue for EventBridge Scheduler」(EventBridge スケジューラのデッドレターキューの設定) を参照してください。

> **Note**
>
> AWS::Serverless::Function リソースタイプには `DeadLetterQueue` という同じようなデータ型があります。これは、ターゲット Lambda 関数の呼び出しが成功した後で発生する失敗を処理します。これらのタイプの失敗の例には、Lambda のスロットリングや、Lambda ターゲット関数によって返されるエラーなどがあります。関数の `DeadLetterQueue` プロパティの詳細については、「AWS Lambda デベロッパーガイド」の「デッドレターキュー」を参照してください。

タイプ: DeadLetterConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Scheduler::Schedule Target` データ型の `DeadLetterConfig` プロパティに似ています。このプロパティの AWS SAM バージョンには、AWS SAM にデッドレターキューを作成させる場合のために、追加のサブプロパティが含まれています。

`Description`

スケジュールの説明。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの Description プロパティに直接渡されます。

EndDate

スケジュールがターゲットを呼び出すことができる日付 (UTC)。スケジュールの繰り返し式によっては、指定した EndDate またはそれより前に呼び出しが停止する場合があります。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの EndDate プロパティに直接渡されます。

FlexibleTimeWindow

スケジュールを呼び出すことができるウィンドウを設定できます。

タイプ: FlexibleTimeWindow

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの FlexibleTimeWindow プロパティに直接渡されます。

GroupName

このスケジュールに関連付けるスケジュールグループの名前。定義されていない場合、デフォルトグループが使用されます。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの GroupName プロパティに直接渡されます。

Input

ターゲットに渡された有効な JSON テキストです。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Scheduler::Schedule Target` リソースの `Input` プロパティに直接渡されます。

`KmsKeyArn`

お客様のデータの暗号化に使用する KMS キーの ARN。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Scheduler::Schedule` リソースの `KmsKeyArn` プロパティに直接渡されます。

`Name`

スケジュールの名前。名前を指定しない場合、AWS SAM は *`Function-Logical-IDEvent-Source-Name`* の形式で名前を生成し、その ID をスケジュール名に使用します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Scheduler::Schedule` リソースの `Name` プロパティに直接渡されます。

`OmitName`

デフォルトの場合、AWS SAM は *`<Function-logical-ID><event-source-name>`* の形式でスケジュール名を生成して使用します。このプロパティをに `true` 設定すると、AWS CloudFormation は一意の固有の物理 ID を生成し、代わりにその物理 ID をスケジュール名として使用します。

タイプ: ブール

必須: いいえ

デフォルト: `false`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`PermissionsBoundary`

ロールのアクセス許可の境界を設定するために使用するポリシーの ARN。

> **Note**
>
> `PermissionsBoundary` が定義されている場合、AWS SAM は同じ境界をスケジューラスケジュールのターゲット IAM ロールに適用します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::IAM::Role` リソースの `PermissionsBoundary` プロパティに直接渡されます。

`RetryPolicy`

再試行ポリシーの設定に関する情報が含まれた RetryPolicy オブジェクトです。

タイプ: RetryPolicy

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Scheduler::Schedule Target` データ型の `RetryPolicy` プロパティに直接渡されます。

`RoleArn`

スケジュールが呼び出されたときに EventBridge スケジューラがターゲットとして使用する IAM ロールの ARN。

タイプ: RoleArn

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Scheduler::Schedule Target` データ型の `RoleArn` プロパティに直接渡されます。

`ScheduleExpression`

スケジューラのスケジュールイベントがいつ、どのくらいの頻度で実行されるかを決定するスケジューリング式です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの ScheduleExpression プロパティに直接渡されます。

ScheduleExpressionTimezone

スケジュール式が評価されるタイムゾーン。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの ScheduleExpressionTimezone プロパティに直接渡されます。

StartDate

スケジュールがターゲットの呼び出しを開始できる日付 (UTC)。スケジュールの繰り返し式によっては、指定した StartDate またはそれより後に呼び出しが発生する場合があります。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの StartDate プロパティに直接渡されます。

State

スケジューラのスケジュールの状態。

使用できる値: DISABLED | ENABLED

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule リソースの State プロパティに直接渡されます。

例

ScheduleV2 リソースを定義する基本的な例

```
Resources:
  Function:
    Properties:
      ...
      Events:
        ScheduleEvent:
          Type: ScheduleV2
          Properties:
            ScheduleExpression: "rate(1 minute)"
        ComplexScheduleEvent:
          Type: ScheduleV2
          Properties:
            ScheduleExpression: rate(1 minute)
            FlexibleTimeWindow:
              Mode: FLEXIBLE
              MaximumWindowInMinutes: 5
            StartDate: '2022-12-28T12:00:00.000Z'
            EndDate: '2023-01-28T12:00:00.000Z'
            ScheduleExpressionTimezone: UTC
            RetryPolicy:
              MaximumRetryAttempts: 5
              MaximumEventAgeInSeconds: 300
            DeadLetterConfig:
              Type: SQS
```

> **Note**
>
> 生成された ScheduleV2 の物理 ID には、スタック名は含まれません。

SelfManagedKafka

`SelfManagedKafka` イベントソースタイプを説明するオブジェクトです。詳細については、「AWS Lambda デベロッパーガイド」の「セルフマネージド Apache Kafka AWS Lambda で を使用する」を参照してください。

AWS Serverless Application Model （ AWS SAM) は を生成します。
AWS::Lambda::EventSourceMapping このイベントタイプが設定されている場合の リソース。

構文

AWS SAM テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
  BatchSize: Integer
  ConsumerGroupId: String
  DestinationConfig: DestinationConfig
  Enabled: Boolean
  FilterCriteria: FilterCriteria
  KafkaBootstrapServers: List
  KmsKeyArn: String
  ProvisionedPollerConfig: ProvisionedPollerConfig
  SourceAccessConfigurations: SourceAccessConfigurations
  StartingPosition: String
  StartingPositionTimestamp: Double
  Topics: List
```

プロパティ

`BatchSize`

Lambda がストリームから取り出し、関数に送信する各バッチ内の最大レコード数。

タイプ: 整数

必須: いいえ

デフォルト: 100

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `BatchSize`プロパティに直接渡されます。

最小: 1

最大: 10000

`ConsumerGroupId`

Kafka トピックからイベントを読み取る方法を設定する文字列。

タイプ: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの SelfManagedKafkaConfigurationプロパティに直接渡されます。

DestinationConfig

Lambda がイベントを処理した後のイベントの送信先を指定する構成オブジェクト。

このプロパティを使用して、自己管理型の Kafka イベントソースからの失敗した呼び出しの送信先を指定します。

タイプ: DestinationConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの DestinationConfigプロパティに直接渡されます。

Enabled

ポーリングと呼び出しを一時停止するために、イベントソースマッピングを無効にします。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの Enabledプロパティに直接渡されます。

FilterCriteria

Lambda がイベントを処理する必要があるかどうかを判断するための基準を定義するオブジェクト。詳細については、AWS Lambda デベロッパーガイドの AWS Lambda イベントのフィルタリングを参照してください。

タイプ: FilterCriteria

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの FilterCriteriaプロパティに直接渡されます。

KafkaBootstrapServers

Kafka ブローカー用のブートストラップサーバーのリスト。ポートを含めます。例: broker.example.com:*xxxx*

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`KmsKeyArn`

このイベントに関連する情報を暗号化するためのキーの Amazon リソースネーム (ARN) 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `KmsKeyArn`プロパティに直接渡されます。

`ProvisionedPollerConfig`

イベントソースマッピングの計算に使用されるポーラーの量を増やすための設定。この設定では、ポーラーを 1 つ以上、ポーラーを最大 20 個にすることができます。例については、「」を参照してください。 ProvisionedPollerConfig 例

タイプ: ProvisionedPollerConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `ProvisionedPollerConfig`プロパティに直接渡されます。

`SourceAccessConfigurations`

イベントソースを保護して定義する認証プロトコル、VPCコンポーネント、または仮想ホストの配列。

有効な値: `BASIC_AUTH | CLIENT_CERTIFICATE_TLS_AUTH | SASL_SCRAM_256_AUTH | SASL_SCRAM_512_AUTH | SERVER_ROOT_CA_CERTIFICATE`

タイプ: SourceAccessConfiguration のリスト

必須: はい

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `SourceAccessConfigurations`プロパティに直接渡されます。

`StartingPosition`

読み取りを開始するストリームの場所です。

- `AT_TIMESTAMP` - レコードの読み取りを開始する時間を指定します。
- `LATEST` - 新しいレコードのみを読み込みます。
- `TRIM_HORIZON` - 使用可能なすべてのレコードを処理します。

有効な値: `AT_TIMESTAMP` | `LATEST` | `TRIM_HORIZON`

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `StartingPosition`プロパティに直接渡されます。

`StartingPositionTimestamp`

Unix タイム秒単位で読み取りをスタートする時間。`StartingPosition` が `AT_TIMESTAMP` として指定されている場合の `StartingPositionTimestamp` を定義します。

型: 倍精度

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `StartingPositionTimestamp`プロパティに直接渡されます。

`Topics`

Kafka トピックの名前です。

タイプ: リスト

必須: はい

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `Topics`プロパティに直接渡されます。

例

ProvisionedPollerConfig 例

```
ProvisionedPollerConfig:
```

```
  MinimumPollers: 1
  MaximumPollers: 20
```

セルフマネージド型の Kafka イベントソース

以下は、SelfManagedKafka イベントソースタイプの例です。

YAML

```
Events:
  SelfManagedKafkaEvent:
    Type: SelfManagedKafka
    Properties:
      BatchSize: 1000
      Enabled: true
      KafkaBootstrapServers:
        - abc.xyz.com:xxxx
      SourceAccessConfigurations:
        -  Type: BASIC_AUTH
           URI: arn:aws:secretsmanager:us-west-2:123456789012:secret:my-path/my-secret-
name-1a2b3c
      Topics:
        - MyKafkaTopic
```

SNS

SNS イベントソースタイプを説明するオブジェクトです。

このイベントタイプが設定されていると、SAM は AWS::SNS::Subscription リソースを生成します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  FilterPolicy: SnsFilterPolicy
  FilterPolicyScope: String
  RedrivePolicy: Json
  Region: String
  SqsSubscription: Boolean | SqsSubscriptionObject
```

```
  Topic: String
```

プロパティ

`FilterPolicy`

サブスクリプションに割り当てられているフィルターポリシー JSON です。詳細については、Amazon Simple Notification Service API リファレンスの「GetSubscriptionAttributes」を参照してください。

タイプ: SNSFilterPolicy

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::SNS::Subscription` リソースの `FilterPolicy` プロパティに直接渡されます。

`FilterPolicyScope`

この属性では、次の文字列値のタイプのいずれかを使用してフィルタリング範囲を選択できます。

- `MessageAttributes` – フィルターはメッセージ属性に適用されます。
- `MessageBody` - フィルターはメッセージ本文に適用されます。

型: 文字列

必須: いいえ

デフォルト: `MessageAttributes`

AWS CloudFormation との互換性: このプロパティは、`AWS::SNS::Subscription` リソースの `FilterPolicyScope` プロパティに直接渡されます。

`RedrivePolicy`

指定すると、指定された Amazon SQS デッドレターキューに配信不能メッセージを送信します。クライアントエラー (例: サブスクライブされたエンドポイントに到達できない) またはサーバーエラー (例: サブスクライブされたエンドポイントに電力を供給するサービスが使用できなくなる) が原因で配信できないメッセージは、詳細な分析や再処理のためにデッドレターキューに保持されます。

再処理ポリシーとデッドレターキューの詳細については、「Amazon Simple Queue Service デベロッパーガイド」の「Amazon SQS デッドレターキュー」を参照してください。

型: Json

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::SNS::Subscription` リソースの `RedrivePolicy` プロパティに直接渡されます。

`Region`

クロスリージョンサブスクリプションの場合に、トピックが格納されるリージョンです。

リージョンが指定されていない場合、CloudFormation は発信者のリージョンをデフォルトとして使用します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::SNS::Subscription` リソースの `Region` プロパティに直接渡されます。

`SqsSubscription`

このプロパティを true に設定するか、`SqsSubscriptionObject` を指定して SQS キューで SNS トピック通知のバッチ処理を有効にします。このプロパティを `true` に設定することによって新しい SQS キューが作成されますが、`SqsSubscriptionObject` を指定すると既存の SQS キューが使用されます。

タイプ: ブール | SqsSubscriptionObject

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Topic`

サブスクライブ先のトピックの ARN です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::SNS::Subscription` リソースの `TopicArn` プロパティに直接渡されます。

例

SNS イベントソースの例

SNS イベントソースの例

YAML

```
Events:
  SNSEvent:
    Type: SNS
    Properties:
      Topic: arn:aws:sns:us-east-1:123456789012:my_topic
      SqsSubscription: true
      FilterPolicy:
        store:
          - example_corp
        price_usd:
          - numeric:
              - ">="
              - 100
```

SqsSubscriptionObject

SNS イベントに既存の SQS キューオプションを指定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  BatchSize: String
  Enabled: Boolean
  QueueArn: String
  QueuePolicyLogicalId: String
  QueueUrl: String
```

プロパティ

`BatchSize`

SQS キューの 単一バッチで取得する項目の最大数です。

型: 文字列

必須: いいえ

デフォルト: 10

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Enabled`

ポーリングと呼び出しを中断する SQS イベントソースマッピングを無効にします。

タイプ: ブール

必須: いいえ

デフォルト: True

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`QueueArn`

既存の SQS キュー ARN を指定します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`QueuePolicyLogicalId`

AWS::SQS::QueuePolicy リソースのカスタム LogicalID 名を指定します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`QueueUrl`

`QueueArn` プロパティに関連付けられたキュー URL を指定します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

SNS イベント用の既存の SQS

SNS トピックにサブスクライブするための既存の SQS キューを追加する例です。

YAML

```
QueuePolicyLogicalId: CustomQueuePolicyLogicalId
QueueArn:
  Fn::GetAtt: MyCustomQueue.Arn
QueueUrl:
  Ref: MyCustomQueue
BatchSize: 5
```

SQS

SQS イベントソースタイプを説明するオブジェクトです。詳細については、「 AWS Lambda デベロッパーガイド」の「Amazon AWS Lambda での の使用SQS」を参照してください。

SAM 生成 AWS::Lambda::EventSourceMapping このイベントタイプが設定されている場合の リソース

構文

AWS Serverless Application Model （ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
  BatchSize: Integer
  Enabled: Boolean
  FilterCriteria: FilterCriteria
  FunctionResponseTypes: List
  KmsKeyArn: String
```

```
  MaximumBatchingWindowInSeconds: Integer
  MetricsConfig: MetricsConfig
  Queue: String
  ScalingConfig: ScalingConfig
```

プロパティ

`BatchSize`

単一のバッチで取得する項目の最大数です。

タイプ: 整数

必須: いいえ

デフォルト: 10

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `BatchSize`プロパティに直接渡されます。

最小: 1

最大: `10000`

`Enabled`

ポーリングと呼び出しを一時停止するために、イベントソースマッピングを無効にします。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::EventSourceMapping`リソースの `Enabled`プロパティに直接渡されます。

`FilterCriteria`

Lambda がイベントを処理する必要があるかどうかを判断するための基準を定義するオブジェクト。詳細については、AWS Lambda デベロッパーガイドの AWS Lambda イベントのフィルタリングを参照してください。

タイプ: FilterCriteria

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの FilterCriteriaプロパティに直接渡されます。

FunctionResponseTypes

現在イベントソースマッピングに適用されているレスポンスタイプのリストです。詳細については、「AWS Lambda デベロッパーガイド」の「バッチアイテムの失敗をレポートする」を参照してください。

有効な値: ReportBatchItemFailures

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの FunctionResponseTypesプロパティに直接渡されます。

KmsKeyArn

このイベントに関連する情報を暗号化するためのキーの Amazon リソースネーム (ARN) 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの KmsKeyArnプロパティに直接渡されます。

MaximumBatchingWindowInSeconds

関数を呼び出すまでのレコード収集の最大時間 (秒) です。

タイプ: 整数

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MaximumBatchingWindowInSecondsプロパティに直接渡されます。

MetricsConfig

処理の各ステージをキャプチャするイベントソースマッピングの拡張メトリクスを取得するためのオプトイン設定。例については、MetricsConfig イベントを参照してください。

タイプ: MetricsConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの MetricsConfigプロパティに直接渡されます。

Queue

キューARNの 。

型: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの EventSourceArnプロパティに直接渡されます。

ScalingConfig

呼び出しレートを制御し、最大同時呼び出しを設定するSQSポーラーのスケーリング設定。

タイプ: ScalingConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::EventSourceMappingリソースの ScalingConfigプロパティに直接渡されます。

例

MetricsConfig イベント

以下は、 MetricsConfigプロパティを使用してイベントソースマッピングの各処理ステージをキャプチャするリソースの例です。

```
Resources:
  FilteredEventsFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/metricsConfig.zip
      Handler: index.handler
      Runtime: nodejs16.x
      Events:
```

```
        KinesisStream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt KinesisStream.Arn
            StartingPosition: LATEST
            MetricsConfig:
              Metrics:
              - EventCount
```

基本SQSイベント

```
Events:
  SQSEvent:
    Type: SQS
    Properties:
      Queue: arn:aws:sqs:us-west-2:012345678901:my-queue
      BatchSize: 10
      Enabled: false
      FilterCriteria:
        Filters:
          - Pattern: '{"key": ["val1", "val2"]}'
```

SQS キューの部分的なバッチレポートを設定する

```
Events:
  SQSEvent:
    Type: SQS
    Properties:
      Enabled: true
      FunctionResponseTypes:
        - ReportBatchItemFailures
      Queue: !GetAtt MySqsQueue.Arn
      BatchSize: 10
```

スケーリングが設定されているSQSイベントを含む Lambda 関数

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MySQSEvent:
```

```
        Type: SQS
        Properties:
          ...
          ScalingConfig:
            MaximumConcurrency: 10
```

FunctionCode

Lambda 関数のデプロイパッケージです。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

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

プロパティ

`Bucket`

関数と同じ AWS リージョンにある Amazon S3 バケット。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::Lambda::FunctionCode`データ型の `S3Bucket` プロパティに直接渡されます。

`Key`

デプロイパッケージの Amazon S3 キーです。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::Lambda::FunctionCode`データ型の `S3Key` プロパティに直接渡されます。

`Version`

バージョニングオブエクトの場合、使用するデプロイパッケージオブジェクトのバージョンです。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Lambda::FunctionCode`データ型の `S3ObjectVersion` プロパティに直接渡されます。

例

FunctionCode

`CodeUri`: 関数コードの例

YAML

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

FunctionUrlConfig

指定した設定パラメータを使用して AWS Lambda 関数 URL を作成します。Lambda 関数 URL は、関数を呼び出すために使用する HTTPS エンドポイントです。

デフォルトでは、作成する関数 URL は Lambda 関数のバージョン `$LATEST` を使用します。Lambda 関数に `AutoPublishAlias` を指定した場合、エンドポイントは指定された関数エイリアスに接続します。

詳細については、「AWS Lambda デベロッパーガイド」の「Lambda 関数 URL」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
AuthType: String
Cors: Cors
InvokeMode: String
```

プロパティ

`AuthType`

関数 URL が使用する認可のタイプ。AWS Identity and Access Management (IAM) を使用してリクエストを認可するには、[AWS_IAM] に設定します。オープンアクセスの場合は、[NONE] に設定します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::Url` リソースの `AuthType` プロパティに直接渡されます。

`Cors`

関数 URL のための、Cross-Origin Resource Sharing (CORS) 設定。

型: Cors

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::Url` リソースの `Cors` プロパティに直接渡されます。

`InvokeMode`

関数 URL が呼び出されるモード。呼び出しの完了後に関数がレスポンスを返すようにするには、`BUFFERED` に設定します。関数がレスポンスをストリーミングするようにするには、`RESPONSE_STREAM` に設定します。デフォルト値は `BUFFERED` です。

有効な値: `BUFFERED` または `RESPONSE_STREAM`

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Lambda::Url` リソースの `InvokeMode` プロパティに直接渡されます。

例

関数 URL

次の例では、関数 URL を使用して Lambda 関数を作成します。関数 URL は IAM 認可を使用します。

YAML

```
HelloWorldFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: hello_world/
    Handler: index.handler
    Runtime: nodejs20.x
    FunctionUrlConfig:
      AuthType: AWS_IAM
      InvokeMode: RESPONSE_STREAM

Outputs:
  MyFunctionUrlEndpoint:
      Description: "My Lambda Function URL Endpoint"
      Value:
        Fn::GetAtt: HelloWorldFunctionUrl.FunctionUrl
```

AWS::Serverless::GraphQLApi

AWS Serverless Application Model（ AWS SAM) `AWS::Serverless::GraphQLApi`リソースタイプを使用して を作成および設定する AWS AppSync GraphQL API サーバーレスアプリケーション用の 。

詳細については AWS AppSync、「 AWS AppSync デベロッパーガイド」の「 AWS AppSyncとは」を参照してください。

構文

YAML

```
LogicalId:
```

```
  Type: AWS::Serverless::GraphQLApi
  Properties:
    ApiKeys: ApiKeys
    Auth: Auth
    Cache: AWS::AppSync::ApiCache
    DataSources: DataSource
    DomainName: AWS::AppSync::DomainName
    Functions: Function
    Logging: LogConfig
    Name: String
    Resolvers: Resolver
    SchemaInline: String
    SchemaUri: String
    Tags:
    - Tag
    XrayEnabled: Boolean
```

プロパティ

`ApiKeys`

実行に使用できる一意のキーを作成する GraphQL API キーを必要とする オペレーション。

タイプ: ApiKeys

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`Auth`

の認証を設定する GraphQL API.

タイプ: Auth

必須: はい

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`Cache`

`CreateApiCache` 操作の入力です。

タイプ: AWS：AppSync:::ApiCache

必須: いいえ

AWS CloudFormation 互換性: このプロパティは AWS::AppSync::ApiCache リソースに直接渡されます。

`DataSources`

関数が接続 AWS AppSync するためのデータソースを に作成します。 は Amazon DynamoDB と AWS Lambda データソース AWS SAM をサポートしています。

タイプ: DataSource

必須: はい

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`DomainName`

のカスタムドメイン名 GraphQL API.

タイプ: AWS：AppSync:::DomainName

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、AWS::AppSync::DomainName resource. AWS SAM automatically によって AWS::AppSync::DomainNameApiAssociation resource に直接渡されます。

`Functions`

で関数を設定する GraphQL APIs 特定のオペレーションを実行する 。

タイプ: Function

必須: はい

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`Logging`

の Amazon CloudWatch ログ記録を設定します。GraphQL API.

このプロパティを指定しない場合、 AWS SAM は次の値を生成`CloudWatchLogsRoleArn`して設定します。

- `ExcludeVerboseContent: true`
- `FieldLogLevel: ALL`

ロギングをオプトアウトするには、以下を指定します。

```
Logging: false
```

タイプ: LogConfig

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::AppSync::GraphQLApi`リソースの`LogConfig`プロパティに直接渡されます。

`LogicalId`

の一意の名前 GraphQL API.

型: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、 `AWS::AppSync::GraphQLApi`リソースの`Name`プロパティに直接渡されます。

`Name`

の名前 GraphQL API。 `LogicalId` 値を上書きするには、このプロパティを指定します。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::AppSync::GraphQLApi`リソースの`Name`プロパティに直接渡されます。

`Resolvers`

のフィールドにリゾルバーを設定する GraphQL APIは をサポートします。 AWS SAM JavaScript パイプラインリゾルバー。

タイプ: Resolver

必須: はい

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

`SchemaInline`

のテキスト表現 GraphQL のスキーマ SDL の形式で設定。

型: 文字列

必須: 条件的。`SchemaInline` または `SchemaUri` を指定する必要があります。

AWS CloudFormation 互換性: このプロパティは、 `AWS::AppSync::GraphQLSchema`リソースの `Definition`プロパティに直接渡されます。

`SchemaUri`

スキーマの Amazon Simple Storage Service (Amazon S3) バケットURIまたはローカルフォルダへのパス。

ローカルフォルダへのパスを指定する場合、 AWS CloudFormation はデプロイ前にファイルを最初に Amazon S3 にアップロードする必要があります。は、 AWS SAM CLI このプロセスを容易にします。詳細については、「デプロイ時にローカルファイル AWS SAM をアップロードする方法」を参照してください。

型: 文字列

必須: 条件的。`SchemaInline` または `SchemaUri` を指定する必要があります。

AWS CloudFormation 互換性: このプロパティは、 `AWS::AppSync::GraphQLSchema`リソースの `DefinitionS3Location`プロパティに直接渡されます。

`Tags`

このタグ (キーと値のペア) GraphQL API。 タグを使用して、 リソースを識別および分類します。

タイプ: Tag のリスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::AppSync::GraphQLApi`リソースの `Tag`プロパティに直接渡されます。

`XrayEnabled`

このリソースに AWS X-Ray トレーシングを使用するかどうかを指定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::AppSync::GraphQLApi`リソースの `XrayEnabled`プロパティに直接渡されます。

戻り値

戻り値のリストについては、「」を参照してください。 AWS::Serverless::GraphQLApi「」(AWS CloudFormation ユーザーガイド) を参照してください。

例

GraphQL API DynamoDB データソースを使用する

この例では、GraphQL API データソースとして DynamoDB テーブルを使用する 。

schema.graphql

```
schema {
  query: Query
  mutation: Mutation
}

type Query {
  getPost(id: String!): Post
}

type Mutation {
  addPost(author: String!, title: String!, content: String!): Post!
}

type Post {
  id: String!
  author: String
  title: String
  content: String
  ups: Int!
  downs: Int!
  version: Int!
}
```

template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  DynamoDBPostsTable:
    Type: AWS::Serverless::SimpleTable

  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      SchemaUri: ./sam_graphql_api/schema.graphql
      Auth:
        Type: AWS_IAM
      DataSources:
        DynamoDb:
          PostsDataSource:
            TableName: !Ref DynamoDBPostsTable
            TableArn: !GetAtt DynamoDBPostsTable.Arn
      Functions:
        preprocessPostItem:
          Runtime:
            Name: APPSYNC_JS
            Version: 1.0.0
          DataSource: NONE
          CodeUri: ./sam_graphql_api/preprocessPostItem.js
        createPostItem:
          Runtime:
            Name: APPSYNC_JS
            Version: "1.0.0"
          DataSource: PostsDataSource
          CodeUri: ./sam_graphql_api/createPostItem.js
        getPostFromTable:
          Runtime:
            Name: APPSYNC_JS
            Version: "1.0.0"
          DataSource: PostsDataSource
          CodeUri: ./sam_graphql_api/getPostFromTable.js
      Resolvers:
        Mutation:
          addPost:
            Runtime:
              Name: APPSYNC_JS
```

```
            Version: "1.0.0"
          Pipeline:
          - preprocessPostItem
          - createPostItem
      Query:
        getPost:
          CodeUri: ./sam_graphql_api/getPost.js
          Runtime:
            Name: APPSYNC_JS
            Version: "1.0.0"
          Pipeline:
          - getPostFromTable
```

createPostItem.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const { key, values } = ctx.prev.result;
  return {
    operation: "PutItem",
    key: util.dynamodb.toMapValues(key),
    attributeValues: util.dynamodb.toMapValues(values),
  };
}

export function response(ctx) {
  return ctx.result;
}
```

getPostFromTable.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  return dynamoDBGetItemRequest({ id: ctx.args.id });
}

export function response(ctx) {
  return ctx.result;
}

/**
```

```
 * A helper function to get a DynamoDB item
 */
function dynamoDBGetItemRequest(key) {
  return {
    operation: "GetItem",
    key: util.dynamodb.toMapValues(key),
  };
}
```

preprocessPostItem.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const id = util.autoId();
  const { ...values } = ctx.args;
  values.ups = 1;
  values.downs = 0;
  values.version = 1;
  return { payload: { key: { id }, values: values } };
}

export function response(ctx) {
  return ctx.result;
}
```

リゾルバーコードは以下のとおりです。

getPost.js

```
export function request(ctx) {
  return {};
}

export function response(ctx) {
  return ctx.prev.result;
}
```

GraphQL API Lambda 関数をデータソースとして使用する

この例では、GraphQL API Lambda 関数をデータソースとして使用する 。

template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs20.x
      CodeUri: ./lambda

  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      Name: MyApi
      SchemaUri: ./gql/schema.gql
      Auth:
        Type: API_KEY
      ApiKeys:
        MyApiKey:
          Description: my api key
      DataSources:
        Lambda:
          MyLambdaDataSource:
            FunctionArn: !GetAtt MyLambdaFunction.Arn
      Functions:
        lambdaInvoker:
          Runtime:
            Name: APPSYNC_JS
            Version: 1.0.0
          DataSource: MyLambdaDataSource
          CodeUri: ./gql/invoker.js
      Resolvers:
        Mutation:
          addPost:
            Runtime:
              Name: APPSYNC_JS
              Version: 1.0.0
            Pipeline:
            - lambdaInvoker
        Query:
          getPost:
            Runtime:
```

```
              Name: APPSYNC_JS
              Version: 1.0.0
            Pipeline:
            - lambdaInvoker

Outputs:
  MyGraphQLAPI:
    Description: AppSync API
    Value: !GetAtt MyGraphQLAPI.GraphQLUrl
  MyGraphQLAPIMyApiKey:
    Description: API Key for authentication
    Value: !GetAtt MyGraphQLAPIMyApiKey.ApiKey
```

schema.graphql

```
schema {
  query: Query
  mutation: Mutation
}
type Query {
  getPost(id: ID!): Post
}
type Mutation {
  addPost(id: ID!, author: String!, title: String, content: String): Post!
}
type Post {
  id: ID!
  author: String!
  title: String
  content: String
  ups: Int
  downs: Int
}
```

関数は次のとおりです。

lambda/index.js

```
exports.handler = async (event) => {
  console.log("Received event {}", JSON.stringify(event, 3));

  const posts = {
    1: {
```

```
      id: "1",
      title: "First book",
      author: "Author1",
      content: "Book 1 has this content",
      ups: "100",
      downs: "10",
    },
  };

  console.log("Got an Invoke Request.");
  let result;
  switch (event.field) {
    case "getPost":
      return posts[event.arguments.id];
    case "addPost":
      // return the arguments back
      return event.arguments;
    default:
      throw new Error("Unknown field, unable to resolve " + event.field);
  }
};
```

invoker.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const { source, args } = ctx;
  return {
    operation: "Invoke",
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

export function response(ctx) {
  return ctx.result;
}
```

ApiKeys

API キーを必要とする GraphQL オペレーションを実行するために使用できる一意のキーを作成します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
LogicalId:
  ApiKeyId: String
  Description: String
  ExpiresOn: Double
```

プロパティ

`ApiKeyId`

API キーの一意の名前です。`LogicalId` 値を上書きするように指定します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::ApiKey` リソースの `ApiKeyId` プロパティに直接渡されます。

`Description`

API キーの説明です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::ApiKey` リソースの `Description` プロパティに直接渡されます。

`ExpiresOn`

API キーが失効してからの時間。日付は、エポックからの秒数で表され、最も近い時間に丸められます。

型: 倍精度

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::ApiKey リソースの Expires プロパティに直接渡されます。

LogicalId

API キーの一意の名前です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::ApiKey リソースの ApiKeyId プロパティに直接渡されます。

Auth

GraphQL API 用の認可を設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Additional:
- AuthProvider
LambdaAuthorizer: LambdaAuthorizerConfig
OpenIDConnect: OpenIDConnectConfig
Type: String
UserPool: UserPoolConfig
```

プロパティ

Additional

GraphQL API 向けの、追加の認可タイプのリストです。

タイプ: AuthProvider のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`LambdaAuthorizer`

Lambda 関数オーソライザーのオプションの認可設定を指定します。このオプションプロパティは、`Type` を `AWS_LAMBDA` として指定する場合に設定できます。

タイプ: LambdaAuthorizerConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` リソースの `LambdaAuthorizerConfig` プロパティに直接渡されます。

`OpenIDConnect`

OpenID Connect 準拠サービスのオプションの認可設定を指定します。このオプションプロパティは、`Type` を `OPENID_CONNECT` として指定する場合に設定できます。

タイプ: OpenIDConnectConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` リソースの `OpenIDConnectConfig` プロパティに直接渡されます。

`Type`

アプリケーションと AWS AppSync GraphQL API 間におけるデフォルトの認可タイプです。

許可される値のリストと説明については、「AWS AppSync デベロッパーガイド」の「認証と認可」を参照してください。

Lambda オーソライザー (`AWS_LAMBDA`) を指定すると、AWS SAM が、GraphQL API と Lambda 関数間の許可をプロビジョニングするための AWS Identity and Access Management (IAM) ポリシーを作成します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` リソースの `AuthenticationType` プロパティに直接渡されます。

`UserPool`

Amazon Cognito ユーザープールを使用するためのオプションの認可設定を指定します。このオプションプロパティは、`Type` を `AMAZON_COGNITO_USER_POOLS` として指定する場合に設定できます。

タイプ: UserPoolConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` リソースの `UserPoolConfig` プロパティに直接渡されます。

例

デフォルトとその他の認可タイプの設定

この例では、まず Lambda オーソライザーを GraphQL API のデフォルトの認可タイプとして設定します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      Auth:
        Type: AWS_LAMBDA
        LambdaAuthorizer:
          AuthorizerUri: !GetAtt Authorizer1.Arn
          AuthorizerResultTtlInSeconds: 10
          IdentityValidationExpression: hello
```

次に、AWS SAM テンプレートに以下を追加して、GraphQL API のその他の認可タイプを設定します。

```
        Additional:
        - Type: AWS_IAM
        - Type: API_KEY
        - Type: OPENID_CONNECT
```

```
          OpenIDConnect:
            AuthTTL: 10
            ClientId: myId
            IatTTL: 10
            Issuer: prod
```

この結果は、以下の AWS SAM テンプレートのようになります。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      Auth:
        Type: AWS_LAMBDA
        LambdaAuthorizer:
          AuthorizerUri: !GetAtt Authorizer1.Arn
          AuthorizerResultTtlInSeconds: 10
          IdentityValidationExpression: hello
        Additional:
        - Type: AWS_IAM
        - Type: API_KEY
        - Type: OPENID_CONNECT
          OpenIDConnect:
            AuthTTL: 10
            ClientId: myId
            IatTTL: 10
            Issuer: prod
```

AuthProvider

追加の GraphQL API 認可タイプ用のオプションの認可設定。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
LambdaAuthorizer: LambdaAuthorizerConfig
```

```
OpenIDConnect: OpenIDConnectConfig
Type: String
UserPool: UserPoolConfig
```

プロパティ

`LambdaAuthorizer`

AWS Lambda 関数オーソライザーのオプションの認可設定を指定します。このオプションプロパティは、`Type` を `AWS_LAMBDA` として指定する場合に設定できます。

タイプ: LambdaAuthorizerConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` `AdditionalAuthenticationProvider` オブジェクトの `LambdaAuthorizerConfig` プロパティに直接渡されます。

`OpenIDConnect`

OpenID Connect 準拠サービスのオプションの認可設定を指定します。このオプションプロパティは、`Type` を `OPENID_CONNECT` として指定する場合に設定できます。

タイプ: OpenIDConnectConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` `OpenIDConnectConfig` オブジェクトの `AdditionalAuthenticationProvider` プロパティに直接渡されます。

`Type`

アプリケーションと AWS AppSync GraphQL API 間におけるデフォルトの認可タイプです。

許可される値のリストと説明については、「AWS AppSync デベロッパーガイド」の「認証と認可」を参照してください。

Lambda オーソライザー (`AWS_LAMBDA`) を指定すると、AWS SAM が、GraphQL API と Lambda 関数間の許可をプロビジョニングするための AWS Identity and Access Management (IAM) ポリシーを作成します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` [AuthenticationType](#) オブジェクトの [AdditionalAuthenticationProvider](#) プロパティに直接渡されます。

`UserPool`

Amazon Cognito ユーザープールを使用するためのオプションの認可設定を指定します。このオプションプロパティは、`Type` を `AMAZON_COGNITO_USER_POOLS` として指定する場合に設定できます。

タイプ: [UserPoolConfig](#)

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::GraphQLApi` [UserPoolConfig](#) オブジェクトの [AdditionalAuthenticationProvider](#) プロパティに直接渡されます。

DataSource

GraphQL API リゾルバーが接続できるデータソースを設定します。AWS Serverless Application Model (AWS SAM) テンプレートを使用して、以下のデータソースへの接続を設定できます。

- Amazon DynamoDB
- AWS Lambda

データソースの詳細については、「AWS AppSync デベロッパーガイド」の「[データソースのアタッチ](#)」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DynamoDb: DynamoDb
```

```
Lambda: Lambda
```

プロパティ

DynamoDb

GraphQL API リゾルバーのデータソースとして DynamoDB テーブルを設定します。

タイプ: DynamoDB

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

Lambda

GraphQL API リゾルバーのデータソースとして Lambda 関数を設定します。

タイプ: Lambda

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

DynamoDb

GraphQL API リゾルバーのデータソースとして Amazon DynamoDB テーブルを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
LogicalId:
  DeltaSync: DeltaSyncConfig
  Description: String
  Name: String
  Permissions: List
  Region: String
```

```
  ServiceRoleArn: String
  TableArn: String
  TableName: String
  UseCallerCredentials: Boolean
  Versioned: Boolean
```

プロパティ

`DeltaSync`

デルタ同期の構成について説明します。

Type: DeltaSyncConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::DataSource` DynamoDBConfig オブジェクトの `DeltaSyncConfig` プロパティに直接渡されます。

`Description`

データソースの説明です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::DataSource` リソースの `Description` プロパティに直接渡されます。

`LogicalId`

データソースの一意の名前です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::DataSource` リソースの `Name` プロパティに直接渡されます。

`Name`

データソースの名前です。このプロパティを指定して、`LogicalId` 値を上書きします。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource リソースの Name プロパティに直接渡されます。

Permissions

AWS SAM コネクタ を使用して、データソースに許可をプロビジョニングします。リストには、以下の値のいずれかを指定できます。

- Read – データソースの読み取りをリゾルバーに許可します。
- Write – データソースの書き込みをリゾルバーに許可します。

AWS SAM は、デプロイ時に変換された AWS::Serverless::Connector リソースを使用して、許可をプロビジョニングします。生成されたリソースについては、「AWS::Serverless::Connector を指定したときに生成された AWS CloudFormation リソース」を参照してください。

> **Note**
>
> Permissions または ServiceRoleArn を指定できます。両方を指定することはできません。どちらも指定されていない場合、AWS SAM は Read および Write のデフォルト値を生成します。データソースへのアクセスを取り消すには、AWS SAM テンプレートから DynamoDB オブジェクトを削除します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。これは、AWS::Serverless::Connector リソースの Permissions プロパティに似ています。

Region

DynamoDB テーブルの AWS リージョン です。指定しない場合、AWS SAM は AWS::Region を使用します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource DynamoDBConfig オブジェクトの AwsRegion プロパティに直接渡されます。

ServiceRoleArn

データソースの AWS Identity and Access Management (IAM) サービスロール ARN です。システムは、データソースにアクセスするときにこのロールを引き受けます。

Permissions または ServiceRoleArn を指定できます。両方を指定することはできません。

型: 文字列

必須: いいえ。指定されていない場合、AWS SAM は Permissions のデフォルト値を適用します。

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource リソースの ServiceRoleArn プロパティに直接渡されます。

TableArn

DynamoDB テーブルの ARN です。

型: 文字列

必須: 条件的。ServiceRoleArn を指定しない場合は、TableArn が必要になります。

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

TableName

テーブルの名前。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource DynamoDBConfig オブジェクトの TableName プロパティに直接渡されます。

UseCallerCredentials

true に設定して、このデータソースで IAM を使用します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::DataSource` `DynamoDBConfig` オブジェクトの `UseCallerCredentials` プロパティに直接渡されます。

`Versioned`

`true` に設定して、このデータソースで Conflict Detection、Conflict Resolution、および Sync を使用します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::DataSource` `DynamoDBConfig` オブジェクトの `Versioned` プロパティに直接渡されます。

Lambda

GraphQL API リゾルバーのデータソースとして AWS Lambda 関数を設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
LogicalId:
  Description: String
  FunctionArn: String
  Name: String
  ServiceRoleArn: String
```

プロパティ

`Description`

データソースの説明です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource リソースの Description プロパティに直接渡されます。

FunctionArn

Lambda 関数の ARN。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource LambdaConfig オブジェクトの LambdaFunctionArn プロパティに直接渡されます。

LogicalId

データソースの一意の名前です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource リソースの Name プロパティに直接渡されます。

Name

データソースの名前です。このプロパティを指定して、LogicalId 値を上書きします。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::DataSource リソースの Name プロパティに直接渡されます。

ServiceRoleArn

データソースの AWS Identity and Access Management (IAM) サービスロール ARN です。システムは、データソースにアクセスするときにこのロールを引き受けます。

> **Note**
>
> データソースへのアクセスを取り消すには、AWS SAM テンプレートから Lambda オブジェクトを削除します。

型: 文字列

必須: いいえ。指定されていない場合は、AWS SAM が AWS SAM コネクタ を使用して `Write` 許可をプロビジョニングします。

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::DataSource` リソースの `ServiceRoleArn` プロパティに直接渡されます。

機能

特定のオペレーションを実行するように GraphQL API の関数を設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
LogicalId:
  CodeUri: String
  DataSource: String
  Description: String
  Id: String
  InlineCode: String
  MaxBatchSize: Integer
  Name: String
  Runtime: Runtime
  Sync: SyncConfig
```

プロパティ

`CodeUri`

関数コードの Amazon Simple Storage Service (Amazon S3) URI、またはローカルフォルダへのパスです。

ローカルフォルダへのパスを指定する場合、AWS CloudFormation では、最初にファイルを Amazon S3 にアップロードしてからデプロイする必要があります。AWS SAM CLI を使用することで、この処理を円滑化することができます。詳細については、「デプロイ時にローカルファイル AWS SAM をアップロードする方法」を参照してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration リソースの CodeS3Location プロパティに直接渡されます。

`DataSource`

この関数がアタッチされるデータソースの名前です。

- AWS::Serverless::GraphQLApi リソース内のデータソースを参照するには、その論理 ID を指定します。
- AWS::Serverless::GraphQLApi リソース外のデータソースを参照するには、Fn::GetAtt 組み込み関数を使用して、その Name 属性を指定します。例えば、!GetAtt MyLambdaDataSource.Name と指定します。
- 別のスタックからのデータソースを参照するには、Fn::ImportValue を使用します。

[NONE | None | none] のバリエーションが指定されている場合、AWS SAM は AWS::AppSync::DataSource Type オブジェクトの None 値を生成します。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration リソースの DataSourceName プロパティに直接渡されます。

`Description`

関数の説明です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration リソースの Description プロパティに直接渡されます。

Id

AWS::Serverless::GraphQLApi リソース外にある関数の関数 ID です。

- 同じ AWS SAM テンプレート内の関数を参照するには、Fn::GetAtt 組み込み関数を使用します。例: Id: !GetAtt createPostItemFunc.FunctionId
- 別のスタックからの関数を参照するには、Fn::ImportValue を使用します。

Id を使用する場合、これ以外のプロパティはいずれも許可されません。AWS SAM は、参照される関数の関数 ID を自動的に渡します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

InlineCode

リクエスト関数とレスポンス関数が含まれる関数コードです。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration リソースの Code プロパティに直接渡されます。

LogicalId

関数の一意の名前です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration リソースの Name プロパティに直接渡されます。

`MaxBatchSize`

`BatchInvoke` オペレーションの単一の AWS Lambda 機能に送信されるリゾルバーリクエスト入力の最大数を指定します。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::FunctionConfiguration` リソースの MaxBatchSize プロパティに直接渡されます。

`Name`

関数の名前 `LogicalId` 値を上書きするように指定します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::FunctionConfiguration` リソースの `Name` プロパティに直接渡されます。

`Runtime`

AWS AppSync パイプラインリゾルバーまたは AWS AppSync 関数が使用するランタイムについて説明します。使用するランタイムの名前とバージョンを指定します。

タイプ: Runtime

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。これは、`AWS::AppSync::FunctionConfiguration` リソースの `Runtime` プロパティに似ています。

`Sync`

関数の Sync 構成について説明します。

関数が呼び出されたときに、どの競合検出戦略および解決戦略を使用するかを指定します。

タイプ: SyncConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration リソースの SyncConfig プロパティに直接渡されます。

ランタイム

パイプラインリゾルバーまたは関数のランタイムです。使用する名前とバージョンを指定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Name: String
Version: String
```

プロパティ

Name

使用するランタイムの名前です。現在許容されている値は、APPSYNC_JS のみです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration AppSyncRuntime オブジェクトの Name プロパティに直接渡されます。

Version

使用するランタイムのバージョンです。現在許容されているバージョンは、1.0.0 のみです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::FunctionConfiguration AppSyncRuntime オブジェクトの RuntimeVersion プロパティに直接渡されます。

[リゾルバ]

GraphQL API のフィールドのリゾルバーを設定します。AWS Serverless Application Model (AWS SAM) は、JavaScript パイプラインリゾルバーをサポートしています。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
OperationType:
  LogicalId:
    Caching: CachingConfig
    CodeUri: String
    FieldName: String
    InlineCode: String
    MaxBatchSize: Integer
    Pipeline: List
    Runtime: Runtime
    Sync: SyncConfig
```

プロパティ

Caching

キャッシュが有効になっているリゾルバーのキャッシュ構成です。

タイプ: CachingConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::Resolver リソースの CachingConfig プロパティに直接渡されます。

`CodeUri`

リゾルバー関数コードの Amazon Simple Storage Service (Amazon S3) URI、またはローカルフォルダへのパスです。

ローカルフォルダへのパスを指定する場合、AWS CloudFormation では、最初にファイルを Amazon S3 にアップロードしてからデプロイする必要があります。AWS SAM CLI を使用することで、この処理を円滑化することができます。詳細については、「デプロイ時にローカルファイル AWS SAM をアップロードする方法」を参照してください。

`CodeUri` または `InlineCode` のどちらも指定されていない場合、AWS SAM は、リクエストを最初のパイプライン関数にリダイレクトし、最後のパイプライン関数からレスポンスを受け取る `InlineCode` を生成します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::Resolver` リソースの `CodeS3Location` プロパティに直接渡されます。

`FieldName`

リゾルバーの名前です。このプロパティを指定して、`LogicalId` 値を上書きします。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::Resolver` リソースの `FieldName` プロパティに直接渡されます。

`InlineCode`

リクエスト関数とレスポンス関数が含まれるリゾルバーコードです。

`CodeUri` または `InlineCode` のどちらも指定されていない場合、AWS SAM は、リクエストを最初のパイプライン関数にリダイレクトし、最後のパイプライン関数からレスポンスを受け取る `InlineCode` を生成します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::Resolver リソースの Code プロパティに直接渡されます。

LogicalId

リゾルバーの一意の名前です。GraphQL スキーマでは、リゾルバーが使用されるフィールド名が、リゾルバー名と一致する必要があります。LogicalId にも、それと同じフィールド名を使用してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

MaxBatchSize

BatchInvoke オペレーションの単一の AWS Lambda 機能に送信されるリゾルバーリクエスト入力の最大数を指定します。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::Resolver リソースの MaxBatchSize プロパティに直接渡されます。

OperationType

リゾルバーに関連付けられている GraphQL オペレーションタイプです。例えば、Query、Mutation、または Subscription などです。単一の OperationType 内にある LogicalId で、複数のリゾルバーをネストすることができます。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::AppSync::Resolver リソースの TypeName プロパティに直接渡されます。

Pipeline

パイプラインリゾルバーにリンクされている関数。リスト内の論理 ID で関数を指定します。

タイプ: リスト

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。これは、`AWS::AppSync::Resolver` リソースの `PipelineConfig` プロパティに似ています。

`Runtime`

パイプラインリゾルバーまたは関数のランタイムです。使用する名前とバージョンを指定します。

タイプ: Runtime

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。これは、`AWS::AppSync::Resolver` リソースの `Runtime` プロパティに似ています。

`Sync`

リゾルバーの同期構成について説明します。

リゾルバーが呼び出されたときに、どの競合検出戦略および解決戦略を使用するかを指定します。

タイプ: SyncConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::Resolver` リソースの `SyncConfig` プロパティに直接渡されます。

例

AWS SAM が生成したリゾルバー関数コードを使用して、フィールドを変数として保存する

以下は、この例の GraphQL スキーマです。

```
schema {
```

```
  query: Query
  mutation: Mutation
}

type Query {
  getPost(id: ID!): Post
}

type Mutation {
  addPost(author: String!, title: String!, content: String!): Post!
}

type Post {
  id: ID!
  author: String
  title: String
  content: String
}
```

以下は、AWS SAM テンプレートのスニペットです。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyGraphQLApi:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      ...
      Functions:
        preprocessPostItem:
          ...
        createPostItem:
          ...
      Resolvers:
        Mutation:
          addPost:
            Runtime:
              Name: APPSYNC_JS
              Version: 1.0.0
            Pipeline:
            - preprocessPostItem
            - createPostItem
```

AWS SAM テンプレートでは、CodeUri または InlineCode を指定しません。デプロイ時に、AWS SAM がリゾルバー用の次のインラインコードを自動生成します。

```
export function request(ctx) {
  return {};
}

export function response(ctx) {
  return ctx.prev.result;
}
```

このデフォルトリゾルバーコードは、リクエストを最初のパイプライン関数にリダイレクトし、最後のパイプライン関数からレスポンスを受け取ります。

最初のパイプライン関数では、提供された args フィールドを使用してリクエストオブジェクトを解析し、変数を作成することができます。作成後、関数内でこれらの変数を使用できます。以下は、preprocessPostItem 関数の例です。

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const author = ctx.args.author;
  const title = ctx.args.title;
  const content = ctx.args.content;

  // Use variables to process data

}

export function response(ctx) {
  return ctx.result;
}
```

ランタイム

パイプラインリゾルバーまたは関数のランタイムです。使用する名前とバージョンを指定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Name: String
Version: String
```

プロパティ

`Name`

使用するランタイムの名前です。現在許容されている値は、`APPSYNC_JS` のみです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::Resolver` `AppSyncRuntime` オブジェクトの `Name` プロパティに直接渡されます。

`Version`

使用するランタイムのバージョンです。現在許容されているバージョンは、`1.0.0` のみです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::AppSync::Resolver` `AppSyncRuntime` オブジェクトの `RuntimeVersion` プロパティに直接渡されます。

AWS::Serverless::HttpApi

REST API よりもレイテンシーとコストが低い RESTful API を作成できる Amazon API Gateway HTTP API を作成します。詳細については、API Gateway デベロッパーガイドの「HTTP API の操作」を参照してください。

API Gateway リソースへのアクセスを制御するオーソライザーがアタッチされていることを確認するため、AWS CloudFormation フックまたは IAM ポリシーを使用することをお勧めします。

AWS CloudFormation フックの使用の詳細については、「AWS CloudFormation CLI ユーザーガイド」の「Registering hooks」(フックの登録) および GitHub リポジトリの apigw-enforce-authorizer を参照してください。

IAM ポリシーの使用の詳細については、「API ゲートウェイデベロッパーガイド」の「API ルートに認可を要求する」を参照してください。

> **Note**
>
> AWS CloudFormation にデプロイすると、AWS SAM は、AWS SAM リソースを AWS CloudFormation リソースに変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Type: AWS::Serverless::HttpApi
Properties:
  AccessLogSettings: AccessLogSettings
  Auth: HttpApiAuth
  CorsConfiguration: String | HttpApiCorsConfiguration
  DefaultRouteSettings: RouteSettings
  DefinitionBody: JSON
  DefinitionUri: String | HttpApiDefinition
  Description: String
  DisableExecuteApiEndpoint: Boolean
  Domain: HttpApiDomainConfiguration
  FailOnWarnings: Boolean
  Name: String
  PropagateTags: Boolean
  RouteSettings: RouteSettings
  StageName: String
  StageVariables: Json
  Tags: Map
```

プロパティ

`AccessLogSettings`

ステージのアクセスロギングのための設定です。

タイプ: AccessLogSettings

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGatewayV2::Stage リソースの AccessLogSettings プロパティに直接渡されます。

`Auth`

API Gateway HTTP API へのアクセスを制御するための認可を設定します。

詳細については、API Gateway デベロッパーガイドの「JWT オーソライザーを使用した HTTP API へのアクセスの制御」を参照してください。

タイプ: HttpApiAuth

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`CorsConfiguration`

すべての API Gateway HTTP API のクロスオリジンリソース共有 (CORS) を管理します。文字列として許可するドメインを指定、または HttpApiCorsConfiguration オブジェクトを指定します。DefinitionBody プロパティが指定されている場合のみに CORS が機能するように、CORS には OpenAPI 定義を変更する AWS SAM が必要であることに注意してください。

詳細については、API Gateway デベロッパーガイドの「HTTP API の CORS の設定」を参照してください。

> **Note**
>
> OpenAPI 定義内とプロパティレベルの両方で CorsConfiguration が設定されている場合、AWS SAM は両方の設定ソースを統合し、プロパティが優先されます。このプロパティが true に設定されている場合、すべてのオリジンが許可されます。

タイプ: 文字列 | HttpApiCorsConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`DefaultRouteSettings`

この HTTP API のデフォルトルート設定です。これらの設定は、特定のルートの `RouteSettings` プロパティによって上書きされる場合を除き、すべてのルートに適用されます。

Type: RouteSettings

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::Stage` リソースの `RouteSettings` プロパティに直接渡されます。

`DefinitionBody`

HTTP API を説明する OpenAPI 定義です。`DefinitionUri` または `DefinitionBody` を指定しない場合は、AWS SAM がテンプレート設定に基づいて `DefinitionBody` を生成します。

Type: JSON

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::Api` リソースの `Body` プロパティに似ています。特定のプロパティが提供されている場合、AWS SAM は `DefinitionBody` を AWS CloudFormation に渡す前に、コンテンツの挿入や変更を行う場合があります。に渡される前に。プロパティには `Auth`、および対応する `AWS::Serverless::Function` のための HttpApi タイプの `EventSource` が含まれます。

`DefinitionUri`

HTTP API を定義する OpenAPI 定義の Amazon Simple Storage Service (Amazon S3) URI、ローカルファイルパス、またはロケーションオブジェクトです。このプロパティが参照する Amazon S3 オブジェクトは、有効な OpenAPI 定義ファイルである必要があります。`DefinitionUri` を指定しない、または `DefinitionBody` が指定されている場合は、AWS SAM がテンプレート設定に基づいて `DefinitionBody` を生成します。

ローカルファイルパスを指定する場合は、定義が適切に変換されるようにするために、テンプレートが `sam deploy` または `sam package` コマンドを含むワークフローを実行する必要があります。

DefinitionUri を使用して参照する外部 OpenApi 定義ファイルでは、組み込み関数はサポートされていません。OpenApi 定義をテンプレートにインポートするには、Include transform が含まれる DefinitionBody プロパティを使用します。

タイプ: 文字列 | HttpApiDefinition

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGatewayV2::Api リソースの BodyS3Location プロパティに似ています。ネストされた Amazon S3 プロパティには異なる名前が付けられています。

Description

HTTP API リソースの説明です。

Description を指定すると、AWS SAM は description フィールドを設定して HTTP API リソースの OpenAPI 定義を変更します。次のシナリオではエラーが発生します。

- DefinitionBody プロパティが Open API 定義で設定された description フィールドで指定されている – これは AWS SAM で解決されない description フィールドの競合を引き起こします。
- DefinitionUri プロパティが指定されている – AWS SAM は Amazon S3 から取得された Open API 定義を変更しません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

DisableExecuteApiEndpoint

クライアントがデフォルトの execute-api エンドポイント https://{api_id}.execute-api.{region}.amazonaws.com を使用して HTTP API を呼び出すことができるかどうかを指定します。デフォルトで、クライアントはデフォルトのエンドポイントを使用して API を呼び出すことができます。クライアントが API の呼び出しにカスタムドメイン名以外を使用しないようにするには、デフォルトのエンドポイントを無効にします。

このプロパティを使用するには、 DefinitionBody プロパティではなく DefinitionUriプロパティを指定するか、OpenAPI 定義の x-amazon-apigateway-endpoint-configuration を disableExecuteApiEndpoint に定義する必要があります。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGatewayV2::Api リソースの DisableExecuteApiEndpoint プロパティに似ています。これは x-amazon-apigateway-endpoint-configuration 拡張機能の disableExecuteApiEndpoint プロパティに直接渡され、AWS::ApiGatewayV2::Api リソースの Body プロパティに追加されます。

`Domain`

この API Gateway HTTP API のカスタムドメインを設定します。

タイプ: HttpApiDomainConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FailOnWarnings`

警告が発生したときに HTTP API の作成をロールバックするか (`true`) しないか (`false`) を指定します。デフォルト値は `false` です。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::Api` リソースの `FailOnWarnings` プロパティに直接渡されます。

`Name`

HTTP API リソースの名前。

`Name` を指定すると、AWS SAM は `title` フィールドを設定して HTTP API リソースの OpenAPI 定義を変更します。次のシナリオではエラーが発生します。

- `DefinitionBody` プロパティが Open API 定義で設定された `title` フィールドで指定されている – これは AWS SAM で解決されない `title` フィールドの競合を引き起こします。
- `DefinitionUri` プロパティが指定されている – AWS SAM は Amazon S3 から取得された Open API 定義を変更しません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`PropagateTags`

AWS::Serverless::HttpApi が生成したリソースに `Tags` プロパティからのタグを渡すかどうかを指定します。`True` を指定して、生成されたリソースにタグを伝播します。

タイプ: ブール

必須: いいえ

デフォルト: `False`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`RouteSettings`

この HTTP API に対するルートごとのルート設定です。詳細については、API Gateway デベロッパーガイドの「HTTP API のルートの使用」を参照してください。

Type: RouteSettings

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::Stage` リソースの `RouteSettings` プロパティに直接渡されます。

`StageName`

API ステージの名前です。名前が指定されていない場合、AWS SAM は API Gateway からの `$default` ステージを使用します。

型: 文字列

必須: いいえ

デフォルト: $default

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGatewayV2::Stage リソースの StageName プロパティに直接渡されます。

StageVariables

ステージ変数を定義するマップです。変数名には、英数字とアンダースコアを使用できます。値は [A-Za-z0-9-._~:/?#&=,]+ に一致する必要があります。

タイプ: Json

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGatewayV2::Stage リソースの StageVariables プロパティに直接渡されます。

Tags

この API Gateway ステージに追加するタグを指定するマップ (文字列対文字列) です。キーの長さは 1 ~ 128 文字の Unicode 文字で、プレフィックス aws: を含めることはできません。以下の文字を使用できます。一連の Unicode 文字、数字、空白、_、.、/、=、+、-。値は 1 ~ 256 文字の Unicode 文字にすることができます。

タイプ: マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

その他の注意点: Tags プロパティは OpenAPI 定義を変更する AWS SAM を必要とするため、タグが追加されるのは DefinitionBody プロパティが指定されている場合のみで、DefinitionUri プロパティが指定されている場合にタグは追加されません。AWS SAM は httpapi:createdBy:SAM タグを自動的に追加します。タグは、AWS::ApiGatewayV2::Stage リソースと AWS::ApiGatewayV2::DomainName リソース (DomainName が指定されている場合) にも追加されます。

戻り値

参照番号

このリソースの論理 ID を Ref 組み込み関数に渡すと、Ref は基盤となる AWS::ApiGatewayV2::Api リソースの API ID (a1bcdef2gh など) を返します。

Ref 関数の使用方法の詳細については、AWS CloudFormation ユーザーガイドの「Ref」を参照してください。

例

シンプルな HttpApi

以下の例は、Lambda 関数によってサポートされる HTTP API エンドポイントをセットアップするために必要な最小限の内容を示しています。この例は、AWS SAM が作成するデフォルトの HTTP API を使用します。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS SAM template with a simple API definition
Resources:
  ApiFunction:
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: HttpApi
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
            return {'body': 'Hello World!', 'statusCode': 200}
      Runtime: python3.7
Transform: AWS::Serverless-2016-10-31
```

Auth を使用した HTTP API

以下の例は、API エンドポイントで認可をセットアップする方法を示しています。

YAML

```
Properties:
  FailOnWarnings: true
  Auth:
    DefaultAuthorizer: OAuth2
    Authorizers:
      OAuth2:
        AuthorizationScopes:
```

```
          - scope4
        JwtConfiguration:
          issuer: "https://www.example.com/v1/connect/oauth2"
          audience:
            - MyApi
        IdentitySource: "$request.querystring.param"
```

OpenAPI 定義を使用した HttpAPI

以下の例は、テンプレートに OpenAPI 定義を追加する方法を示しています。

AWS SAM は、この HTTP API を参照する HttpApi イベントに欠落しているすべての Lambda 統合を補充することに注意してください。AWS SAM は、HttpApi イベントが参照するパスが欠落している場合もそれらを追加します。

YAML

```
Properties:
  FailOnWarnings: true
  DefinitionBody:
    info:
      version: '1.0'
      title:
        Ref: AWS::StackName
    paths:
      "/":
        get:
          security:
          - OpenIdAuth:
            - scope1
            - scope2
          responses: {}
    openapi: 3.0.1
    securitySchemes:
      OpenIdAuth:
        type: openIdConnect
        x-amazon-apigateway-authorizer:
          identitySource: "$request.querystring.param"
          type: jwt
          jwtConfiguration:
            audience:
            - MyApi
            issuer: https://www.example.com/v1/connect/oidc
```

```
          openIdConnectUrl: https://www.example.com/v1/connect/oidc/.well-known/openid-
configuration
```

構成設定を使用した HttpApi

以下の例は、テンプレートに HTTP API とステージ構成を追加する方法を示しています。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Parameters:
  StageName:
    Type: String
    Default: Prod

Resources:
  HttpApiFunction:
    Type: AWS::Serverless::Function
    Properties:
      InlineCode: |
          def handler(event, context):
              import json
              return {
                  "statusCode": 200,
                  "body": json.dumps(event),
              }
      Handler: index.handler
      Runtime: python3.7
      Events:
        ExplicitApi: # warning: creates a public endpoint
          Type: HttpApi
          Properties:
            ApiId: !Ref HttpApi
            Method: GET
            Path: /path
            TimeoutInMillis: 15000
            PayloadFormatVersion: "2.0"
            RouteSettings:
              ThrottlingBurstLimit: 600

  HttpApi:
    Type: AWS::Serverless::HttpApi
    Properties:
```

```
      StageName: !Ref StageName
      Tags:
        Tag: Value
      AccessLogSettings:
        DestinationArn: !GetAtt AccessLogs.Arn
        Format: $context.requestId
      DefaultRouteSettings:
        ThrottlingBurstLimit: 200
      RouteSettings:
        "GET /path":
          ThrottlingBurstLimit: 500 # overridden in HttpApi Event
      StageVariables:
        StageVar: Value
      FailOnWarnings: true

  AccessLogs:
    Type: AWS::Logs::LogGroup

Outputs:
  HttpApiUrl:
    Description: URL of your API endpoint
    Value:
      Fn::Sub: 'https://${HttpApi}.execute-api.${AWS::Region}.${AWS::URLSuffix}/
${StageName}/'
  HttpApiId:
    Description: Api id of HttpApi
    Value:
      Ref: HttpApi
```

HttpApiAuth

Amazon API Gateway HTTP API へのアクセスを制御するための認可を設定します。

HTTP API へのアクセス権の設定に関する詳細については、API Gateway デベロッパーガイドの「API Gateway での HTTP API へのアクセスの制御と管理」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Authorizers: OAuth2Authorizer | LambdaAuthorizer
```

```
  DefaultAuthorizer: String
  EnableIamAuthorizer: Boolean
```

プロパティ

`Authorizers`

API Gateway API へのアクセスを制御するために使用されるオーソライザーです。

タイプ: OAuth2Authorizer | LambdaAuthorizer

必須: いいえ

デフォルト: なし

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

その他の注意点: AWS SAM は、OpenAPI 定義にオーソライザーを追加します。

`DefaultAuthorizer`

API Gateway API に対する API コールの認証に使用するデフォルトのオーソライザーを指定します。`EnableIamAuthorizer` が `true` に設定されている場合、`AWS_IAM` をデフォルトの承認者として指定できます。それ以外の場合は、`Authorizers` で定義した承認者を指定します。

型: 文字列

必須: いいえ

デフォルト: なし

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`EnableIamAuthorizer`

API ルートに IAM 認可を使用するかどうかを指定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

OAuth 2.0 オーソライザー

OAuth 2.0 オーソライザーの例

YAML

```
Auth:
  Authorizers:
    OAuth2Authorizer:
      AuthorizationScopes:
        - scope1
        - scope2
      JwtConfiguration:
        issuer: "https://www.example.com/v1/connect/oauth2"
        audience:
          - MyApi
      IdentitySource: "$request.querystring.param"
  DefaultAuthorizer: OAuth2Authorizer
```

IAM 承認者

IAM 承認者の例

YAML

```
Auth:
  EnableIamAuthorizer: true
  DefaultAuthorizer: AWS_IAM
```

LambdaAuthorizer

AWS Lambda 関数を使用して、Amazon API Gateway HTTP API へのアクセスを制御するように Lambda オーソライザーを設定します。

詳細については、API Gateway デベロッパーガイドの「HTTP API の AWS Lambda オーソライザーの使用」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  AuthorizerPayloadFormatVersion: String
  EnableFunctionDefaultPermissions: Boolean
  EnableSimpleResponses: Boolean
  FunctionArn: String
  FunctionInvokeRole: String
  Identity: LambdaAuthorizationIdentity
```

プロパティ

`AuthorizerPayloadFormatVersion`

HTTP API Lambda オーソライザーに送信されるペイロードの形式を指定します。HTTP API Lambda オーソライザーに必要です。

これは、OpenAPI 定義の `securitySchemes` セクションにある `x-amazon-apigateway-authorizer` の `authorizerPayloadFormatVersion` セクションに渡されます。

有効な値: `1.0` または `2.0`

タイプ: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`EnableFunctionDefaultPermissions`

デフォルトでは、HTTP API リソースには Lambda オーソライザーを呼び出すための許可が付与されていません。HTTP API リソースと Lambda オーソライザーの間の許可を自動的に作成するには、このプロパティを `true` として指定します。

タイプ: ブール

必須: いいえ

デフォルト値: `false`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`EnableSimpleResponses`

Lambda オーソライザーがシンプルな形式でレスポンスを返すかどうかを指定します。デフォルトで、Lambda オーソライザーは AWS Identity and Access Management (IAM) ポリシーを返す必要があります。有効にした場合、Lambda オーソライザーは IAM ポリシーの代わりにブール値を返すことができます。

これは、OpenAPI 定義の `securitySchemes` セクションにある `x-amazon-apigateway-authorizer` の `enableSimpleResponses` セクションに渡されます。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionArn`

API の認可を提供する Lambda 関数の Amazon リソースネーム (ARN) です。

これは、OpenAPI 定義の `securitySchemes` セクションにある `x-amazon-apigateway-authorizer` の `authorizerUri` セクションに渡されます。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`FunctionInvokeRole`

API Gateway がオーソライザー関数を呼び出すために必要な認証情報を持つ IAM ロールの ARN です。このパラメータは、関数のリソースベースのポリシーが API Gateway に `lambda:InvokeFunction` 許可を付与しない場合に指定します。

これは、OpenAPI 定義の `securitySchemes` セクションにある `x-amazon-apigateway-authorizer` の `authorizerCredentials` セクションに渡されます。

詳細については、API Gateway デベロッパーガイドの「Lambda オーソライザーの作成」を参照してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Identity`

オーソライザーの受信リクエストに `IdentitySource` を指定します。

これは、OpenAPI 定義の `securitySchemes` セクションにある `x-amazon-apigateway-authorizer` の `identitySource` セクションに渡されます。

タイプ: LambdaAuthorizationIdentity

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

LambdaAuthorizer

LambdaAuthorizer の例

YAML

```
Auth:
  Authorizers:
    MyLambdaAuthorizer:
      AuthorizerPayloadFormatVersion: 2.0
      FunctionArn:
        Fn::GetAtt:
          - MyAuthFunction
          - Arn
      FunctionInvokeRole:
        Fn::GetAtt:
          - LambdaAuthInvokeRole
          - Arn
      Identity:
        Headers:
          - Authorization
```

LambdaAuthorizationIdentity

このプロパティは、Lambda オーソライザーの受信リクエストに IdentitySource を指定するために使用できます。アイデンティティソースの詳細については、API Gateway デベロッパーガイドの「ID ソース」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

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

プロパティ

`Context`

所定のコンテキスト文字列を、`$context.contextString` 形式のマッピング式のリストに変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Headers`

ヘッダーを、`$request.header.name` 形式のマッピング式のリストに変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`QueryStrings`

所定のクエリ文字列を、`$request.querystring.queryString` 形式のマッピング式のリストに変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ReauthorizeEvery`

API Gateway がオーソライザー結果をキャッシュする時間を指定する有効期限 (TTL) (秒) です。0 より大きい値を指定する場合、API Gateway が認証レスポンスをキャッシュします。最大値は 3600、つまり 1 時間です。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`StageVariables`

所定のステージ変数を、`$stageVariables.stageVariable` 形式のマッピング式のリストに変換します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

LambdaRequestIdentity

Lambda リクエストアイデンティティの例

YAML

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

OAuth2Authorizer

JSON Web トークン (JWT) オーソライザーとしても知られる OAuth 2.0 オーソライザーの定義です。

詳細については、API Gateway デベロッパーガイドの「JWT オーソライザーを使用した HTTP API へのアクセスの制御」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  AuthorizationScopes: List
  IdentitySource: String
  JwtConfiguration: Map
```

プロパティ

`AuthorizationScopes`

このオーソライザーの認可スコープのリストです。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IdentitySource`

このオーソライザーのアイデンティティソース式です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`JwtConfiguration`

このオーソライザーの JWT 設定です。

これは、OpenAPI 定義の `securitySchemes` セクションにある `x-amazon-apigateway-authorizer` の `jwtConfiguration` セクションに渡されます。

> **Note**
>
> プロパティ `issuer` と `audience` は大文字と小文字を区別せず、OpenAPI のように小文字を使用することも、AWS::ApiGatewayV2::Authorizer のように大文字の `Issuer` および `Audience` を使用することもできます。

タイプ: マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

OAuth 2.0 オーソライザー

OAuth 2.0 オーソライザーの例

YAML

```
Auth:
```

```
  Authorizers:
    OAuth2Authorizer:
      AuthorizationScopes:
        - scope1
      JwtConfiguration:
        issuer: "https://www.example.com/v1/connect/oauth2"
        audience:
          - MyApi
      IdentitySource: "$request.querystring.param"
  DefaultAuthorizer: OAuth2Authorizer
```

HttpApiCorsConfiguration

HTTP API のクロスオリジンリソース共有 (CORS) を管理します。許可するドメインを文字列として指定するか、追加の CORS 設定でディクショナリを指定します。注意: CORS では OpenAPI 定義の変更に SAM が必要となるため、これが機能するのは `DefinitionBody` で Inline OpenApi が定義されている場合のみです。

CORS の詳細については、API Gateway デベロッパーガイドの「HTTP API の CORS の設定」を参照してください。

注意: OpenAPI とプロパティレベルの両方で HTTPAPICorsConfiguration が設定されている場合、AWS SAM はそれらを統合し、プロパティが優先されます。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
AllowCredentials: Boolean
AllowHeaders: List
AllowMethods: List
AllowOrigins: List
ExposeHeaders: List
MaxAge: Integer
```

プロパティ

`AllowCredentials`

CORS リクエストに認証情報を含めるかどうかを指定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AllowHeaders`

許可されるヘッダーのコレクションを表します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AllowMethods`

許可される HTTP メソッドのコレクションを表します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AllowOrigins`

許可されたオリジンのコレクションを表します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ExposeHeaders`

公開されたヘッダーのコレクションを表します。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`MaxAge`

ブラウザがプリフライトリクエスト結果をキャッシュする秒数。

タイプ: 整数

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

HttpApiCorsConfiguration

HTTP API Cors 設定の例です。

YAML

```
CorsConfiguration:
  AllowOrigins:
    - "https://example.com"
  AllowHeaders:
    - x-apigateway-header
  AllowMethods:
    - GET
  MaxAge: 600
  AllowCredentials: true
```

HttpApiDefinition

を定義する OpenAPI ドキュメントAPI。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

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

プロパティ

`Bucket`

OpenAPI ファイルが保存されている Amazon S3 バケットの名前。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::ApiGatewayV2::ApiBodyS3Location`データ型の `Bucket` プロパティに直接渡されます。

`Key`

OpenAPI ファイルの Amazon S3 キー。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::ApiGatewayV2::ApiBodyS3Location`データ型の `Key` プロパティに直接渡されます。

`Version`

バージョニングされたオブジェクトの場合、OpenAPI ファイルのバージョン。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::ApiGatewayV2::ApiBodyS3Location`データ型の `Version` プロパティに直接渡されます。

例

定義 URI の例

API 定義の例

YAML

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

HttpApiDomainConfiguration

API のカスタムドメインを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  BasePath: List
  CertificateArn: String
  DomainName: String
  EndpointConfiguration: String
  MutualTlsAuthentication: MutualTlsAuthentication
  OwnershipVerificationCertificateArn: String
  Route53: Route53Configuration
  SecurityPolicy: String
```

プロパティ

`BasePath`

Amazon API Gateway ドメイン名で設定する basepaths のリストです。

タイプ: リスト

必須: いいえ

デフォルト: /

AWS CloudFormation との互換性: このプロパティは AWS::ApiGatewayV2::ApiMapping リソースの ApiMappingKey プロパティと似ています。AWS SAM は複数の AWS::ApiGatewayV2::ApiMapping リソースを作成します (このプロパティに指定された値につき 1 つ)。

`CertificateArn`

このドメイン名のエンドポイント用の AWS マネージド証明書の Amazon リソースネーム (ARN) です。サポートされるソースは AWS Certificate Manager のみです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway2::DomainName DomainNameConfiguration リソースの CertificateArn プロパティに直接渡されます。

`DomainName`

API Gateway API のカスタムドメイン名です。大文字はサポートされていません。

このプロパティが設定されていると、AWS SAM は AWS::ApiGatewayV2::DomainName リソースを生成します。このシナリオの詳細については、「DomainName プロパティが指定されている」を参照してください。生成された AWS CloudFormation リソースについては、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::ApiGateway2::DomainName リソースの DomainName プロパティに直接渡されます。

`EndpointConfiguration`

カスタムドメインにマップする API Gateway エンドポイントのタイプを定義します。このプロパティの値は、CertificateArn プロパティが AWS CloudFormation でマップされる方法を決定します。

HTTP API に有効な値は REGIONAL のみです。

型: 文字列

必須: いいえ

デフォルト: REGIONAL

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`MutualTlsAuthentication`

カスタムドメイン名の相互 Transport Layer Security (TLS) 認証設定です。

タイプ: MutualTlsAuthentication

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::DomainName` リソースの `MutualTlsAuthentication` プロパティに直接渡されます。

`OwnershipVerificationCertificateArn`

カスタムドメインの所有権を検証するために ACM によって発行されたパブリック証明書の ARN。相互 TLS を設定し、ACM にインポートされた、またはプライベート CA 証明書の ARN を `CertificateArn` に指定する場合のみ必須です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::DomainName` `DomainNameConfiguration` データ型の `OwnershipVerificationCertificateArn` プロパティに直接渡されます。

`Route53`

Amazon Route 53 設定を定義します。

タイプ: Route53Configuration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SecurityPolicy`

このドメイン名に対するセキュリティポリシーの TLS バージョンです。

HTTP API に有効な値は `TLS_1_2` のみです。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::ApiGatewayV2::DomainName` `DomainNameConfiguration` データ型の `SecurityPolicy` プロパティに直接渡されます。

例

DomainName

DomainName の例

YAML

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

Route53Configuration

API の Route53 レコードセットを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  DistributionDomainName: String
```

```
  EvaluateTargetHealth: Boolean
  HostedZoneId: String
  HostedZoneName: String
  IpV6: Boolean
  Region: String
  SetIdentifier: String
```

プロパティ

`DistributionDomainName`

API カスタムドメイン名のカスタムディストリビューションを設定します。

型: 文字列

必須: いいえ

デフォルト: API Gateway ディストリビューションを使用します。

AWS CloudFormation との互換性: このプロパティは、`AWS::Route53::RecordSetGroup` `AliasTarget` リソースの `DNSName` プロパティに直接渡されます。

その他の注意点: CloudFront ディストリビューションのドメイン名です。

`EvaluateTargetHealth`

EvaluateTargetHealth が true の場合は、参照される AWS リソース (Elastic Load Balancing ロードバランサーやホストゾーン内の別のレコードなど) のヘルスがエイリアスレコードに継承されます。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Route53::RecordSetGroup` `AliasTarget` リソースの `EvaluateTargetHealth` プロパティに直接渡されます。

その他の注意点: エイリアスターゲットが CloudFront ディストリビューションの場合は EvaluateTargetHealth を true に設定できません。

`HostedZoneId`

レコードを作成するホストゾーンの ID です。

HostedZoneName または HostedZoneId を指定します。両方を指定することはできません。同じドメイン名のホストゾーンが複数ある場合は、HostedZoneId を使用してホストゾーンを指定する必要があります。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Route53::RecordSetGroup RecordSet リソースの HostedZoneId プロパティに直接渡されます。

HostedZoneName

レコードを作成するホストゾーンの名前です。HostedZoneName の一部には、末尾のドット (www.example.com. など) を含める必要があります。

HostedZoneName または HostedZoneId を指定します。両方を指定することはできません。同じドメイン名のホストゾーンが複数ある場合は、HostedZoneId を使用してホストゾーンを指定する必要があります。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Route53::RecordSetGroup RecordSet リソースの HostedZoneName プロパティに直接渡されます。

IpV6

このプロパティを設定すると、AWS SAM が AWS::Route53::RecordSet リソースを作成し、提供された HostedZone のタイプを AAAA に設定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

Region

レイテンシーベースのリソースレコードセットのみ: このリソースレコードセットが参照するリソースを作成した Amazon EC2 リージョン。リソースは通常、AWS リソース (EC2 インスタンスや ELB ロードバランサーなど) であり、レコードタイプに応じて IP アドレスまたは DNS ドメイン名で参照されます。

レイテンシーリソースレコードセットが作成されているドメインの名前や種類を要求する DNS クエリを受け取ると、Amazon Route 53 は、エンドユーザーとそのユーザーに関連付けられている Amazon EC2 リージョンとの間でレイテンシーが最も小さいレイテンシーリソースレコードセットを選択します。その後、Route 53 は、選択したリソースレコードセットに関連付けられている値を返します。

次の点に注意してください。

- レイテンシーリソースレコードセットごとに 1 つの `ResourceRecord` のみ指定できます。
- 作成できるレイテンシーリソースレコードセットは、各 Amazon EC2 リージョンにつき 1 つだけです。
- すべての Amazon EC2 リージョンに対してレイテンシーリソースレコードセットを作成する必要はありません。レイテンシーリソースレコードセットを作成したリージョンの中から、レイテンシーの最も小さいリージョンが Route 53 によって選択されます。
- レイテンシーリソースレコードセットリソースと `Name` および `Type` 要素の値が同じである非レイテンシーリソースレコードセットを作成することはできません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Route53::RecordSetGroup` `RecordSet` データ型の `Region` プロパティに直接渡されます。

`SetIdentifier`

シンプル以外のルーティングポリシーを持つリソースレコードセット: タイプが A である acme.example.com という名前の複数の加重リソースレコードセットなど、名前とタイプの組み合わせが同じである複数のリソースレコードセットを区別する識別子。名前とタイプが同じであるリソースレコードセットのグループでは、リソースレコードセットごとに `SetIdentifier` の値が一意である必要があります。

ルーティングポリシーの詳細については、「Amazon Route 53 デベロッパーガイド」の「ルーティングポリシーの選択」を参照してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Route53::RecordSetGroup` `RecordSet` データ型の `SetIdentifier` プロパティに直接渡されます。

例

Route 53 設定の例

この例は、Route 53 の設定方法を示しています。

YAML

```
Domain:
  DomainName: www.example.com
  CertificateArn: arn-example
  EndpointConfiguration: EDGE
  Route53:
    HostedZoneId: Z1PA6795UKMFR9
    EvaluateTargetHealth: true
    DistributionDomainName: xyz
```

AWS::Serverless::LayerVersion

Lambda 関数に必要なライブラリまたはランタイムコード LayerVersion を含む Lambda を作成します。

AWS::Serverless::LayerVersion リソースは Metadataリソース属性もサポートしているため、アプリケーションに含まれるレイヤーを構築する AWS SAM ように に指示できます。レイヤーの構築に関する詳細については、「AWS SAM での Lambda レイヤーの構築」を参照してください。

重要な注意: でのUpdateReplacePolicyリソース属性のリリース以降 AWS CloudFormation、AWS::Lambda::LayerVersion (推奨) には、 と同じ利点がありますAWS::Serverless::LayerVersion。

サーバーレス LayerVersion が変換されると、 はリソースの論理 ID SAMも変換するため、リソースが更新され CloudFormation ても LayerVersions によって古い が自動的に削除されることはありません。

> **Note**
>
> にデプロイすると AWS CloudFormation、 は AWS SAM リソースを AWS CloudFormation リソース AWS SAM に変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
Type: AWS::Serverless::LayerVersion
Properties:
  CompatibleArchitectures: List
  CompatibleRuntimes: List
  ContentUri: String | LayerContent
  Description: String
  LayerName: String
  LicenseInfo: String
  PublishLambdaVersion: Boolean
  RetentionPolicy: String
```

プロパティ

`CompatibleArchitectures`

レイヤバージョンでサポートされる命令セットアーキテクチャを指定します。

このプロパティの詳細については、AWS Lambda デベロッパーガイドの「Lambda 命令セットアーキテクチャ」を参照してください。

有効な値: `x86_64`、`arm64`

タイプ: リスト

必須: いいえ

デフォルト: `x86_64`

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::LayerVersion`リソースの `CompatibleArchitectures`プロパティに直接渡されます。

`CompatibleRuntimes`

これと互換性のあるランタイムのリスト LayerVersion。

タイプ: リスト

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::LayerVersion`リソースの `CompatibleRuntimes`プロパティに直接渡されます。

`ContentUri`

Amazon S3 Uri、ローカルフォルダへのパス、またはレイヤーコードの LayerContent オブジェクト。

Amazon S3 Uri または LayerContent オブジェクトが指定されている場合、参照される Amazon S3 オブジェクトは、Lambda レイヤーの内容を含む有効なZIPアーカイブである必要があります。

ローカルフォルダへのパスが提供されている場合は、コンテンツが適切に変換されるようにするために、sam build が含まれ、その後に sam deploy または sam package が続くワークフローをテンプレートが実行する必要があります。デフォルトでは、相対パスは AWS SAM テンプレートの場所に関して解決されます。

タイプ: 文字列 | LayerContent

必須: はい

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::LayerVersion`リソースの `Content`プロパティに似ています。ネストされた Amazon S3 プロパティには異なる名前が付けられています。

`Description`

このレイヤーの説明です。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 `AWS::Lambda::LayerVersion`リソースの `Description`プロパティに直接渡されます。

`LayerName`

レイヤーの名前または Amazon リソースネーム (ARN) 。

型: 文字列

必須: いいえ

デフォルト: リソースの論理 ID

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::LayerVersionリソースの LayerNameプロパティに似ています。名前を指定しない場合は、リソースの論理 ID が名前として使用されます。

LicenseInfo

このライセンスに関する情報 LayerVersion。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、 AWS::Lambda::LayerVersionリソースの LicenseInfoプロパティに直接渡されます。

PublishLambdaVersion

参照されるLayerVersionリソースに変更があるたびに新しい Lambda バージョンを作成するオプトインプロパティ。接続された Lambda 関数AutoPublishAliasAllPropertiesで AutoPublishAliasと を使用して有効にすると、LayerVersionリソースに加えられた変更ごとに新しい Lambda バージョンが作成されます。

タイプ: ブール

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

RetentionPolicy

このプロパティは、リソースを削除するときに LayerVersion の古いバージョンを保持するか削除するかを指定します。リソースを更新または置き換えるときに LayerVersion の古いバージョンを保持する必要がある場合は、 UpdateReplacePolicy 属性を有効にする必要があります。これを行う方法については、「AWS CloudFormation ユーザーガイド」の「UpdateReplacePolicy 属性」を参照してください。

有効な値: Retain または Delete

タイプ：文字列

必須: いいえ

AWS CloudFormation 互換性: このプロパティは に固有 AWS SAM であり、 AWS CloudFormation 同等のものはありません。

その他の注意点: を指定するとRetain、 は変換されたAWS::Lambda::LayerVersionリソースDeletionPolicy: Retainに AWS SAM でサポートされているリソース属性 の AWS SAM を追加します。

戻り値

参照番号

このリソースの論理 ID がRef組み込み 関数に提供されると、基盤となる Lambda ARNのリソースが返されます LayerVersion。

Ref 関数の使用方法の詳細については、「AWS CloudFormation ユーザーガイド」の「Ref」を参照してください。

例

LayerVersionExample

の例 LayerVersion

YAML

```
Properties:
  LayerName: MyLayer
  Description: Layer description
  ContentUri: 's3://amzn-s3-demo-bucket/my-layer.zip'
  CompatibleRuntimes:
    - nodejs10.x
    - nodejs12.x
  LicenseInfo: 'Available under the MIT-0 license.'
  RetentionPolicy: Retain
```

LayerContent

Lambda レイヤー の内容を含むZIPアーカイブ。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

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

プロパティ

`Bucket`

レイヤーアーカイブの Amazon S3 バケットです。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::Lambda::LayerVersionContent`データ型の `S3Bucket` プロパティに直接渡されます。

`Key`

レイヤーアーカイブの Amazon S3 キーです。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、`AWS::Lambda::LayerVersionContent`データ型の `S3Key` プロパティに直接渡されます。

`Version`

バージョン管理されたオブジェクトの場合に使用する、レイヤーアーカイブオブジェクトのバージョンです。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Lambda::LayerVersionContent`データ型の `S3ObjectVersion` プロパティに直接渡されます。

例

LayerContent

LayerContent の例

YAML

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

AWS::Serverless::SimpleTable

単一属性のプライマリキーで DynamoDB テーブルを作成します。これは、データへのアクセスがプライマリキー経由でのアクセスに限定されている場合に役立ちます。

より高度な機能については、 AWS::DynamoDB::Table のリソース AWS CloudFormation。これらのリソースは、 で使用できます AWS SAM。これらは包括的であり、次のようなさらなるカスタマイズを提供します。 key schema および resource policy カスタマイズ。

> **Note**
>
> にデプロイすると AWS CloudFormation、 は AWS SAM リソースを AWS CloudFormation リソース AWS SAM に変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model （ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
Type: AWS::Serverless::SimpleTable
Properties:
  PointInTimeRecoverySpecification: PointInTimeRecoverySpecification
  PrimaryKey: PrimaryKeyObject
  ProvisionedThroughput: ProvisionedThroughput
  SSESpecification: SSESpecification
  TableName: String
  Tags: Map
```

プロパティ

`PointInTimeRecoverySpecification`

ポイントインタイムリカバリを有効にするための設定。

タイプ : PointInTimeRecoverySpecification

必須: いいえ

AWS CloudFormation 互換性 : このプロパティは、`AWS::DynamoDB::Table`リソースの`PointInTimeRecoverySpecification`プロパティに直接渡されます。

`PrimaryKey`

テーブルのプライマリキーとして使用される属性の名前とタイプです。指定しない場合、プライマリキーは値が `id` の `String` になります。

> **Note**
> このプロパティの値は、このリソースが作成された後で変更することはできません。

タイプ : PrimaryKeyObject

必須: いいえ

AWS CloudFormation 互換性 : このプロパティは に一意 AWS SAM であり、 AWS CloudFormation 同等の はありません。

`ProvisionedThroughput`

読み取りおよび書き込みスループットのプロビジョニング情報です。

ProvisionedThroughput が指定されていない場合、BillingMode は PAY_PER_REQUEST として指定されます。

タイプ : ProvisionedThroughput

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::DynamoDB::Tableリソースの ProvisionedThroughputプロパティに直接渡されます。

SSESpecification

サーバー側の暗号化を有効にする設定を指定します。

タイプ : SSESpecification

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::DynamoDB::Tableリソースの SSESpecificationプロパティに直接渡されます。

TableName

DynamoDB テーブルの名前です。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::DynamoDB::Tableリソースの TableNameプロパティに直接渡されます。

Tags

この に追加するタグを指定するマップ (文字列から文字列) SimpleTable。タグの有効なキーと値の詳細については、AWS CloudFormation ユーザーガイドのリソースタグを参照してください。

タイプ: マップ

必須: いいえ

AWS CloudFormation 互換性 : このプロパティは、AWS::DynamoDB::Tableリソースの Tagsプロパティに似ています。の Tags プロパティSAMは Key:Value ペアで構成され CloudFormation 、その中でタグオブジェクトのリストで構成されます。

戻り値

参照番号

このリソースの論理 ID が Ref 組み込み関数に提供されると、基盤となる DynamoDB テーブルのリソース名 が返されます。

Ref 関数の使用方法の詳細については、AWS CloudFormation ユーザーガイドの「Ref」を参照してください。

例

SimpleTableExample

の例 SimpleTable

YAML

```
Properties:
  TableName: my-table
  Tags:
    Department: Engineering
    AppType: Serverless
```

PrimaryKeyObject

プライマリキーのプロパティを説明するオブジェクトです。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Name: String
  Type: String
```

プロパティ

Name

プライマリキーの属性名です。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::DynamoDB::Table AttributeDefinition` データ型の `AttributeName` プロパティに直接渡されます。

その他の注意点: このプロパティは、`AWS::DynamoDB::Table KeySchema` データ型の AttributeName プロパティにも渡されます。

`Type`

プライマリキーのデータ型です。

有効な値: `String`、`Number`、`Binary`

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::DynamoDB::Table AttributeDefinition` データ型の `AttributeType` プロパティに直接渡されます。

例

PrimaryKey

プライマリキーの例です。

YAML

```
Properties:
  PrimaryKey:
    Name: MyPrimaryKey
    Type: String
```

AWS::Serverless::StateMachine

AWS Step Functions ステートマシンを作成します。これは、AWS Lambda 関数と AWS リソースのオーケストレーションを行って、複雑で堅牢なワークフローを形成するために使用できます。

Step Functions の詳細については、AWS Step Functions デベロッパーガイドを参照してください。

> **Note**
>
> AWS CloudFormation にデプロイすると、AWS SAM は、AWS SAM リソースを AWS CloudFormation リソースに変換します。詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Type: AWS::Serverless::StateMachine
Properties:
  AutoPublishAlias: String
  UseAliasAsEventTarget: Boolean
  Definition: Map
  DefinitionSubstitutions: Map
  DefinitionUri: String | S3Location
  DeploymentPreference: DeploymentPreference
  Events: EventSource
  Logging: LoggingConfiguration
  Name: String
  PermissionsBoundary: String
  Policies: String | List | Map
  PropagateTags: Boolean
  RolePath: String
  Role: String
  Tags: Map
  Tracing: TracingConfiguration
  Type: String
```

プロパティ

`AutoPublishAlias`

ステートマシンエイリアスの名前です。Step Functions ステートマシンエイリアスの使用に関する詳細については、「AWS Step Functions デベロッパーガイド」の「Versions and Aliases を使用して継続的なデプロイを管理する」を参照してください。

DeploymentPreference を使用して、エイリアスのデプロイ設定を実行します。DeploymentPreference を指定しない場合、AWS SAM がトラフィックを設定して、新しいステートマシンバージョンに一度に移行するようにします。

デフォルトで、AWS SAM はバージョンの DeletionPolicy および UpdateReplacePolicy を Retain に設定します。以前のバージョンは自動的に削除されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::StepFunctions::StateMachineAlias リソースの Name プロパティに直接渡されます。

UseAliasAsEventTarget

AutoPublishAlias プロパティを使用して作成されたエイリアスを、Eventsとして定義されたイベントソースのターゲットに渡すかどうかを指定します。

エイリアスをイベントのターゲットとして使用するには、True を指定します。

タイプ: ブール

必須: いいえ

デフォルト: False

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

Definition

ステートマシン定義はオブジェクトで、オブジェクトの形式は、AWS SAM テンプレートファイルの形式 (JSON や YAML など) と一致します。ステートマシンの定義は、Amazon States Language に準拠しています。

インラインステートマシンの定義例については、「例」を参照してください。

Definition または DefinitionUri を提供する必要があります。

タイプ: マップ

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`DefinitionSubstitutions`

ステートマシン定義内のプレースホルダー変数のマッピングを指定する文字列対文字列のマップです。これは、実行時に取得した値 (組み込み関数からの値など) をステートマシン定義に挿入できるようにします。

タイプ: マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::StepFunctions::StateMachine` リソースの `DefinitionSubstitutions` プロパティに似ています。インラインステートマシン定義で組み込み関数が指定されている場合は、AWS SAM がこのプロパティにエントリを追加して、ステートマシン定義にそれらを挿入します。

`DefinitionUri`

Amazon States Language で記述されたステートマシン定義の Amazon Simple Storage Service (Amazon S3) URI またはローカルファイルパスです。

ローカルファイルパスを指定する場合は、定義が適切に変換されるようにするために、テンプレートが `sam deploy` または `sam package` コマンドを含むワークフローを実行する必要があります。これを行うには、バージョン 0.52.0 以降の AWS SAM CLI を使用する必要があります。

`Definition` または `DefinitionUri` を提供する必要があります。

タイプ: 文字列 | S3Location

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは、`AWS::StepFunctions::StateMachine` リソースの `DefinitionS3Location` プロパティに直接渡されます。

`DeploymentPreference`

段階的なステートマシンデプロイを有効化して設定する設定です。Step Functions の段階的なデプロイに関する詳細については、「AWS Step Functions デベロッパーガイド」の「Versions and Aliases を使用して継続的なデプロイを管理する」を参照してください。

このプロパティを設定する前に、AutoPublishAlias を指定してください。DeploymentPreference 設定は、AutoPublishAlias で指定されたエイリアスに適用されます。

DeploymentPreference を指定すると、AWS SAM が StateMachineVersionArn サブプロパティ値を自動的に生成します。

タイプ: DeploymentPreference

必須: いいえ

AWS CloudFormation との互換性: AWS SAM がStateMachineVersionArn プロパティ値を生成して DeploymentPreference にアタッチし、AWS::StepFunctions::StateMachineAlias リソースの DeploymentPreference プロパティに DeploymentPreference を渡します。

Events

このステートマシンをトリガーするイベントを指定します。イベントは、1 つのタイプと、そのタイプに依存する一連のプロパティで構成されます。

タイプ: EventSource

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

Logging

どの実行履歴イベントがどこにログされるかを定義します。

タイプ: LoggingConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::StepFunctions::StateMachine リソースの LoggingConfiguration プロパティに直接渡されます。

Name

ステートマシンの名前です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::StepFunctions::StateMachine` リソースの `StateMachineName` プロパティに直接渡されます。

`PermissionsBoundary`

このステートマシンの実行ロールに使用するアクセス許可境界の ARN です。このプロパティは、ユーザーのためにロールが生成される場合にのみ機能します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::IAM::Role` リソースの `PermissionsBoundary` プロパティに直接渡されます。

`Policies`

このステートマシンの許可ポリシー。ポリシーは、ステートマシンのデフォルト AWS Identity and Access Management (IAM) 実行ロールに付加されます。

このプロパティは、単一の値または値のリストを受け入れます。使用できる値は次のとおりです。

- AWS SAMポリシーテンプレート.
- AWS 管理ポリシーまたはカスタマー管理ポリシーの ARN。
- 次のリストにある AWS マネージドポリシーの名前。
- マップとして YAML でフォーマットされたインライン IAM ポリシー。

> **Note**
>
> `Role` プロパティを設定する場合、このプロパティは無視されます。

タイプ: 文字列 | リスト | マップ

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`PropagateTags`

AWS::Serverless::StateMachine が生成したリソースに `Tags` プロパティからのタグを渡すかどうかを指定します。`True` を指定して、生成されたリソースにタグを伝播します。

タイプ: ブール

必須: いいえ

デフォルト: `False`

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Role`

このステートマシンの実行ロールとして使用する IAM ロールの ARN です。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは、`AWS::StepFunctions::StateMachine` リソースの `RoleArn` プロパティに直接渡されます。

`RolePath`

ステートマシンの IAM 実行ロールへのパス。

このプロパティは、ユーザーのためにロールが生成される場合にのみ機能します。`Role` プロパティでロールが指定されている場合は使用しないでください。

型: 文字列

必須: 条件に応じて異なります

AWS CloudFormation との互換性: このプロパティは、`AWS::IAM::Role` リソースの `Path` プロパティに直接渡されます。

`Tags`

ステートマシンと対応する実行ロールに追加されるタグを指定する文字列対文字列マップです。タグの有効なキーと値についての情報は、AWS::StepFunctions::StateMachine リソースの Tags プロパティを参照してください。

タイプ: マップ

必須: いいえ

AWS CloudFormation の互換性: このプロパティは、`AWS::StepFunctions::StateMachine` リソースの `Tags` プロパティに似ています。AWS SAM は自動的にこのリソースに `stateMachine:createdBy:SAM` タグを付けて、そのリソースに対して生成されるデフォルトのロールにタグを付けます。

`Tracing`

ステートマシンに対して AWS X-Ray が有効化されるかどうかを選択します。X-Ray の Step Functions との使用に関する詳細については、AWS Step Functions デベロッパーガイドの「AWS X-Ray and Step Functions」を参照してください。

タイプ: TracingConfiguration

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::StepFunctions::StateMachine` リソースの `TracingConfiguration` プロパティに直接渡されます。

`Type`

ステートマシンのタイプです。

有効な値: `STANDARD` または `EXPRESS`

タイプ : 文字列

必須: いいえ

デフォルト: `STANDARD`

AWS CloudFormation との互換性: このプロパティは、`AWS::StepFunctions::StateMachine` リソースの `StateMachineType` プロパティに直接渡されます。

戻り値

参照番号

このリソースの論理 ID を Ref 組み込み関数に提供すると、Ref は基盤となる AWS::StepFunctions::StateMachine リソースの Amazon リソースネーム (ARN) を返します。

Ref 関数の使用方法の詳細については、AWS CloudFormation ユーザーガイドの「Ref」を参照してください。

Fn::GetAtt

Fn::GetAtt は、このタイプの指定された属性の値を返します。利用可能な属性とサンプル戻り値は以下のとおりです。

Fn::GetAtt の使用の詳細については、AWS CloudFormation ユーザーガイドの「Fn::GetAtt」を参照してください。

Name

ステートマシンの名前 (HelloWorld-StateMachine など) を返します。

例

ステートマシン定義ファイル

以下は、Lambda 関数によるステートマシンの呼び出しを許可するインラインステートマシン定義の例です。この例では、呼び出しを許可する適切なポリシーが Role プロパティで設定されることを想定している点に注意してください。my_state_machine.asl.json ファイルは Amazon States Language で記述される必要があります。

この例では、DefinitionSubstitution エントリによって、ステートマシンに AWS SAM テンプレートファイルで宣言されているリソースを含めることが許可されます。

YAML

```
MySampleStateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
```

```
    DefinitionUri: statemachine/my_state_machine.asl.json
    Role: arn:aws:iam::123456123456:role/service-role/my-sample-role
    Tracing:
      Enabled: true
    DefinitionSubstitutions:
      MyFunctionArn: !GetAtt MyFunction.Arn
      MyDDBTable: !Ref TransactionTable
```

インラインステートマシン定義

以下は、インラインステートマシン定義の例です。

この例では、AWS SAM テンプレートファイルが YAML で記述されるため、ステートマシン定義も YAML になります。インラインステートマシン定義を JSON で宣言するには、AWS SAM テンプレートファイルを JSON で記述してください。

YAML

```
MySampleStateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
    Definition:
      StartAt: MyLambdaState
      States:
        MyLambdaState:
          Type: Task
          Resource: arn:aws:lambda:us-east-1:123456123456:function:my-sample-lambda-app
          End: true
    Role: arn:aws:iam::123456123456:role/service-role/my-sample-role
    Tracing:
      Enabled: true
```

EventSource

ステートマシンをトリガーするイベントのソースを説明するオブジェクトです。各イベントは、1 つのタイプと、そのタイプに依存する一連のプロパティで構成されます。各イベントソースのプロパティの詳細については、そのタイプに対応するサブトピックを参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Properties: Schedule | ScheduleV2 | CloudWatchEvent | EventBridgeRule | Api
Type: String
```

プロパティ

`Properties`

このイベントマッピングのプロパティを説明するオブジェクトです。プロパティのセットは、定義された `Type` に準拠する必要があります。

タイプ: Schedule | ScheduleV2 | CloudWatchEvent | EventBridgeRule | Api

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Type`

イベントタイプです。

有効な値: `Api`、`Schedule`、`ScheduleV2`、`CloudWatchEvent`、`EventBridgeRule`

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

API

以下は、`API` タイプのイベントの例です。

YAML

```
ApiEvent:
  Type: Api
```

```
  Properties:
    Method: get
    Path: /group/{user}
    RestApiId:
      Ref: MyApi
```

Api

Api イベントソースタイプを説明するオブジェクトです。AWS::Serverless::Api リソースが定義されている場合、パスとメソッドの値は、API の OpenApi 定義にあるオペレーションに対応している必要があります。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Auth: ApiStateMachineAuth
Method: String
Path: String
RestApiId: String
UnescapeMappingTemplate: Boolean
```

プロパティ

`Auth`

この API、パス、およびメソッドの認可設定です。

このプロパティを使用して、`DefaultAuthorizer` が指定されていない場合、または デフォルトの `ApiKeyRequired` 設定を上書きするために、個々のパスに対する API の `DefaultAuthorizer` 設定を上書きします。

タイプ: ApiStateMachineAuth

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Method`

この関数が呼び出される HTTP メソッドです。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Path`

この関数が呼び出される URI パスです。値は / で始める必要があります。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`RestApiId`

`RestApi` リソースの識別子で、所定のパスとメソッドでのオペレーションが含まれている必要があります。これは通常、このテンプレートで定義された AWS::Serverless::Api リソースを参照するように設定されます。

このプロパティを定義しない場合は、AWS SAM が生成された `OpenApi` ドキュメントを使用してデフォルトの AWS::Serverless::Api リソースを作成します。そのリソースには、`RestApiId` を指定しない同じテンプレート内の `Api` イベントによって定義されるすべてのパスとメソッドの和集合が含まれます。

このプロパティは、別のテンプレートで定義された AWS::Serverless::Api リソースを参照できません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`UnescapeMappingTemplate`

`\'` を `'` に置き換えて、ステートマシンに渡される入力の一重引用符のエスケープを解除します。入力に一重引用符が含まれている場合に使用します。

> **Note**
>
> `False` に設定し、入力に一重引用符が含まれている場合、エラーが発生します。

タイプ: ブール

必須: いいえ

デフォルト: False

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

ApiEvent

以下は、`Api` タイプのイベントの例です。

YAML

```
Events:
  ApiEvent:
    Type: Api
    Properties:
      Path: /path
      Method: get
```

ApiStateMachineAuth

特定の API、パス、およびメソッドに対して、イベントレベルで認可を設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  ApiKeyRequired: Boolean
  AuthorizationScopes: List
  Authorizer: String
  ResourcePolicy: ResourcePolicyStatement
```

プロパティ

`ApiKeyRequired`

この API、パス、およびメソッドの API キーが必要です。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AuthorizationScopes`

この API、パス、およびメソッドに適用する認可スコープです。

指定するスコープは、`DefaultAuthorizer` プロパティが適用するスコープ (指定されている場合) を上書きします。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Authorizer`

特定のステートマシンの `Authorizer` です。

API のグローバルオーソライザーが指定されているときにこのステートマシンを公開したい場合は、`Authorizer` を `NONE` に設定してグローバルオーソライザーを上書きします。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`ResourcePolicy`

API とパスのリソースポリシーを設定します。

タイプ: ResourcePolicyStatement

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

Statemachine-Auth

以下の例は、ステートマシンレベルで認可を指定します。

YAML

```
Auth:
  ApiKeyRequired: true
  Authorizer: NONE
```

ResourcePolicyStatement

API 上のすべてのメソッドとパスのリソースポリシーを設定します。リソースポリシーの詳細については、API Gateway デベロッパーガイドの「API Gateway リソースポリシーを使用して API へのアクセスを制御する」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  AwsAccountBlacklist: List
  AwsAccountWhitelist: List
  CustomStatements: List
  IntrinsicVpcBlacklist: List
```

```
  IntrinsicVpcWhitelist: List
  IntrinsicVpceBlacklist: List
  IntrinsicVpceWhitelist: List
  IpRangeBlacklist: List
  IpRangeWhitelist: List
  SourceVpcBlacklist: List
  SourceVpcWhitelist: List
```

プロパティ

`AwsAccountBlacklist`

ブロックする AWS アカウントです。

型: 文字列のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`AwsAccountWhitelist`

許可する AWS アカウントです。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

型: 文字列のリスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`CustomStatements`

この API に適用するカスタムリソースポリシーステートメントのリストです。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpcBlacklist`

ブロックする仮想プライベートクラウド (VPC) のリストで、各 VPC が動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpcWhitelist`

許可する VPC のリストで、各 VPC が動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpceBlacklist`

ブロックする VPC エンドポイントのリストで、各 VPC エンドポイントが動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IntrinsicVpceWhitelist`

許可する VPC エンドポイントのリストで、各 VPC エンドポイントが動的参照または `Ref` 組み込み関数などのリファレンスとして指定されます。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IpRangeBlacklist`

ブロックする IP アドレスまたはアドレス範囲です。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`IpRangeWhitelist`

許可する IP アドレスまたはアドレス範囲です。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceVpcBlacklist`

ブロックするソース VPC またはソース VPC エンドポイントです。ソース VPC 名は `"vpc-"` で始まり、ソース VPC エンドポイント名は `"vpce-"` で始まる必要があります。このプロパティの使用例については、このページの下部にある「例」セクションを参照してください。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`SourceVpcWhitelist`

許可するソース VPC またはソース VPC エンドポイントです。ソース VPC 名は `"vpc-"` で始まり、ソース VPC エンドポイント名は `"vpce-"` で始まる必要があります。

タイプ: リスト

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

リソースポリシーの例

以下の例は、2 つの IP アドレスと 1 つのソース VPC をブロックし、AWS アカウントを許可します。

YAML

```
Auth:
  ResourcePolicy:
    CustomStatements: [{
                         "Effect": "Allow",
                         "Principal": "*",
                         "Action": "execute-api:Invoke",
                         "Resource": "execute-api:/Prod/GET/pets",
                         "Condition": {
                           "IpAddress": {
                             "aws:SourceIp": "1.2.3.4"
                           }
                         }
                       }]
    IpRangeBlacklist:
      - "10.20.30.40"
      - "1.2.3.4"
    SourceVpcBlacklist:
      - "vpce-1a2b3c4d"
    AwsAccountWhitelist:
      - "111122223333"
    IntrinsicVpcBlacklist:
      - "{{resolve:ssm:SomeVPCReference:1}}"
      - !Ref MyVPC
    IntrinsicVpceWhitelist:
      - "{{resolve:ssm:SomeVPCEReference:1}}"
      - !Ref MyVPCE
```

CloudWatchEvent

`CloudWatchEvent` イベントソースタイプを説明するオブジェクトです。

このイベントタイプが設定されていると、AWS Serverless Application Model (AWS SAM) は AWS::Events::Rule リソースを生成します。

重要な注意点: EventBridgeRule は、`CloudWatchEvent` の代わりに使用することが推奨されるイベントソースタイプです。`EventBridgeRule` と `CloudWatchEvent` は同じ基盤となるサービス、API、および AWS CloudFormation リソースを使用しますが、AWS SAM は `EventBridgeRule` のみに新しい機能のサポートを追加します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
EventBusName: String
Input: String
InputPath: String
Pattern: EventPattern
```

プロパティ

`EventBusName`

このルールに関連付けるイベントバスです。このプロパティを省略する場合、AWS SAM はデフォルトのイベントバスを使用します。

型: 文字列

必須: いいえ

デフォルト値: デフォルトのイベントバス

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの EventBusName プロパティに直接渡されます。

`Input`

ターゲットに渡された有効な JSON テキストです。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target リソースの Input プロパティに直接渡されます。

InputPath

一致するイベント全体をターゲットに渡したくない場合は、InputPath プロパティを使用して、イベントのどの部分を渡すかを説明します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target リソースの InputPath プロパティに直接渡されます。

Pattern

どのイベントが指定されたターゲットにルーティングされるかを説明します。詳細については、Amazon EventBridge ユーザーガイドの「Events andEvent Patterns in EventBridge」を参照してください。

タイプ: EventPattern

必須: はい

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの EventPattern プロパティに直接渡されます。

例

CloudWatchEvent

以下は、CloudWatchEvent イベントソースタイプの例です。

YAML

```
CWEvent:
  Type: CloudWatchEvent
  Properties:
```

```
    Input: '{"Key": "Value"}'
    Pattern:
      detail:
        state:
          - running
```

EventBridgeRule

EventBridgeRule イベントソースタイプを説明するオブジェクトです。これは、ステートマシンを Amazon EventBridge ルールのターゲットとして設定します。詳細については、Amazon EventBridge ユーザーガイドの「Amazon EventBridge とは」を参照してください。

このイベントタイプが設定されていると、AWS SAM は AWS::Events::Rule リソースを生成します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DeadLetterConfig: DeadLetterConfig
EventBusName: String
Input: String
InputPath: String
InputTransformer: InputTransformer
Pattern: EventPattern
RetryPolicy: RetryPolicy
RuleName: String
State: String
Target: Target
```

プロパティ

DeadLetterConfig

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを設定します。呼び出しは、存在しない Lambda 関数にイベントを送信した場合、または Lambda 関数を呼び出すために十分な許可が EventBridge にない場合などに失敗します。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

タイプ: DeadLetterConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target データ型の DeadLetterConfig プロパティに似ています。このプロパティの AWS SAM バージョンには、AWS SAM にデッドレターキューを作成させる場合のために、追加のサブプロパティが含まれています。

EventBusName

このルールに関連付けるイベントバスです。このプロパティを省略する場合、AWS SAM はデフォルトのイベントバスを使用します。

型: 文字列

必須: いいえ

デフォルト値: デフォルトのイベントバス

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの EventBusName プロパティに直接渡されます。

Input

ターゲットに渡された有効な JSON テキストです。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target リソースの Input プロパティに直接渡されます。

InputPath

一致するイベント全体をターゲットに渡したくない場合は、InputPath プロパティを使用して、イベントのどの部分を渡すかを説明します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target リソースの InputPath プロパティに直接渡されます。

`InputTransformer`

特定のイベントデータに基づいてターゲットにカスタム入力を提供できるための設定。イベントから 1 つ以上のキーと値のペアを抽出し、そのデータを使用して、カスタマイズされた入力をターゲットに送信できます。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge の入力変換」を参照してください。

Type: InputTransformer

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` データ型の `InputTransformer` プロパティに直接渡されます。

`Pattern`

どのイベントが指定されたターゲットにルーティングされるかを説明します。詳細については、Amazon EventBridge ユーザーガイドの「Events andEvent Patterns in EventBridge」を参照してください。

タイプ: EventPattern

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `EventPattern` プロパティに直接渡されます。

`RetryPolicy`

再試行ポリシーの設定に関する情報が含まれた `RetryPolicy` オブジェクトです。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

タイプ: RetryPolicy

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` データ型の `RetryPolicy` プロパティに直接渡されます。

`RuleName`

ルールの名前。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの Name プロパティに直接渡されます。

State

ルールの状態。

有効な値: [DISABLED | ENABLED]

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの State プロパティに直接渡されます。

Target

ルールがトリガーされるときに EventBridge が呼び出す AWS リソースです。このプロパティを使用して、ターゲットの論理 ID を指定できます。このプロパティが指定されていない場合は、AWS SAM がターゲットの論理 ID を生成します。

タイプ: Target

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの Targets プロパティに似ています。このプロパティの AWS SAM バージョンでは、単一のターゲットの論理 ID しか指定できません。

例

EventBridgeRule

以下は、EventBridgeRule イベントソースタイプの例です。

YAML

```
EBRule:
  Type: EventBridgeRule
  Properties:
    Input: '{"Key": "Value"}'
    Pattern:
```

```
      detail:
        state:
          - terminated
```

DeadLetterConfig

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを指定するために使用されるオブジェクトです。呼び出しコールは、存在しないステートマシンにイベントを送信した場合、またはステートマシンを呼び出すために十分な許可がない場合などに失敗します。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
Arn: String
QueueLogicalId: String
Type: String
```

プロパティ

`Arn`

デッドレターキューのターゲットとして指定された Amazon SQS キューの Amazon リソースネーム (ARN) です。

> **Note**
>
> `Type` プロパティと `Arn` プロパティは、両方ではなく、どちらか一方を指定してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule DeadLetterConfig` データ型の `Arn` プロパティに直接渡されます。

`QueueLogicalId`

Type が指定されている場合に AWS SAM が作成するデッドレターキューのカスタム名です。

> **Note**
>
> Type プロパティが設定されていない場合、このプロパティは無視されます。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Type`

キューのタイプです。このプロパティが設定されていると、AWS SAM がデッドレターキューを自動的に作成し、そのキューにイベントを送信する許可をルールリソースに付与するために必要なリソースベースのポリシーをアタッチします。

> **Note**
>
> Type プロパティと `Arn` プロパティは、両方ではなく、どちらか一方を指定してください。

有効な値: `SQS`

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

ルールがトリガーされるときに EventBridge が呼び出す AWS リソースを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Id: String
```

プロパティ

`Id`

ターゲットの論理 ID です。

`Id` の値には、英数字、ピリオド (`.`)、ハイフン (`-`)、およびアンダースコア (`_`) を含めることができます。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` データ型の `Id` プロパティに直接渡されます。

例

Target

YAML

```
EBRule:
```

```
  Type: EventBridgeRule
  Properties:
    Target:
      Id: MyTarget
```

Schedule

Schedule イベントソースタイプを説明するオブジェクトです。これは、ステートマシンをスケジュールに従ってトリガーする EventBridge ルールのターゲットとして設定します。詳細については、Amazon EventBridge ユーザーガイドの「Amazon EventBridge とは」を参照してください。

このイベントタイプが設定されていると、AWS Serverless Application Model (AWS SAM) は AWS::Events::Rule リソースを生成します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
DeadLetterConfig: DeadLetterConfig
Description: String
Enabled: Boolean
Input: String
Name: String
RetryPolicy: RetryPolicy
RoleArn: String
Schedule: String
State: String
Target: Target
```

プロパティ

DeadLetterConfig

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを設定します。呼び出しは、存在しない Lambda 関数にイベントを送信した場合、または Lambda 関数を呼び出すために十分な許可が EventBridge にない場合などに失敗します。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

タイプ: DeadLetterConfig

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target データ型の DeadLetterConfig プロパティに似ています。このプロパティの AWS SAM バージョンには、AWS SAM にデッドレターキューを作成させる場合のために、追加のサブプロパティが含まれています。

`Description`

ルールの説明です。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの Description プロパティに直接渡されます。

`Enabled`

ルールが有効かどうかを示します。

ルールを無効にするには、このプロパティを false に設定します。

> **Note**
>
> Enabled プロパティと State プロパティは、両方ではなく、どちらか一方を指定してください。

タイプ: ブール

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの State プロパティに似ています。このプロパティが true に設定されている場合は、AWS SAM が ENABLED を渡します。それ以外の場合は DISABLED を渡します。

`Input`

ターゲットに渡された有効な JSON テキストです。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target リソースの Input プロパティに直接渡されます。

Name

ルールの名前です。名前を指定しない場合、AWS CloudFormation が一意の物理 ID を生成し、その ID をルール名として使用します。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule リソースの Name プロパティに直接渡されます。

RetryPolicy

再試行ポリシーの設定に関する情報が含まれた RetryPolicy オブジェクトです。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

タイプ: RetryPolicy

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule Target データ型の RetryPolicy プロパティに直接渡されます。

RoleArn

スケジュールが呼び出されたときに EventBridge スケジューラがターゲットとして使用する IAM ロールの ARN。

タイプ: RoleArn

必須:いいえ。指定しない場合、新しいロールが作成されて適用されます。

AWS CloudFormation との互換性: このプロパティは、AWS::Scheduler::Schedule Target データ型の RoleArn プロパティに直接渡されます。

`Schedule`

ルールがいつ、どのくらいの頻度で実行されるかを決定するスケジューリング式です。詳細については、「Schedule Expressions for Rules」を参照してください。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `ScheduleExpression` プロパティに直接渡されます。

`State`

ルールの状態。

使用できる値: `DISABLED | ENABLED`

> **Note**
>
> `Enabled` プロパティと `State` プロパティは、両方ではなく、どちらか一方を指定してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `State` プロパティに直接渡されます。

`Target`

ルールがトリガーされるときに EventBridge が呼び出す AWS リソースです。このプロパティを使用して、ターゲットの論理 ID を指定できます。このプロパティが指定されていない場合は、AWS SAM がターゲットの論理 ID を生成します。

タイプ: Target

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule` リソースの `Targets` プロパティに似ています。このプロパティの AWS SAM バージョンでは、単一のターゲットの論理 ID しか指定できません。

例

CloudWatch スケジュールイベント

CloudWatch スケジュールイベントの例

YAML

```
CWSchedule:
  Type: Schedule
  Properties:
    Schedule: 'rate(1 minute)'
    Name: TestSchedule
    Description: test schedule
    Enabled: false
```

DeadLetterConfig

ターゲットの呼び出しに失敗した後で EventBridge がイベントを送信する Amazon Simple Queue Service (Amazon SQS) キューを指定するために使用されるオブジェクトです。呼び出しコールは、存在しないステートマシンにイベントを送信した場合、またはステートマシンを呼び出すために十分な許可がない場合などに失敗します。詳細については、Amazon EventBridge ユーザーガイドの「Event retry policy and using dead-letter queues」を参照してください。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Arn: String
  QueueLogicalId: String
  Type: String
```

プロパティ

`Arn`

デッドレターキューのターゲットとして指定された Amazon SQS キューの Amazon リソースネーム (ARN) です。

> **Note**
>
> Type プロパティと Arn プロパティは、両方ではなく、どちらか一方を指定してください。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは、AWS::Events::Rule DeadLetterConfig データ型の Arn プロパティに直接渡されます。

`QueueLogicalId`

Type が指定されている場合に AWS SAM が作成するデッドレターキューのカスタム名です。

> **Note**
>
> Type プロパティが設定されていない場合、このプロパティは無視されます。

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

`Type`

キューのタイプです。このプロパティが設定されていると、AWS SAM がデッドレターキューを自動的に作成し、そのキューにイベントを送信する許可をルールリソースに付与するために必要なリソースベースのポリシーをアタッチします。

> **Note**
>
> Type プロパティと Arn プロパティは、両方ではなく、どちらか一方を指定してください。

有効な値: SQS

型: 文字列

必須: いいえ

AWS CloudFormation との互換性: このプロパティは AWS SAM に固有であり、AWS CloudFormation に同等のものはありません。

例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

ルールがトリガーされるときに EventBridge が呼び出す AWS リソースを設定します。

構文

AWS Serverless Application Model (AWS SAM) テンプレートでこのエンティティを宣言するには、以下の構文を使用します。

YAML

```
  Id: String
```

プロパティ

`Id`

ターゲットの論理 ID です。

`Id` の値には、英数字、ピリオド (.)、ハイフン (-)、およびアンダースコア (_) を含めることができます。

型: 文字列

必須: はい

AWS CloudFormation との互換性: このプロパティは、`AWS::Events::Rule Target` データ型の `Id` プロパティに直接渡されます。

例

Target

YAML

```
EBRule:
  Type: Schedule
  Properties:
    Target:
      Id: MyTarget
```

ScheduleV2

`ScheduleV2` イベントソースタイプを記述するオブジェクト。ステートマシンをスケジュールでトリガーする Amazon EventBridge スケジューライベントのターゲットとして設定します。詳細については、スEventBridge ケジューラユーザーガイドの「Amazon スケジューラとは？」を参照してください。 EventBridge

AWS Serverless Application Model（ AWS SAM) は を生成します。 AWS::Scheduler::Schedule このイベントタイプが設定されている場合のリソース。

構文

AWS Serverless Application Model（ AWS SAM) テンプレートでこのエンティティを宣言するには、次の構文を使用します。

YAML

```
DeadLetterConfig: DeadLetterConfig
Description: String
EndDate: String
FlexibleTimeWindow: FlexibleTimeWindow
```

```
GroupName: String
Input: String
KmsKeyArn: String
Name: String
OmitName: Boolean
PermissionsBoundary: String
RetryPolicy: RetryPolicy
RoleArn: String
ScheduleExpression: String
ScheduleExpressionTimezone: String
StartDate: String
State: String
```

プロパティ

`DeadLetterConfig`

が失敗したターゲット呼び出しの後にイベント EventBridge を送信する Amazon Simple Queue Service (Amazon SQS) キューを設定します。呼び出しは、存在しない Lambda 関数にイベントを送信する場合や、Lambda 関数を呼び出すためのアクセス許可 EventBridge が不十分な場合などに失敗する可能性があります。詳細については、ス EventBridge ケジューラユーザーガイドの「スケジューラのデッドレターキューの設定」を参照してください。 EventBridge

タイプ : DeadLetterConfig

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::ScheduleTarget`データ型の `DeadLetterConfig` プロパティに似ています。このプロパティ AWS SAM のバージョンには、デッドレターキュー AWS SAM を作成する場合に備えて、追加のサブプロパティが含まれています。

`Description`

スケジュールの説明。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::Schedule`リソースの `Description`プロパティに直接渡されます。

`EndDate`

スケジュールがターゲットを呼び出すことができる UTCの日付。スケジュールの繰り返し式によっては、指定した EndDate またはそれより前に呼び出しが停止する場合があります。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::Schedule`リソースの`EndDate`プロパティに直接渡されます。

`FlexibleTimeWindow`

スケジュールを呼び出すことができるウィンドウを設定できます。

タイプ : FlexibleTimeWindow

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::Schedule`リソースの`FlexibleTimeWindow`プロパティに直接渡されます。

`GroupName`

このスケジュールに関連付けるスケジュールグループの名前。定義されていない場合、デフォルトグループが使用されます。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::Schedule`リソースの`GroupName`プロパティに直接渡されます。

`Input`

ターゲットに渡された有効なJSONテキスト。このプロパティを使用する場合、イベントテキスト自体からはターゲットに何も渡されません。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::Schedule Target`リソースの `Input`プロパティに直接渡されます。

`KmsKeyArn`

顧客データの暗号化に使用されるKMSキーARNの 。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::Schedule`リソースの `KmsKeyArn`プロパティに直接渡されます。

`Name`

スケジュールの名前。名前を指定しない場合、 は 形式で名前 AWS SAM を生成*`StateMachine-Logical-IDEvent-Source-Name`*し、スケジュール名にその ID を使用します。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、`AWS::Scheduler::Schedule`リソースの `Name`プロパティに直接渡されます。

`OmitName`

デフォルトでは、 は の形式でスケジュール名 AWS SAM を生成して使用します。*`<State-machine-logical-ID><event-source-name>`*。 このプロパティを に設定`true`して一意の物理 ID AWS CloudFormation を生成し、代わりにスケジュール名にそれを使用します。

タイプ: ブール

必須: いいえ

デフォルト: `false`

AWS CloudFormation 互換性： このプロパティは に一意 AWS SAM であり、同等の AWS CloudFormation はありません。

`PermissionsBoundary`

ロールのアクセス許可境界を設定するために使用されるポリシーARNの 。

> **Note**
>
> PermissionsBoundary が定義されている場合、 AWS SAM はスケジューラスケジュールのターゲットIAMロールに同じ境界を適用します。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::IAM::Roleリソースの PermissionsBoundaryプロパティに直接渡されます。

RetryPolicy

再試行ポリシーの設定に関する情報が含まれた RetryPolicy オブジェクトです。

タイプ : RetryPolicy

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Scheduler::ScheduleTargetデータ型の RetryPolicy プロパティに直接渡されます。

RoleArn

スケジュールが呼び出されたときにス EventBridge ケジューラがターゲットに使用するIAMロールARNの 。

タイプ : RoleArn

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Scheduler::ScheduleTargetデータ型の RoleArn プロパティに直接渡されます。

ScheduleExpression

スケジュールがいつ、どのくらいの頻度で実行されるかを決定するスケジューリング式です。

型: 文字列

必須: はい

AWS CloudFormation 互換性： このプロパティは、AWS::Scheduler::Scheduleリソースの ScheduleExpressionプロパティに直接渡されます。

ScheduleExpressionTimezone

スケジュール式が評価されるタイムゾーン。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Scheduler::Scheduleリソースの ScheduleExpressionTimezoneプロパティに直接渡されます。

StartDate

の日付。UTCその後、スケジュールはターゲットの呼び出しを開始できます。スケジュールの繰り返し式によっては、指定した StartDate またはそれより後に呼び出しが発生する場合があります。

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Scheduler::Scheduleリソースの StartDateプロパティに直接渡されます。

State

スケジュールの状態。

使用できる値: DISABLED | ENABLED

型: 文字列

必須: いいえ

AWS CloudFormation 互換性： このプロパティは、AWS::Scheduler::Scheduleリソースの Stateプロパティに直接渡されます。

例

ScheduleV2 リソースを定義する基本的な例

```
StateMachine:
```

```
  Type: AWS::Serverless::StateMachine
  Properties:
    Name: MyStateMachine
    Events:
      ScheduleEvent:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: "rate(1 minute)"
      ComplexScheduleEvent:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: rate(1 minute)
          FlexibleTimeWindow:
            Mode: FLEXIBLE
            MaximumWindowInMinutes: 5
          StartDate: '2022-12-28T12:00:00.000Z'
          EndDate: '2023-01-28T12:00:00.000Z'
          ScheduleExpressionTimezone: UTC
          RetryPolicy:
            MaximumRetryAttempts: 5
            MaximumEventAgeInSeconds: 300
          DeadLetterConfig:
            Type: SQS
    DefinitionUri:
      Bucket: sam-amzn-s3-demo-bucket
      Key: my-state-machine.asl.json
      Version: 3
    Policies:
      - LambdaInvokePolicy:
          FunctionName: !Ref MyFunction
```

AWS SAM 向けに生成された AWS CloudFormation リソース

このセクションでは、AWS SAM が AWS テンプレートを処理するときに作成される AWS CloudFormation リソースの詳細を提供しています。AWS SAM が生成する AWS CloudFormation リソースのセットは、ユーザーが指定するシナリオに応じて異なります。シナリオとは、テンプレートファイルで指定される AWS SAM リソースとプロパティの組み合わせのことです。テンプレートファイル内の他の部分で生成された AWS CloudFormation リソースは、テンプレートファイルで明示的に宣言するリファレンスの参照方法と同じように参照できます。

例えば、AWS SAM テンプレートファイルで `AWS::Serverless::Function` リソースを指定すると、AWS SAM は常に `AWS::Lambda::Function` ベースのリソースを生成します。オプションの

AutoPublishAlias プロパティも指定すると、AWS SAM が追加で AWS::Lambda::Alias および AWS::Lambda::Version リソースを生成します。

このセクションでは、シナリオとそれらが生成する AWS CloudFormation リソースをリストして、AWS SAM テンプレートファイル内で生成された AWS CloudFormation リソースを参照する方法を説明します。

生成された AWS CloudFormation リソースの参照

AWS SAM テンプレートファイル内で生成された AWS CloudFormation リソースを参照するためのオプションには、LogicalId によるものと、参照可能なプロパティによるものの 2 つのオプションがあります。

生成された AWS CloudFormation リソースの LogicalId による参照

AWS SAM が生成する各 AWS CloudFormation リソースには LogicalId があります。これは、テンプレートファイル内で一意の英数字 (A～Z、a～z、0～9) 識別子です。AWS SAM は、テンプレートファイル内の AWS SAM リソースの LogicalIds を使用して、それが生成する AWS CloudFormation リソースの LogicalIds を構築します。明示的に宣言した AWS CloudFormation の場合と同様に、生成された AWS CloudFormation リソースの LogicalId を使用して、テンプレートファイル内のそのリソースのプロパティにアクセスすることができます。AWS CloudFormation および AWS SAM テンプレート内の LogicalIds の詳細については、AWS CloudFormation ユーザーガイドの「Resources」を参照してください。

> **Note**
>
> 生成されたリソースの LogicalIds には、名前空間の衝突を避けるための一意のハッシュ値が含まれるものがあります。これらのリソースの LogicalIds は、スタックの作成時に導出されます。これらを取得できるのは、スタックが AWS Management Console、AWS CLI、または AWS の 1 つを使用して作成された後のみです。ハッシュ値が変更される可能性があるため、LogicalId によるこれらのリソースの参照は推奨されません。

生成された AWS CloudFormation リソースの参照可能なプロパティによる参照

AWS SAM は、生成された一部のリソースに、AWS SAM リソースの参照可能なプロパティを提供します。このプロパティを使用して、AWS SAM テンプレート内の生成された AWS CloudFormation リソースとそのプロパティを参照できます。

> **Note**
>
> 生成された AWS CloudFormation リソースのすべてに参照可能なプロパティがあるわけではありません。これらのリソースには、`LogicalId` を使用する必要があります。

生成された AWS CloudFormation リソースのシナリオ

以下の表は、AWS CloudFormation リソースを生成するシナリオを構成する AWS SAM リソースとプロパティを要約したものです。シナリオ列のトピックには、そのシナリオに対して AWS SAM が生成する追加の AWS CloudFormation リソースの詳細が説明されています。

AWS SAM リソース	ベース AWS CloudFormation リソース	シナリオ
AWS::Serverless::Api	AWS::ApiGateway::RestApi	• DomainName プロパティが指定されている • UsagePlan プロパティが指定されている
AWS::Serverless::Application	AWS::CloudFormation::Stack	• このサーバーレスリソースには、ベース AWS CloudFormation リソースの生成以外の追加のシナリオはありません。
AWS::Serverless::Function	AWS::Lambda::Function	• AutoPublishAlias プロパティが指定されている • Role プロパティが指定されていない • DeploymentPreference プロパティが指定されている • Api イベントソースが指定されている • HttpApi イベントソースが指定されている • ストリーミングイベントソースが指定されている •

AWS SAM リソース	ベース AWS CloudFormation リソース	シナリオ
		イベントブリッジ (またはイベントバス) イベントソースが指定されている • IotRule イベントソースが指定されている • Amazon SNS イベントに対して OnSuccess (または OnFailure) プロパティが指定されている • Amazon SQS イベントに対して OnSuccess (または OnFailure) プロパティが指定されている
AWS::Serverless::HttpApi	AWS::ApiGatewayV2::Api	• StageName プロパティが指定されている • StageName プロパティが指定されていない • DomainName プロパティが指定されている
AWS::Serverless::LayerVersion	AWS::Lambda::LayerVersion	• このサーバーレスリソースには、ベース AWS CloudFormation リソースの生成以外の追加のシナリオはありません。
AWS::Serverless::SimpleTable	AWS::DynamoDB::Table	• このサーバーレスリソースには、ベース AWS CloudFormation リソースの生成以外の追加のシナリオはありません。
AWS::Serverless::StateMachine	AWS::StepFunctions::StateMachine	• Role プロパティが指定されていない • Api イベントソースが指定されている • イベントブリッジ (またはイベントバス) イベントソースが指定されている

トピック

AWS::Serverless::Api が指定された場合、生成される AWS CloudFormation リソース

`AWS::Serverless::Api` が指定されている場合、AWS Serverless Application Model (AWS SAM) は常に `AWS::ApiGateway::RestApi` のベース AWS CloudFormation リソースを生成します。これに加えて、`AWS::ApiGateway::Stage` と `AWS::ApiGateway::Deployment` リソースも常に生成します。

`AWS::ApiGateway::RestApi`

`LogicalId`: *`<api-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::ApiGateway::Stage`

`LogicalId`: *`<api-LogicalId><stage-name>`*`Stage`

`<stage-name>` は、`StageName` プロパティが設定されている文字列です。例えば、`StageName` を `Gamma` に設定とすると、`LogicalId` が *`MyRestApiGamma`*`Stage` になります。

参照可能なプロパティ: *`<api-LogicalId>`*`.Stage`

`AWS::ApiGateway::Deployment`

`LogicalId`: *`<api-LogicalId>`*`Deployment`*`<sha>`*

<sha> は、スタックが作成されるときに生成される一意のハッシュ値です。例えば、*MyRestApi*Deployment*926eeb5ff1* と指定します。

参照可能なプロパティ: *<api-LogicalId>*.Deployment

これらの AWS CloudFormation リソースに加えて、AWS::Serverless::Api が指定されているときは、AWS SAM が以下のシナリオのために追加の AWS CloudFormation リソースを生成します。

シナリオ

- DomainName プロパティが指定されている
- UsagePlan プロパティが指定されている

DomainName プロパティが指定されている

AWS::Serverless::Api の Domain プロパティの DomainName プロパティが指定されている場合、AWS SAM は AWS::ApiGateway::DomainName AWS CloudFormation リソースを生成します。

AWS::ApiGateway::DomainName

LogicalId: ApiGatewayDomainName*<sha>*

<sha> は、スタックが作成されるときに生成される一意のハッシュ値です。例: ApiGatewayDomainName*926eeb5ff1*。

参照可能なプロパティ: *<api-LogicalId>*.DomainName

UsagePlan プロパティが指定されている

AWS::Serverless::Api の UsagePlan プロパティの Auth プロパティが指定されている場合、AWS SAM は、AWS::ApiGateway::UsagePlan、AWS::ApiGateway::UsagePlanKey、および AWS::ApiGateway::ApiKey の AWS CloudFormation リソースを生成します。

AWS::ApiGateway::UsagePlan

LogicalId: *<api-LogicalId>*UsagePlan

参照可能なプロパティ: *<api-LogicalId>*.UsagePlan

AWS::ApiGateway::UsagePlanKey

LogicalId: *<api-LogicalId>*UsagePlanKey

参照可能なプロパティ: *<api-LogicalId>*.UsagePlanKey

AWS::ApiGateway::ApiKey

LogicalId: *<api-LogicalId>*ApiKey

参照可能なプロパティ: *<api-LogicalId>*.ApiKey

AWS::Serverless::Application が指定された場合、生成される AWS CloudFormation リソース

AWS::Serverless::Application が指定されている場合、AWS Serverless Application Model (AWS SAM) は AWS::CloudFormation::Stack のベース AWS CloudFormation リソースを生成します。

AWS::CloudFormation::Stack

LogicalId: *<application-LogicalId>*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::Serverless::Connector を指定したときに生成された AWS CloudFormation リソース

> **Note**
>
> 埋め込み Connectors プロパティを介してコネクタを定義すると、これらのリソースを生成する前に、まず AWS::Serverless::Connector リソースに変換されます。

AWS SAM テンプレートで AWS::Serverless::Connector リソースを指定すると、AWS SAM は必要に応じて次の AWS CloudFormation リソースを生成します。

AWS::IAM::ManagedPolicy

LogicalId:*<connector-LogicalId>*Policy

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)。

`AWS::SNS::TopicPolicy`

`LogicalId`:*`<connector-LogicalId>`*`TopicPolicy`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)。

`AWS::SQS::QueuePolicy`

`LogicalId`:*`<connector-LogicalId>`*`QueuePolicy`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)。

`AWS::Lambda::Permission`

`LogicalId`:*`<connector-LogicalId>`*`<permission>`*`LambdaPermission`

`<permission>` は、`Permissions` プロパティによって指定されたアクセス許可です。例えば、`Write` と指定します。

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)。

AWS::Serverless::Function が指定された場合、生成される AWS CloudFormation リソース

`AWS::Serverless::Function` が指定されている場合、AWS Serverless Application Model (AWS SAM) は常に `AWS::Lambda::Function` のベース AWS CloudFormation リソースを生成します。

`AWS::Lambda::Function`

`LogicalId`: *`<function-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::Serverless::Function` が指定されている場合、AWS SAM はこの AWS CloudFormation リソースに加えて、以下のシナリオに対する AWS CloudFormation リソースも生成します。

シナリオ

- AutoPublishAlias プロパティが指定されている
- Role プロパティが指定されていない
- DeploymentPreference プロパティが指定されている
- Api イベントソースが指定されている
- HttpApi イベントソースが指定されている
- ストリーミングイベントソースが指定されている
- イベントブリッジ (またはイベントバス) イベントソースが指定されている
- IotRule イベントソースが指定されている
- Amazon SNS イベントに対して OnSuccess (または OnFailure) プロパティが指定されている
- Amazon SQS イベントに対して OnSuccess (または OnFailure) プロパティが指定されている

AutoPublishAlias プロパティが指定されている

`AWS::Serverless::Function` の `AutoPublishAlias` プロパティが指定されている場合、AWS SAM は `AWS::Lambda::Alias` と `AWS::Lambda::Version` の AWS CloudFormation リソースを生成します。

`AWS::Lambda::Alias`

`LogicalId`: *`<function-LogicalId>`*`Alias`*`<alias-name>`*

`<alias-name>` は、`AutoPublishAlias` が設定されている文字列です。例えば、`AutoPublishAlias` を `live` に設定すると、`LogicalId` は *`MyFunction`*`Alias`*`live`* になります。

参照可能なプロパティ: *`<function-LogicalId>`*`.Alias`

`AWS::Lambda::Version`

`LogicalId`: *`<function-LogicalId>`*`Version`*`<sha>`*

`<sha>` は、スタックが作成されるときに生成される一意のハッシュ値です。例えば、*`MyFunction`*`Version`*`926eeb5ff1`* などです。

参照可能なプロパティ: *`<function-LogicalId>`*`.Version`

AutoPublishAlias プロパティの詳細については、AWS::Serverless::Function のプロパティセクションを参照してください。

Role プロパティが指定されていない

AWS::Serverless::Function の Role プロパティが指定されていない場合、AWS SAM は AWS::IAM::Role AWS CloudFormation リソースを生成します。

AWS::IAM::Role

LogicalId: *<function-LogicalId>*Role

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

DeploymentPreference プロパティが指定されている

AWS::Serverless::Function の DeploymentPreference プロパティが指定されている場合、AWS SAM は AWS::CodeDeploy::Application および AWS::CodeDeploy::DeploymentGroup の AWS CloudFormation リソースを生成します。さらに、DeploymentPreference オブジェクトの Role プロパティが指定されていない場合、AWS SAM は AWS::IAM::Role AWS CloudFormation リソースも生成します。

AWS::CodeDeploy::Application

LogicalId: ServerlessDeploymentApplication

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::CodeDeploy::DeploymentGroup

LogicalId: *<function-LogicalId>*DeploymentGroup

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::IAM::Role

LogicalId: CodeDeployServiceRole

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

Api イベントソースが指定されている

`Event` の `AWS::Serverless::Function` プロパティが `Api` に設定されているが、`RestApiId` プロパティは指定されていないという場合、AWS SAM は `AWS::ApiGateway::RestApi` AWS CloudFormation リソースを生成します。

`AWS::ApiGateway::RestApi`

`LogicalId`: `ServerlessRestApi`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

HttpApi イベントソースが指定されている

`Event` の `AWS::Serverless::Function` プロパティが `HttpApi` に設定されているが、`ApiId` プロパティは指定されていないという場合、AWS SAM は `AWS::ApiGatewayV2::Api` AWS CloudFormation リソースを生成します。

`AWS::ApiGatewayV2::Api`

`LogicalId`: `ServerlessHttpApi`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

ストリーミングイベントソースが指定されている

`AWS::Serverless::Function` の `Event` プロパティがストリーミングタイプのいずれかに設定されている場合、AWS SAM は `AWS::Lambda::EventSourceMapping` AWS CloudFormation リソースを生成します。これは、`DynamoDB`、`Kinesis`、`MQ`、`MSK`、および `SQS` の各タイプに適用されます。

`AWS::Lambda::EventSourceMapping`

`LogicalId`: *`<function-LogicalId><event-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

イベントブリッジ (またはイベントバス) イベントソースが指定されている

`AWS::Serverless::Function` の `Event` プロパティがイベントブリッジ (またはイベントバス) タイプのいずれかに設定されている場合、AWS SAM は `AWS::Events::Rule` AWS CloudFormation リソースを生成します。これは、`EventBridgeRule`、`Schedule`、および `CloudWatchEvents` の各タイプに適用されます。

`AWS::Events::Rule`

`LogicalId`: *`<function-LogicalId><event-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

IotRule イベントソースが指定されている

`AWS::Serverless::Function` の `Event` プロパティが IoTRule に設定されている場合、AWS SAM は `AWS::IoT::TopicRule` AWS CloudFormation リソースを生成します。

`AWS::IoT::TopicRule`

`LogicalId`: *`<function-LogicalId><event-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

Amazon SNS イベントに対して OnSuccess (または OnFailure) プロパティが指定されている

`AWS::Serverless::Function` の `EventInvokeConfig` プロパティの `DestinationConfig` プロパティの `OnSuccess` (または `OnFailure`) プロパティが指定されており、送信先タイプが `SNS` になっているが、送信先 ARN は指定されていないという場合、AWS SAM は `AWS::Lambda::EventInvokeConfig` および `AWS::SNS::Topic` の AWS CloudFormation リソースを生成します。

`AWS::Lambda::EventInvokeConfig`

`LogicalId`: *`<function-LogicalId>`*`EventInvokeConfig`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::SNS::Topic

LogicalId: *<function-LogicalId>*OnSuccessTopic (または *<function-LogicalId>*OnFailureTopic)

参照可能なプロパティ: *<function-LogicalId>*.DestinationTopic

OnSuccess と OnFailure の両方が Amazon SNS イベントに指定されている場合、生成されたリソースを区別するには LogicalId を使用する必要があります。

Amazon SQS イベントに対して OnSuccess (または OnFailure) プロパティが指定されている

AWS::Serverless::Function の EventInvokeConfig プロパティの DestinationConfig プロパティの OnSuccess (または OnFailure) プロパティが指定されており、送信先タイプが SQS になっているが、送信先 ARN は指定されていないという場合、AWS SAM は AWS::Lambda::EventInvokeConfig および AWS::SQS::Queue の AWS CloudFormation リソースを生成します。

AWS::Lambda::EventInvokeConfig

LogicalId: *<function-LogicalId>*EventInvokeConfig

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::SQS::Queue

LogicalId: *<function-LogicalId>*OnSuccessQueue (または *<function-LogicalId>*OnFailureQueue)

参照可能なプロパティ: *<function-LogicalId>*.DestinationQueue

OnSuccess と OnFailure の両方が Amazon SQS イベントに指定されている場合、生成されたリソースを区別するには LogicalId を使用する必要があります。

AWS::Serverless::GraphQLApi が指定された場合、生成される AWS CloudFormation リソース

AWS Serverless Application Model (AWS SAM) テンプレートで `AWS::Serverless::GraphQLApi` リソースを指定すると、AWS SAM は常に以下のベース AWS CloudFormation リソースを作成します。

`AWS::AppSync::DataSource`

`LogicalId`: *`<graphqlapi-LogicalId><datasource-RelativeId><datasource-Type>`*`DataSource`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::AppSync::FunctionConfiguration`

`LogicalId`: *`<graphqlapi-LogicalId><function-RelativeId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::AppSync::GraphQLApi`

`LogicalId`: *`<graphqlapi-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::AppSync::GraphQLSchema`

`LogicalId`: *`<graphqlapi-LogicalId>`*`Schema`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::AppSync::Resolver`

`LogicalId`: *`<graphqlapi-LogicalId><OperationType><resolver-RelativeId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

これらの AWS CloudFormation リソースに加えて、AWS::Serverless::GraphQLApi が指定されている場合は、AWS SAM が以下の AWS CloudFormation リソースも生成する場合があります。

AWS::AppSync::ApiCache

LogicalId: *<graphqlapi-LogicalId>*ApiCache

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::AppSync::ApiKey

LogicalId: *<graphqlapi-LogicalId><apikey-RelativeId>*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::AppSync::DomainName

LogicalId: *<graphqlapi-LogicalId>*DomainName

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::AppSync::DomainNameApiAssociation

LogicalId: *<graphqlapi-LogicalId>*DomainNameApiAssociation

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS SAM は、AWS::Serverless::Connector リソースを使用して許可をプロビジョニングする場合もあります。詳細については、「AWS::Serverless::Connector を指定したときに生成された AWS CloudFormation リソース」を参照してください。

AWS::Serverless::HttpApi が指定されている時に生成された AWS CloudFormation リソース

AWS::Serverless::HttpApi が指定されている場合、AWS Serverless Application Model (AWS SAM) は AWS::ApiGatewayV2::Api のベース AWS CloudFormation リソースを生成します。

AWS::ApiGatewayV2::Api

LogicalId: *<httpapi-LogicalId>*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::Serverless::HttpApi` が指定されている場合、AWS SAM はこの AWS CloudFormation リソースに加えて、以下のシナリオに対する AWS CloudFormation リソースも生成します。

シナリオ

- StageName プロパティが指定されている
- StageName プロパティが指定されていない
- DomainName プロパティが指定されている

StageName プロパティが指定されている

`AWS::Serverless::HttpApi` の `StageName` プロパティが指定されていない場合、AWS SAM は `AWS::ApiGatewayV2::Stage` AWS CloudFormation リソースを生成します。

`AWS::ApiGatewayV2::Stage`

`LogicalId`: *`<httpapi-LogicalId><stage-name>`*`Stage`

`<stage-name>` は、`StageName` プロパティが設定されている文字列です。例えば、`StageName` を `Gamma` に設定すると、`LogicalId` は *`MyHttpapigamma`*`Stage` になります。

参照可能なプロパティ: *`<httpapi-LogicalId>`*`.Stage`

StageName プロパティが指定されていない

`AWS::Serverless::HttpApi` の `StageName` プロパティが指定されていない場合、AWS SAM は `AWS::ApiGatewayV2::Stage` AWS CloudFormation リソースを生成します。

`AWS::ApiGatewayV2::Stage`

`LogicalId`: *`<httpapi-LogicalId>`*`ApiGatewayDefaultStage`

参照可能なプロパティ: *`<httpapi-LogicalId>`*`.Stage`

DomainName プロパティが指定されている

AWS::Serverless::HttpApi の Domain プロパティの DomainName プロパティが指定されている場合、AWS SAM は AWS::ApiGatewayV2::DomainName AWS CloudFormation リソースを生成します。

AWS::ApiGatewayV2::DomainName

LogicalId: ApiGatewayDomainNameV2*<sha>*

<sha> は、スタックが作成されるときに生成される一意のハッシュ値です。例えば、ApiGatewayDomainNameV2*926eeb5ff1* などです。

参照可能なプロパティ: *<httpapi-LogicalId>*.DomainName

AWS::Serverless::LayerVersion が指定された場合、生成される AWS CloudFormation リソース

AWS::Serverless::LayerVersion が指定されている場合、AWS Serverless Application Model (AWS SAM) は AWS::Lambda::LayerVersion のベース AWS CloudFormation リソースを生成します。

AWS::Lambda::LayerVersion

LogicalId: *<layerversion-LogicalId>*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::Serverless::SimpleTable が指定された場合、生成される AWS CloudFormation リソース

AWS::Serverless::SimpleTable が指定されている場合、AWS Serverless Application Model (AWS SAM) は AWS::DynamoDB::Table のベース AWS CloudFormation リソースを生成します。

AWS::DynamoDB::Table

LogicalId: *<simpletable-LogicalId>*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、LogicalId を使用する必要があります)

AWS::Serverless::StateMachine が指定された場合、生成される AWS CloudFormation リソース

`AWS::Serverless::StateMachine` が指定されている場合、AWS Serverless Application Model (AWS SAM) は `AWS::StepFunctions::StateMachine` のベース AWS CloudFormation リソースを生成します。

`AWS::StepFunctions::StateMachine`

`LogicalId`: *`<statemachine-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

`AWS::Serverless::StateMachine` が指定されている場合、AWS SAM はこの AWS CloudFormation リソースに加えて、以下のシナリオに対する AWS CloudFormation リソースも生成します。

シナリオ

- Role プロパティが指定されていない
- Api イベントソースが指定されている
- イベントブリッジ (またはイベントバス) イベントソースが指定されている

Role プロパティが指定されていない

`AWS::Serverless::StateMachine` の `Role` プロパティが指定されていない場合、AWS SAM は `AWS::IAM::Role` AWS CloudFormation リソースを生成します。

`AWS::IAM::Role`

`LogicalId`: *`<statemachine-LogicalId>`*`Role`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

Api イベントソースが指定されている

`Event` の `AWS::Serverless::StateMachine` プロパティが `Api` に設定されているが、`RestApiId` プロパティは指定されていないという場合、AWS SAM は `AWS::ApiGateway::RestApi` AWS CloudFormation リソースを生成します。

`AWS::ApiGateway::RestApi`

`LogicalId`: `ServerlessRestApi`

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

イベントブリッジ (またはイベントバス) イベントソースが指定されている

`AWS::Serverless::StateMachine` の `Event` プロパティがイベントブリッジ (またはイベントバス) タイプのいずれかに設定されている場合、AWS SAM は `AWS::Events::Rule` AWS CloudFormation リソースを生成します。これは、`EventBridgeRule`、`Schedule`、および `CloudWatchEvents` の各タイプに適用されます。

`AWS::Events::Rule`

`LogicalId`: *`<statemachine-LogicalId><event-LogicalId>`*

参照可能なプロパティ: なし (この AWS CloudFormation リソースを参照するには、`LogicalId` を使用する必要があります)

AWS SAM でサポートされているリソース属性

リソース属性は、追加の動作と関係性を制御するために AWS SAM と AWS CloudFormation のリソースに追加できる属性です。リソース属性の詳細については、AWS CloudFormation ユーザーガイドの「Resource Attribute Reference」を参照してください。

AWS SAM は、AWS CloudFormation によって定義されるリソース属性のサブセットをサポートします。サポートされているリソース属性には、対応する AWS SAM リソースから生成されたベース AWS CloudFormation リソースのみにコピーされるものもあれば、対応する AWS SAM リソースから生成されたすべての AWS CloudFormation リソースにコピーされるものもあります。対応する AWS SAM から生成された AWS CloudFormation リソースの詳細については、「AWS SAM 向けに生成された AWS CloudFormation リソース」を参照してください。

以下の表は、AWS SAM がサポートするリソース属性を要約したもので、下記の「例外」が適用されます。

リソース属性	送信先で生成されるリソース
DependsOn Metadata [1、2]	生成されたベース AWS CloudFormation リソースのみ。AWS SAM リソースとベース AWS CloudFormation リソース間のマッピングについては、「生成された AWS CloudFormation リソースのシナリオ」を参照してください。
条件 DeletionPolicy UpdateReplacePolicy	対応する AWS SAM リソースから生成されたすべての AWS CloudFormation リソース。生成された AWS CloudFormation リソースのシナリオについては、「生成された AWS CloudFormation リソースのシナリオ」を参照してください。

注意:

1. `AWS::Serverless::Function` リソースタイプとの `Metadata` リソース属性の使用に関する詳細については、「でのカスタムランタイムを使用した Lambda 関数の構築 AWS SAM」を参照してください。
2. `AWS::Serverless::LayerVersion` リソースタイプとの `Metadata` リソース属性の使用に関する詳細については、「AWS SAM での Lambda レイヤーの構築」を参照してください。

例外

前述のリソース属性ルールには、いくつかの例外があります。

- `AWS::Lambda::LayerVersion` については、AWS SAM 限定のカスタムフィールド `RetentionPolicy` が、生成された AWS CloudFormation リソースの `DeletionPolicy` を設定します。この設定は、`DeletionPolicy` 自体よりも優先されます。どちらも設定されていない場合、`DeletionPolicy` はデフォルトで `Retain` に設定されます。
- `AWS::Lambda::Version` については、`DeletionPolicy` が指定されていない場合のデフォルトは `Retain` です。
- サーバーレス関数に `DeploymentPreferences` が指定されているシナリオでは、以下の生成された AWS CloudFormation リソースにリソース属性がコピーされません。

  - `AWS::CodeDeploy::Application`
  - `AWS::CodeDeploy::DeploymentGroup`
  - このシナリオ用に作成された、`CodeDeployServiceRole` という名前の `AWS::IAM::Role`
- 暗黙的に作成された API イベントソースを伴う複数の関数が AWS SAM テンプレートに含まれている場合、これらの関数は生成された `AWS::ApiGateway::RestApi` リソースを共有します。このシナリオでは、関数に異なるリソース属性がある場合、生成された `AWS::ApiGateway::RestApi` リソースに対し、AWS SAM が以下の優先リストに従ってリソース属性をコピーします。
  - `UpdateReplacePolicy`:
    1. `Retain`
    2. `Snapshot`
    3. `Delete`
  - `DeletionPolicy`:
    1. `Retain`
    2. `Delete`

AWS SAM 向けの API Gateway 拡張機能

API Gateway 拡張機能は AWS 用に特別に設計されており、API を設計および管理するためのカスタマイズや機能が追加されています。API Gateway 拡張機能は OpenAPI 仕様の拡張であり、AWS 固有の認可とAPI Gateway 固有の API 統合をサポートしています。API Gateway 拡張機能の詳細情報については、「API Gateway の OpenAPI 拡張機能」を参照してください。

AWS SAM は、API Gateway 拡張機能のサブセットをサポートします。AWS SAM がサポートする API Gateway 拡張機能の種類は、次の表でご確認いただけます。

API Gateway 拡張機能	AWS SAM によりサポート
x-amazon-apigateway-any-method オブジェクト	可能
x-amazon-apigateway-api-key-source プロパティ	不可
x-amazon-apigateway-auth オブジェクト	可能

x-amazon-apigateway-authorizer オブジェクト	可能
x amazon-apigateway-authtype プロパティ	可能
x-amazon-apigateway-binary-media-types のプロパティ	可能
x-amazon-apigateway-documentation オブジェクト	不可
x-amazon-apigateway-endpoint-configuration オブジェクト	不可
x-amazon-apigateway-gateway-responses オブジェクト	可能
x-amazon-apigateway-gateway-responses.gatewayResponse オブジェクト	可能
x-amazon-apigateway-gateway-responses.responseParameters オブジェクト	可能
x-amazon-apigateway-gateway-responses.responseTemplates オブジェクト	可能
x-amazon-apigateway-integration オブジェクト	可能
x-amazon-apigateway-integration.requestTemplates オブジェクト	可能
x-amazon-apigateway-integration.requestParameters オブジェクト	不可
x-amazon-apigateway-integration.responses オブジェクト	可能
x-amazon-apigateway-integration.response オブジェクト	可能
x-amazon-apigateway-integration.responseTemplates オブジェクト	可能
x-amazon-apigateway-integration.responseParameters オブジェクト	可能
x-amazon-apigateway-request-validator プロパティ	不可
x-amazon-apigateway-request-validators オブジェクト	不可
x-amazon-apigateway-request-validators.requestValidator オブジェクト	不可

AWS SAM の組み込み関数

組み込み関数は、ランタイムにのみ使用可能なプロパティに値を割り当てるために組み込まれた関数です。AWS SAM では、特定の組み込み関数プロパティにサポートが制限されているため、一部の組み込み関数を解決することはできません。したがって、これを解決するためには、`AWS::LanguageExtensions` 変換を追加することをお勧めします。`AWS::LanguageExtensions` は、AWS CloudFormation によってホストされたマクロであり、組み込み関数や、デフォルトでは AWS CloudFormation に含まれていないその他の機能の使用を可能にしています。

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

> **Note**
>
> 注: CodeUri プロパティで組み込み関数を使用すると、AWS SAM が値を正しく解析できなくなります。代わりに `AWS::LanguageExtensions` 変換の使用を検討してください。詳細情報については、AWS::Serverless::Function のプロパティセクションを参照してください。

組み込み関数の詳細については、AWS CloudFormation ユーザーガイドの「Intrinsic Function Reference」を参照してください。

AWS SAM を使用してサーバーレスアプリケーションを開発する

このセクションには、AWS SAM テンプレートの検証と、依存関係を使用したアプリケーションの構築に関するトピックが含まれています。また、Lambda レイヤーの操作、ネストされたアプリケーションの使用、API Gateway API へのアクセスの制御、Step Functions を使用した AWS リソースのオーケストレーション、およびアプリケーションのコード署名などの特定のユースケースのための AWS SAM の使用に関するトピックもあります。アプリケーションを開発するために完了する必要がある 3 つの主要なマイルストーンを以下に示します。

トピック

- AWS SAM でアプリケーションを作成する
- AWS SAMを使用してインフラストラクチャを定義する
- AWS SAM を使用してアプリケーションを構築する

AWS SAM でアプリケーションを作成する

使用を開始し「AWS Serverless Application Model (AWS SAM) を使用する方法」を読み終えれば、デベロッパー環境で AWS SAM プロジェクトを作成する準備が整います。AWS SAM プロジェクトは、サーバーレスアプリケーションを記述するための出発点の働きをします。AWS SAM CLI `sam init` コマンドオプションのリストについては、「sam init」を参照してください。

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam init` コマンドは、次で構成される新しいサーバーレスアプリケーションを初期化するためのオプションを提供します。

- インフラストラクチャコードを定義するための AWS SAM テンプレート。
- アプリケーションを整理するフォルダ構造。
- AWS Lambda 関数の設定。

AWS SAM プロジェクトを作成するには、このセクションのトピックを参照してください。

トピック

- 新しいサーバーレスアプリケーションを初期化する

- sam init のオプション
- トラブルシューティング
- 例
- 詳細はこちら
- 次のステップ

新しいサーバーレスアプリケーションを初期化する

AWS SAM CLI を使用して新しいサーバーレスアプリケーションを初期化するには

1. cd を実行して開始ディレクトリに移動します。
2. コマンドラインで次を実行します。

```
$ sam init
```

3. AWS SAM CLI は、新しいサーバーレスアプリケーションを作成するためのインタラクティブフローを通じてユーザーをガイドします。

> **Note**
>
> チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM で説明されているように、このコマンドはサーバーレスアプリケーションを初期化し、プロジェクトディレクトリを作成します。このディレクトリには、ファイルとフォルダがいくつか含まれています。最も重要なファイルは `template.yaml` です。これが使用する AWS SAM テンプレートです。ご使用の Python のバージョンは、sam init コマンドで作成された `template.yaml` ファイルにリストされている Python のものと、一致する必要があります。

開始テンプレートを選択する

テンプレートは次で構成されます。

1. インフラストラクチャコードの AWS SAM テンプレート。
2. プロジェクトファイルを整理する開始プロジェクトディレクトリ。例えば、これには次が含まれる場合があります。
   a. Lambda 関数コードとその依存関係の構造。

b. ローカルテスト用のテストイベントを含む events フォルダ。

c. ユニットテストをサポートする tests フォルダ。

d. プロジェクトの設定を構成する samconfig.toml ファイル。

e. ReadMe ファイルおよび他の基本的な開始プロジェクトファイル。

開始プロジェクトディレクトリの例を次に示します。

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

利用可能な AWS クイックスタートテンプレートのリストから選択することも、独自のカスタムテンプレートの場所を指定することもできます。

AWS クイックスタートテンプレートを選択するには

1. プロンプトが表示されたら、[AWS クイックスタートテンプレート] を選択します。
2. まずは AWS クイックスタートテンプレートを選択します。以下に例を示します。

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

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

独自のカスタムテンプレートの場所を選択するには

1. プロンプトが表示されたら、[カスタムテンプレートの場所] を選択します。

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

2. AWS SAM CLI は、テンプレートの場所を指定するよう促すプロンプトを表示します。

   ```
   Template location (git, mercurial, http(s), zip, path):
   ```

   テンプレート .zip ファイルアーカイブに対して次のいずれかの場所を指定します。

   - GitHub リポジトリ – GitHub リポジトリ内の .zip ファイルへのパス。ファイルはリポジトリのルートに存在する必要があります。
   - Mercurial リポジトリ – Mercurial リポジトリ内の .zip ファイルへのパス。ファイルはリポジトリのルートに存在する必要があります。
   - .zip パス – .zip ファイルへの HTTPS またはローカルパス。

3. AWS SAM CLI は、カスタムテンプレートを使用してサーバーレスアプリケーションを初期化します。

ランタイムを選択する

AWS クイックスタートテンプレートを選択すると、AWS SAM CLI は Lambda 関数のランタイムを選択するよう促すプロンプトを表示します。AWS SAM CLI によって表示されるオプションのリストは、Lambda によってネイティブにサポートされるランタイムです。

- ランタイムでは、実行環境で実行される言語固有の環境が提供されます。
- AWS クラウド にデプロイされると、Lambda サービスは実行環境で関数を呼び出します。

カスタムランタイムでは他のプログラミング言語を使用できます。これを実行するには、開始アプリケーション構造を手動で作成する必要があります。その後、カスタムテンプレートの場所を設定することで、`sam init` を使用してアプリケーションを迅速に初期化できます。

選択内容に基づいて、AWS SAM CLI は Lambda 関数コードと依存関係の開始ディレクトリを作成します。

Lambda がランタイムについて複数の依存関係マネージャーをサポートしている場合は、優先する依存関係マネージャーを選択するよう促すプロンプトが表示されます。

パッケージタイプを選択する

AWS クイックスタートテンプレートとランタイムを選択すると、AWS SAM CLI はパッケージタイプを選択するよう促すプロンプトを表示します。パッケージタイプによって、Lambda サービスで使用するために Lambda 関数がどのようにデプロイされるかが決まります。サポートされているパッケージタイプは次の 2 つです。

1. コンテナイメージ – 基本オペレーティングシステム、ランタイム、Lambda 拡張機能、アプリケーションコード、およびその依存関係が含まれています。
2. .zip ファイルアーカイブ – アプリケーション コードとその依存関係が含まれます。

デプロイパッケージタイプの詳細については、「AWS Lambda デベロッパーガイド」の「Lambda デプロイパッケージ」を参照してください。

Lambda 関数がコンテナイメージとしてパッケージ化されたアプリケーションのディレクトリ構造の例を次に示します。AWS SAM CLI はイメージをダウンロードし、関数のディレクトリで `Dockerfile` を作成してイメージを指定します。

```
sam-app
```

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

関数が .zip ファイルアーカイブとしてパッケージ化されたアプリケーションのディレクトリ構造の例を次に示します。

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

AWS X-Ray トレースを設定する

AWS X-Ray トレースをアクティブ化するかどうかを選択できます。詳細については、「AWS X-Ray デベロッパーガイド」の「AWS X-Ray とは」を参照してください。

アクティブ化すると、AWS SAM CLI は AWS SAM テンプレートを設定します。以下に例を示します。

```
Globals:
  Function:
    ...
    Tracing: Active
  Api:
    TracingEnabled: True
```

Amazon CloudWatch Application Insights でモニタリングを設定する

Amazon CloudWatch Application Insights を使用してモニタリングをアクティブ化することを選択できます。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Application Insights」を参照してください。

アクティブ化すると、AWS SAM CLI は AWS SAM テンプレートを設定します。以下に例を示します。

```
Resources:
  ApplicationResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
          - Ref: AWS::StackName
      ResourceQuery:
        Type: CLOUDFORMATION_STACK_1_0
  ApplicationInsightsMonitoring:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName:
        Fn::Join:
        - ''
```

```
        - - ApplicationInsights-SAM-
          - Ref: AWS::StackName
      AutoConfigurationEnabled: 'true'
    DependsOn: ApplicationResourceGroup
```

アプリケーションに名前を付ける

アプリケーションの名前を入力します。AWS SAM CLI は、この名前を使用してアプリケーションの最上位フォルダを作成します。

sam init のオプション

`sam init` コマンドで使用できる主なオプションの一部を次に示します。すべてのオプションのリストについては、「sam init」を参照してください。

カスタムテンプレートの場所を使用してアプリケーションを初期化する

`--location` オプションを使用して、サポートされているカスタムテンプレートの場所を指定します。以下に例を示します。

```
$ sam init --location https://github.com/aws-samples/sessions-with-aws-sam/raw/master/
starter-templates/web-app.zip
```

インタラクティブフローを使用せずにアプリケーションを初期化する

`--no-interactive` オプションを使用して、コマンドラインで設定の選択内容を指定し、インタラクティブフローをスキップします。以下に例を示します。

```
$ sam init --no-interactive --runtime go1.x --name go-demo --dependency-manager mod --
app-template hello-world
```

トラブルシューティング

AWS SAM CLI をトラブルシューティングするには、「AWS SAMCLI トラブルシューティング」を参照してください。

例

Hello World AWS スターターテンプレートを使用して新しいサーバーレスアプリケーションを初期化する

この例については、「チュートリアル: Hello World アプリケーションのデプロイ」の「ステップ 1: サンプルの Hello World アプリケーションを初期化する」を参照してください。

カスタムテンプレートの場所を使用して新しいサーバーレスアプリケーションを初期化する

カスタムテンプレートに GitHub の場所を提供する例を次に示します。

```
$ sam init --location gh:aws-samples/cookiecutter-aws-sam-python
$ sam init --location git+sh://git@github.com/aws-samples/cookiecutter-aws-sam-python.git
$ sam init --location hg+ssh://hg@bitbucket.org/repo/template-name
```

ローカルファイルパスの例を次に示します。

```
$ sam init --location /path/to/template.zip
```

HTTPS によって到達可能なパスの例を次に示します。

```
$ sam init --location https://github.com/aws-samples/sessions-with-aws-sam/raw/master/starter-templates/web-app.zip
```

詳細はこちら

`sam init` コマンドの使用方法の詳細については、次を参照してください。

- AWS SAM の学習: sam init – YouTube の Serverless Land の「AWS SAM の学習」シリーズ。
- AWS SAM CLI で使用するサーバーレスアプリケーションの構築 (SAM S2E7 を使用したセッション) – YouTube の AWS SAM を使用したセッションのシリーズ。

次のステップ

AWS SAM プロジェクトを作成したら、アプリケーションのオーサリングを開始できます。これを行うために必要なタスクの詳細については、「AWS SAMを使用してインフラストラクチャを定義する」を参照してください。

AWS SAMを使用してインフラストラクチャを定義する

プロジェクトを作成したら、AWS SAM を使用してアプリケーションインフラストラクチャを定義できます。これを行うには、AWS SAM テンプレートを設定して、AWS SAM プロジェクトの `template.yaml` ファイルであるアプリケーションのリソースとプロパティを定義します。

このセクションのトピックでは、AWS SAM テンプレート (`template.yaml` ファイル) でインフラストラクチャを定義する方法について説明します。また、Lambda レイヤーの操作、ネストされたアプリケーションの使用、API Gateway API へのアクセスの制御、Step Functions を使用した AWS リソースのオーケストレーション、およびアプリケーションのコード署名、AWS SAM テンプレートの検証などの特定のユースケースのためのリソース定義に関するトピックも含まれます。

トピック

- AWS SAM テンプレートでアプリケーションリソースを定義する
- AWS SAM テンプレート内でリソースアクセスを設定および管理する
- AWS SAM テンプレートを使用して API アクセスを制御する
- AWS SAM で Lambda レイヤーを使用して効率を向上させる
- AWS SAM 内で、ネストされたアプリケーションを使用して、コードとリソースを再使用する
- AWS SAM で EventBridge スケジューラを使用して時間ベースのイベントを管理する
- AWS Step Functions を使用した AWS SAM リソースのオーケストレーション
- AWS SAM アプリケーションのコード署名を設定する
- AWS SAM テンプレートファイルを検証する

AWS SAM テンプレートでアプリケーションリソースを定義する

サーバーレスアプリケーションで使用する AWS リソースは、AWS SAM テンプレートの `Resources` セクションで定義します。リソースを定義する際は、リソースとは何か、他のリソースとやり取りする方法、およびリソースへのアクセス方法 (リソースのアクセス許可) を特定します。

AWS SAM テンプレートの `Resources` セクションには、AWS CloudFormation リソースと AWS SAM リソースの組み合わせを含めることができます。さらに、次のリソースには AWS SAM の省略構文を使用できます。

AWS SAM の省略構文	関連する AWS リソースでの動作
AWS::Serverless::Api	HTTPS エンドポイント経由で呼び出すことができる API Gateway リソースとメソッドのコレクションを作成します。
AWS::Serverless::Application	AWS Serverless Application Repository から、または Amazon S3 バケットからのサーバーレスアプリケーションを、ネストされたアプリケーションとして埋め込みます。
AWS::Serverless::Connector	2 つのリソース間のアクセス許可を設定します。コネクタの概要については、「AWS SAM コネクタによるリソースに対するアクセス許可の管理」を参照してください。
AWS::Serverless::Function	AWS Lambda 関数、AWS Identity and Access Management (IAM) 実行ロール、およびこの関数をトリガーするイベントソースマッピングを作成します。
AWS::Serverless::GraphQLApi	サーバーレスアプリケーション用の AWS AppSync GraphQL API を作成し、設定します。
AWS::Serverless::HttpApi	REST API よりもレイテンシーとコストが低い RESTful API を作成できる Amazon API Gateway HTTP API を作成します。
AWS::Serverless::LayerVersion	Lambda 関数に必要なライブラリまたはランタイムコードが含まれる Lambda LayerVersion を作成します。

AWS SAM の省略構文	関連する AWS リソースでの動作
AWS::Serverless::SimpleTable	単一属性のプライマリキーで DynamoDB テーブルを作成します。
AWS::Serverless::StateMachine	AWS Step Functions ステートマシンを作成します。これは、AWS Lambda 関数と AWS リソースのオーケストレーションを行って、複雑で堅牢なワークフローを形成するために使用できます。

上記のリソースは、AWS SAM リソースとプロパティ にも記載されています。

AWS CloudFormation および AWS SAM によってサポートされるすべての AWS リソースとプロパティの参照情報については、AWS CloudFormation ユーザーガイドの「AWS リソースおよびプロパティタイプのリファレンス」を参照してください。

AWS SAM テンプレート内でリソースアクセスを設定および管理する

AWS リソースで相互のやり取りを行わせるには、そのリソース間で適切なアクセスとアクセス許可を設定する必要があります。これにより安全な方法でのやり取りを完了するためには、AWS Identity and Access Management (IAM) ユーザー、ロール、ポリシーの設定が必要です。

このセクションのトピックはすべて、テンプレートの中で定義されているリソースへのアクセス設定に関連があります。このセクションの最初では、一般的なベストプラクティスを扱っています。次の 2 つのトピックでは、サーバーレスアプリケーションで参照されるリソース (AWS SAM コネクタと AWS SAM ポリシーテンプレート) 間のアクセスとアクセス許可を設定するために用意されている、2 つのオプションについて説明します。最後のトピックでは、AWS CloudFormation がユーザーの管理に使用するのと同じメカニズムを使用して、ユーザーアクセスを管理する方法の詳細について説明します。

詳細については、「AWS CloudFormation ユーザーガイド」の「AWS Identity and Access Management によるアクセスの制御」を参照してください。

AWS Serverless Application Model (AWS SAM) には、サーバーレスアプリケーションのアクセスとアクセス許可の管理を簡素化する 2 つのオプションがあります。

1. AWS SAM コネクタ

2. AWS SAM ポリシーテンプレート

AWS SAM コネクタ

コネクタを使用することで、2 つのリソース間の許可をプロビジョニングできます。これを実行するには、AWS SAM テンプレートで相互にインタラクションする方法を記述します。これらは、`Connectors` リソース属性または `AWS::Serverless::Connector` リソースタイプのいずれかを使用して定義できます。コネクタは、AWS リソースの組み合わせの間におけるデータとイベントの `Read` および `Write` アクセスのプロビジョニングをサポートします。AWS SAM コネクタの詳細については、「AWS SAM コネクタによるリソースに対するアクセス許可の管理」を参照してください。

AWS SAM ポリシーテンプレート

AWS SAM ポリシーテンプレートは事前に定義されたアクセス許可のセットであり、AWS SAM テンプレートに追加して、AWS Lambda 関数、AWS Step Functions ステートマシン、およびそれらが相互作用するリソース間のアクセスとアクセス許可を管理できます。AWS SAM ポリシーテンプレートの詳細については、「AWS SAMポリシーテンプレート」を参照してください。

AWS CloudFormation メカニズム

AWS CloudFormation メカニズムには、AWS リソース間のアクセス許可を管理するための IAM ユーザー、ロール、ポリシーの設定が含まれます。詳細については、「AWS CloudFormation メカニズムを使用した AWS SAM のアクセス許可の管理」を参照してください。

ベストプラクティス

サーバーレスアプリケーション全体で、複数の方法を使用してリソース間のアクセス許可を設定できます。そのため、各シナリオに最適なオプションを選択し、アプリケーション全体で複数のオプションを一緒に使用できます。最適なオプションを選択する際に留意すべき点がいくつかあります。

- AWS SAM コネクタとポリシーテンプレートはどちらも、AWS リソース間の安全なやりとりを促進するために必要な IAM の専門知識が少なく済みます。サポートされている場合は、コネクタとポリシーテンプレートを使用してください。
- AWS SAM コネクタには、AWS SAM テンプレート内のアクセス許可を定義するためのシンプルで直感的な、簡潔な構文が用意されており、IAM の専門知識は最小限で済みます。AWS SAM コネクタとポリシーテンプレートの両方がサポートされている場合は、AWS SAM コネクタを使用してください。

- AWS SAM コネクタは、サポートされている AWS SAM ソースと宛先リソース間でデータやイベントの `Read` および `Write` アクセスをプロビジョニングできます。サポートされているリソースの一覧については、「AWS SAM コネクタリファレンス」を参照してください。サポートされている場合は、AWS SAM コネクタを使用してください。
- AWS SAM ポリシーテンプレートは、Lambda 関数、Step Functions ステートマシン、およびそれらが操作する AWS リソース間のアクセス許可に限定されますが、ポリシーテンプレートはすべての CRUD 操作をサポートします。サポートされている場合、および対象シナリオ用の AWS SAM ポリシーテンプレートが利用可能な場合は、AWS SAM ポリシーテンプレートを使用してください。使用可能なポリシーテンプレートのリストについては、「AWS SAMポリシーテンプレート」を参照してください。
- それ以外のシナリオや、きめ細かな制御が求められる場合は、AWS CloudFormation メカニズムを使用してください。

AWS SAM コネクタによるリソースに対するアクセス許可の管理

コネクタは、サーバーレスアプリケーションリソース間にシンプルで範囲の広いアクセス許可を提供する AWS Serverless Application Model（ AWS SAM) 抽象リソースタイプで`AWS::Serverless::Connector`、 として識別されます。

AWS SAM コネクタの利点

リソース間で適切なアクセスポリシーを自動的に作成することで、コネクタは、 AWS 認可機能、ポリシー言語、サービス固有のセキュリティ設定に関する専門知識を必要とせずに、サーバーレスアプリケーションをオーサリングし、アプリケーションアーキテクチャに集中できるようになります。したがって、コネクタは、サーバーレス開発に慣れていない開発者や、開発速度を上げたいと考えている経験豊富な開発者にとって大きなメリットです。

AWS SAM コネクタの使用

`Connectors` リソース属性をソースリソース内に埋め込んで使用します。次に、送信先リソースを定義し、これらのリソース間でデータまたはイベントがどのように流れるかを説明します。 AWS SAM 次に、必要なインタラクションを容易にするために必要なアクセスポリシーを構成します。

以下に、このリソース属性の記述方法の概要を示します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
```

```
Resources:
  <source-resource-logical-id>:
    Type: <resource-type>
    ...
    Connectors:
      <connector-name>:
        Properties:
          Destination:
            <properties-that-identify-destination-resource>
          Permissions:
            <permission-types-to-provision>
  ...
```

コネクタの仕組み

> **Note**
>
> このセクションでは、コネクタがバックグラウンドで必要なリソースをプロビジョニングする方法について説明します。これは、コネクタを使用する際に自動的に実行されます。

まず、埋め込まれた Connectors リソース属性が AWS::Serverless::Connector リソースタイプに変換されます。その論理 ID は として自動的に作成されます。*<source-resource-logical-id><embedded-connector-logical-id>*.

例えば、これは埋め込みコネクタです。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
```

```
    Type: AWS::DynamoDB::Table
```

これにより、次の AWS::Serverless::Connector リソースが生成されます。

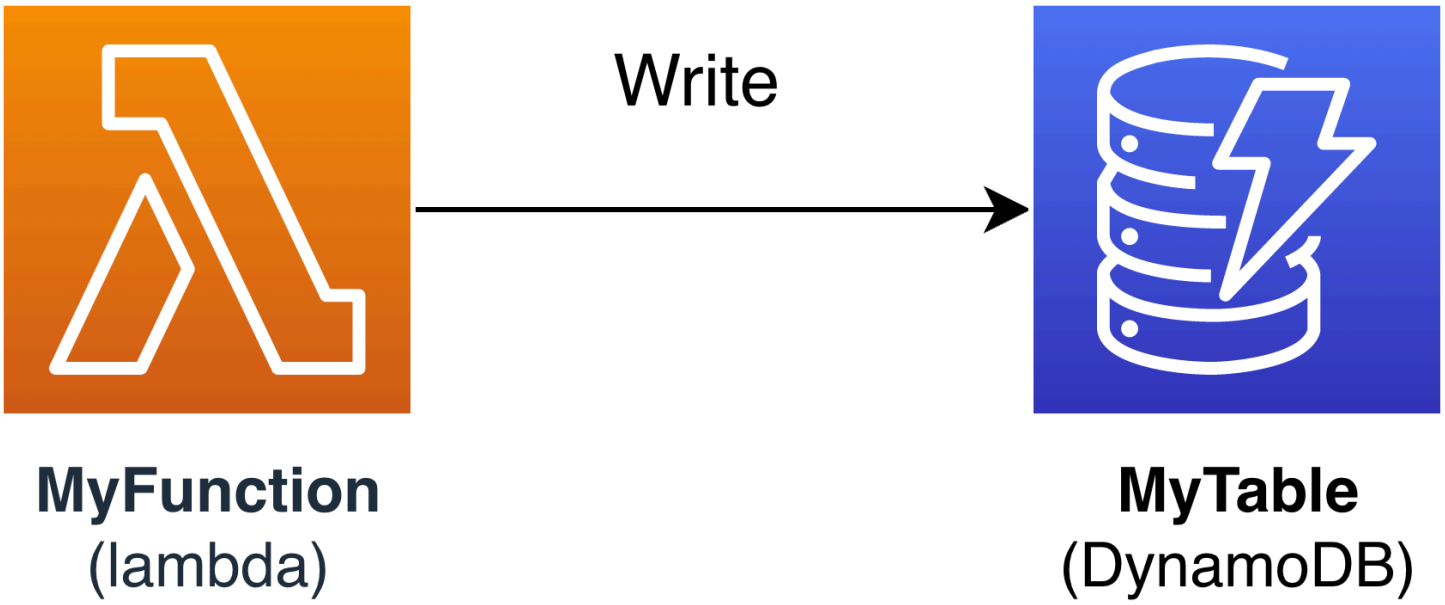
```
Transform: AWS::Serverless-2016-10-31
Resources:
  ...
  MyFunctionMyConn:
    Type: AWS::Serverless::Connector
    Properties:
      Source:
        Id: MyFunction
      Destination:
        Id: MyTable
      Permissions:
        - Read
        - Write
```

> **Note**
>
> この構文を使用して、 AWS SAM テンプレート内のコネクタを定義することもできます。これは、ソースリソースがコネクタとは別のテンプレートで定義されている場合に推奨されます。

次に、この接続に必要なアクセスポリシーが自動的に作成されます。コネクタによって生成されるリソースの詳細については、「AWS::Serverless::Connector を指定したときに生成された AWS CloudFormation リソース」を参照してください。

コネクタの例

次の例は、コネクタを使用して AWS Lambda 関数から Amazon DynamoDB テーブルにデータを書き込む方法を示しています。




```
Transform: AWS::Serverless-2016-10-31
Resources:
  MyTable:
    Type: AWS::Serverless::SimpleTable
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Write
    Properties:
      Runtime: nodejs16.x
      Handler: index.handler
      InlineCode: |
        const AWS = require("aws-sdk");
        const docClient = new AWS.DynamoDB.DocumentClient();
        exports.handler = async (event, context) => {
          await docClient.put({
            TableName: process.env.TABLE_NAME,
            Item: {
              id: context.awsRequestId,
              event: JSON.stringify(event)
            }
          }).promise();
```

```
        }
      Environment:
        Variables:
          TABLE_NAME: !Ref MyTable
```

Connectors リソース属性は、Lambda 関数のソースリソース内に埋め込まれています。DynamoDB テーブルは、Id プロパティを使用して送信先リソースとして定義されます。コネクタは、これら 2 つのリソース間の Write 許可をプロビジョニングします。

AWS SAM テンプレートを にデプロイすると AWS CloudFormation、 AWS SAM は、この接続が機能するために必要なアクセスポリシーを自動的に作成します。

送信元リソースと送信先リソースの間でサポートされている接続

コネクタは、ソースと送信先のリソース接続の選択された組み合わせの間におけるデータとイベントの Read および Write 許可タイプをサポートします。例えば、コネクタは AWS::ApiGateway::RestApi ソースリソースと AWS::Lambda::Function 送信先リソースの間の Write 接続をサポートします。

ソースリソースと送信先リソースは、サポートされているプロパティを組み合わせて定義できます。プロパティの要件は、使用する接続と、リソースが定義されている場所によって異なります。

> **Note**
>
> コネクタは、サポートされているサーバーレスリソースタイプと非サーバーレスリソースタイプの間の許可をプロビジョニングできます。

サポートされているリソース接続とそのプロパティの要件のリストについては、「コネクタに対してサポートされている送信元リソースと送信先リソースのタイプ」を参照してください。

で読み取りおよび書き込みアクセス許可を定義する AWS SAM

では AWS SAM、 Read および アクセスWrite許可は 1 つのコネクタ内でプロビジョニングできます。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
```

```
    Type: AWS::Lambda::Function
    Connectors:
      MyTableConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
```

コネクタの使用の詳細については、「AWS SAM コネクタリファレンス」を参照してください。

でサポートされている他のプロパティを使用してリソースを定義する AWS SAM

ソースリソースと送信先リソースの両方について、同じテンプレート内で定義されている場合は、Id プロパティを使用します。オプションで、Qualifier を追加して、定義したリソースの範囲を絞り込むことができます。リソースが同じテンプレート内にない場合は、サポートされているプロパティの組み合わせを使用してください。

- ソースリソースと送信先リソースでサポートされているプロパティの組み合わせのリストについては、「コネクタに対してサポートされている送信元リソースと送信先リソースのタイプ」を参照してください。
- コネクタで使用できるプロパティの説明については、「AWS::Serverless::Connector」を参照してください。

Id 以外のプロパティでソースリソースを定義する場合は、SourceReference プロパティを使用します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  <source-resource-logical-id>:
    Type: <resource-type>
    ...
    Connectors:
      <connector-name>:
        Properties:
          SourceReference:
```

```
            Qualifier: <optional-qualifier>
            <other-supported-properties>
          Destination:
            <properties-that-identify-destination-resource>
          Permissions:
            <permission-types-to-provision>
```

以下は、 Qualifier を使用して Amazon API Gateway リソースの範囲を絞り込む例です。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Connectors:
      ApiToLambdaConn:
        Properties:
          SourceReference:
            Qualifier: Prod/GET/foobar
          Destination:
            Id: MyFunction
          Permissions:
            - Write
  ...
```

サポートされている Arn と Type の組み合わせを使用して、別のテンプレートから送信先リソースを定義する例を次に示します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      TableConn:
        Properties:
          Destination:
            Type: AWS::DynamoDB::Table
            Arn: !GetAtt MyTable.Arn
  ...
```

コネクタの使用の詳細については、「AWS SAM コネクタリファレンス」を参照してください。

で 1 つのソースから複数のコネクタを作成する AWS SAM

ソースリソース内で、それぞれが異なる宛先リソースを持つ複数のコネクタを定義できます。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      BucketConn:
        Properties:
          Destination:
            Id: amzn-s3-demo-bucket
          Permissions:
            - Read
            - Write
      SQSConn:
        Properties:
          Destination:
            Id: MyQueue
          Permissions:
            - Read
            - Write
      TableConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
      TableConnWithTableArn:
        Properties:
          Destination:
            Type: AWS::DynamoDB::Table
            Arn: !GetAtt MyTable.Arn
          Permissions:
            - Read
            - Write
...
```

コネクタの使用の詳細については、「AWS SAM コネクタリファレンス」を参照してください。

でマルチ宛先コネクタを作成する AWS SAM

ソースリソース内で、複数の宛先リソースを持つ単一のコネクタを定義できます。Amazon Simple Storage Service (Amazon S3) バケットと DynamoDB テーブルに接続する Lambda 関数のソースリソースの例を次に示します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      WriteAccessConn:
        Properties:
          Destination:
            - Id: OutputBucket
            - Id: CredentialTable
          Permissions:
            - Write
  ...
  OutputBucket:
    Type: AWS::S3::Bucket
  CredentialTable:
    Type: AWS::DynamoDB::Table
```

コネクタの使用の詳細については、「AWS SAM コネクタリファレンス」を参照してください。

でコネクタを使用してリソース属性を定義する AWS SAM

リソース用にリソース属性を定義して、追加の動作や関係を指定できます。リソース属性の詳細については、「AWS CloudFormation ユーザーガイド」の「リソース属性リファレンス」を参照してください。

リソース属性をコネクタプロパティと同じレベルで定義することで、埋め込みコネクタに追加できます。テンプレート AWS SAM がデプロイ時に変換されると、属性は生成されたリソースに渡されます。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
```

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        DeletionPolicy: Retain
        DependsOn: AnotherFunction
        Properties:
          ...
```

コネクタの使用の詳細については、「AWS SAM コネクタリファレンス」を参照してください。

詳細

AWS SAM コネクタの使用の詳細については、以下のトピックを参照してください。

- AWS::Serverless::Connector
- で読み取りおよび書き込みアクセス許可を定義する AWS SAM
- でサポートされている他のプロパティを使用してリソースを定義する AWS SAM
- で 1 つのソースから複数のコネクタを作成する AWS SAM
- でマルチ宛先コネクタを作成する AWS SAM
- で読み取りおよび書き込みアクセス許可を定義する AWS SAM
- でコネクタを使用してリソース属性を定義する AWS SAM

フィードバックを送信する

コネクタに関するフィードバックを提供するには、serverless-application-model AWS GitHubリポジトリ で新しい問題を送信します。

AWS SAMポリシーテンプレート

AWS Serverless Application Model （ AWS SAM) を使用すると、ポリシーテンプレートのリストから選択して、Lambda 関数と AWS Step Functions ステートマシンのアクセス許可をアプリケーションで使用されるリソースに絞り込むことができます。

AWS SAM ポリシーテンプレート AWS Serverless Application Repository を使用する のアプリケーションでは、 からアプリケーションをデプロイするために特別な顧客確認は必要ありません AWS Serverless Application Repository。

新しいポリシーテンプレートの追加をリクエストしたい場合は、以下を実行します。

1. AWS SAM GitHub プロジェクトの developブランチの policy_templates.json ソースファイルに対してプルリクエストを送信します。ソースファイルは、 GitHub ウェブサイトの policy_templates.json にあります。
2. プルリクエストの理由とリクエストへのリンク AWS SAM GitHub を含むプロジェクトの問題を送信します。新しい問題を送信するには、AWS Serverless Application Model: 問題リンクを使用してください。

構文

AWS SAM テンプレートファイルで指定するポリシーテンプレートごとに、ポリシーテンプレートのプレースホルダー値を含むオブジェクトを常に指定する必要があります。ポリシーテンプレートにプレースホルダ値が必要ない場合は、空のオブジェクトを指定する必要があります。

YAML

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    Policies:
      - PolicyTemplateName1:        # Policy template with placeholder value
          Key1: Value1
      - PolicyTemplateName2: {}     # Policy template with no placeholder value
```

例

例 1: プレースホルダー値が含まれるポリシーテンプレート

以下の例は、SQSPollerPolicy ポリシーテンプレートがリソースとして QueueName を期待する例です。 AWS SAM テンプレートは、MyQueue「」Amazon SQSキューの名前を取得します。これは、同じアプリケーションで作成することも、アプリケーションのパラメータとしてリクエストすることもできます。

```
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    CodeUri: ${codeuri}
    Handler: hello.handler
    Runtime: python2.7
    Policies:
      - SQSPollerPolicy:
```

```
          QueueName:
            !GetAtt MyQueue.QueueName
```

例 2: プレースホルダー値が含まれないポリシーテンプレート

以下の例には、プレースホルダー値がない CloudWatchPutMetricPolicy ポリシーテンプレートが含まれています。

> **Note**
>
> プレースホルダ値がない場合でも、空のオブジェクトを指定する必要があります。指定されない場合は、エラーが発生します。

```
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    CodeUri: ${codeuri}
    Handler: hello.handler
    Runtime: python2.7
    Policies:
      - CloudWatchPutMetricPolicy: {}
```

ポリシーテンプレート表

以下は、使用可能なポリシーテンプレートの表です。

ポリシーテンプレート	説明		
AcmGetCertificatePolicy	証明書を読み取るアクセス許可を付与します AWS Certificate Manager。		
AMIDescribePolicy	Amazon マシンイメージ () を記述するアクセス許可を付与しますAMIs。		
AthenaQueryPolicy	Athena クエリを実行する許可を付与します。		

ポリシーテンプレート	説明		
AWSSecretsManagerGetSecretValuePolicy	指定された AWS Secrets Manager シークレットのシークレット値を取得する許可を付与します。		
AWSSecretsManagerRotationPolicy	AWS Secrets Managerのシークレットをローテーションを行う許可を付与します。		
CloudFormationDescribeStacksPolicy	AWS CloudFormation スタックを記述するアクセス許可を付与します。		
CloudWatchDashboardPolicy	CloudWatch ダッシュボードで運用するメトリクスを配置するアクセス許可を付与します。		
CloudWatchDescribeAlarmHistoryPolicy	CloudWatch アラーム履歴を記述するアクセス許可を付与します。		
CloudWatchPutMetricPolicy	メトリクスを送信するアクセス許可を付与します CloudWatch。		
CodeCommitCrudPolicy	特定の CodeCommitリポジトリ内のcreate/read/update/deleteオブジェクトにアクセス許可を付与します。		
CodeCommitReadPolicy	特定の CodeCommit リポジトリ内のオブジェクトを読み取るアクセス許可を付与します。		

ポリシーテンプレート	説明		
CodePipelineLambdaExecutionPolicy	によって呼び出された Lambda 関数 CodePipeline に、ジョブのステータスを報告するアクセス許可を付与します。		
CodePipelineReadOnlyPolicy	CodePipeline パイプラインの詳細を取得する読み取りアクセス許可を付与します。		
ComprehendBasicAccessPolicy	エンティティ、キーフレーズ、言語、およびセンチメントを検出する許可を付与します。		
CostExplorerReadOnlyPolicy	請求履歴APIsの読み取り専用 Cost Explorer に読み取り専用アクセス許可を付与します。		
DynamoDBBackupFullAccessPolicy	テーブルの DynamoDB オンデマンドバックアップに読み取りおよび書き込み許可を付与します。		
DynamoDBCrudPolicy	Amazon DynamoDB テーブルに作成、読み取り、更新、および削除許可を付与します。		
DynamoDBReadPolicy	DynamoDB テーブルに読み取り専用許可を付与します。		
DynamoDBReconfigurePolicy	DynamoDB テーブルを再構成する許可を付与します。		
DynamoDBRestoreFromBackupPolicy	バックアップから DynamoDB テーブルを復元する許可を付与します。		

ポリシーテンプレート	説明		
DynamoDBStreamReadPolicy	DynamoDB のストリームとレコードを記述および読み取る許可を付与します。		
DynamoDBWritePolicy	DynamoDB テーブルに書き込み専用許可を付与します。		
EC2CopyImagePolicy	Amazon EC2イメージをコピーするアクセス許可を付与します。		
EC2DescribePolicy	Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを記述するアクセス許可を付与します。		
EcsRunTaskPolicy	タスク定義の新しいタスクを開始する許可を付与します。		
EFSWriteAccessPolicy	書き込みアクセスを使用して Amazon EFS ファイルシステムをマウントするアクセス許可を付与します。		
EKSDescribePolicy	Amazon EKSクラスターを記述または一覧表示するアクセス許可を付与します。		
ElasticMapReduceAddJobFlowStepsPolicy	実行中のクラスターに新しいステップを追加する許可を付与します。		
ElasticMapReduceCancelStepsPolicy	実行中のクラスターで保留中のステップ (1つ、または複数) をキャンセルする許可を付与します。		

ポリシーテンプレート	説明		
ElasticMapReduceModifyInstanceFleetPolicy	クラスター内のインスタンスフリートの詳細をリスト化し、容量を変更する許可を付与します。		
ElasticMapReduceModifyInstanceGroupsPolicy	クラスター内のインスタンスグループの詳細をリスト化し、設定を変更する許可を付与します。		
ElasticMapReduceSetTerminationProtectionPolicy	クラスターの終了保護を設定する許可を付与します。		
ElasticMapReduceTerminateJobFlowsPolicy	クラスターをシャットダウンする許可を付与します。		
ElasticsearchHttpPostPolicy	Amazon OpenSearch Service にアクセスPOST許可を付与します。		
EventBridgePutEventsPolicy	イベントを送信するアクセス許可を付与します EventBridge。		
FilterLogEventsPolicy	指定された CloudWatch ロググループからログイベントをフィルタリングするアクセス許可を付与します。		

ポリシーテンプレート	説明		
FirehoseCrudPolicy	Firehose の配信ストリームに対し作成、書き込み、更新、および削除を行うアクセス許可を付与します。		
FirehoseWritePolicy	Firehose の配信ストリームに書き込むアクセス許可を付与します		
KinesisCrudPolicy	Amazon Kinesis のストリームを作成、発行、および削除する許可を付与します。		
KinesisStreamReadPolicy	Amazon Kinesis のストリームをリスト化して読み取る許可を付与します。		
KMSDecryptPolicy	AWS Key Management Service（AWS KMS) キーで復号するアクセス許可を付与します。		
KMSEncryptPolicy	AWS Key Management Service（AWS KMS) キーで暗号化するアクセス許可を付与します。		
LambdaInvokePolicy	AWS Lambda 関数、エイリアス、またはバージョンを呼び出すアクセス許可を付与します。		
MobileAnalyticsWriteOnlyAccessPolicy	すべてのアプリケーションリソースのイベントデータを配置するための書き込み専用許可を付与します。		
OrganizationsListAccountsPolicy	子アカウント名 と を一覧表示する読み取り専用アクセス許可を付与しますIDs。		

ポリシーテンプレート	説明		
PinpointEndpointAccessPolicy	Amazon Pinpoint アプリケーションのエンドポイントを取得して更新する許可を付与します		
PollyFullAccessPolicy	Amazon Polly のレキシコンリソースへのフルアクセス許可を付与します。		
RekognitionDetectOnlyPolicy	顔、ラベル、およびテキストを検出する許可を付与します。		
RekognitionFacesManagementPolicy	Amazon Rekognition コレクション内の顔を追加、削除、および検索する許可を付与します。		
RekognitionFacesPolicy	顔とラベルを比較および検出する許可を付与します。		
RekognitionLabelsPolicy	オブジェクトとモデレーションラベルを検出する許可を付与します。		
RekognitionNoDataAccessPolicy	顔とラベルを比較および検出する許可を付与します。		
RekognitionReadPolicy	顔をリスト化して検索する許可を付与します。		
RekognitionWriteOnlyAccessPolicy	顔のコレクションを作成してインデックス化する許可を付与します。		

ポリシーテンプレート	説明		
Route53ChangeResourceRecordSetsPolicy	Route 53 のリソースレコードセットを変更する許可を付与します。		
S3CrudPolicy	Amazon S3 バケット内のオブジェクトでアクションを実行するための作成、読み取り、更新、および削除許可を付与します。		
S3FullAccessPolicy	Amazon S3 バケット内のオブジェクトでアクションを実行するためのフルアクセス許可を付与します。		
S3ReadPolicy	Amazon Simple Storage Service (Amazon S3) バケットにあるオブジェクトを読み取るための読み取り専用許可を付与します。		
S3WritePolicy	Amazon S3 バケットにオブジェクトを書き込むための書き込み許可を付与します。		
SageMakerCreateEndpointConfigPolicy	SageMaker AI でエンドポイント設定を作成するアクセス許可を付与します。		
SageMakerCreateEndpointPolicy	SageMaker AI でエンドポイントを作成するアクセス許可を付与します。		
ServerlessRepoReadWriteAccessPolicy	AWS Serverless Application Repository サービスでアプリケーションを作成および一覧表示するアクセス許可を付与します。		

ポリシーテンプレート	説明		
SESBulkTemplatedCrudPolicy	E メール、テンプレート化された E メール、テンプレート化されたバルク E メールを送信し、アイデンティティを確認する許可を付与します。		
SESBulkTemplatedCrudPolicy_v2	Amazon E SESメール、テンプレート化された E メール、およびテンプレート化された一括 E メールを送信し、ID を検証するアクセス許可を付与します。		
SESCrudPolicy	E メールを送信し、アイデンティティを確認する許可を付与します。		
SESEmailTemplateCrudPolicy	Amazon E SESメールテンプレートを作成、取得、一覧表示、更新、削除するアクセス許可を付与します。		
SESSendBounncePolicy	Amazon Simple Email Service (Amazon SES) ID にアクセス SendBounce 許可を付与します。		
SNSCrudPolicy	Amazon SNSトピックを作成、公開、サブスクライブするアクセス許可を付与します。		
SNSPublishMessagePolicy	Amazon Simple Notification Service (Amazon SNS) トピックにメッセージを発行するアクセス許可を付与します。		
SQSPollerPolicy	Amazon Simple Queue Service (Amazon SQS) キューをポーリングするアクセス許可を付与します。		
SQSSendMessagePolicy	Amazon SQSキューにメッセージを送信するアクセス許可を付与します。		

ポリシーテンプレート	説明		
SSMParameterReadPolicy	Amazon EC2 Systems Manager (SSM) パラメータストアからパラメータにアクセスして、このアカウントにシークレットをロードするアクセス許可を付与します。パラメータ名にスラッシュプレフィックスが含まれていない場合に使用します。		
SSMParameterWithSlashPrefixReadPolicy	Amazon EC2 Systems Manager (SSM) パラメータストアからパラメータにアクセスして、このアカウントにシークレットをロードするアクセス許可を付与します。パラメータ名にスラッシュプレフィックスが含まれている場合に使用します。		
StepFunctionsExecutionPolicy	Step Functions ステートマシンの実行を開始する許可を付与します。		
TextractDetectAnalyzePolicy	Amazon Textract でドキュメントを検出して分析するためのアクセス権を付与します。		
TextractGetResultPolicy	検出および分析されたドキュメントを Amazon Textract から取得するためのアクセス権を付与します。		
TextractPolicy	Amazon Textract へのフルアクセス権を付与します。		
VPCAccessPolicy	Elastic Network Interface を作成、削除、記述、およびデタッチするアクセス権を付与します。		

トラブルシューティング

SAM CLI エラー：「ポリシーテンプレート '<policy-template-name>' に有効なパラメータ値を指定する必要があります」

`sam build` の実行時に、以下のエラーが表示されます。

```
"Must specify valid parameter values for policy template '<policy-template-name>'"
```

これは、プレースホルダ値がないポリシーテンプレートを宣言したときに、空のオブジェクトが渡されなかったことを意味します。

これを修正するには、以下の CloudWatchPutMetricPolicy 例のようにポリシーを宣言します。

```
MyFunction:
  Policies:
    - CloudWatchPutMetricPolicy: {}
```

AWS SAM ポリシーテンプレートリスト

以下は、使用可能なポリシーテンプレートと、それぞれに適用されるアクセス許可です。 AWS Serverless Application Model （ AWS SAM) は、プレースホルダー項目 (AWS リージョンやアカウント ID など) に適切な情報を自動的に入力します。

トピック

- AcmGetCertificatePolicy
- AMIDescribePolicy
- AthenaQueryPolicy
- AWSSecretsManagerGetSecretValuePolicy
- AWSSecretsManagerRotationPolicy
- CloudFormationDescribeStacksPolicy
- CloudWatchDashboardPolicy
- CloudWatchDescribeAlarmHistoryPolicy
- CloudWatchPutMetricPolicy
- CodePipelineLambdaExecutionPolicy

- CodePipelineReadOnlyPolicy
- CodeCommitCrudPolicy
- CodeCommitReadPolicy
- ComprehendBasicAccessPolicy
- CostExplorerReadOnlyPolicy
- DynamoDBBackupFullAccessPolicy
- DynamoDBCrudPolicy
- DynamoDBReadPolicy
- DynamoDBReconfigurePolicy
- DynamoDBRestoreFromBackupPolicy
- DynamoDBStreamReadPolicy
- DynamoDBWritePolicy
- EC2CopyImagePolicy
- EC2DescribePolicy
- EcsRunTaskPolicy
- EFSWriteAccessPolicy
- EKSDescribePolicy
- ElasticMapReduceAddJobFlowStepsPolicy
- ElasticMapReduceCancelStepsPolicy
- ElasticMapReduceModifyInstanceFleetPolicy
- ElasticMapReduceModifyInstanceGroupsPolicy
- ElasticMapReduceSetTerminationProtectionPolicy
- ElasticMapReduceTerminateJobFlowsPolicy
- ElasticsearchHttpPostPolicy
- EventBridgePutEventsPolicy
- FilterLogEventsPolicy
- FirehoseCrudPolicy
- FirehoseWritePolicy
- KinesisCrudPolicy
- KinesisStreamReadPolicy

- KMSDecryptPolicy
- KMSEncryptPolicy
- LambdaInvokePolicy
- MobileAnalyticsWriteOnlyAccessPolicy
- OrganizationsListAccountsPolicy
- PinpointEndpointAccessPolicy
- PollyFullAccessPolicy
- RekognitionDetectOnlyPolicy
- RekognitionFacesManagementPolicy
- RekognitionFacesPolicy
- RekognitionLabelsPolicy
- RekognitionNoDataAccessPolicy
- RekognitionReadPolicy
- RekognitionWriteOnlyAccessPolicy
- Route53ChangeResourceRecordSetsPolicy
- S3CrudPolicy
- S3FullAccessPolicy
- S3ReadPolicy
- S3WritePolicy
- SageMakerCreateEndpointConfigPolicy
- SageMakerCreateEndpointPolicy
- ServerlessRepoReadWriteAccessPolicy
- SESBulkTemplatedCrudPolicy
- SESBulkTemplatedCrudPolicy_v2
- SESCrudPolicy
- SESEmailTemplateCrudPolicy
- SESSendBouncePolicy
- SNSCrudPolicy
- SNSPublishMessagePolicy
- SQSPollerPolicy

- SQSSendMessagePolicy
- SSMParameterReadPolicy
- SSMParameterWithSlashPrefixReadPolicy
- StepFunctionsExecutionPolicy
- TextractDetectAnalyzePolicy
- TextractGetResultPolicy
- TextractPolicy
- VPCAccessPolicy

AcmGetCertificatePolicy

証明書を読み取るアクセス許可を付与します AWS Certificate Manager。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "acm:GetCertificate"
    ],
    "Resource": {
      "Fn::Sub": [
        "${certificateArn}",
        {
          "certificateArn": {
            "Ref": "CertificateArn"
          }
        }
      ]
    }
  }
]
```

AMIDescribePolicy

Amazon マシンイメージ () を記述するアクセス許可を付与しますAMIs。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "ec2:DescribeImages"
    ],
    "Resource": "*"
  }
]
```

AthenaQueryPolicy

Athena クエリを実行する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "athena:ListWorkGroups",
      "athena:GetExecutionEngine",
      "athena:GetExecutionEngines",
      "athena:GetNamespace",
      "athena:GetCatalogs",
      "athena:GetNamespaces",
      "athena:GetTables",
      "athena:GetTable"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "athena:StartQueryExecution",
      "athena:GetQueryResults",
      "athena:DeleteNamedQuery",
      "athena:GetNamedQuery",
      "athena:ListQueryExecutions",
      "athena:StopQueryExecution",
      "athena:GetQueryResultsStream",
      "athena:ListNamedQueries",
      "athena:CreateNamedQuery",
      "athena:GetQueryExecution",
      "athena:BatchGetNamedQuery",
      "athena:BatchGetQueryExecution",
      "athena:GetWorkGroup"
    ],
    "Resource": {
```

```
      "Fn::Sub": [
        "arn:${AWS::Partition}:athena:${AWS::Region}:${AWS::AccountId}:workgroup/
${workgroupName}",
        {
          "workgroupName": {
            "Ref": "WorkGroupName"
          }
        }
      ]
    }
  }
]
```

AWSSecretsManagerGetSecretValuePolicy

指定されたシークレットの AWS Secrets Manager シークレット値を取得するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetSecretValue"
    ],
    "Resource": {
      "Fn::Sub": [
        "${secretArn}",
        {
          "secretArn": {
            "Ref": "SecretArn"
          }
        }
      ]
    }
  }
]
```

AWSSecretsManagerRotationPolicy

AWS Secrets Managerのシークレットをローテーションを行う許可を付与します。

```
"Statement": [
  {
```

```
    "Effect": "Allow",
    "Action": [
      "secretsmanager:DescribeSecret",
      "secretsmanager:GetSecretValue",
      "secretsmanager:PutSecretValue",
      "secretsmanager:UpdateSecretVersionStage"
    ],
    "Resource": {
      "Fn::Sub": "arn:${AWS::Partition}:secretsmanager:${AWS::Region}:
${AWS::AccountId}:secret:*"
    },
    "Condition": {
      "StringEquals": {
        "secretsmanager:resource/AllowRotationLambdaArn": {
          "Fn::Sub": [
            "arn:${AWS::Partition}:lambda:${AWS::Region}:${AWS::AccountId}:function:
${functionName}",
            {
              "functionName": {
                "Ref": "FunctionName"
              }
            }
          ]
        }
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetRandomPassword"
    ],
    "Resource": "*"
  }
]
```

CloudFormationDescribeStacksPolicy

AWS CloudFormation スタックを記述するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
```

```
      "cloudformation:DescribeStacks"
    ],
    "Resource": {
      "Fn::Sub": "arn:${AWS::Partition}:cloudformation:${AWS::Region}:
${AWS::AccountId}:stack/*"
    }
  }
]
```

CloudWatchDashboardPolicy

CloudWatch ダッシュボードを操作するメトリクスを配置するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:GetDashboard",
      "cloudwatch:ListDashboards",
      "cloudwatch:PutDashboard",
      "cloudwatch:ListMetrics"
    ],
    "Resource": "*"
  }
]
```

CloudWatchDescribeAlarmHistoryPolicy

Amazon CloudWatch アラーム履歴を記述するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:DescribeAlarmHistory"
    ],
    "Resource": "*"
  }
]
```

CloudWatchPutMetricPolicy

メトリクスを送信するアクセス許可を付与します CloudWatch。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:PutMetricData"
    ],
    "Resource": "*"
  }
]
```

CodePipelineLambdaExecutionPolicy

によって呼び出された Lambda 関数 AWS CodePipeline に、ジョブのステータスを報告するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codepipeline:PutJobSuccessResult",
      "codepipeline:PutJobFailureResult"
    ],
    "Resource": "*"
  }
]
```

CodePipelineReadOnlyPolicy

CodePipeline パイプラインの詳細を取得する読み取りアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codepipeline:ListPipelineExecutions"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:codepipeline:${AWS::Region}:${AWS::AccountId}:
${pipelinename}",
        {
          "pipelinename": {
```

```
            "Ref": "PipelineName"
          }
        }
      ]
    }
  }
]
```

CodeCommitCrudPolicy

特定の CodeCommitリポジトリ内のオブジェクトを作成、読み取り、更新、削除するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codecommit:GitPull",
      "codecommit:GitPush",
      "codecommit:CreateBranch",
      "codecommit:DeleteBranch",
      "codecommit:GetBranch",
      "codecommit:ListBranches",
      "codecommit:MergeBranchesByFastForward",
      "codecommit:MergeBranchesBySquash",
      "codecommit:MergeBranchesByThreeWay",
      "codecommit:UpdateDefaultBranch",
      "codecommit:BatchDescribeMergeConflicts",
      "codecommit:CreateUnreferencedMergeCommit",
      "codecommit:DescribeMergeConflicts",
      "codecommit:GetMergeCommit",
      "codecommit:GetMergeOptions",
      "codecommit:BatchGetPullRequests",
      "codecommit:CreatePullRequest",
      "codecommit:DescribePullRequestEvents",
      "codecommit:GetCommentsForPullRequest",
      "codecommit:GetCommitsFromMergeBase",
      "codecommit:GetMergeConflicts",
      "codecommit:GetPullRequest",
      "codecommit:ListPullRequests",
      "codecommit:MergePullRequestByFastForward",
      "codecommit:MergePullRequestBySquash",
      "codecommit:MergePullRequestByThreeWay",
```

```
      "codecommit:PostCommentForPullRequest",
      "codecommit:UpdatePullRequestDescription",
      "codecommit:UpdatePullRequestStatus",
      "codecommit:UpdatePullRequestTitle",
      "codecommit:DeleteFile",
      "codecommit:GetBlob",
      "codecommit:GetFile",
      "codecommit:GetFolder",
      "codecommit:PutFile",
      "codecommit:DeleteCommentContent",
      "codecommit:GetComment",
      "codecommit:GetCommentsForComparedCommit",
      "codecommit:PostCommentForComparedCommit",
      "codecommit:PostCommentReply",
      "codecommit:UpdateComment",
      "codecommit:BatchGetCommits",
      "codecommit:CreateCommit",
      "codecommit:GetCommit",
      "codecommit:GetCommitHistory",
      "codecommit:GetDifferences",
      "codecommit:GetObjectIdentifier",
      "codecommit:GetReferences",
      "codecommit:GetTree",
      "codecommit:GetRepository",
      "codecommit:UpdateRepositoryDescription",
      "codecommit:ListTagsForResource",
      "codecommit:TagResource",
      "codecommit:UntagResource",
      "codecommit:GetRepositoryTriggers",
      "codecommit:PutRepositoryTriggers",
      "codecommit:TestRepositoryTriggers",
      "codecommit:GetBranch",
      "codecommit:GetCommit",
      "codecommit:UploadArchive",
      "codecommit:GetUploadArchiveStatus",
      "codecommit:CancelUploadArchive"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:codecommit:${AWS::Region}:${AWS::AccountId}:
${repositoryName}",
        {
          "repositoryName": {
            "Ref": "RepositoryName"
```

```
          }
        }
      ]
    }
  }
]
```

CodeCommitReadPolicy

特定の CodeCommit リポジトリ内のオブジェクトを読み取るアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codecommit:GitPull",
      "codecommit:GetBranch",
      "codecommit:ListBranches",
      "codecommit:BatchDescribeMergeConflicts",
      "codecommit:DescribeMergeConflicts",
      "codecommit:GetMergeCommit",
      "codecommit:GetMergeOptions",
      "codecommit:BatchGetPullRequests",
      "codecommit:DescribePullRequestEvents",
      "codecommit:GetCommentsForPullRequest",
      "codecommit:GetCommitsFromMergeBase",
      "codecommit:GetMergeConflicts",
      "codecommit:GetPullRequest",
      "codecommit:ListPullRequests",
      "codecommit:GetBlob",
      "codecommit:GetFile",
      "codecommit:GetFolder",
      "codecommit:GetComment",
      "codecommit:GetCommentsForComparedCommit",
      "codecommit:BatchGetCommits",
      "codecommit:GetCommit",
      "codecommit:GetCommitHistory",
      "codecommit:GetDifferences",
      "codecommit:GetObjectIdentifier",
      "codecommit:GetReferences",
      "codecommit:GetTree",
      "codecommit:GetRepository",
      "codecommit:ListTagsForResource",
      "codecommit:GetRepositoryTriggers",
```

```
      "codecommit:TestRepositoryTriggers",
      "codecommit:GetBranch",
      "codecommit:GetCommit",
      "codecommit:GetUploadArchiveStatus"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:codecommit:${AWS::Region}:${AWS::AccountId}:
${repositoryName}",
        {
          "repositoryName": {
            "Ref": "RepositoryName"
          }
        }
      ]
    }
  }
]
```

ComprehendBasicAccessPolicy

エンティティ、キーフレーズ、言語、およびセンチメントを検出する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "comprehend:BatchDetectKeyPhrases",
      "comprehend:DetectDominantLanguage",
      "comprehend:DetectEntities",
      "comprehend:BatchDetectEntities",
      "comprehend:DetectKeyPhrases",
      "comprehend:DetectSentiment",
      "comprehend:BatchDetectDominantLanguage",
      "comprehend:BatchDetectSentiment"
    ],
    "Resource": "*"
  }
]
```

CostExplorerReadOnlyPolicy

請求履歴APIsの読み取り専用 AWS Cost Explorer (Cost Explorer) に読み取り専用アクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ce:GetCostAndUsage",
      "ce:GetDimensionValues",
      "ce:GetReservationCoverage",
      "ce:GetReservationPurchaseRecommendation",
      "ce:GetReservationUtilization",
      "ce:GetTags"
    ],
    "Resource": "*"
  }
]
```

DynamoDBBackupFullAccessPolicy

テーブルの DynamoDB オンデマンドバックアップに読み取りおよび書き込み許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:CreateBackup",
      "dynamodb:DescribeContinuousBackups"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  },
```

```
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:DeleteBackup",
      "dynamodb:DescribeBackup",
      "dynamodb:ListBackups"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/backup/*",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  }
]
```

DynamoDBCrudPolicy

Amazon DynamoDB テーブルに作成、読み取り、更新、および削除許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:DeleteItem",
      "dynamodb:PutItem",
      "dynamodb:Scan",
      "dynamodb:Query",
      "dynamodb:UpdateItem",
      "dynamodb:BatchWriteItem",
      "dynamodb:BatchGetItem",
      "dynamodb:DescribeTable",
      "dynamodb:ConditionCheckItem"
    ],
    "Resource": [
      {
        "Fn::Sub": [
```

```
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
]
```

DynamoDBReadPolicy

DynamoDB テーブルに読み取り専用許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:Scan",
      "dynamodb:Query",
      "dynamodb:BatchGetItem",
      "dynamodb:DescribeTable"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
          {
```

```
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
]
```

DynamoDBReconfigurePolicy

DynamoDB テーブルを再構成する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:UpdateTable"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  }
```

```
]
```

DynamoDBRestoreFromBackupPolicy

バックアップから DynamoDB テーブルを復元する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:RestoreTableFromBackup"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/backup/*",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:PutItem",
      "dynamodb:UpdateItem",
      "dynamodb:DeleteItem",
      "dynamodb:GetItem",
      "dynamodb:Query",
      "dynamodb:Scan",
      "dynamodb:BatchWriteItem"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
```

```
      ]
    }
  }
]
```

DynamoDBStreamReadPolicy

DynamoDB のストリームとレコードを記述および読み取る許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:DescribeStream",
      "dynamodb:GetRecords",
      "dynamodb:GetShardIterator"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/stream/${streamName}",
        {
          "tableName": {
            "Ref": "TableName"
          },
          "streamName": {
            "Ref": "StreamName"
          }
        }
      ]
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:ListStreams"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/stream/*",
        {
          "tableName": {
            "Ref": "TableName"
```

```
          }
        }
      ]
    }
  }
]
```

DynamoDBWritePolicy

DynamoDB テーブルに書き込み専用許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:PutItem",
      "dynamodb:UpdateItem",
      "dynamodb:BatchWriteItem"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
```

```
]
```

EC2CopyImagePolicy

Amazon EC2イメージをコピーするアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ec2:CopyImage"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ec2:${AWS::Region}:${AWS::AccountId}:image/${imageId}",
        {
          "imageId": {
            "Ref": "ImageId"
          }
        }
      ]
    }
  }
]
```

EC2DescribePolicy

Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを記述するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ec2:DescribeRegions",
      "ec2:DescribeInstances"
    ],
    "Resource": "*"
  }
]
```

EcsRunTaskPolicy

タスク定義の新しいタスクを開始する許可を付与します。

```
"Statement": [
  {
    "Action": [
      "ecs:RunTask"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ecs:${AWS::Region}:${AWS::AccountId}:task-definition/
${taskDefinition}",
        {
          "taskDefinition": {
            "Ref": "TaskDefinition"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

EFSWriteAccessPolicy

書き込みアクセスを使用して Amazon EFS ファイルシステムをマウントするアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "elasticfilesystem:ClientMount",
      "elasticfilesystem:ClientWrite"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticfilesystem:${AWS::Region}:${AWS::AccountId}:file-
system/${FileSystem}",
        {
          "FileSystem": {
            "Ref": "FileSystem"
```

```
          }
        }
      ]
    },
    "Condition": {
      "StringEquals": {
        "elasticfilesystem:AccessPointArn": {
          "Fn::Sub": [
            "arn:${AWS::Partition}:elasticfilesystem:${AWS::Region}:
${AWS::AccountId}:access-point/${AccessPoint}",
            {
              "AccessPoint": {
                "Ref": "AccessPoint"
              }
            }
          ]
        }
      }
    }
  }
]
```

EKSDescribePolicy

Amazon Elastic Kubernetes Service (Amazon EKS) クラスターを記述または一覧表示するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "eks:DescribeCluster",
      "eks:ListClusters"
    ],
    "Resource": "*"
  }
]
```

ElasticMapReduceAddJobFlowStepsPolicy

実行中のクラスターに新しいステップを追加する許可を付与します。

```
"Statement": [
```

```
  {
    "Action": "elasticmapreduce:AddJobFlowSteps",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceCancelStepsPolicy

実行中のクラスターで保留中のステップ (1 つ、または複数) をキャンセルする許可を付与します。

```
"Statement": [
  {
    "Action": "elasticmapreduce:CancelSteps",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceModifyInstanceFleetPolicy

クラスター内のインスタンスフリートの詳細をリスト化し、容量を変更する許可を付与します。

```
"Statement": [
  {
    "Action": [
      "elasticmapreduce:ModifyInstanceFleet",
      "elasticmapreduce:ListInstanceFleets"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceModifyInstanceGroupsPolicy

クラスター内のインスタンスグループの詳細をリスト化し、設定を変更する許可を付与します。

```
"Statement": [
  {
    "Action": [
      "elasticmapreduce:ModifyInstanceGroups",
      "elasticmapreduce:ListInstanceGroups"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
```

```
  }
]
```

ElasticMapReduceSetTerminationProtectionPolicy

クラスターの終了保護を設定する許可を付与します。

```
"Statement": [
  {
    "Action": "elasticmapreduce:SetTerminationProtection",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceTerminateJobFlowsPolicy

クラスターをシャットダウンする許可を付与します。

```
"Statement": [
  {
    "Action": "elasticmapreduce:TerminateJobFlows",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
```

```
    },
    "Effect": "Allow"
  }
]
```

ElasticsearchHttpPostPolicy

Amazon OpenSearch Service に POSTおよび アクセスPUT許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "es:ESHttpPost",
      "es:ESHttpPut"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:es:${AWS::Region}:${AWS::AccountId}:domain/
${domainName}/*",
        {
          "domainName": {
            "Ref": "DomainName"
          }
        }
      ]
    }
  }
]
```

EventBridgePutEventsPolicy

Amazon にイベントを送信するアクセス許可を付与します EventBridge。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": "events:PutEvents",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:events:${AWS::Region}:${AWS::AccountId}:event-bus/
${eventBusName}",
        {
```

```
          "eventBusName": {
            "Ref": "EventBusName"
          }
        }
      ]
    }
  }
]
```

FilterLogEventsPolicy

指定された CloudWatch ロググループからログイベントをフィルタリングするアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "logs:FilterLogEvents"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-group:
${logGroupName}:log-stream:*",
        {
          "logGroupName": {
            "Ref": "LogGroupName"
          }
        }
      ]
    }
  }
]
```

FirehoseCrudPolicy

Firehose の配信ストリームに対し作成、書き込み、更新、および削除を行うアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "firehose:CreateDeliveryStream",
      "firehose:DeleteDeliveryStream",
      "firehose:DescribeDeliveryStream",
      "firehose:PutRecord",
      "firehose:PutRecordBatch",
      "firehose:UpdateDestination"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:firehose:${AWS::Region}:
${AWS::AccountId}:deliverystream/${deliveryStreamName}",
        {
          "deliveryStreamName": {
            "Ref": "DeliveryStreamName"
          }
        }
      ]
    }
  }
]
```

FirehoseWritePolicy

Firehose の配信ストリームに書き込むアクセス許可を付与します

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "firehose:PutRecord",
      "firehose:PutRecordBatch"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:firehose:${AWS::Region}:
${AWS::AccountId}:deliverystream/${deliveryStreamName}",
        {
          "deliveryStreamName": {
            "Ref": "DeliveryStreamName"
          }
        }
      ]
    }
```

```
  }
]
```

KinesisCrudPolicy

Amazon Kinesis のストリームを作成、発行、および削除する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "kinesis:AddTagsToStream",
      "kinesis:CreateStream",
      "kinesis:DecreaseStreamRetentionPeriod",
      "kinesis:DeleteStream",
      "kinesis:DescribeStream",
      "kinesis:DescribeStreamSummary",
      "kinesis:GetShardIterator",
      "kinesis:IncreaseStreamRetentionPeriod",
      "kinesis:ListTagsForStream",
      "kinesis:MergeShards",
      "kinesis:PutRecord",
      "kinesis:PutRecords",
      "kinesis:SplitShard",
      "kinesis:RemoveTagsFromStream"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kinesis:${AWS::Region}:${AWS::AccountId}:stream/
${streamName}",
        {
          "streamName": {
            "Ref": "StreamName"
          }
        }
      ]
    }
  }
]
```

KinesisStreamReadPolicy

Amazon Kinesis のストリームをリスト化して読み取る許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "kinesis:ListStreams",
      "kinesis:DescribeLimits"
    ],
    "Resource": {
      "Fn::Sub": "arn:${AWS::Partition}:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/*"
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "kinesis:DescribeStream",
      "kinesis:DescribeStreamSummary",
      "kinesis:GetRecords",
      "kinesis:GetShardIterator"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kinesis:${AWS::Region}:${AWS::AccountId}:stream/
${streamName}",
        {
          "streamName": {
            "Ref": "StreamName"
          }
        }
      ]
    }
  }
]
```

KMSDecryptPolicy

AWS Key Management Service（ AWS KMS) キーで復号するアクセス許可を付与します。は AWS KMS キーエイリアスではなく、キー ID `keyId`である必要があります。

```
"Statement": [
  {
    "Action": "kms:Decrypt",
    "Effect": "Allow",
```

```
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kms:${AWS::Region}:${AWS::AccountId}:key/${keyId}",
        {
          "keyId": {
            "Ref": "KeyId"
          }
        }
      ]
    }
  }
]
```

KMSEncryptPolicy

AWS KMS キーで暗号化するアクセス許可を付与します。は AWS KMS キーエイリアスではなく、キー ID keyId である必要があります。

```
"Statement": [
  {
    "Action": "kms:Encrypt",
    "Effect": "Allow",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kms:${AWS::Region}:${AWS::AccountId}:key/${keyId}",
        {
          "keyId": {
            "Ref": "KeyId"
          }
        }
      ]
    }
  }
]
```

LambdaInvokePolicy

AWS Lambda 関数、エイリアス、またはバージョンを呼び出すアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "lambda:InvokeFunction"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:lambda:${AWS::Region}:${AWS::AccountId}:function:
${functionName}*",
        {
          "functionName": {
            "Ref": "FunctionName"
          }
        }
      ]
    }
  }
]
```

MobileAnalyticsWriteOnlyAccessPolicy

すべてのアプリケーションリソースのイベントデータを配置するための書き込み専用許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "mobileanalytics:PutEvents"
    ],
    "Resource": "*"
  }
]
```

OrganizationsListAccountsPolicy

子アカウント名 と を一覧表示する読み取り専用アクセス許可を付与しますIDs。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "organizations:ListAccounts"
    ],
```

```
    "Resource": "*"
  }
]
```

PinpointEndpointAccessPolicy

Amazon Pinpoint アプリケーションのエンドポイントを取得して更新する許可を付与します

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "mobiletargeting:GetEndpoint",
      "mobiletargeting:UpdateEndpoint",
      "mobiletargeting:UpdateEndpointsBatch"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:mobiletargeting:${AWS::Region}:${AWS::AccountId}:apps/
${pinpointApplicationId}/endpoints/*",
        {
          "pinpointApplicationId": {
            "Ref": "PinpointApplicationId"
          }
        }
      ]
    }
  }
]
```

PollyFullAccessPolicy

Amazon Polly のレキシコンリソースへのフルアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "polly:GetLexicon",
      "polly:DeleteLexicon"
    ],
    "Resource": [
      {
```

```
        "Fn::Sub": [
          "arn:${AWS::Partition}:polly:${AWS::Region}:${AWS::AccountId}:lexicon/
${lexiconName}",
          {
            "lexiconName": {
              "Ref": "LexiconName"
            }
          }
        ]
      }
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "polly:DescribeVoices",
      "polly:ListLexicons",
      "polly:PutLexicon",
      "polly:SynthesizeSpeech"
    ],
    "Resource": [
      {
        "Fn::Sub": "arn:${AWS::Partition}:polly:${AWS::Region}:
${AWS::AccountId}:lexicon/*"
      }
    ]
  }
]
```

RekognitionDetectOnlyPolicy

顔、ラベル、およびテキストを検出する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:DetectFaces",
      "rekognition:DetectLabels",
      "rekognition:DetectModerationLabels",
      "rekognition:DetectText"
    ],
    "Resource": "*"
  }
```

```
]
```

RekognitionFacesManagementPolicy

Amazon Rekognition コレクション内の顔を追加、削除、および検索する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:IndexFaces",
      "rekognition:DeleteFaces",
      "rekognition:SearchFaces",
      "rekognition:SearchFacesByImage",
      "rekognition:ListFaces"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
]
```

RekognitionFacesPolicy

顔とラベルを比較および検出する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:CompareFaces",
      "rekognition:DetectFaces"
    ],
    "Resource": "*"
  }
```

```
]
```

RekognitionLabelsPolicy

オブジェクトとモデレーションラベルを検出する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:DetectLabels",
      "rekognition:DetectModerationLabels"
    ],
    "Resource": "*"
  }
]
```

RekognitionNoDataAccessPolicy

顔とラベルを比較および検出する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:CompareFaces",
      "rekognition:DetectFaces",
      "rekognition:DetectLabels",
      "rekognition:DetectModerationLabels"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
]
```

RekognitionReadPolicy

顔をリスト化して検索する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:ListCollections",
      "rekognition:ListFaces",
      "rekognition:SearchFaces",
      "rekognition:SearchFacesByImage"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
]
```

RekognitionWriteOnlyAccessPolicy

顔のコレクションを作成してインデックス化する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:CreateCollection",
      "rekognition:IndexFaces"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
```

```
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
]
```

Route53ChangeResourceRecordSetsPolicy

Route 53 のリソースレコードセットを変更する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "route53:ChangeResourceRecordSets"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:route53:::hostedzone/${HostedZoneId}",
        {
          "HostedZoneId": {
            "Ref": "HostedZoneId"
          }
        }
      ]
    }
  }
]
```

S3CrudPolicy

Amazon S3 バケット内のオブジェクトでアクションを実行するための作成、読み取り、更新、および削除許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket",
```

```
      "s3:GetBucketLocation",
      "s3:GetObjectVersion",
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:GetLifecycleConfiguration",
      "s3:PutLifecycleConfiguration",
      "s3:DeleteObject"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  }
]
```

S3FullAccessPolicy

Amazon S3 バケット内のオブジェクトでアクションを実行するためのフルアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
```

```
      "s3:GetObjectAcl",
      "s3:GetObjectVersion",
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:DeleteObject",
      "s3:DeleteObjectTagging",
      "s3:DeleteObjectVersionTagging",
      "s3:GetObjectTagging",
      "s3:GetObjectVersionTagging",
      "s3:PutObjectTagging",
      "s3:PutObjectVersionTagging"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:ListBucket",
      "s3:GetBucketLocation",
      "s3:GetLifecycleConfiguration",
      "s3:PutLifecycleConfiguration"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
```

```
    ]
  }
]
```

S3ReadPolicy

Amazon Simple Storage Service (Amazon S3) バケットにあるオブジェクトを読み取るための読み取り専用許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket",
      "s3:GetBucketLocation",
      "s3:GetObjectVersion",
      "s3:GetLifecycleConfiguration"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  }
]
```

S3WritePolicy

Amazon S3 バケットにオブジェクトを書き込むための書き込み許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:PutLifecycleConfiguration"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  }
]
```

SageMakerCreateEndpointConfigPolicy

SageMaker AI でエンドポイント設定を作成するアクセス許可を付与します。

```
"Statement": [
  {
```

```
    "Action": [
      "sagemaker:CreateEndpointConfig"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint-
config/${endpointConfigName}",
        {
          "endpointConfigName": {
            "Ref": "EndpointConfigName"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

SageMakerCreateEndpointPolicy

SageMaker AI でエンドポイントを作成するアクセス許可を付与します。

```
"Statement": [
  {
    "Action": [
      "sagemaker:CreateEndpoint"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint/
${endpointName}",
        {
          "endpointName": {
            "Ref": "EndpointName"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ServerlessRepoReadWriteAccessPolicy

AWS Serverless Application Repository（ AWS SAM) サービスでアプリケーションを作成および一覧表示するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "serverlessrepo:CreateApplication",
      "serverlessrepo:CreateApplicationVersion",
      "serverlessrepo:GetApplication",
      "serverlessrepo:ListApplications",
      "serverlessrepo:ListApplicationVersions"
    ],
    "Resource": [
      {
        "Fn::Sub": "arn:${AWS::Partition}:serverlessrepo:${AWS::Region}:
${AWS::AccountId}:applications/*"
      }
    ]
  }
]
```

SESBulkTemplatedCrudPolicy

Amazon E SESメール、テンプレート化された E メール、およびテンプレート化された一括 E メールを送信し、ID を検証するアクセス許可を付与します。

> **Note**
>
> `ses:SendTemplatedEmail` アクションにはテンプレート が必要ですARN。代わりに `SESBulkTemplatedCrudPolicy_v2` を使用します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ses:GetIdentityVerificationAttributes",
      "ses:SendEmail",
      "ses:SendRawEmail",
```

```
      "ses:SendTemplatedEmail",
      "ses:SendBulkTemplatedEmail",
      "ses:VerifyEmailIdentity"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
        {
          "identityName": {
            "Ref": "IdentityName"
          }
        }
      ]
    }
  }
]
```

SESBulkTemplatedCrudPolicy_v2

Amazon E SESメール、テンプレート化された E メール、およびテンプレート化された一括 E メールを送信し、ID を検証するアクセス許可を付与します。

```
"Statement": [
  {
    "Action": [
      "ses:SendEmail",
      "ses:SendRawEmail",
      "ses:SendTemplatedEmail",
      "ses:SendBulkTemplatedEmail"
    ],
    "Effect": "Allow",
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
          {
            "identityName": {
              "Ref": "IdentityName"
            }
          }
        ]
```

```
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:template/
${templateName}",
          {
            "templateName": {
              "Ref": "TemplateName"
            }
          }
        ]
      }
    ]
  },
  {
    "Action": [
      "ses:GetIdentityVerificationAttributes",
      "ses:VerifyEmailIdentity"
    ],
    "Effect": "Allow",
    "Resource": "*"
  }
]
```

SESCrudPolicy

E メールを送信し、アイデンティティを確認する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ses:GetIdentityVerificationAttributes",
      "ses:SendEmail",
      "ses:SendRawEmail",
      "ses:VerifyEmailIdentity"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
        {
          "identityName": {
            "Ref": "IdentityName"
```

```
          }
        }
      ]
    }
  }
]
```

SESEmailTemplateCrudPolicy

Amazon E SESメールテンプレートを作成、取得、一覧表示、更新、削除するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ses:CreateTemplate",
      "ses:GetTemplate",
      "ses:ListTemplates",
      "ses:UpdateTemplate",
      "ses:DeleteTemplate",
      "ses:TestRenderTemplate"
    ],
    "Resource": "*"
  }
]
```

SESSendBouncePolicy

Amazon Simple Email Service (Amazon SES) ID にアクセス SendBounce 許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ses:SendBounce"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
        {
          "identityName": {
```

```
            "Ref": "IdentityName"
          }
        }
      ]
    }
  }
]
```

SNSCrudPolicy

Amazon SNSトピックを作成、公開、サブスクライブするアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sns:ListSubscriptionsByTopic",
      "sns:CreateTopic",
      "sns:SetTopicAttributes",
      "sns:Subscribe",
      "sns:Publish"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sns:${AWS::Region}:${AWS::AccountId}:${topicName}*",
        {
          "topicName": {
            "Ref": "TopicName"
          }
        }
      ]
    }
  }
]
```

SNSPublishMessagePolicy

Amazon Simple Notification Service (Amazon SNS) トピックにメッセージを発行するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "sns:Publish"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sns:${AWS::Region}:${AWS::AccountId}:${topicName}",
        {
          "topicName": {
            "Ref": "TopicName"
          }
        }
      ]
    }
  }
]
```

SQSPollerPolicy

Amazon Simple Queue Service (Amazon SQS) キューをポーリングするアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sqs:ChangeMessageVisibility",
      "sqs:ChangeMessageVisibilityBatch",
      "sqs:DeleteMessage",
      "sqs:DeleteMessageBatch",
      "sqs:GetQueueAttributes",
      "sqs:ReceiveMessage"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:${queueName}",
        {
          "queueName": {
            "Ref": "QueueName"
          }
        }
      ]
    }
  }
```

```
]
```

SQSSendMessagePolicy

Amazon SQSキューにメッセージを送信するアクセス許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sqs:SendMessage*"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:${queueName}",
        {
          "queueName": {
            "Ref": "QueueName"
          }
        }
      ]
    }
  }
]
```

SSMParameterReadPolicy

Amazon EC2 Systems Manager (SSM) パラメータストアからパラメータにアクセスして、このアカウントにシークレットをロードするアクセス許可を付与します。パラメータ名にスラッシュプレフィックスが含まれていない場合に使用します。

> **Note**
>
> デフォルトのキーを使用していない場合は、KMSDecryptPolicy ポリシーも必要になります。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ssm:DescribeParameters"
```

```
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "ssm:GetParameters",
      "ssm:GetParameter",
      "ssm:GetParametersByPath"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ssm:${AWS::Region}:${AWS::AccountId}:parameter/
${parameterName}",
        {
          "parameterName": {
            "Ref": "ParameterName"
          }
        }
      ]
    }
  }
]
```

SSMParameterWithSlashPrefixReadPolicy

Amazon EC2 Systems Manager (SSM) パラメータストアからパラメータにアクセスして、このアカウントにシークレットをロードするアクセス許可を付与します。パラメータ名にスラッシュプレフィックスが含まれている場合に使用します。

> **Note**
>
> デフォルトのキーを使用していない場合は、KMSDecryptPolicy ポリシーも必要になります。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ssm:DescribeParameters"
    ],
```

```
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "ssm:GetParameters",
      "ssm:GetParameter",
      "ssm:GetParametersByPath"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ssm:${AWS::Region}:${AWS::AccountId}:parameter
${parameterName}",
        {
          "parameterName": {
            "Ref": "ParameterName"
          }
        }
      ]
    }
  }
]
```

StepFunctionsExecutionPolicy

Step Functions ステートマシンの実行を開始する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "states:StartExecution"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:stateMachine:
${stateMachineName}",
        {
          "stateMachineName": {
            "Ref": "StateMachineName"
          }
        }
      ]
    }
```

```
  }
]
```

TextractDetectAnalyzePolicy

Amazon Textract でドキュメントを検出して分析するためのアクセス権を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "textract:DetectDocumentText",
      "textract:StartDocumentTextDetection",
      "textract:StartDocumentAnalysis",
      "textract:AnalyzeDocument"
    ],
    "Resource": "*"
  }
]
```

TextractGetResultPolicy

検出および分析されたドキュメントを Amazon Textract から取得するためのアクセス権を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "textract:GetDocumentTextDetection",
      "textract:GetDocumentAnalysis"
    ],
    "Resource": "*"
  }
]
```

TextractPolicy

Amazon Textract へのフルアクセス権を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "textract:*"
    ],
    "Resource": "*"
  }
]
```

VPCAccessPolicy

Elastic Network Interface を作成、削除、記述、およびデタッチするアクセス権を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ec2:CreateNetworkInterface",
      "ec2:DeleteNetworkInterface",
      "ec2:DescribeNetworkInterfaces",
      "ec2:DetachNetworkInterface"
    ],
    "Resource": "*"
  }
]
```

AWS CloudFormation メカニズムを使用した AWS SAM のアクセス許可の管理

AWS Serverless Application Model (AWS SAM) は、AWS リソースへのアクセスを制御するために、AWS CloudFormation と同じメカニズムを使用します。詳細については、AWS CloudFormation ユーザーガイドの「Controlling access with AWS Identity and Access Management」を参照してください。

サーバーレスアプリケーションを管理するためのユーザー権限の付与には、3 つの主なオプションがあります。各オプションは、ユーザーに異なるレベルのアクセスコントロールを提供します。

- 管理者権限を付与する。
- 必要な AWS 管理ポリシーをアタッチする。
- 特定の AWS Identity and Access Management (IAM) 許可を付与する。

選択するオプションに応じて、ユーザーは、アクセス許可を持つ AWS リソースが含まれたサーバーレスアプリケーションのみを管理できます。

以下のセクションでは、各オプションがより詳しく説明されています。

管理者権限を付与する

ユーザーに管理者権限を付与すると、ユーザーは、任意の組み合わせの AWS リソースが含まれたサーバーレスアプリケーションを管理できます。これは最もシンプルなオプションですが、ユーザーに最も広範な許可のセットを付与するため、最も大きな影響を与えるアクションの実行が可能になります。

ユーザーへの管理者権限の付与に関する詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびユーザーグループの作成」を参照してください。

必要な AWS 管理ポリシーをアタッチする

ユーザーに完全な管理者権限を付与するのではなく、AWS 管理ポリシーを使用して権限のサブセットを付与することができます。このオプションを使用する場合は、ユーザーが管理するサーバーレスアプリケーションに必要なすべてのアクションとリソースが、AWS 管理ポリシーセットの対象範囲内であることを確認してください。

例えば、以下の AWS 管理ポリシーは、サンプル Hello World アプリケーションをデプロイするために十分なポリシーです。

- AWSCloudFormationFullAccess
- IAMFullAccess
- AWSLambda_FullAccess
- AmazonAPIGatewayAdministrator
- AmazonS3FullAccess
- AmazonEC2ContainerRegistryFullAccess

IAM ユーザーへのポリシーのアタッチに関する詳細については、IAM ユーザーガイドの「IAM ユーザーのアクセス権限の変更」を参照してください。

特定の IAM 許可を付与する

最もきめ細かなレベルのアクセスコントロールには、ポリシステートメントを使用して、ユーザーに特定の IAM 許可を付与することができます。このオプションを使用する場合は、ユーザーが管理するサーバーレスアプリケーションに必要なすべてのアクションとリソースがポリシーステートメントに含まれていることを確認してください。

このオプションのベストプラクティスは、ユーザーが昇格された許可を自分自身に付与できないように、Lambda 実行ロールを含めたロールを作成するための許可をユーザーに付与しないことです。このため、管理者は、ユーザーが管理するサーバーレスアプリケーションで指定される Lambda 実行ロールを最初に作成する必要があります。Lambda 実行ロールの作成については、「IAM コンソールでの実行ロールの作成」を参照してください。

サンプル Hello World アプリケーションの実行には、AWSLambdaBasicExecutionRole で十分です。Lambda 実行ロールを作成したら、サンプル Hello World アプリケーションの AWS SAM テンプレートファイルを変更して、以下のプロパティを `AWS::Serverless::Function` リソースに追加します。

```
  Role: lambda-execution-role-arn
```

変更された Hello World アプリケーションを使用すると、以下のポリシーステートメントが、アプリケーションをデプロイ、更新、および削除するために十分な許可をユーザーに付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudFormationTemplate",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateChangeSet"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:aws:transform/Serverless-2016-10-31"
            ]
        },
        {
            "Sid": "CloudFormationStack",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateChangeSet",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeChangeSet",
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeStacks",
                "cloudformation:ExecuteChangeSet",
                "cloudformation:GetTemplateSummary",
                "cloudformation:ListStackResources",
```

```
                "cloudformation:UpdateStack"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/*"
            ]
        },
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*/*"
            ]
        },
        {
            "Sid": "ECRRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreateRepository",
                "ecr:DeleteRepository",
                "ecr:DescribeImages",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:InitiateLayerUpload",
                "ecr:ListImages",
                "ecr:PutImage",
                "ecr:SetRepositoryPolicy",
                "ecr:UploadLayerPart"
            ],
            "Resource": [
                "arn:aws:ecr:*:111122223333:repository/*"
            ]
        },
        {
            "Sid": "ECRAuthToken",
            "Effect": "Allow",
```

```
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "Lambda",
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:ListTags",
                "lambda:RemovePermission",
                "lambda:TagResource",
                "lambda:UntagResource",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:*"
            ]
        },
        {
            "Sid": "IAM",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DeleteRole",
                "iam:DetachRolePolicy",
                "iam:GetRole",
                "iam:TagRole"
            ],
            "Resource": [
                "arn:aws:iam::111122223333:role/*"
            ]
        },
        {
            "Sid": "IAMPassRole",
```

```
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "lambda.amazonaws.com"
                }
            }
        },
        {
            "Sid": "APIGateway",
            "Effect": "Allow",
            "Action": [
                "apigateway:DELETE",
                "apigateway:GET",
                "apigateway:PATCH",
                "apigateway:POST",
                "apigateway:PUT"
            ],
            "Resource": [
                "arn:aws:apigateway:*::*"
            ]
        }
    ]
}
```

> **Note**
>
> このセクションのポリシーステートメントの例では、Hello World アプリケーションのサンプルをデプロイ、更新、および削除するための十分な権限を付与します。アプリケーションにリソースタイプを追加する場合は、ポリシーステートメントを更新して以下を含める必要があります。
>
> 1. アプリケーションがサービスのアクションを呼び出すための権限。
> 2. サービスのアクションに必要な場合は、サービスプリンシパル。
>
> 例えば、Step Functions ワークフローを追加する場合は、ここにリストされているアクションに対するアクセス許可と、`states.amazonaws.com` サービスプリンシパルを追加する必要があります。

IAM ポリシーの詳細については、IAM ユーザーガイドの「IAM ポリシーの管理」を参照してください。

AWS SAM テンプレートを使用して API アクセスを制御する

API Gateway API へのアクセスを制御すると、サーバーレスアプリケーションの安全性を担保し、有効化されている認証を通じてのみアクセスが行われることを保証できます。API Gateway API にアクセスできるユーザーを制御するには、AWS SAMテンプレートでの認可を有効にします。

AWS SAM は API Gateway API へのアクセスを制御する複数のメカニズムをサポートしています。サポートされているメカニズムのセットは、`AWS::Serverless::HttpApi` と `AWS::Serverless::Api` のリソースタイプ間で異なります。

以下の表は、各リソースタイプがサポートするメカニズムの要約です。

アクセスを制御するためのメカニズム	AWS::Serverless::HttpApi	AWS::Serverless::Api
Lambda オーソライザー	✓	✓
IAM アクセス許可		✓
Amazon Cognito ユーザープール	✓ *	✓
API キー		✓
リソースポリシー		✓
OAuth 2.0/JWT オーソライザー	✓	

* `AWS::Serverless::HttpApi` リソースタイプでは、JSON Web トークン (JWT) 発行者として Amazon Cognito を使用できます。

- Lambda オーソライザー - Lambda オーソライザー (これまで カスタムオーソライザー と呼ばれていたもの) は、API へのアクセスを制御するためにユーザーが提供する Lambda 関数です。API が呼び出されると、クライアントアプリケーションが提供するリクエストコンテキストまたは認可

トークンによって、この Lambda 関数が呼び出されます。Lambda 関数は、発信者がリクエストされたオペレーションの実行を許可されているかどうかについて応答します。

`AWS::Serverless::HttpApi` と `AWS::Serverless::Api` リソースタイプの両方が Lambda オーソライザーをサポートします。

`AWS::Serverless::HttpApi` での Lambda オーソライザーの詳細については、API Gateway デベロッパーガイドの「AWS Lambda オーソライザーを使用する」で HTTP API について参照してください。`AWS::Serverless::Api` での Lambda オーソライザーの詳細については、API Gateway デベロッパーガイドの「API Gateway Lambda オーソライザーを使用する」を参照してください。

どちらか一方のリソースタイプ向けの Lambda オーソライザーの例については、「AWS SAM 向け Lambda オーソライザーの例」を参照してください。

- IAM 許可 - AWS Identity and Access Management (IAM) 許可を使用して、API を呼び出すことができるユーザーを制御できます。API を呼び出すユーザーは、IAM 認証情報を使用して認証される必要があります。API の呼び出しは、API 発信者を表す IAM ユーザー、ユーザーが含まれる IAM グループ、またはユーザーが引き受ける IAM ロールに IAM ポリシーがアタッチされていなければ成功しません。

  IAM 許可をサポートするのは、`AWS::Serverless::Api` リソースタイプのみです。

  詳細については、API Gateway デベロッパーガイドの「IAM アクセス許可により API へのアクセスを制御する」を参照してください。例については、「AWS SAM 向け IAM 許可の例」を参照してください。
- Amazon Cognito ユーザープール - Amazon Cognito ユーザープールは、Amazon Cognito 内のユーザーディレクトリです。API のクライアントは、まずユーザーをユーザープールにサインインし、ユーザーのアイデンティティまたはアクセストークンを取得する必要があります。その後、返されたトークンの 1 つを使用して、クライアントが API を呼び出します。API コールは、必要なトークンが有効な場合にのみ成功します。

  `AWS::Serverless::Api` リソースタイプが Amazon Cognito ユーザープールをサポートします。`AWS::Serverless::HttpApi` リソースタイプは、JWT 発行者としての Amazon Cognito の使用をサポートします。

  詳細については、API Gateway デベロッパーガイドの「Amazon Cognito ユーザープールをオーソライザーとして使用して REST API へのアクセスを制御する」を参照してください 例については、「AWS SAM 向け Amazon Cognito ユーザープールの例」を参照してください。

- API キー - API キーは、API へのアクセス権を付与するためにアプリケーションデベロッパーのお客様に配信する、英数字の文字列値です。

  API キーをサポートするのは、`AWS::Serverless::Api` リソースタイプのみです。

  詳細については、API Gateway デベロッパーガイドの「API キーを使用した使用量プランの作成と使用」を参照してください。API キーの例については、「AWS SAM 向け API キーの例」を参照してください。

- リソースポリシー - リソースポリシーは、API Gateway API にアタッチできる JSON ポリシードキュメントです。リソースポリシーは、指定されたプリンシパル (通常は IAM のユーザーまたはロール) が API を呼び出せるかどうかを制御します。

  API Gateway API へのアクセスを制御するメカニズムとしてのリソースポリシーをサポートするのは、`AWS::Serverless::Api` リソースタイプのみです。

  リソースポリシーの詳細については、API Gateway デベロッパーガイドの「API Gateway リソースポリシーを使用して API へのアクセスを制御する」を参照してください。リソースポリシーの例については、「AWS SAM 向けリソースポリシーの例」を参照してください。

- OAuth 2.0/JWT オーソライザー - OpenID Connect (OIDC) および OAuth 2.0 フレームワークの一部として JWT を使用して、API へのアクセスを制御できます。API Gateway は、クライアントが API リクエストと共に送信する JWT を検証し、トークン検証、およびオプションでトークン内のスコープに基づいて、リクエストを許可または拒否します。

  OAuth 2.0/JWT オーソライザーをサポートするのは、`AWS::Serverless::HttpApi` リソースタイプのみです。

  詳細については、API Gateway デベロッパーガイドの「JWT オーソライザーを使用した HTTP API へのアクセスの制御」を参照してください。例については、「AWS SAM 向け OAuth 2.0/JWT オーソライザーの例」を参照してください。

アクセスを制御するためのメカニズムの選択

API Gateway API へのアクセスを制御するために選択するメカニズムは、いくつかの要因に応じて異なります。例えば、認可またはアクセスコントロールが設定されていないグリーンフィールドプロジェクトの場合は、Amazon Cognito ユーザープールが最適なオプションになり得ます。ユーザープールのセットアップ時には、認証とアクセスコントロールがどちらも自動的にセットアップされるからです。

ただし、アプリケーションで認証がすでにセットアップされている場合は、Lambda オーソライザーの使用が最適なオプションになり得ます。これは、既存の認証サービスを呼び出し、レスポンスに基づいてポリシードキュメントを返すことができるためです。また、アプリケーションに、ユーザープールがサポートしないカスタム認証またはアクセスコントロールロジックが必要な場合は、Lambda オーソライザーが最適なオプションになり得ます。

使用するメカニズムを選択したら、そのメカニズムを使用するために AWS SAM を使用してアプリケーションを設定する方法について、「例」の対応するセクションを参照してください。

エラーレスポンスのカスタマイズ

AWS SAM を使用して、一部の API Gateway エラーレスポンスの内容をカスタマイズできます。カスタマイズされた API Gateway レスポンスをサポートするのは、`AWS::Serverless::Api` リソースタイプのみです。

API Gateway レスポンスの詳細については、API Gateway デベロッパーガイドの「API Gateway でのゲートウェイレスポンス」を参照してください。カスタマイズされたレスポンスの例については、「AWS SAM のカスタマイズされたレスポンスの例」を参照してください。

例

- AWS SAM 向け Lambda オーソライザーの例
- AWS SAM 向け IAM 許可の例
- AWS SAM 向け Amazon Cognito ユーザープールの例
- AWS SAM 向け API キーの例
- AWS SAM 向けリソースポリシーの例
- AWS SAM 向け OAuth 2.0/JWT オーソライザーの例
- AWS SAM のカスタマイズされたレスポンスの例

AWS SAM 向け Lambda オーソライザーの例

`AWS::Serverless::Api` リソースタイプは、`TOKEN` オーソライザーと `REQUEST` オーソライザーの 2 つのタイプの Lambda オーソライザーをサポートします。`AWS::Serverless::HttpApi` リソースタイプは `REQUEST` オーソライザーのみをサポートします。以下は、各タイプの例です。

Lambda **TOKEN** オーソライザーの例 (AWS::Serverless::Api)

API へのアクセスは、AWS SAM テンプレート内で Lambda TOKEN オーソライザーを定義することによって制御できます。これを実行するには、ApiAuth データ型を使用します。

以下は、Lambda TOKEN オーソライザーの AWS SAM テンプレートセクションの例です。

> **Note**
>
> 次の例では、SAM FunctionRole が暗黙的に生成されます。

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Auth:
        DefaultAuthorizer: MyLambdaTokenAuthorizer
        Authorizers:
          MyLambdaTokenAuthorizer:
            FunctionArn: !GetAtt MyAuthFunction.Arn

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        GetRoot:
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: get

  MyAuthFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: authorizer.handler
```

```
      Runtime: nodejs12.x
```

Lambda オーソライザーの詳細については、API Gateway デベロッパーガイドの「API Gateway Lambda オーソライザーを使用する」を参照してください。

Lambda **REQUEST** オーソライザーの例 (AWS::Serverless::Api)

API へのアクセスは、AWS SAM テンプレート内で Lambda REQUEST オーソライザーを定義することによって制御できます。これを実行するには、ApiAuth データ型を使用します。

以下は、Lambda REQUEST オーソライザーの AWS SAM テンプレートセクションの例です。

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Auth:
        DefaultAuthorizer: MyLambdaRequestAuthorizer
        Authorizers:
          MyLambdaRequestAuthorizer:
            FunctionPayloadType: REQUEST
            FunctionArn: !GetAtt MyAuthFunction.Arn
            Identity:
              QueryStrings:
                - auth

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        GetRoot:
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: get

  MyAuthFunction:
    Type: AWS::Serverless::Function
```

```
    Properties:
      CodeUri: ./src
      Handler: authorizer.handler
      Runtime: nodejs12.x
```

Lambda オーソライザーの詳細については、API Gateway デベロッパーガイドの「API Gateway Lambda オーソライザーを使用する」を参照してください。

Lambda オーソライザーの例 (AWS::Serverless::HttpApi)

HTTP API へのアクセスは、AWS SAM テンプレート内で Lambda オーソライザーを定義することによって制御できます。これを実行するには、HttpApiAuth データ型を使用します。

以下は、Lambda オーソライザーの AWS SAM テンプレートセクションの例です。

```
Resources:
  MyApi:
    Type: AWS::Serverless::HttpApi
    Properties:
      StageName: Prod
      Auth:
        DefaultAuthorizer: MyLambdaRequestAuthorizer
        Authorizers:
          MyLambdaRequestAuthorizer:
            FunctionArn: !GetAtt MyAuthFunction.Arn
            FunctionInvokeRole: !GetAtt MyAuthFunctionRole.Arn
            Identity:
              Headers:
                - Authorization
            AuthorizerPayloadFormatVersion: 2.0
            EnableSimpleResponses: true

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        GetRoot:
          Type: HttpApi
          Properties:
            ApiId: !Ref MyApi
```

```
            Path: /
            Method: get
            PayloadFormatVersion: "2.0"

  MyAuthFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: authorizer.handler
      Runtime: nodejs12.x
```

AWS SAM 向け IAM 許可の例

API へのアクセスは、AWS SAM テンプレート内で IAM 許可を定義することによって制御できます。これを実行するには、ApiAuth データ型を使用します。

以下は、IAM での許可に使用する AWS SAM テンプレートの例です。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Description: 'API with IAM authorization'
      Auth:
        DefaultAuthorizer: AWS_IAM #sets AWS_IAM auth for all methods in this API
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: python3.10
      Events:
        GetRoot:
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: get
      InlineCode: |
        def handler(event, context):
          return {'body': 'Hello World!', 'statusCode': 200}
```

IAM 許可の詳細については、API Gateway デベロッパーガイドの「API を呼び出すためのアクセスの制御」を参照してください。

AWS SAM 向け Amazon Cognito ユーザープールの例

API へのアクセスは、AWS SAM テンプレート内で Amazon Cognito ユーザープールを定義することによって制御できます。これを実行するには、ApiAuth データ型を使用します。

以下は、ユーザープールの AWS SAM テンプレートセクションの例です。

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Cors: "'*'"
      Auth:
        DefaultAuthorizer: MyCognitoAuthorizer
        Authorizers:
          MyCognitoAuthorizer:
            UserPoolArn: !GetAtt MyCognitoUserPool.Arn

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: lambda.handler
      Runtime: nodejs12.x
      Events:
        Root:
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: GET

  MyCognitoUserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      UserPoolName: !Ref CognitoUserPoolName
      Policies:
        PasswordPolicy:
          MinimumLength: 8
```

```
      UsernameAttributes:
        - email
      Schema:
        - AttributeDataType: String
          Name: email
          Required: false

  MyCognitoUserPoolClient:
    Type: AWS::Cognito::UserPoolClient
    Properties:
      UserPoolId: !Ref MyCognitoUserPool
      ClientName: !Ref CognitoUserPoolClientName
      GenerateSecret: false
```

Amazon Cognito ユーザープールの詳細については、API Gateway デベロッパーガイドの「Amazon Cognito ユーザープールをオーソライザーとして使用して REST API へのアクセスを制御する」を参照してください

AWS SAM 向け API キーの例

API へのアクセスは、AWS SAM テンプレート内で API キーを必須とすることによって制御できます。これを実行するには、ApiAuth データ型を使用します。

以下は、API キーの AWS SAM テンプレートセクションの例です。

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Auth:
        ApiKeyRequired: true # sets for all methods

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        ApiKey:
          Type: Api
          Properties:
```

```
            RestApiId: !Ref MyApi
            Path: /
            Method: get
            Auth:
              ApiKeyRequired: true
```

詳細については、API Gateway デベロッパーガイドの「API キーを使用した使用量プランの作成と使用」を参照してください。

AWS SAM 向けリソースポリシーの例

API へのアクセスは、AWS SAM テンプレート内でリソースポリシーをアタッチすることによって制御できます。これを実行するには、ApiAuth データ型を使用します。

次は、プライベート API 向けの AWS SAM テンプレートの例です。プライベート API には、デプロイするリソースポリシーが必要です。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyPrivateApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      EndpointConfiguration: PRIVATE  # Creates a private API. Resource policies are
 required for all private APIs.
      Auth:
        ResourcePolicy:
          CustomStatements: {
              Effect: 'Allow',
              Action: 'execute-api:Invoke',
              Resource: ['execute-api:/*/*/*'],
              Principal: '*'
            }
  MyFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      InlineCode: |
        def handler(event, context):
          return {'body': 'Hello World!', 'statusCode': 200}
      Handler: index.handler
      Runtime: python3.10
      Events:
```

```
        AddItem:
          Type: Api
          Properties:
            RestApiId:
              Ref: MyPrivateApi
            Path: /
            Method: get
```

リソースポリシーの詳細については、API Gateway デベロッパーガイドの「API Gateway リソースポリシーを使用して API へのアクセスを制御する」を参照してください。プライベート API の詳細情報については、「API Gateway デベロッパーガイド」の「Amazon API Gateway でのプライベート API の作成」を参照してください。

AWS SAM 向け OAuth 2.0/JWT オーソライザーの例

API へのアクセスは、OpenID Connect (OIDC) および OAuth 2.0 フレームワークの一部として JWT を使用することによって制御できます。これを実行するには、HttpApiAuth データ型を使用します。

以下は、OAuth 2.0/JWT オーソライザーの AWS SAM テンプレートセクションの例です。

```
Resources:
  MyApi:
    Type: AWS::Serverless::HttpApi
    Properties:
      Auth:
        Authorizers:
          MyOauth2Authorizer:
            AuthorizationScopes:
              - scope
            IdentitySource: $request.header.Authorization
            JwtConfiguration:
              audience:
                - audience1
                - audience2
              issuer: "https://www.example.com/v1/connect/oidc"
        DefaultAuthorizer: MyOauth2Authorizer
      StageName: Prod
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Events:
```

```
        GetRoot:
          Properties:
            ApiId: MyApi
            Method: get
            Path: /
            PayloadFormatVersion: "2.0"
          Type: HttpApi
      Handler: index.handler
      Runtime: nodejs12.x
```

OAuth 2.0/JWT オーソライザーの詳細については、API Gateway デベロッパーガイドの「JWT オーソライザーを使用した HTTP API へのアクセスの制御」を参照してください。

AWS SAM のカスタマイズされたレスポンスの例

一部の API Gateway エラーレスポンスは、AWS SAM テンプレート内でレスポンスヘッダーを定義することによってカスタマイズできます。これを実行するには、Gateway Response Object データ型を使用します。

以下は、DEFAULT_5XX エラーに対するカスタマイズされたレスポンスを作成する AWS SAM テンプレートの例です。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      GatewayResponses:
        DEFAULT_5XX:
          ResponseParameters:
            Headers:
              Access-Control-Expose-Headers: "'WWW-Authenticate'"
              Access-Control-Allow-Origin: "'*'"
              ErrorHeader: "'MyCustomErrorHeader'"
          ResponseTemplates:
            application/json: "{\"message\": \"Error on the $context.resourcePath
 resource\" }"

  GetFunction:
    Type: AWS::Serverless::Function
```

```
    Properties:
      Runtime: python3.10
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
          raise Exception('Check out the new response!')
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /error
            Method: get
            RestApiId: !Ref MyApi
```

API Gateway レスポンスの詳細については、API Gateway デベロッパーガイドの「API Gateway でのゲートウェイレスポンス」を参照してください。

AWS SAM で Lambda レイヤーを使用して効率を向上させる

AWS SAM を使用すると、サーバーレスアプリケーションにレイヤーを含めることができます。AWS Lambda レイヤーは Lambda 関数から Lambda レイヤーへのコードの抽出を可能にし、そのコードは複数の Lambda 関数で使用できます。これにより、デプロイパッケージのサイズを縮小し、コア関数ロジックを依存関係から分離し、複数の関数間で依存関係を共有できます。レイヤーの詳細情報については、「AWS Lambda デベロッパーガイド」の「Lambda レイヤー」を参照してください。

このトピックでは、以下に関する情報を提供します。

- アプリケーションへのレイヤーの包含
- レイヤーがローカルにキャッシュされる方法

カスタムレイヤーの構築については、「AWS SAM での Lambda レイヤーの構築」を参照してください。

アプリケーションへのレイヤーの包含

アプリケーションにレイヤーを含めるには、AWS::Serverless::Function リソースタイプの `Layers` プロパティを使用します。

以下は、レイヤーが含まれる Lambda 関数を使用した AWS SAM テンプレートの例です。

```
ServerlessFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: .
    Handler: my_handler
    Runtime: Python3.7
    Layers:
        - <LayerVersion ARN>
```

レイヤーがローカルにキャッシュされる方法

`sam local` コマンドの 1 つを使用して関数を呼び出すと、関数のレイヤーパッケージがダウンロードされ、ローカルホストにキャッシュされます。

以下の表は、異なるオペレーティングシステムのデフォルトのキャッシュディレクトリの場所を示しています。

OS	ロケーション
Windows 7	`C:\Users\<user>\AppData\Roaming\AWS SAM`
Windows 8	`C:\Users\<user>\AppData\Roaming\AWS SAM`
Windows 10	`C:\Users\<user>\AppData\Roaming\AWS SAM`
macOS	`~/.aws-sam/layers-pkg`
Unix	`~/.aws-sam/layers-pkg`

パッケージがキャッシュされると、AWS SAM CLI が、関数の呼び出しに使用される Docker イメージにレイヤーをオーバーレイします。AWS SAM CLI は、それが構築するイメージの名前と、キャッシュに保持される LayerVersions を生成します。スキーマの詳細は、後続のセクションで説明されています。

オーバーレイされたレイヤーを調査するには、以下のコマンドを実行して、調査したいイメージで bash セッションを開始します。

```
docker run -it --entrypoint=/bin/bash samcli/lambda:<Tag following the schema outlined
 in Docker Image Tag Schema> -i
```

レイヤーキャッシュディレクトリ名のスキーマ

テンプレートで定義されている LayerVersionArn を提供すると、AWS SAM CLI が ARN から LayerName と Version を抽出し、レイヤーコンテンツを配置するための `LayerName-Version-<first 10 characters of sha256 of ARN>` という名前のディレクトリを作成します。

例：

```
ARN = arn:aws:lambda:us-west-2:111111111111:layer:myLayer:1
Directory name = myLayer-1-926eeb5ff1
```

Docker イメージのタグスキーマ

一意のレイヤーのハッシュを計算するには、「-」の区切り記号を使用してすべての一意のレイヤー名を結合し、SHA256 ハッシュを指定してから、最初の 10 文字を指定します。

例：

```
ServerlessFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: .
    Handler: my_handler
    Runtime: Python3.7
    Layers:
        - arn:aws:lambda:us-west-2:111111111111:layer:myLayer:1
        - arn:aws:lambda:us-west-2:111111111111:layer:mySecondLayer:1
```

一意の名前は、レイヤーキャッシュディレクトリ名のスキーマと同じ方法で計算されます。

```
arn:aws:lambda:us-west-2:111111111111:layer:myLayer:1 = myLayer-1-926eeb5ff1
arn:aws:lambda:us-west-2:111111111111:layer:mySecondLayer:1 =
 mySecondLayer-1-6bc1022bdf
```

一意のレイヤーのハッシュを計算するには、「-」の区切り記号を使用してすべての一意のレイヤー名を結合し、SHA256 ハッシュを指定してから、最初の 25 文字を指定します。

```
myLayer-1-926eeb5ff1-mySecondLayer-1-6bc1022bdf = 2dd7ac5ffb30d515926aef
```

次に、この値と関数のランタイムおよびアーキテクチャを区切り記号「-」で結合します。

```
python3.7-x86_64-2dd7ac5ffb30d515926aefffd
```

AWS SAM 内で、ネストされたアプリケーションを使用して、コードとリソースを再使用する

サーバーレスアプリケーションには、1 つ、または複数のネストされたアプリケーションを含めることができます。ネストされたアプリケーションはより大きなアプリケーションの一部であり、スタンドアロンのアーティファクトとして、またはより大きなアプリケーションのコンポーネントとしてパッケージ化とデプロイができます。ネストされたアプリケーションを使用すると、頻繁に使用されるコードを、それ独自のアプリケーションに変換でき、より大きなサーバーレスアプリケーションまたは複数のサーバーレスアプリケーション間で再利用することができます。

サーバーレスアーキテクチャが大きくなるにつれて起こる共通のパターンに、同じコンポーネントが複数のアプリケーションテンプレートで定義されているということがあります。ネストされたアプリケーションでは、共通のコード、機能、リソース、分離したAWS SAM テンプレート内の設定などを再利用できるため、1 つのソースからコードを保守できるようになります。これにより、コードと設定の重複を減らします。さらに、このモジュラーアプローチは、開発を合理化し、コードの編成を強化し、サーバーレスアプリケーション間の一貫性を促進します。ネストされたアプリケーションを使用することで、アプリケーションに固有のビジネスロジックにさらに集中できるようになります。

サーバーレスアプリケーションでネストされたアプリケーションを定義するには、AWS::Serverless::Application リソースタイプを使用します。

ネストされたアプリケーションは、以下の 2 つのソースから定義できます。

- AWS Serverless Application Repository アプリケーション - AWS Serverless Application Repository にある、アカウントで利用可能なアプリケーションを使用して、ネストされたアプリケーションを定義できます。これには、アカウント内のプライベートアプリケーション、アカウントとプライベートに共有されたアプリケーション、または AWS Serverless Application Repository で一般公開されているアプリケーションを使用できます。異なるデプロイ許可レベルの詳細については、AWS Serverless Application Repository デベロッパーガイドの「Application Deployment Permissions」と「Publishing Applications」を参照してください。
- ローカルアプリケーション - ローカルファイルシステムに保存されているアプリケーションを使用して、ネストされたアプリケーションを定義できます。

サーバーレスアプリケーションで両タイプのネストされたアプリケーションを定義するために AWS SAM を使用する方法の詳細については、以下のセクションを参照してください。

> **Note**
>
> サーバーレスアプリケーションでネストできるアプリケーションの最大数は 200 です。ネストされたアプリケーションで使用できるパラメータの最大数は 60 です。

AWS Serverless Application Repository からのネストされたアプリケーションの定義

ネストされたアプリケーションは、AWS Serverless Application Repository で利用可能なアプリケーションを使用することによって定義できます。また、AWS Serverless Application Repository を使用して、ネストされたアプリケーションが含まれるアプリケーションを保存し、配信することもできます。AWS Serverless Application Repository にあるネストされたアプリケーションの詳細を確認するには、AWS SDK、AWS CLI、または Lambda コンソールを使用できます。

AWS Serverless Application Repository でホストされるアプリケーションをサーバーレスアプリケーションの AWS SAM で定義するには、すべての AWS Serverless Application Repository アプリケーションの詳細ページにある [Copy as SAM Resource] (SAM リソースとしてコピーする) ボタンを使用します。これを実行するには、以下の手順を実行します。

1. AWS Management Console にサインインしていることを確認します。
2. AWS Serverless Application Repository デベロッパーガイドの「Browsing, Searching, and Deploying Applications」セクションにある手順に従って、ネストしたいアプリケーションを AWS Serverless Application Repository で検索します。
3. [Copy as SAM Resource] (SAM リソースとしてコピーする) ボタンをクリックします。これで、表示されているアプリケーションの SAM テンプレートセクションがクリップボードにコピーされます。
4. このアプリケーションにネストしたいアプリケーションの SAM テンプレートファイルの `Resources:` セクションに、この SAM テンプレートセクションを貼り付けます。

以下は、AWS Serverless Application Repository でホストされているネストされたアプリケーションの SAM テンプレートセクションの例です。

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

Resources:
  applicationaliasname:
    Type: AWS::Serverless::Application
```

```
    Properties:
      Location:
        ApplicationId: arn:aws:serverlessrepo:us-
east-1:123456789012:applications/application-alias-name
        SemanticVersion: 1.0.0
      Parameters:
        # Optional parameter that can have default value overridden
        # ParameterName1: 15 # Uncomment to override default value
        # Required parameter that needs value to be provided
        ParameterName2: YOUR_VALUE
```

必要なパラメータ設定がない場合は、テンプレートの Parameters: セクションを省略できます。

> **Important**
>
> AWS Serverless Application Repository でホストされているネストされたアプリケーションが含まれるアプリケーションは、ネストされたアプリケーションの共有制約を継承します。例えば、あるアプリケーションが一般公開されていて、それにこの親アプリケーションを作成した AWS アカウントとプライベート限定で共有されている、ネストされたアプリケーションが含まれているとします。この場合、このネストされたアプリケーションをデプロイする許可が AWS アカウントにないときに、親アプリケーションをデプロイすることはできません。アプリケーションをデプロイするための許可の詳細については、AWS Serverless Application Repository デベロッパーガイドの「Application Deployment Permissions」と「Publishing Applications」を参照してください。

ローカルファイルシステムからのネストされたアプリケーションの定義

ローカルファイルシステムに保存されているアプリケーションを使用して、ネストされたアプリケーションを定義できます。これは、ローカルファイルシステムに保存されている AWS SAM テンプレートファイルへのパスを指定することによって実行します。

以下は、ネストされたローカルアプリケーションの SAM テンプレートセクションの例です。

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

Resources:
  applicationaliasname:
    Type: AWS::Serverless::Application
    Properties:
```

```
      Location: ../my-other-app/template.yaml
      Parameters:
        # Optional parameter that can have default value overridden
        # ParameterName1: 15 # Uncomment to override default value
        # Required parameter that needs value to be provided
        ParameterName2: YOUR_VALUE
```

パラメータ設定がない場合は、テンプレートの `Parameters:` セクションを省略できます。

ネストされたアプリケーションのデプロイ

AWS SAM CLI コマンド `sam deploy` を使用して、ネストされたアプリケーションをデプロイできます。詳細については、「AWS SAM を使用してアプリケーションとリソースをデプロイする」を参照してください。

> **Note**
>
> ネストされたアプリケーションが含まれるアプリケーションをデプロイする場合、お客様は、これらが含まれることを承認する必要があります。これは、`CAPABILITY_AUTO_EXPAND` を CreateCloudFormationChangeSet API に渡すか、`aws serverlessrepo create-cloud-formation-change-set` AWS CLI コマンドを使用することで実行します。
> ネストされたアプリケーションの承認に関する詳細については、AWS Serverless Application Repository デベロッパーガイドの「Acknowledging IAM Roles, Resource Policies, and Nested Applications when Deploying Applications」を参照してください。

AWS SAM で EventBridge スケジューラを使用して時間ベースのイベントを管理する

このトピックのコンテンツでは、Amazon EventBridge スケジューラについて、AWS SAM が提供するサポート内容、スケジューライベントの作成方法、およびスケジューライベントの作成時に参照できる例について詳細を説明します。

Amazon EventBridge スケジューラとは

EventBridge スケジューラを使用して、AWS SAM テンプレート内でイベントをスケジュールします。Amazon EventBridge スケジューラは、すべての AWS サービスで数千万に及ぶイベントやタス

クを作成、開始、管理できるスケジューリングサービスです。このサービスは、時間関連のイベントに特に役立ちます。これは、イベントや定期的な時間ベースの呼び出しをスケジュールするために使用します。またこれは、1 回限りのイベントだけでなく、開始時刻と終了時刻を指定したレート式と Chron 式もサポートしています。

Amazon EventBridge スケジューラの詳細については、「EventBridge スケジューラユーザーガイド」の「What is Amazon EventBridge Scheduler?」(Amazon EventBridge スケジューラとは) を参照してください。

トピック

- AWS SAM での EventBridge スケジューラのサポート
- AWS SAM での EventBridge スケジューライベントの作成
- 例
- 詳細はこちら

AWS SAM での EventBridge スケジューラのサポート

AWS Serverless Application Model (AWS SAM) テンプレートの仕様には、EventBridge スケジューラを使用して AWS Lambda や AWS Step Functions のイベントをスケジュールできる、シンプルで簡潔な構文が用意されています。

AWS SAM での EventBridge スケジューライベントの作成

`ScheduleV2` プロパティを AWS SAM テンプレートのイベントタイプとして設定して、EventBridge スケジューライベントを定義します。このプロパティは、`AWS::Serverless::Function` および `AWS::Serverless::StateMachine` リソースタイプをサポートします。

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    Events:
      CWSchedule:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: 'rate(1 minute)'
          Name: TestScheduleV2Function
          Description: Test schedule event
```

```
MyStateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
    Events:
      CWSchedule:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: 'rate(1 minute)'
          Name: TestScheduleV2StateMachine
          Description: Test schedule event
```

EventBridge スケジューラのイベントスケジューリングでは、未処理のイベントのデッドレターキュー (DLQ) もサポートされています。デッドレターキューの詳細については、「EventBridge スケジューラユーザーガイド」の「Configuring a dead-letter queue for EventBridge Scheduler」(EventBridge スケジューラのデッドレターキューの設定) を参照してください。

DLQ ARN を指定すると、AWS SAM はスケジューラのスケジュールのアクセス許可を設定して DLQ にメッセージを送信します。DLQ ARN が指定されていない場合、AWS SAM は DLQ リソースを作成します。

例

AWS SAM を使用して EventBridge スケジューライベントを定義する基本的な例

```
Transform: AWS::Serverless-2016-10-31
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: python3.8
      InlineCode: |
        def handler(event, context):
            print(event)
            return {'body': 'Hello World!', 'statusCode': 200}
      MemorySize: 128
      Events:
        Schedule:
          Type: ScheduleV2
          Properties:
            ScheduleExpression: rate(1 minute)
            Input: '{"hello": "simple"}'
```

```
MySFNFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: python3.8
    InlineCode: |
      def handler(event, context):
          print(event)
          return {'body': 'Hello World!', 'statusCode': 200}
    MemorySize: 128

StateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
    Type: STANDARD
    Definition:
      StartAt: MyLambdaState
      States:
        MyLambdaState:
          Type: Task
          Resource: !GetAtt MySFNFunction.Arn
          End: true
    Policies:
      - LambdaInvokePolicy:
          FunctionName: !Ref MySFNFunction
    Events:
      Events:
      Schedule:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: rate(1 minute)
          Input: '{"hello": "simple"}'
```

詳細はこちら

ScheduleV2 EventBridge スケジューラプロパティの定義の詳細については、以下を参照してください。

- `AWS::Serverless::Function` 用の ScheduleV2。
- `AWS::Serverless::StateMachine` 用の ScheduleV2。

AWS Step Functions を使用した AWS SAM リソースのオーケストレーション

AWS Step Functions を使用して、複雑で堅牢なワークフローを形成するために AWS Lambda 関数とその他の AWS リソースをオーケストレーションすることができます。Step Functions では、AWS Lambda 関数などの AWS リソースがいつ、どのような条件で使用されるかがアプリケーションに指示されます。これにより、複雑で堅牢なワークフローを形成するプロセスが簡素化されます。AWS::Serverless::StateMachine を使用して、ワークフロー内の個々のステップを定義し、各ステップでリソースを関連付けてから、これらのステップを順番に並べます。また、必要な移行および条件も追加します。これにより、複雑で堅牢なワークフローを作成するプロセスが簡素化されます。

> **Note**
>
> Step Functions ステートマシンが含まれた AWS SAM テンプレートを管理するには、AWS SAM CLI のバージョン 0.52.0 以降を使用する必要があります。使用しているバージョンを確認するには、`sam --version` コマンドを実行します。

Step Functions はタスクとステートマシンの概念に基づいています。ステートマシンは、JSON ベースの Amazon States Language を使用して定義します。Step Functions コンソール にはステートマシンの構造のグラフィカルなビューが表示されるので、ステートマシンのロジックを視覚的にチェックし、実行をモニタリングできます。

AWS Serverless Application Model (AWS SAM) の Step Functions サポートを使用して、以下を実行することができます。

- AWS SAM テンプレート内で直接、または個別のファイルでステートマシンを定義する
- AWS SAM ポリシーテンプレート、インラインポリシー、またはマネージドポリシーを使用してステートマシンの実行ロールを作成する
- API Gateway または Amazon EventBridge イベントを使用して、AWS SAM テンプレート内のスケジュールに従って、または API を直接呼び出すことによってステートマシンの実行をトリガーする
- 一般的な Step Functions 開発パターン向けに利用できる AWS SAM ポリシーテンプレートを使用する

例

AWS SAM テンプレートファイルからの以下のサンプルスニペットは、定義ファイル内のStep Functions ステートマシンを定義します。my_state_machine.asl.json ファイルは Amazon States Language で記述される必要があることに注意してください。

```
AWSTemplateFormatVersion: "2010-09-09"
Transform: AWS::Serverless-2016-10-31
Description: Sample SAM template with Step Functions State Machine

Resources:
  MyStateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      DefinitionUri: statemachine/my_state_machine.asl.json
      ...
```

Step Functions ステートマシンが含まれたサンプル AWS SAM アプリケーションをダウンロードするには、AWS Step Functions デベロッパーガイドの「Create a Step Functions State Machine Using AWS SAM」を参照してください。

詳細情報

Step Functions と AWS SAM での Step Functions の使用に関する詳細については、以下を参照してください。

- AWS Step Functions のしくみ
- AWS Step Functions および AWS Serverless Application Model
- チュートリアル: AWS SAM を使用して Step Functions ステートマシンを作成
- AWS SAM の仕様: AWS::Serverless::StateMachine

AWS SAM アプリケーションのコード署名を設定する

信頼できるコードのみがデプロイされるようにするには、 AWS SAM を使用してサーバーレスアプリケーションでコード署名を有効にすることができます。コードに署名すると、署名後にコードが変更されず、信頼できるパブリッシャーから署名されたコードパッケージのみが Lambda 関数で実行されるようになります。これにより組織は、デプロイパイプラインにゲートキーパーコンポーネントを構築する負担から解放されます。

コード署名の詳細については、「AWS Lambda デベロッパーガイド」の「Lambda 関数のコード署名の設定」を参照してください。

サーバーレスアプリケーションのコード署名を設定する前に、 AWS Signer を使用して署名プロファイルを作成する必要があります。この署名プロファイルは、以下のタスクに使用します。

1. コード署名設定の作成 - `AWS::Lambda::CodeSigningConfig` リソースを宣言して、信頼できる発行元の署名プロファイルの指定と、検証チェックのためのポリシーアクションの設定を行います。このオブジェクトは、サーバーレス関数と同じ AWS SAM テンプレート、別の AWS SAM テンプレート、または AWS CloudFormation テンプレートで宣言できます。次に、`AWS::Lambda::CodeSigningConfig`リソースの Amazon リソースネーム (ARN) で関数の`CodeSigningConfigArn`プロパティを指定して、サーバーレス関数のコード署名を有効にします。
2. コードの署名 - `--signing-profiles` オプションを用いた `sam package` または `sam deploy` コマンドを使用します。

> **Note**
>
> `sam package` または `sam deploy` コマンドを使用したコードの署名が正常に行われるには、これらのコマンドで使用する Amazon S3 バケットでバージョニングが有効化されている必要があります。 AWS SAM が作成した Amazon S3 バケットを使用している場合、バージョニングは自動的に有効になります。Amazon S3 バケットのバージョニング、および提供する Amazon S3 バケットでバージョニングを有効化する手順の詳細については、Amazon Simple Storage Service ユーザーガイドの「S3 バケットでのバージョニングの使用」を参照してください。

サーバーレスアプリケーションをデプロイするときは、コード署名を有効にしたすべての関数に対して Lambda が検証チェックを実行します。Lambda は、これらの関数が依存するレイヤーにも検証チェックを実行します。Lambda の検証チェックの詳細については、AWS Lambda デベロッパーガイドの「署名の検証」を参照してください。

例

署名プロファイルの作成

署名プロファイルを作成するには、以下のコマンドを実行します。

```
aws signer put-signing-profile --platform-id "AWSLambda-SHA384-ECDSA" --profile-
name MySigningProfile
```

前のコマンドが成功すると、署名プロファイルARNが返されます。例:

```
{
    "arn": "arn:aws:signer:us-east-1:111122223333:/signing-profiles/MySigningProfile",
    "profileVersion": "SAMPLEverx",
    "profileVersionArn": "arn:aws:signer:us-east-1:111122223333:/signing-
profiles/MySigningProfile/SAMPLEverx"
}
```

profileVersionArn フィールドにはARN、コード署名設定を作成するときに使用する が含まれます。

コード署名設定の作成と関数のコード署名の有効化

次の AWS SAM テンプレート例では、AWS::Lambda::CodeSigningConfigリソースを宣言し、Lambda 関数のコード署名を有効にします。この例では、信頼できるプロファイルが 1 つあり、署名チェックが失敗するとデプロイが拒否されます。

```
Resources:
  HelloWorld:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.7
      CodeSigningConfigArn: !Ref MySignedFunctionCodeSigningConfig

  MySignedFunctionCodeSigningConfig:
    Type: AWS::Lambda::CodeSigningConfig
    Properties:
      Description: "Code Signing for MySignedLambdaFunction"
      AllowedPublishers:
        SigningProfileVersionArns:
          - MySigningProfile-profileVersionArn
      CodeSigningPolicies:
        UntrustedArtifactOnDeployment: "Enforce"
```

コードの署名

コードは、アプリケーションをパッケージ化またはデプロイするときに署名できます。以下のコマンド例にあるように、sam package または sam deploy コマンドで --signing-profiles オプションを指定します。

アプリケーションをパッケージ化するときの関数コードへの署名:

```
sam package --signing-profiles HelloWorld=MySigningProfile --s3-bucket amzn-s3-demo-bucket --output-template-file packaged.yaml
```

アプリケーションをパッケージ化するときの関数コードと関数が依存するレイヤー両方への署名:

```
sam package --signing-profiles HelloWorld=MySigningProfile MyLayer=MySigningProfile --s3-bucket amzn-s3-demo-bucket --output-template-file packaged.yaml
```

関数コードの署名と、署名後のデプロイの実行:

```
sam deploy --signing-profiles HelloWorld=MySigningProfile MyLayer=MySigningProfile --s3-bucket amzn-s3-demo-bucket --template-file packaged.yaml --stack-name --region us-east-1 --capabilities CAPABILITY_IAM
```

> **Note**
>
> sam package または sam deploy コマンドを使用したコードの署名が正常に行われるには、これらのコマンドで使用する Amazon S3 バケットでバージョニングが有効化されている必要があります。 AWS SAM が作成した Amazon S3 バケットを使用している場合、バージョニングは自動的に有効になります。Amazon S3 バケットのバージョニング、および提供する Amazon S3 バケットでバージョニングを有効化する手順の詳細については、Amazon Simple Storage Service ユーザーガイドの「S3 バケットでのバージョニングの使用」を参照してください。

sam deploy --guided での署名プロファイルの提供

コード署名で設定されたサーバーレスアプリケーションで sam deploy --guided コマンドを実行すると、 はコード署名に使用する署名プロファイルを指定するように AWS SAM プロンプトします。sam deploy --guided プロンプトの詳細については、sam deploy「」の「」を参照してください。 AWS SAM CLI コマンドリファレンス。

AWS SAM テンプレートファイルを検証する

sam validate を使用してテンプレートを検証します。現在、このコマンドは、提供されたテンプレートが有効な JSON/YAML であることを検証します。ほとんどの AWS SAM CLI コマンドと同様に、このコマンドはデフォルトで、現行の作業ディレクトリ内の template.[yaml|yml] ファイルを検索します。-t または --template オプションを使用して、異なるテンプレートファイル/場所を指定できます。

例：

```
$ sam validate
<path-to-template>/template.yaml is a valid SAM Template
```

> **Note**
>
> sam validate コマンドには、AWS 認証情報を設定する必要があります。詳細については、「AWS SAM CLI の設定」を参照してください。

AWS SAM を使用してアプリケーションを構築する

AWS SAM テンプレートに Infrastructure as Code (IaC) を追加したら、sam build コマンドを使用してアプリケーションの構築を開始できます。このコマンドでは、アプリケーションプロジェクトディレクトリ内のファイル (AWS SAM テンプレートファイル、アプリケーションコード、および該当する言語固有のファイルと依存関係) からビルドアーティファクトを作成します。これらのビルドアーティファクトにより、ローカルテストや AWS クラウドへのデプロイなど、アプリケーション開発の後のステップのためにサーバーレスアプリケーションが準備されます。テストとデプロイの両方で、ビルドアーティファクトは入力として使用されます。

sam build を使用して、サーバーレスアプリケーション全体を構築できます。さらに、特定の関数、レイヤー、カスタムランタイムなど、カスタマイズされたビルドを作成できます。sam build の使用方法と理由の詳細については、このセクションのトピックを参照してください。sam build コマンドの使用方法の概要については、「AWS SAM を使用した構築の概要」を参照してください。

トピック

- AWS SAM を使用した構築の概要
- AWS SAM を使用したデフォルトのビルド
- AWS SAM を使用してビルドをカスタマイズする

AWS SAM を使用した構築の概要

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam build` コマンドを使用して、ローカルテストや AWS クラウド へのデプロイなど、開発ワークフローの後続のステップに備えてサーバーレスアプリケーションを準備します。このコマンドは、`sam local` および `sam deploy` が必要とする形式と場所でアプリケーションを構造化する `.aws-sam` ディレクトリを作成します。

- AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください。
- `sam build` コマンドオプションのリストについては、「sam build」を参照してください。
- 一般的な開発ワークフローでの `sam build` の使用例については、「ステップ 2: アプリケーションを構築する」を参照してください。

> **Note**
>
> `sam build` を使用するには、開発マシン上のサーバーレスアプリケーションの基本コンポーネントから始める必要があります。これには、AWS SAM テンプレート、AWS Lambda 関数コード、言語固有のファイルと依存関係が含まれます。詳細については、「AWS SAM でアプリケーションを作成する」を参照してください。

トピック

sam build を使用したアプリケーションの構築

`sam build` を使用する前に、次を設定することを検討してください。

1. Lambda 関数とレイヤー – `sam build` コマンドは Lambda 関数とレイヤーを構築できます。Lambda レイヤーの詳細については、「AWS SAM での Lambda レイヤーの構築」を参照してください。
2. Lambda ランタイム – ランタイムは、呼び出されたときに実行環境で関数を実行する言語固有の環境を提供します。ネイティブランタイムとカスタムランタイムを設定できます。
   a. ネイティブランタイム – サポートされている Lambda ランタイムで Lambda 関数を作成し、AWS クラウド でネイティブ Lambda ランタイムを使用する関数を構築します。
   b. カスタムランタイム – 任意のプログラミング言語を使用して Lambda 関数を作成し、esbuild など、makefile やサードパーティービルダーで定義されたカスタムプロセスを使用してランタイムを構築します。詳細については、「でのカスタムランタイムを使用した Lambda 関数の構築 AWS SAM」を参照してください。
3. Lambda パッケージタイプ – Lambda 関数は、次の Lambda デプロイパッケージタイプにパッケージ化できます。
   a. .zip ファイルアーカイブ – アプリケーション コードとその依存関係が含まれます。
   b. コンテナイメージ – 基本オペレーティングシステム、ランタイム、Lambda 拡張機能、アプリケーションコードとその依存関係が含まれています。

これらのアプリケーション設定は、`sam init` を使用してアプリケーションを初期化するときに設定できます。

- `sam init` の使用の詳細については、「AWS SAM でアプリケーションを作成する」を参照してください。
- アプリケーションでのこれらの設定の詳細については、「AWS SAM を使用したデフォルトのビルド」を参照してください。

アプリケーションを構築するには

1. cd を実行してプロジェクトのルートに移動します。これは AWS SAM テンプレートと同じ場所です。

```
$ cd sam-app
```

2. 下記を実行します。

```
sam-app $ sam build <arguments> <options>
```

> **Note**
>
> 一般的に使用されるオプションは `--use-container` です。詳細については、「指定されたコンテナ内における Lambda 関数の構築」を参照してください。

次は AWS SAM CLI 出力の例です。

```
sam-app $ sam build
Starting Build use cache
Manifest file is changed (new hash: 3298f1304...d4d421) or dependency folder (.aws-
sam/deps/4d3dfad6-a267-47a6-a6cd-e07d6fae318c) is missing for (HelloWorldFunction),
 downloading dependencies and copying/building source
Building codeuri: /Users/.../sam-app/hello_world runtime: python3.12 metadata: {}
 architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CleanUp
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource
Running PythonPipBuilder:CopySource

Build Succeeded

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

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

3. AWS SAM CLI は `.aws-sam` ビルドディレクトリを作成します。以下に例を示します。

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

```
### build.toml
```

アプリケーションの設定に応じて、AWS SAM CLI は次を実行します。

1. .aws-sam/build ディレクトリ内の依存関係をダウンロード、インストール、整理します。
2. Lambda コードを準備します。これには、コードのコンパイル、実行可能バイナリの作成、コンテナイメージの構築などが含まれます。
3. ビルドアーティファクトを .aws-sam ディレクトリにコピーします。形式はアプリケーションパッケージの種類によって異なります。
   a. .zip パッケージタイプの場合、アーティファクトはローカルテストに使用できるようにまだ圧縮されていません。AWS SAM CLI は、sam deploy を使用するときにアプリケーションを圧縮します。
   b. コンテナイメージパッケージタイプの場合、コンテナイメージはローカルで作成され、.aws-sam/build.toml ファイル内で参照されます。
4. AWS SAM テンプレートを .aws-sam ディレクトリにコピーし、必要に応じて新しいファイルパスで変更します。

.aws-sam ディレクトリ内のビルドアーティファクトを構成する主要なコンポーネントを次に示します。

- ビルドディレクトリ – Lambda 関数と、互いに独立して構造化されたレイヤーが含まれます。これにより、.aws-sam/build ディレクトリ内の各関数またはレイヤー用に固有の構造が作成されます。
- AWS SAM テンプレート – 構築プロセス中の変更に基づいて更新された値で変更されます。
- build.toml ファイル – AWS SAM CLI で使用されるビルド設定を含む設定ファイル。

ローカルでのテストとデプロイ

sam local を使用してローカルテストを実行する場合、または sam deploy を使用してデプロイを実行する場合、AWS SAM CLI は次を実行します。

1. まず、.aws-sam ディレクトリが存在するかどうか、およびそのディレクトリ内に AWS SAM テンプレートが存在するかどうかを確認します。これらの条件が満たされる場合、AWS SAM CLI はこれをアプリケーションのルートディレクトリとみなします。

2. これらの条件が満たされない場合、AWS SAM CLI は AWS SAM テンプレートの元の場所をアプリケーションのルートディレクトリとみなします。

開発中、元のアプリケーションファイルに変更が加えられた場合は、ローカルでテストする前に、`sam build` を実行して `.aws-sam` ディレクトリを更新します。

ベストプラクティス

- `.aws-sam/build` ディレクトリ内のコードは編集しないでください。代わりに、プロジェクトフォルダ内の元のソースコードを更新し、`sam build` を実行して `.aws-sam/build` ディレクトリを更新します。
- 元のファイルを変更する場合は、`sam build` を実行して `.aws-sam/build` ディレクトリを更新します。
- `sam local` を使用して開発およびテストする場合など、AWS SAM CLI が `.aws-sam` ディレクトリではなくプロジェクトの元のルートディレクトリを参照するようにしたい場合があります。`.aws-sam` ディレクトリまたは `.aws-sam` ディレクトリ内の AWS SAM テンプレートを削除して、AWS SAM CLI が元のプロジェクトディレクトリをルートプロジェクトディレクトリとして認識するようにします。準備ができたら、再度 `sam build` を実行して `.aws-sam` ディレクトリを作成します。
- `sam build` を実行すると、`.aws-sam/build` ディレクトリは毎回上書きされます。`.aws-sam` ディレクトリは毎回上書きしません。ログなどのファイルを保存する場合は、上書きされないように `.aws-sam` に保存します。

sam build のオプション

単一のリソースの構築

リソースの論理 ID を指定して、そのリソースのみを構築します。以下に例を示します。

```
$ sam build HelloWorldFunction
```

ネストされたアプリケーションまたはスタックのリソースを構築するには、*<stack-logical-id>*/*<resource-logical-id>* 形式を使用して、リソースの論理 ID と共に、アプリケーションまたはスタックの論理 ID を指定します。

```
$ sam build MyNestedStack/MyFunction
```

指定されたコンテナ内における Lambda 関数の構築

`--use-container` オプションはコンテナイメージをダウンロードし、それを使用して Lambda 関数を構築します。その後、ローカルコンテナが `.aws-sam/build.toml` ファイル内で参照されます。

このオプションでは Docker をインストールする必要があります。手順については、Docker のインストール を参照してください。

このコマンドの例を次に示します。

```
$ sam build --use-container
```

`--build-image` オプションで使用するコンテナイメージを指定できます。以下に例を示します。

```
$ sam build --use-container --build-image amazon/aws-sam-cli-build-image-nodejs20.x
```

単一の関数に使用するコンテナイメージを指定するには、関数の論理 ID を指定します。以下に例を示します。

```
$ sam build --use-container --build-image Function1=amazon/aws-sam-cli-build-image-
python3.12
```

環境変数をビルドコンテナに渡す

`--container-env-var` を使用して、環境変数をビルドコンテナに渡します。以下に例を示します。

```
$ sam build --use-container --container-env-var Function1.GITHUB_TOKEN=<token1> --
container-env-var GLOBAL_ENV_VAR=<global-token>
```

ファイルから環境変数を渡すには、`--container-env-var-file` オプションを使用します。以下に例を示します。

```
$ sam build --use-container --container-env-var-file <env.json>
```

`env.json` ファイルの例:

```
{
```

```
  "MyFunction1": {
    "GITHUB_TOKEN": "TOKEN1"
  },
  "MyFunction2": {
    "GITHUB_TOKEN": "TOKEN2"
  }
}
```

複数の関数を含むアプリケーションの構築を高速化する

複数の関数を含むアプリケーションで sam build を実行する場合、AWS SAM CLI は各関数を一度に 1 つずつ構築します。構築プロセスを高速化するには、--parallel オプションを使用します。これにより、すべての関数とレイヤーが同時に構築されます。

このコマンドの例を次に示します。

```
$ sam build --parallel
```

ソースフォルダにプロジェクトを構築することでビルド時間を短縮する

サポートされているランタイムとビルドメソッドについては、--build-in-source オプションを使用してプロジェクトをソースフォルダに直接構築できます。デフォルトでは、AWS SAM CLI は一時ディレクトリに構築されます。これは、ソースコードとプロジェクトファイルのコピーを伴います。--build-in-source を使用すると、AWS SAM CLI はソースフォルダに直接構築するため、ファイルを一時ディレクトリにコピーする必要がなくなり、ビルドプロセスが高速化されます。

サポートされているランタイムとビルドメソッドのリストについては、「--build-in-source」を参照してください。

トラブルシューティング

AWS SAM CLI をトラブルシューティングするには、「AWS SAMCLI トラブルシューティング」を参照してください。

例

ネイティブランタイムと .zip パッケージタイプを使用するアプリケーションの構築

この例については、「チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM」を参照してください。

ネイティブランタイムとイメージパッケージタイプを使用するアプリケーションの構築

まず、新しいアプリケーションを初期化するために sam init を実行します。インタラクティブなフロー中に、Image パッケージタイプを選択します。以下に例を示します。

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

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

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

Which runtime would you like to use?
        ...
        10 - java8
        11 - nodejs20.x
        12 - nodejs18.x
        13 - nodejs16.x
        ...
Runtime: 12

What package type would you like to use?
        1 - Zip
        2 - Image
Package type: 2
```

```
Based on your selections, the only dependency manager available is npm.
We will proceed copying the template using npm.

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

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

Project name [sam-app]: ENTER

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

    -----------------------
    Generating application:
    -----------------------
    Name: sam-app
    Base Image: amazon/nodejs18.x-base
    Architectures: x86_64
    Dependency Manager: npm
    Output Directory: .
    Configuration file: sam-app/samconfig.toml

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

...
```

AWS SAM CLI はアプリケーションを初期化し、次のプロジェクトディレクトリを作成します。

```
sam-app
### README.md
### events
#   ### event.json
### hello-world
#   ### Dockerfile
#   ### app.mjs
#   ### package.json
#   ### tests
#       ### unit
#           ### test-handler.mjs
### samconfig.toml
```

```
### template.yaml
```

次に、アプリケーションを構築するために sam build を実行します。

```
sam-app $ sam build
Building codeuri: /Users/.../build-demo/sam-app runtime: None metadata: {'DockerTag':
 'nodejs18.x-v1', 'DockerContext': '/Users/.../build-demo/sam-app/hello-world',
 'Dockerfile': 'Dockerfile'} architecture: arm64 functions: HelloWorldFunction
Building image for HelloWorldFunction function
Setting DockerBuildArgs: {} for HelloWorldFunction function
Step 1/4 : FROM public.ecr.aws/lambda/nodejs:18
 ---> f5b68038c080
Step 2/4 : COPY app.mjs package*.json ./
 ---> Using cache
 ---> 834e565aae80
Step 3/4 : RUN npm install
 ---> Using cache
 ---> 31c2209dd7b5
Step 4/4 : CMD ["app.lambdaHandler"]
 ---> Using cache
 ---> 2ce2a438e89d
Successfully built 2ce2a438e89d
Successfully tagged helloworldfunction:nodejs18.x-v1

Build Succeeded

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

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

コンパイルされたプログラミング言語を含むアプリケーションの構築

この例では、Go ランタイムを使用して Lambda 関数を含むアプリケーションを構築します。

まず、sam init を使用して新しいアプリケーションを初期化し、アプリケーションが Go を使用するように設定します。

```
$ sam init

...

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

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        3 - Serverless API
        ...
Template: 1

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

Which runtime would you like to use?
        ...
        4 - dotnetcore3.1
        5 - go1.x
        6 - go (provided.al2)
        ...
Runtime: 5

What package type would you like to use?
        1 - Zip
        2 - Image
Package type: 1

Based on your selections, the only dependency manager available is mod.
We will proceed copying the template using mod.

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

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

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

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

    -----------------------
    Generating application:
    -----------------------
    Name: sam-app
    Runtime: go1.x
    Architectures: x86_64
    Dependency Manager: mod
    Application Template: hello-world
    Output Directory: .
    Configuration file: sam-app/samconfig.toml

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

...
```

AWS SAM CLI はアプリケーションを初期化します。アプリケーションのディレクトリ構造の例を次に示します。

```
sam-app
### Makefile
### README.md
### events
#   ### event.json
### hello-world
#   ### go.mod
#   ### go.sum
#   ### main.go
#   ### main_test.go
### samconfig.toml
### template.yaml
```

このアプリケーションの要件については、README.md ファイルを参照します。

```
...
## Requirements
* AWS CLI already configured with Administrator permission
* [Docker installed](https://www.docker.com/community-edition)
* [Golang](https://golang.org)
* SAM CLI - [Install the SAM CLI](https://docs.aws.amazon.com/serverless-application-
model/latest/developerguide/serverless-sam-cli-install.html)
```

```
...
```

次に、関数をテストするために sam local invoke を実行します。Go がローカルマシンにインストールされていないため、このコマンドはエラーになります。

```
sam-app $ sam local invoke
Invoking hello-world (go1.x)
Local image was not found.
Removing rapid images for repo public.ecr.aws/sam/emulation-go1.x
Building
 image.....................................................................................
Using local image: public.ecr.aws/lambda/go:1-rapid-x86_64.

Mounting /Users/.../Playground/build/sam-app/hello-world as /var/task:ro,delegated
 inside runtime container
START RequestId: c6c5eddf-042b-4e1e-ba66-745f7c86dd31 Version: $LATEST
fork/exec /var/task/hello-world: no such file or directory: PathError
null
END RequestId: c6c5eddf-042b-4e1e-ba66-745f7c86dd31
REPORT RequestId: c6c5eddf-042b-4e1e-ba66-745f7c86dd31  Init Duration: 0.88 ms
 Duration: 175.75 ms Billed Duration: 176 ms Memory Size: 128 MB     Max Memory Used:
 128 MB
{"errorMessage":"fork/exec /var/task/hello-world: no such file or
 directory","errorType":"PathError"}%
```

次に、アプリケーションを構築するために sam build を実行します。Go がローカルマシンにインストールされていないため、エラーが発生しました:

```
sam-app $ sam build
Starting Build use cache
Cache is invalid, running build and copying resources for following functions
 (HelloWorldFunction)
Building codeuri: /Users/.../Playground/build/sam-app/hello-world runtime: go1.x
 metadata: {} architecture: x86_64 functions: HelloWorldFunction

Build Failed
Error: GoModulesBuilder:Resolver - Path resolution for runtime: go1.x of binary: go was
 not successful
```

関数を適切に構築するようにローカルマシンを設定することもできますが、代わりに sam build で --use-container オプションを使用します。AWS SAM CLI はコンテナイメージをダウンロー

ドし、ネイティブ GoModulesBuilder を使用して関数を構築し、結果として得られるバイナリを `.aws-sam/build/HelloWorldFunction` ディレクトリにコピーします。

```
sam-app $ sam build --use-container
Starting Build use cache
Starting Build inside a container
Cache is invalid, running build and copying resources for following functions
 (HelloWorldFunction)
Building codeuri: /Users/.../build/sam-app/hello-world runtime: go1.x metadata: {}
 architecture: x86_64 functions: HelloWorldFunction

Fetching public.ecr.aws/sam/build-go1.x:latest-x86_64 Docker container
 image.....................................................................................
Mounting /Users/.../build/sam-app/hello-world as /tmp/samcli/source:ro,delegated inside
 runtime container
Running GoModulesBuilder:Build

Build Succeeded

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

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

`.aws-sam` ディレクトリの例を次に示します。

```
.aws-sam
### build
#   ### HelloWorldFunction
#   #   ### hello-world
#   ### template.yaml
### build.toml
### cache
#   ### c860d011-4147-4010-addb-2eaa289f4d95
#       ### hello-world
### deps
```

次に、sam local invoke を実行します。関数は正常に呼び出されます。

```
sam-app $ sam local invoke
Invoking hello-world (go1.x)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/go:1-rapid-x86_64.

Mounting /Users/.../Playground/build/sam-app/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated inside runtime container
START RequestId: cfc8ffa8-29f2-49d4-b461-45e8c7c80479 Version: $LATEST
END RequestId: cfc8ffa8-29f2-49d4-b461-45e8c7c80479
REPORT RequestId: cfc8ffa8-29f2-49d4-b461-45e8c7c80479  Init Duration: 1.20 ms
 Duration: 1782.46 ms        Billed Duration: 1783 ms        Memory Size: 128 MB
 Max Memory Used: 128 MB
{"statusCode":200,"headers":null,"multiValueHeaders":null,"body":"Hello,
 72.21.198.67\n"}%
```

詳細はこちら

sam build コマンドの使用方法の詳細については、次を参照してください。

- AWS SAM の学習: sam build – YouTube での Serverless Land の「AWS SAM の学習」シリーズ。
- AWS SAM の学習 | sam build | E3 – YouTube での Serverless Land の「AWS SAM の学習」シリーズ。
- AWS SAM ビルド: デプロイ用のアーティファクトを提供する方法 (SAM S2E8 を使用したセッション) – YouTube での AWS SAM を使用したセッションのシリーズ。
- AWS SAM カスタムビルド: Makefile を使用して SAM (S2E9) でビルドをカスタマイズする方法 – YouTube での AWS SAM を使用したセッションのシリーズ。

AWS SAM を使用したデフォルトのビルド

サーバーレスアプリケーションを構築するには、sam build コマンドを使用します。このコマンドは、アプリケーションの依存関係のビルドアーティファクトも収集して、ローカルでのテスト、パッケージ化、およびデプロイなどの次のステップのために、それらを適切な形式と場所に設定します。

アプリケーションの依存関係は、マニフェストファイル (requirements.txt (Python) または package.json (Node.js) など) で指定するか、関数リソースの Layers プロパティを使用して指定します。Layers プロパティには、Lambda 関数が依存する AWS Lambda レイヤーリソースのリストが含まれています。

アプリケーションのビルドアーティファクトの形式は、各関数の PackageType プロパティに応じて異なります。このプロパティのオプションには次のものがあります。

- **Zip** - アプリケーションコードとその依存関係が含まれる .zip ファイルアーカイブ。コードを .zip ファイルアーカイブとしてパッケージ化する場合は、関数の Lambda ランタイムを指定する必要があります。
- **Image** - アプリケーションコードとその依存関係に加えて、ベースオペレーティングシステム、ランタイム、および拡張機能が含まれているコンテナイメージ。

Lambda パッケージタイプに関する詳細については、AWS Lambda デベロッパーガイドの「Lambda デプロイパッケージ」を参照してください。

トピック

- .zip ファイルアーカイブの構築
- コンテナイメージの構築
- コンテナ環境変数ファイル
- ソースフォルダにプロジェクトを構築することでビルド時間を短縮する
- 例
- AWS SAM の外部に関数を構築する

.zip ファイルアーカイブの構築

サーバーレスアプリケーションを.zip ファイルアーカイブとして構築するには、サーバーレス関数に `PackageType: Zip` を宣言します。

AWS SAM は、指定したアーキテクチャーでアプリケーションを構築します。アーキテクチャを指定していない場合、AWS SAM はデフォルトで x86_64 を使用します。

ネイティブにコンパイルされたプログラムが含まれるパッケージに Lambda 関数が依存する場合は、`--use-container` フラグを使用します。このフラグは、Lambda 環境のように動作する Docker コンテナで関数をローカルにコンパイルするため、AWS クラウドへのデプロイ時にはそれらが適切な形式になります。

`--use-container` オプションを使用するときは、AWS SAM がデフォルトで Amazon ECR Public からコンテナイメージをプルします。DockerHub などの別のリポジトリからコンテナイメージをプルしたい場合は、`--build-image` オプションを使用して、代替コンテナイメージの URI を提供す

ることができます。以下は、DockerHub リポジトリからのコンテナイメージを使用してアプリケーションを構築するための 2 つのコマンド例です。

```
# Build a Node.js 20 application using a container image pulled from DockerHub
sam build --use-container --build-image amazon/aws-sam-cli-build-image-nodejs20.x

# Build a function resource using the Python 3.12 container image pulled from DockerHub
sam build --use-container --build-image Function1=amazon/aws-sam-cli-build-image-
python3.12
```

`--build-image` で使用できる URI のリストについては、サポートされている多数のランタイムに対する DockerHub URI が記載された「のイメージリポジトリ AWS SAM」を参照してください。

.zip ファイルアーカイブアプリケーションのその他の構築例については、このトピックの後半にある「例」セクションを参照してください。

コンテナイメージの構築

サーバーレスアプリケーションをコンテナイメージとして構築するには、サーバーレス関数に `PackageType: Image` を宣言します。また、以下のエントリを使って `Metadata` リソース属性を宣言する必要もあります。

`Dockerfile`

Lambda 関数に関連付けられた Dockerfile の名前。

`DockerContext`

Dockerfile の場所。

`DockerTag`

(オプション) 構築されたイメージに適用するタグ。

`DockerBuildArgs`

ビルド用のビルド引数。

> **Important**
>
> AWS SAM CLI では、`DockerBuildArgs` 引数に含める情報の編集または難読化は行われません。このセクションを使用してパスワードやシークレットなどの機密情報を保存しないことを強くお勧めします。

以下は、Metadata リソース属性セクションの例です。

```
    Metadata:
      Dockerfile: Dockerfile
      DockerContext: ./hello_world
      DockerTag: v1
```

Image パッケージタイプで設定されたサンプルアプリケーションをダウンロードするには、「チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM」を参照してください。インストールしたいパッケージタイプをたずねるプロンプトで、[Image] を選択します。

> **Note**
>
> Dockerfile でマルチアーキテクチャベースイメージを指定した場合、AWS SAM はホストマシンのアーキテクチャ用のコンテナイメージを構築します。別のアーキテクチャ用に構築するには、特定のターゲットアーキテクチャを使用するベースイメージを指定します。

コンテナ環境変数ファイル

ビルドコンテナの環境変数が含まれた JSON ファイルを提供するには、sam build コマンドで --container-env-var-file 引数を使用します。すべてのサーバーレスリソースに適用される単一の環境変数を提供する、または各リソースに異なる環境変数を提供することができます。

[形式]

環境変数をビルドコンテナに渡す形式は、リソースに提供する環境変数の数に応じて異なります。

すべてのリソースに対して単一の環境変数を提供するには、以下のように Parameters オブジェクトを指定します。

```
{
  "Parameters": {
    "GITHUB_TOKEN": "TOKEN_GLOBAL"
  }
}
```

各リソースに異なる環境変数を提供するには、以下のようにリソースごとにオブジェクトを指定します。

```
{
  "MyFunction1": {
    "GITHUB_TOKEN": "TOKEN1"
  },
  "MyFunction2": {
    "GITHUB_TOKEN": "TOKEN2"
  }
}
```

環境変数をファイル (env.json と命名されたファイルなど) として保存します。以下のコマンドは、このファイルを使用して環境変数をビルドコンテナに渡します。

```
sam build --use-container --container-env-var-file env.json
```

優先順位

- すべてのリソースに対する単一の環境変数よりも、特定のリソースに提供する環境変数が優先されます。
- ファイル内の環境変数よりも、コマンドラインで提供する環境変数が優先されます。

ソースフォルダにプロジェクトを構築することでビルド時間を短縮する

サポートされているランタイムとビルドメソッドについては、--build-in-source オプションを使用してプロジェクトをソースフォルダに直接構築できます。デフォルトでは、AWS SAM CLI は一時ディレクトリに構築されます。これは、ソースコードとプロジェクトファイルのコピーを伴います。--build-in-source を使用すると、AWS SAM CLI はソースフォルダに直接構築するため、ファイルを一時ディレクトリにコピーする必要がなくなり、ビルドプロセスが高速化されます。

サポートされているランタイムとビルドメソッドのリストについては、「--build-in-source」を参照してください。

例

例 1。 .zip ファイルアーカイブ

以下の sam build コマンドは、.zip ファイルアーカイブを構築します。

```
# Build all functions and layers, and their dependencies
sam build
```

```
# Run the build process inside a Docker container that functions like a Lambda
 environment
sam build --use-container

# Build a Node.js 20 application using a container image pulled from DockerHub
sam build --use-container --build-image amazon/aws-sam-cli-build-image-nodejs20.x

# Build a function resource using the Python 3.12 container image pulled from DockerHub
sam build --use-container --build-image Function1=amazon/aws-sam-cli-build-image-
python3.12

# Build and run your functions locally
sam build && sam local invoke

# For more options
sam build --help
```

例 2: コンテナイメージ

以下の AWS SAM テンプレートは、コンテナイメージとして構築します。

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      PackageType: Image
      ImageConfig:
        Command: ["app.lambda_handler"]
    Metadata:
      Dockerfile: Dockerfile
      DockerContext: ./hello_world
      DockerTag: v1
```

以下は、Dockerfile 例です。

```
FROM public.ecr.aws/lambda/python:3.12

COPY app.py requirements.txt ./

RUN python3.12 -m pip install -r requirements.txt

# Overwrite the command by providing a different command directly in the template.
CMD ["app.lambda_handler"]
```

例 3: npm ci

Node.js アプリケーションでは、npm installの代わりに、npm ciを使用して依存関係をインストールします。npm ci を使用するには、Lambda 関数の Metadata リソース属性の BuildProperties の下に UseNpmCi: True を指定します。npm ci を使用するには、アプリケーションは Lambda 関数の CodeUri に package-lock.json または npm-shrinkwrap.json ファイルが必要です。

次の例では、sam build を実行するときに、npm ci を使用して依存関係をインストールします。

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello-world/
      Handler: app.handler
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /hello
            Method: get
    Metadata:
      BuildProperties:
        UseNpmCi: True
```

AWS SAM の外部に関数を構築する

デフォルトでは、sam build を実行すると、AWS SAM はすべての関数リソースを構築します。その他のオプションには以下が含まれます。

- すべての関数リソースを AWS SAM の外部で構築する – すべての関数リソースを手動で、または別のツールを通じて構築する場合、sam build は必要ありません。sam build をスキップして、ローカルテストの実行やアプリケーションのデプロイなど、プロセスの次のステップに進むことができます。
- 一部の関数リソースを AWS SAM の外部で構築する – AWS SAM に一部の関数リソースを構築させつつ、他の関数リソースが AWS SAM の外部で構築されるようにするには、これを AWS SAM テンプレートで指定できます。

一部の関数リソースを AWS SAM の外部に構築する

sam build を使用する際に AWS SAM が関数をスキップするようにするには、AWS SAM テンプレートで次を設定します。

1. `SkipBuild: True` メタデータプロパティを関数に追加します。
2. 構築した関数リソースへのパスを指定します。

`TestFunction` がスキップされるように設定した例を次に示します。構築されたリソースは `built-resources/TestFunction.zip` にあります。

```
TestFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: built-resources/TestFunction.zip
    Handler: TimeHandler::handleRequest
    Runtime: java11
  Metadata:
    SkipBuild: True
```

これで、sam build を実行すると、AWS SAM は次を実行するようになりました。

1. AWS SAM は `SkipBuild: True` で設定された関数をスキップします。
2. AWS SAM は他のすべての関数リソースを構築し、`.aws-sam` ビルドディレクトリにキャッシュします。
3. スキップされた関数の場合、`.aws-sam` ビルドディレクトリ内のテンプレートは、構築された関数リソースへの指定されたパスを参照するように自動的に更新されます。

   `.aws-sam` ビルドディレクトリの `TestFunction` についてキャッシュされたテンプレートの例を次に示します。

   ```
   TestFunction:
     Type: AWS::Serverless::Function
     Properties:
       CodeUri: ../../built-resources/TestFunction.zip
       Handler: TimeHandler::handleRequest
       Runtime: java11
     Metadata:
       SkipBuild: True
   ```

AWS SAM を使用してビルドをカスタマイズする

ビルドをカスタマイズして、特定の Lambda 関数または Lambda レイヤーを含めることができます。関数とは、Lambda でコードを実行するために呼び出すことができるリソースです。Lambda レイヤーを使用すると、Lambda 関数からコードを抽出し、複数の Lambda 関数で再利用できます。共有の依存関係やリソースの管理を複雑にすることなく、個々のサーバーレス関数の開発とデプロイに集中する場合は、特定の Lambda 関数を使用してビルドをカスタマイズできます。さらに、デプロイパッケージのサイズを縮小し、コア関数ロジックを依存関係から分離し、複数の関数間で依存関係を共有できるように、Lambda レイヤーを構築することもできます。

このセクションのトピックでは、AWS SAM を使用して Lambda 関数を構築するさまざまな方法について説明します。これには、カスタマーランタイムを使用して Lambda 関数を構築し、Lambda レイヤーを構築することが含まれます。カスタムランタイムを使用すると、AWS Lambda デベロッパーガイドの Lambda ランタイムにリストされていない言語をインストールして使用できます。これにより、サーバーレス関数およびアプリケーションを実行するための特殊な実行環境を作成できます。(アプリケーション全体を構築するのではなく) Lambda レイヤーのみを構築すると、いくつかの点で利点があります。これにより、デプロイパッケージのサイズを縮小し、コア関数ロジックを依存関係から分離し、複数の関数間で依存関係を共有できます。

詳細については、「AWS Lambda デベロッパーガイド」の「Lambda の概念」を参照してください。

トピック

- AWS SAM での esbuild を使用した Node.js Lambda関数の構築
- AWS SAM でのネイティブ AOT コンパイルを使用した .NET Lambda 関数の構築
- を使用した Rust Lambda 関数の構築 Cargo Lambda の AWS SAM
- でのカスタムランタイムを使用した Lambda 関数の構築 AWS SAM
- AWS SAM での Lambda レイヤーの構築

AWS SAM での esbuild を使用した Node.js Lambda関数の構築

Node.js AWS Lambda 関数を構築してパッケージ化する場合、AWS SAM CLI と esbuild JavaScript バンドラーを使用できます。esbuild バンドラーは TypeScript で記述されて Lambda 関数をサポートします。

esbuild を使用して Node.js Lambda 関数を構築するには、`AWS:Serverless::Function` リソースに `Metadata` オブジェクトを追加し、`BuildMethod` に `esbuild` を指定します。sam build コマンドを実行すると、AWS SAM は esbuild を使用して、Lambda 関数コードをバンドルします。

Metadata プロパティ

`Metadata` オブジェクトは esbuild の以下のプロパティをサポートします。

BuildMethod

アプリケーションのバンドルを指定します。`esbuild` はサポートされる唯一の値です。

BuildProperties

Lambda 関数コードの構築プロパティを指定します。

`BuildProperties` オブジェクトは esbuild の以下のプロパティをサポートします。プロパティはすべてオプションです。デフォルトでは、AWS SAM は、エントリポイントに Lambda 関数ハンドラを使用します。

EntryPoint

アプリケーションのエントリポイントを指定します。

外部

構築から除外するパッケージのリストを指定します。詳細については、esbuild ウェブサイトの「External」を参照してください。

[形式]

アプリケーションで生成される JavaScript ファイルの出力形式を指定します。詳細については、esbuild ウェブサイトの「Format」を参照してください。

[ローダー]

特定のファイルタイプのデータをロードするための設定のリストを指定します。

MainFields

パッケージを解決する際にインポートを試行する `package.json` フィールドを指定します。デフォルト値は `main,module` です。

Minify

バンドルされた出力コードを縮小するかどうかを指定します。デフォルト値は `true` です。

OutExtension

esbuild が生成するファイルのファイル拡張子をカスタマイズします。詳細については、esbuild ウェブサイトの「Out extension」を参照してください。

Sourcemap

バンドラーでソースマップファイルを生成するかどうかを指定します。デフォルト値は `false` です。

`true` に設定すると、`NODE_OPTIONS: --enable-source-maps` が Lambda 関数の環境変数に追加され、ソースマップが生成されて関数に含まれます。

また、`NODE_OPTIONS: --enable-source-maps` が関数の環境変数に含まれると、`Sourcemap` が自動的に `true` に設定されます。

競合する場合、`Sourcemap: false` は `NODE_OPTIONS: --enable-source-maps` より優先されます。

> **Note**
>
> デフォルトでは、Lambda は保存中のすべての環境変数を AWS Key Management Service (AWS KMS) で暗号化します。デプロイを正常に行うには、ソースマップを使用するとき、関数の実行ロールに `kms:Encrypt` アクションを実行するアクセス許可が必要です。

SourcesContent

ソースマップファイルにソースコードを含めるかどうかを指定します。`Sourcemap` が `'true'` に設定されている場合、このプロパティを設定します。

- すべてのソースコードを含めるには `SourcesContent: 'true'` を指定します。
- すべてのソースコードを除外するには `SourcesContent: 'false'` を指定します。これにより、ソースマップのファイルサイズが小さくなり、起動時間が短縮されるため本番環境で役立ちます。ただし、ソースコードはデバッガーでは使用できません。

デフォルト値は `SourcesContent: true` です。

詳細については、esbuild ウェブサイトの「ソースコンテンツ」を参照してください。

Target

ターゲットの ECMAScript バージョンを指定します。デフォルト値は `es2020` です。

TypeScript Lambda 関数の例

次のサンプル AWS SAM テンプレートスニペットは、esbuild を使用して、`hello-world/app.ts` の TypeScript コードから Node.js Lambda 関数を作成します。

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello-world/
      Handler: app.handler
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /hello
            Method: get
      Environment:
        Variables:
          NODE_OPTIONS: --enable-source-maps
    Metadata:
      BuildMethod: esbuild
      BuildProperties:
        Format: esm
        Minify: false
        OutExtension:
          - .js=.mjs
        Target: "es2020"
        Sourcemap: true
        EntryPoints:
          - app.ts
        External:
          - "<package-to-exclude>"
```

AWS SAM でのネイティブ AOT コンパイルを使用した .NET Lambda 関数の構築

.NET 8 AWS Lambda 関数を AWS Serverless Application Model (AWS SAM) で構築してパッケージ化し、ネイティブの事前 (AOT) コンパイルを利用することで、AWS Lambda のコールドスタート時間を短縮します。

トピック

.NET 8 ネイティブ AOT の概要

従来の .NET Lambda 関数にはコールドスタート時間があり、それがユーザーエクスペリエンスを損ね、システムのレイテンシーやサーバーレスアプリケーションの使用コストに影響を及ぼしていました。.NET ネイティブ AOT コンパイルを使用すると、Lambda 関数のコールドスタート時間を短縮できます。.NET 8 のネイティブ AOT の詳細については、GitHub の dotnet リポジトリで「ネイティブ AOT の使用」を参照してください。

AWS SAM での .NET 8 Lambda 関数の使用

AWS Serverless Application Model (AWS SAM) を使用して.NET 8 Lambda 関数を設定するには、次の手順を実行します。

- 開発マシンに前提条件をインストールする。
- .NET 8 Lambda 関数を AWS SAM テンプレートで定義します。
- AWS SAM CLI を使用してアプリケーションを構築します。

インストール条件

以下がインストールの前提条件となります。

- AWS SAMCLI

- .NET Core CLI
- Amazon.Lambda.Tools .NET Core Global Tool
- Docker

AWS SAM CLI のインストール

1. AWS SAM CLI がインストールされているかを確認するには、次のコマンドを実行します。

```
sam --version
```

2. AWS SAM CLI をインストールするには、「AWS SAM CLI のインストール」をご参照ください。
3. インストールされている AWS SAM CLI のバージョンをアップグレードするには、「AWS SAM CLI のアップグレード」を参照してください。

.NET Core CLI をインストールする

1. .NET Core CLI をダウンロードしてインストールするには、Microsoft の Web サイトで「.NET のダウンロード」を参照してください。
2. NET Core CLI の詳細については、「AWS Lambda 開発者ガイド」の「.NET Core CLI」を参照してください。

Amazon.Lambda.Tools .NET Core Global Tool をインストールする

1. 次のコマンドを実行します。

```
dotnet tool install -g Amazon.Lambda.Tools
```

2. ツールがすでにインストールされている場合には、次のコマンドを実行し、最新バージョンを使用していることを確認します。

```
dotnet tool update -g Amazon.Lambda.Tools
```

3. Amazon.Lambda.Tools の .NET Core Global Tool の詳細については、GitHub の AWS Extensions for .NET CLI リポジトリを参照してください。

Docker をインストールする

- ネイティブ AOT を構築に使用するには、Docker をインストールする必要があります。インストール手順については、「で使用する Docker のインストール AWS SAM CLI」を参照してください。

AWS SAM テンプレートで .NET 8 Lambda 関数を定義する

AWS SAM テンプレートで .NET 8 Lambda 関数を定義するには、次の手順を実行します。

1. 選択した開始ディレクトリから次のコマンドを実行します。

```
sam init
```

2. AWS Quick Start Templates を選択して開始テンプレートを選択します。
3. Hello World Example テンプレートを選択します。
4. n を入力して、最も人気のあるランタイムとパッケージタイプを使用しないことを選択します
5. ランタイムで、dotnet8 を選択します。
6. パッケージタイプで、Zip を選択します。
7. スターターテンプレートで、Hello World Example using native AOT を選択します。

Docker をインストールする

- ネイティブ AOT を構築に使用するには、Docker をインストールする必要があります。インストール手順については、「で使用する Docker のインストール AWS SAM CLI」を参照してください。

```
Resources:
HelloWorldFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: ./src/HelloWorldAot/
    Handler: bootstrap
    Runtime: dotnet8
    Architectures:
      - x86_64
    Events:
      HelloWorldAot:
```

```
        Type: Api
        Properties:
          Path: /hello
          Method: get
```

AWS SAM CLI を使用してアプリケーションを構築する

プロジェクトのルートディレクトリから、`sam build` を実行してアプリケーションの構築を開始します。.NET 8 プロジェクトファイルで `PublishAot` プロパティを定義済みの場合、AWS SAM CLI はネイティブ AOT コンパイルにより構築されます。`PublishAot` プロパティの詳細については、Microsoft の .NET ドキュメントの「ネイティブ AOT のデプロイ」を参照してください。

関数を構築する際、 AWS SAM CLI は Amazon.Lambda.Tools .NET Core Global Tool を使用する .NET Core CLI を呼び出します。

> **Note**
>
> 構築時にプロジェクトの同じディレクトリか親ディレクトリに `.sln` ファイルが存在する場合、`.sln` ファイルが格納されたディレクトリがコンテナにマウントされます。`.sln` ファイルが見つからない場合は、プロジェクトフォルダだけがマウントされます。そのためマルチプロジェクトアプリケーションを構築する場合は、`.sln` ファイルが存在することを確認してください。

詳細はこちら

.NET 8 Lambda 関数の構築の詳細については、「AWS Lambda の .NET 8 ランタイムの紹介」を参照してください。

sam build コマンドのリファレンスについては、「sam build」を参照してください。

を使用した Rust Lambda 関数の構築 Cargo Lambda の AWS SAM

この機能は のプレビューリリースにあり AWS SAM 、変更される可能性があります。

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) と Rust AWS Lambda 関数。

トピック

- 前提条件
- Rust Lambda 関数で使用する AWS SAM ための の設定
- 例

前提条件

Rust language

をインストールするには Rust「インストール」を参照してください。Rust ()Rust 言語ウェブサイト 。

Cargo Lambda

の AWS SAM CLI のインストールが必要です Cargo Lambda、 のサブコマンド Cargo。 インストール手順については、「」の「 のインストール」を参照してください。 Cargo Lambda ドキュメント 。

Docker

構築とテスト Rust Lambda 関数には、 が必要です。Docker。 インストール手順については、「」を参照してくださいDocker のインストール。

へのオプトイン AWS SAM CLI ベータ機能

この機能はプレビュー段階にあるため、次のいずれかの方法を使用してオプトインする必要があります。

1. 次の環境変数を使用します: `SAM_CLI_BETA_RUST_CARGO_LAMBDA=1`。
2. 次のコードを `samconfig.toml` ファイルに追加します。

```
[default.build.parameters]
beta_features = true
[default.sync.parameters]
beta_features = true
```

3. サポートされている を使用する場合は、 `--beta-features`オプションを使用します。 AWS SAM CLI コマンド。例:

```
$ sam build --beta-features
```

4. y で オプションを選択する AWS SAM CLI はオプトインするように促します。以下に例を示します。

```
$ sam build
Starting Build use cache
Build method "rust-cargolambda" is a beta feature.
Please confirm if you would like to proceed
You can also enable this beta feature with "sam build --beta-features". [y/N]: y
```

Rust Lambda 関数で使用する AWS SAM ための の設定

ステップ 1: AWS SAM テンプレートを設定する

以下を使用して AWS SAM テンプレートを設定します。

- Binary – オプション。テンプレートに複数の Rust Lambda 関数が含まれる場合に指定します。
- BuildMethod – `rust-cargolambda`.
- CodeUri – `Cargo.toml` ファイルへのパス。
- Handler – `bootstrap`。
- Runtime – `provided.al2`。

カスタムランタイムの詳細については、AWS Lambda 「 デベロッパーガイド」の「カスタム AWS Lambda ランタイム」を参照してください。

設定済みの AWS SAM テンプレートの例を次に示します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
      BuildProperties: function_a
    Properties:
      CodeUri: ./rust_app
      Handler: bootstrap
      Runtime: provided.al2
...
```

ステップ 2: を使用する AWS SAM CLI Rust Lambda 関数を使用する

任意の を使用する AWS SAM CLI AWS SAM テンプレートを使用した コマンド。詳細については、「AWS SAMCLI」を参照してください。

例

Hello World の例

この例では、 を使用してサンプル Hello World アプリケーションを構築します。Rust ランタイムとして

まず、sam init を使用して新しいサーバーレスアプリケーションを初期化します。インタラクティブフロー中に、[Hello World アプリケーション] を選択し、[Rust] ランタイムを選択します。

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

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        3 - Serverless API
        ...
Template: 1

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

Which runtime would you like to use?
        1 - aot.dotnet7 (provided.al2)
        2 - dotnet6
        3 - dotnet5.0
        ...
        18 - python3.7
        19 - python3.10
        20 - ruby2.7
        21 - rust (provided.al2)
Runtime: 21

Based on your selections, the only Package type available is Zip.
```

```
We will proceed to selecting the Package type as Zip.

Based on your selections, the only dependency manager available is cargo.
We will proceed copying the template using cargo.

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

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

Project name [sam-app]: hello-rust

    -----------------------
    Generating application:
    -----------------------
    Name: hello-rust
    Runtime: rust (provided.al2)
    Architectures: x86_64
    Dependency Manager: cargo
    Application Template: hello-world
    Output Directory: .
    Configuration file: hello-rust/samconfig.toml

    Next steps can be found in the README file at hello-rust/README.md


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

Hello World アプリケーションの構造を次に示します。

```
hello-rust
### README.md
### events
#   ### event.json
### rust_app
#   ### Cargo.toml
```

```
#   ### src
#       ### main.rs
### samconfig.toml
### template.yaml
```

AWS SAM テンプレートでは、Rust 関数は次のように定義されます。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
    Properties:
      CodeUri: ./rust_app
      Handler: bootstrap
      Runtime: provided.al2
      Architectures:
        - x86_64
      Events:
        HelloWorld:
          Type: Api
            Path: /hello
            Method: get
```

次に、`sam build` を実行してアプリケーションを構築し、デプロイの準備をします。の AWS SAM CLI は`.aws-sam`ディレクトリを作成し、そこでビルドアーティファクトを整理します。関数は を使用して構築されます。Cargo Lambda および は、 で実行可能バイナリとして保存されます`.aws-sam/build/HelloWorldFunction/bootstrap`。

> **Note**
>
> MacOS で sam local invoke コマンドを実行する予定がある場合は、呼び出す前に関数を別の方法で構築する必要があります。これを行うには、次のコマンドを使用します。
>
> - SAM_BUILD_MODE=debug sam build

このコマンドは、ローカルテストが行われる場合にのみ必要です。これは、デプロイ用に構築する場合は推奨されません。

```
hello-rust$ sam build
Starting Build use cache
Build method "rust-cargolambda" is a beta feature.
Please confirm if you would like to proceed
You can also enable this beta feature with "sam build --beta-features". [y/N]: y

Experimental features are enabled for this session.
Visit the docs page to learn more about the AWS Beta terms https://aws.amazon.com/
service-terms/.

Cache is invalid, running build and copying resources for following functions
 (HelloWorldFunction)
Building codeuri: /Users/.../hello-rust/rust_app runtime: provided.al2 metadata:
 {'BuildMethod': 'rust-cargolambda'} architecture: x86_64 functions: HelloWorldFunction
Running RustCargoLambdaBuilder:CargoLambdaBuild
Running RustCargoLambdaBuilder:RustCopyAndRename

Build Succeeded

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

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

次に、sam deploy --guided を使用してアプリケーションをデプロイします。

```
hello-rust$ sam deploy --guided

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

        Looking for config file [samconfig.toml] :  Found
```

```
        Reading default arguments  :  Success

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

        Looking for resources needed for deployment:

        ...

        Uploading to hello-rust/56ba6585d80577dd82a7eaaee5945c0b  817973 / 817973
 (100.00%)

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

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

        Uploading to hello-rust/a4fc54cb6ab75dd0129e4cdb564b5e89.template  1239 / 1239
 (100.00%)
```

```
Waiting for changeset to be created..

CloudFormation stack changeset
---------------------------------------------------------------------------------------------------
Operation                   LogicalResourceId          ResourceType
 Replacement
---------------------------------------------------------------------------------------------------
+ Add                       HelloWorldFunctionHelloW   AWS::Lambda::Permission    N/A

                            orldPermissionProd

...
---------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1681427201/
f0ef1563-5ab6-4b07-9361-864ca3de6ad6


Previewing CloudFormation changeset before deployment
=====================================================
Deploy this changeset? [y/N]: y

2023-04-13 13:07:17 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
---------------------------------------------------------------------------------------------------
ResourceStatus              ResourceType               LogicalResourceId
 ResourceStatusReason
---------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS          AWS::IAM::Role             HelloWorldFunctionRole     -

CREATE_IN_PROGRESS          AWS::IAM::Role             HelloWorldFunctionRole
 Resource creation
...
---------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
---------------------------------------------------------------------------------------------------
Outputs

---------------------------------------------------------------------------------------------------
```

```
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/hello-rust-
HelloWorldFunctionRole-10II2P13AUDUY

Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World function

Value               https://ggdxec9le9.execute-api.us-west-2.amazonaws.com/Prod/hello/


Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN

Value               arn:aws:lambda:us-west-2:012345678910:function:hello-rust-
HelloWorldFunction-
yk4HzGzYeZBj

---------------------------------------------------------------------------------------------


Successfully created/updated stack - hello-rust in us-west-2
```

テストするには、APIエンドポイントを使用して Lambda 関数を呼び出すことができます。

```
$ curl https://ggdxec9le9.execute-api.us-west-2.amazonaws.com/Prod/hello/
Hello World!%
```

関数をローカルでテストするには、まず関数の Architectures プロパティがローカルマシンと一致するようにします。

```
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # More info about Function Resource:
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Metadata:
```

```
      BuildMethod: rust-cargolambda # More info about Cargo Lambda: https://github.com/
cargo-lambda/cargo-lambda
    Properties:
      CodeUri: ./rust_app   # Points to dir of Cargo.toml
      Handler: bootstrap    # Do not change, as this is the default executable name
 produced by Cargo Lambda
      Runtime: provided.al2
      Architectures:
        - arm64
...
```

この例ではアーキテクチャを x86_64 から arm64 に変更したため、ビルドアーティファクトを更新するために sam build を実行します。その後、ローカルで関数を呼び出すために sam local invoke を実行します。

```
hello-rust$ sam local invoke
Invoking bootstrap (provided.al2)
Local image was not found.
Removing rapid images for repo public.ecr.aws/sam/emulation-provided.al2
Building
 image.....................................................................................
Using local image: public.ecr.aws/lambda/provided:al2-rapid-arm64.

Mounting /Users/.../hello-rust/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated, inside runtime container
START RequestId: fbc55e6e-0068-45f9-9f01-8e2276597fc6 Version: $LATEST
{"statusCode":200,"body":"Hello World!"}END RequestId:
 fbc55e6e-0068-45f9-9f01-8e2276597fc6
REPORT RequestId: fbc55e6e-0068-45f9-9f01-8e2276597fc6  Init Duration: 0.68 ms
 Duration: 130.63 ms     Billed Duration: 131 ms     Memory Size: 128 MB     Max Memory
 Used: 128 MB
```

単一 Lambda 関数プロジェクト

1 つの Rust Lambda 関数を含むサーバーレスアプリケーションの例を次に示します。

プロジェクトのディレクトリ構造:

```
.
### Cargo.lock
### Cargo.toml
### src
#   ### main.rs
```

```
### template.yaml
```

AWS SAM テンプレート：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
    Properties:
      CodeUri: ./
      Handler: bootstrap
      Runtime: provided.al2
...
```

複数の Lambda 関数プロジェクト

複数の Rust Lambda 関数を含むサーバーレスアプリケーションの例を次に示します。

プロジェクトのディレクトリ構造:

```
.
### Cargo.lock
### Cargo.toml
### src
#   ### function_a.rs
#   ### function_b.rs
### template.yaml
```

AWS SAM テンプレート：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  FunctionA:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
      BuildProperties:
```

```
        Binary: function_a
    Properties:
      CodeUri: ./
      Handler: bootstrap
      Runtime: provided.al2
  FunctionB:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
      BuildProperties:
        Binary: function_b
    Properties:
      CodeUri: ./
      Handler: bootstrap
      Runtime: provided.al2
```

Cargo.toml ファイル:

```
[package]
name = "test-handler"
version = "0.1.0"
edition = "2021"

[dependencies]
lambda_runtime = "0.6.0"
serde = "1.0.136"
tokio = { version = "1", features = ["macros"] }
tracing = { version = "0.1", features = ["log"] }
tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }

[[bin]]
name = "function_a"
path = "src/function_a.rs"

[[bin]]
name = "function_b"
path = "src/function_b.rs"
```

でのカスタムランタイムを使用した Lambda 関数の構築 AWS SAM

sam build コマンドを使用して、Lambda 関数に必要なカスタムランタイムを構築できます。Lambda 関数に Runtime: provided を指定することによって、その関数がカスタムランタイムを使用するように宣言します。

カスタムランタイムを構築するには、BuildMethod: makefile エントリを使用して Metadata リソース属性を宣言します。ユーザーは、ランタイムの build コマンドが含まれた build-*function-logical-id* フォームのビルドターゲットを宣言するカスタム makefile を提供します。Makefile は、必要に応じてカスタムランタイムをコンパイルして、ワークフローにおける後続のステップに必要となる適切な場所にビルドアーティファクトをコピーする責任を担います。Makefile の場所は、関数リソースの CodeUri プロパティによって指定され、Makefile と命名される必要があります。

例

例 1: Rust で記述された関数用のカスタムランタイム

> **Note**
>
> を使用して Lambda 関数を構築することをお勧めします Cargo Lambda。 詳細については、「」を参照してください[を使用した Rust Lambda 関数の構築 Cargo Lambda の AWS SAM](#)。

次の AWS SAM テンプレートは、Rust で記述された Lambda 関数にカスタムランタイムを使用する関数を宣言し、build-HelloRustFunctionビルドターゲットのコマンドを実行するsam buildように指示します。

```
Resources:
  HelloRustFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: HelloRust
      Handler: bootstrap.is.real.handler
      Runtime: provided
      MemorySize: 512
      CodeUri: .
    Metadata:
      BuildMethod: makefile
```

以下の makefile には、ビルドターゲットと実行されるコマンドが含まれています。CodeUri プロパティが . に設定されていることから、makefile がプロジェクトのルートディレクトリ (つまり、アプリケーションの AWS SAM テンプレートファイルと同じディレクトリ) に置かれている必要があることに注意してください。ファイル名は Makefile にする必要があります。

```
build-HelloRustFunction:
  cargo build --release --target x86_64-unknown-linux-musl
  cp ./target/x86_64-unknown-linux-musl/release/bootstrap $(ARTIFACTS_DIR)
```

上記の makefile で cargo build コマンドを実行するための開発環境のセットアップに関する詳細については、「Rust Runtime for AWS Lambda」ブログ記事を参照してください。

例 2: Python 3.12 用の makefile ビルダー (バンドルされたビルダーの代わりに使用)

バンドルされたビルダーに含まれていないライブラリやモジュールを使用したい場合があります。この例では、 makefile ビルダーを使用した Python3.12 ランタイムの AWS SAM テンプレートを示しています。

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.12
    Metadata:
      BuildMethod: makefile
```

以下の makefile には、ビルドターゲットと実行されるコマンドが含まれています。CodeUri プロパティが hello_world に設定されているため、makefile は hello_world サブディレクトリのルートに置き、ファイル名を Makefile にする必要があることに注意してください。

```
build-HelloWorldFunction:
  cp *.py $(ARTIFACTS_DIR)
  cp requirements.txt $(ARTIFACTS_DIR)
  python -m pip install -r requirements.txt -t $(ARTIFACTS_DIR)
  rm -rf $(ARTIFACTS_DIR)/bin
```

AWS SAM での Lambda レイヤーの構築

カスタム Lambda レイヤーの構築には AWS SAM を使用できます。Lambda レイヤーを使用すると、Lambda 関数からコードを抽出し、複数の Lambda 関数で再利用できます。(アプリケーション全体を構築するのではなく) Lambda レイヤーのみを構築すると、いくつかの点で利点があります。これにより、デプロイパッケージのサイズを縮小し、コア関数ロジックを依存関係から分離し、複数

の関数間で依存関係を共有できます。レイヤーの詳細については、AWS Lambda デベロッパーガイドの「AWS Lambda レイヤー」参照してください。

AWS SAM で Lambda レイヤーを構築する方法

> **Note**
>
> Lambda レイヤーを構築する前に、まず AWS SAM テンプレートに Lambda レイヤーを記述する必要があります。これを行う方法についての詳細と例については、「AWS SAM で Lambda レイヤーを使用して効率を向上させる」を参照してください。

カスタムレイヤーを構築するには、それを AWS Serverless Application Model (AWS SAM) テンプレートファイルで宣言し、`BuildMethod` エントリがある `Metadata` リソース属性セクションを含めます。`BuildMethod` に有効な値は、AWS Lambda ランタイムの識別子、または `makefile` です。`BuildArchitecture` エントリを含めて、レイヤーがサポートする命令セットアーキテクチャを指定します。`BuildArchitecture` の有効値は、Lambda 命令セットアーキテクチャです。

`makefile` を指定する場合は、レイヤーの build コマンドが含まれた `build-`*`layer-logical-id`* フォームのビルドターゲットを宣言するカスタム makefile を提供します。Makefile は、必要に応じてレイヤーをコンパイルして、ワークフローにおける後続のステップに必要となる適切な場所にビルドアーティファクトをコピーする責任を担います。Makefile の場所は、レイヤーリソースの `ContentUri` プロパティによって指定され、`Makefile` と命名される必要があります。

> **Note**
>
> カスタムレイヤーを作成するときは、AWS Lambda が環境変数に頼ってレイヤーコードを見つけます。Lambda ランタイムは、レイヤーコードがコピーされた `/opt` ディレクトリにパスを含めます。カスタムレイヤーコードを見つけられるようにするため、プロジェクトのビルドアーティファクトのフォルダ構造が、ランタイムが期待するフォルダ構造と一致している必要があります。
> 例えば、Python ではコードを `python/` サブディレクトリに置くことができます。NodeJS では、コードを `nodejs/node_modules/` サブディレクトリに置くことができます。
> 詳細については、AWS Lambda デベロッパーガイドの「ライブラリの依存関係をレイヤーに含める」を参照してください。

以下は、`Metadata` リソース属性セクションの例です。

```
    Metadata:
      BuildMethod: python3.8
      BuildArchitecture: arm64
```

> **Note**
>
> Metadata リソース属性セクションを含めない場合、AWS SAM はレイヤーを構築しません。その代わりに、レイヤーリソースの CodeUri プロパティで指定された場所からビルドアーティファクトをコピーします。詳細については、AWS::Serverless::LayerVersion リソースタイプの「ContentUri」プロパティを参照してください。

Metadata リソース属性セクションを含めると、sam build コマンドを使用して、レイヤーを独立したオブジェクト、または AWS Lambda 関数の依存関係として構築することができます。

- 独立したオブジェクトとして。レイヤーに対するコード変更をローカルでテストしており、アプリケーション全体を構築する必要がないときなど、レイヤーオブジェクトだけを構築したい場合があります。レイヤーを個別に構築するには、sam build *layer-logical-id* コマンドでレイヤーリソースを指定します。
- Lambda 関数の依存関係として。同じ AWS SAM テンプレートファイルにある Lambda 関数の Layers プロパティにレイヤーの論理 ID を含めると、レイヤーはその Lambda 関数の依存関係になります。そのレイヤーに BuildMethod エントリがある Metadata リソース属性セクションも含める場合は、sam build コマンドを使用してアプリケーション全体を構築する、または sam build *function-logical-id* コマンドで関数リソースを指定することによってレイヤーを構築します。

例

テンプレート例 1: Python 3.9 ランタイム環境に対してレイヤーを構築する

以下の AWS SAM テンプレート例では、Python 3.9 ランタイム環境に対してレイヤーを構築します。

```
Resources:
  MyLayer:
    Type: AWS::Serverless::LayerVersion
    Properties:
      ContentUri: my_layer
```

```
      CompatibleRuntimes:
        - python3.9
    Metadata:
      BuildMethod: python3.9   # Required to have AWS SAM build this layer
```

テンプレート例 2: カスタム makefile を使用してレイヤーを構築する

以下の AWS SAM テンプレート例は、カスタム `makefile` を使用してレイヤーを構築します。

```
Resources:
  MyLayer:
    Type: AWS::Serverless::LayerVersion
    Properties:
      ContentUri: my_layer
      CompatibleRuntimes:
        - python3.8
    Metadata:
      BuildMethod: makefile
```

以下の `makefile` には、ビルドターゲットと実行されるコマンドが含まれています。`ContentUri` プロパティが `my_layer` に設定されているため、makefile は `my_layer` サブディレクトリのルートに置き、ファイル名を `Makefile` にする必要があることに注意してください。また、AWS Lambda がレイヤーコードを見つけられるように、ビルドアーティファクトが `python/` サブディレクトリにコピーされることにも注意してください。

```
build-MyLayer:
  mkdir -p "$(ARTIFACTS_DIR)/python"
  cp *.py "$(ARTIFACTS_DIR)/python"
  python -m pip install -r requirements.txt -t "$(ARTIFACTS_DIR)/python"
```

sam build コマンドの例

以下の `sam build` コマンドは、`Metadata` リソース属性セクションが含まれたレイヤーを構築します。

```
# Build the 'layer-logical-id' resource independently
$ sam build layer-logical-id

# Build the 'function-logical-id' resource and layers that this function depends on
$ sam build function-logical-id
```

```
# Build the entire application, including the layers that any function depends on
$ sam build
```

AWS SAM を使用してサーバーレスアプリケーションをテストする

アプリケーションを記述して構築したら、アプリケーションが正しく機能することをテストできます。AWS SAM コマンドラインインターフェイス (CLI) では、サーバーレスアプリケーションを AWS クラウドにアップロードする前にローカルでテストできます。アプリケーションをテストすることで、アプリケーションの機能、信頼性、パフォーマンスをすべて確認しながら、対処する必要がある問題 (バグ) を特定できます。

このセクションでは、アプリケーションをテストするために実行できる一般的な方法に関するガイダンスを提供します。このセクションのトピックは、主に AWS クラウドにデプロイする前に実行できるローカルテストに焦点を当てています。デプロイ前にテストを行うことで、問題を積極的に特定し、デプロイの問題に関連する不要なコストを削減するのに役立ちます。このセクションの各トピックでは、実行できるテストについて説明し、それを使用する利点について説明します。また、テストの実行方法を示す例についても示します。アプリケーションをテストすることで、見つかった問題をデバッグする準備が整います。

トピック

- sam local コマンドを使用したテストの概要
- を使用して Lambda 関数をローカルに呼び出す AWS SAM
- を使用して API Gateway をローカルで実行する AWS SAM
- sam remote test-event を使用したクラウドテストの概要
- を使用したクラウドでのテストの概要 sam remote invoke
- AWS SAM を使用してローカル統合テストを自動化する
- AWS SAM を使用してサンプルのイベントペイロードを生成する

sam local コマンドを使用したテストの概要

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local` コマンドを使用して、サーバーレスアプリケーションをローカルでテストします。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください。

前提条件

`sam local` を使用するには、次を実行して AWS SAM CLI をインストールします。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

`sam local` を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.
- でのデプロイの概要 AWS SAM.

sam local コマンドの使用

`sam local` コマンドとそのサブコマンドのいずれかを使用して、アプリケーションについて、さまざまな種類のローカルテストを実行します。

```
$ sam local <subcommand>
```

各サブコマンドの詳細については、次を参照してください。

- の紹介 sam local generate-event – ローカルテストのために AWS のサービス イベントを生成します。
- sam local invoke の概要 – AWS Lambda 関数の 1 回限りの呼び出しをローカルで開始します。
- sam local start-api の概要 – ローカル HTTP サーバーを使用して Lambda 関数を実行します。
- の概要 sam local start-lambda – AWS CLI または SDK で使用するローカル HTTP サーバーを使用して Lambda 関数を実行します。

を使用したテストの概要 sam local generate-event

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local generate-event`サブコマンドを使用して、サポートされている のイベントペイロードサンプルを生成します AWS のサービス。その後、これらのイベントを変更して、テストのためにローカルリソースに渡すことができます。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- `sam local generate-event` コマンドオプションのリストについては、「sam local generate-event」を参照してください。

イベントは、 がアクションまたはタスク AWS のサービス を実行するときに生成されるJSONオブジェクトです。これらのイベントには、処理されたデータやイベントのタイムスタンプなどの特定の情報が含まれています。ほとんどの AWS のサービス はイベントを生成し、各サービスのイベントはそのサービスに合わせて独自にフォーマットされます。

1 つのサービスによって生成されたイベントは、イベントソースとして他のサービスに渡されます。例えば、Amazon Simple Storage Service (Amazon S3) バケットに配置された項目はイベントを生成できます。このイベントは、データをさらに処理する AWS Lambda 関数のイベントソースとして使用できます。

で生成するイベント`sam local generate-event`は、 AWS サービスによって作成された実際のイベントと同じ構造でフォーマットされます。これらのイベントの内容を変更し、それらを使用してアプリケーション内のリソースをテストできます。

前提条件

を使用するには`sam local generate-event`、 をインストールします。 AWS SAM CLI 以下を完了します。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

`sam local generate-event` を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.
- でのデプロイの概要 AWS SAM.

サンプルイベントを生成する

を使用する AWS SAM CLI `sam local generate-event` サポートされている のイベントを生成するサブコマンド AWS のサービス。

サポートされている のリストを表示するには AWS のサービス

1. 下記を実行します。

```
$ sam local generate-event
```

2. サポートされている のリスト AWS のサービス が表示されます。以下に例を示します。

```
$ sam local generate-event
...
Commands:
  alb
  alexa-skills-kit
  alexa-smart-home
  apigateway
  appsync
  batch
  cloudformation
  ...
```

ローカルイベントを生成するには

1. sam local generate-event を実行して、サポートされているサービス名を指定します。これにより、生成できるイベントタイプのリストが表示されます。以下に例を示します。

```
$ sam local generate-event s3

Usage: sam local generate-event s3 [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  batch-invocation  Generates an Amazon S3 Batch Operations Invocation Event
  delete            Generates an Amazon S3 Delete Event
  put               Generates an Amazon S3 Put Event
```

2. サンプルイベントを生成するには、サービスとイベントタイプを指定して sam local generate-event を実行します。

```
$ sam local generate-event <service> <event>
```

以下に例を示します。

```
$ sam local generate-event s3 put
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test/key",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
```

```
  ]
}
```

これらのサンプルイベントにはプレースホルダー値が含まれています。これらの値を変更して、アプリケーション内の実際のリソースを参照したり、ローカルテストに役立つ値を参照したりできます。

サンプルイベントを変更するには

1. コマンドプロンプトでサンプルイベントを変更できます。オプションを確認するには、次のコマンドを実行します。

   ```
   $ sam local generate-event <service> <event> --help
   ```

   以下に例を示します。

   ```
   $ sam local generate-event s3 put --help

   Usage: sam local generate-event s3 put [OPTIONS]

   Options:
     --region TEXT       Specify the region name you'd like, otherwise the
                         default = us-east-1
     --partition TEXT    Specify the partition name you'd like, otherwise the
                         default = aws
     --bucket TEXT       Specify the bucket name you'd like, otherwise the
                         default = example-bucket
     --key TEXT          Specify the key name you'd like, otherwise the default =
                         test/key
     --debug             Turn on debug logging to print debug message generated
                         by AWS SAM CLI and display timestamps.
     --config-file TEXT  Configuration file containing default parameter values.
                         [default: samconfig.toml]
     --config-env TEXT   Environment name specifying default parameter values in
                         the configuration file.  [default: default]
     -h, --help          Show this message and exit.
   ```

2. コマンドプロンプトでこれらのオプションのいずれかを使用して、サンプルイベントペイロードを変更します。以下に例を示します。

   ```
   $ sam local generate-event s3 put--bucket amzn-s3-demo-bucket
   ```

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test/key",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

生成されたイベントをローカルテストに使用する

生成されたイベントをローカルで保存し、他の sam local サブコマンドを使用してテストします。

生成したイベントをローカルで保存するには

- 下記を実行します。

```
$ sam local generate-event <service> <event> <event-option> > <filename.json>
```

プロジェクトの events フォルダに s3.json ファイルとして保存されるイベントの例を次に示します。

```
sam-app$ sam local generate-event s3 put --bucket amzn-s3-demo-bucket > events/
s3.json
```

生成されたイベントをローカルテストに使用するには

- --event オプションを使用して、他の sam local サブコマンドでイベントを渡します。

s3.json イベントを使用して Lambda 関数をローカルで呼び出す例を次に示します。

```
sam-app$ sam local invoke --event events/s3.json S3JsonLoggerFunction

Invoking src/handlers/s3-json-logger.s3JsonLoggerHandler (nodejs18.x)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/nodejs:18-rapid-x86_64.

Mounting /Users/.../sam-app/.aws-sam/build/S3JsonLoggerFunction as /var/
task:ro,delegated, inside runtime container
START RequestId: f4f45b6d-2ec6-4235-bc7b-495ec2ae0128 Version: $LATEST
END RequestId: f4f45b6d-2ec6-4235-bc7b-495ec2ae0128
REPORT RequestId: f4f45b6d-2ec6-4235-bc7b-495ec2ae0128  Init Duration: 1.23 ms
 Duration: 9371.93 ms      Billed Duration: 9372 ms        Memory Size: 128 MB
 Max Memory Used: 128 MB
```

詳細

すべての `sam local generate-event` オプションのリストについては、「sam local generate-event」を参照してください。

`sam local` の使用のデモについては、「ローカル開発用のAWS SAM 」を参照してください。SAM シリーズが の Serverless Land Sessions でのローカル開発環境からの AWS クラウド リソースのテスト YouTube.

sam local invoke を使用したテストの概要

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local invoke` サブコマンドを使用して、AWS Lambda 関数の 1 回限りの呼び出しをローカルで開始します。

- AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください。
- `sam local invoke` コマンドオプションのリストについては、「sam local invoke」を参照してください。
- 一般的な開発ワークフローでの `sam local invoke` の使用例については、「ステップ 7: (オプション) アプリケーションをローカルでテストする」を参照してください。

前提条件

`sam local invoke` を使用するには、次を実行して AWS SAM CLI をインストールします。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

`sam local invoke` を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.
- でのデプロイの概要 AWS SAM.

Lambda 関数をローカルで呼び出す

`sam local invoke` を実行すると、AWS SAM CLI は現在の作業ディレクトリがプロジェクトのルートディレクトリであると想定します。AWS SAM CLI は最初に `.aws-sam` サブフォルダ内の `template.[yaml|yml]` ファイルを検索します。見つからない場合、AWS SAM CLI は現在の作業ディレクトリ内で `template.[yaml|yml]` ファイルを探します。

Lambda 関数をローカルで呼び出すには

1. プロジェクトのルートディレクトリから次のコマンドを実行します。

   ```
   $ sam local invoke <options>
   ```

2. アプリケーションに複数の関数が含まれている場合は、関数の論理 ID を指定します。以下に例を示します。

   ```
   $ sam local invoke HelloWorldFunction
   ```

3. AWS SAM CLI は、Docker を使用してローカルコンテナに関数を構築します。その後、関数を呼び出し、関数のレスポンスを出力します。

   以下に例を示します。

   ```
   $ sam local invoke
   Invoking app.lambda_handler (python3.9)
   Local image is out of date and will be updated to the latest runtime. To skip this,
    pass in the parameter --skip-pull-image
   Building
    image.....................................................................................
   Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

   Mounting /Users/.../sam-app/.aws-sam/build/HelloWorldFunction as /var/
   task:ro,delegated, inside runtime container
   START RequestId: 64bf7e54-5509-4762-a97c-3d740498d3df Version: $LATEST
   END RequestId: 64bf7e54-5509-4762-a97c-3d740498d3df
   REPORT RequestId: 64bf7e54-5509-4762-a97c-3d740498d3df  Init Duration: 1.09 ms
     Duration: 608.42 ms       Billed Duration: 609 ms Memory Size: 128 MB     Max
    Memory Used: 128 MB
   {"statusCode": 200, "body": "{\"message\": \"hello world\"}"}%
   ```

ログを管理する

`sam local invoke` を使用する場合、Lambda 関数のランタイム出力 (ログなど) は `stderr` に出力され、Lambda 関数の結果は `stdout` に出力されます。

基本的な Lambda 関数の例を次に示します。

```
def handler(event, context):
    print("some log") # this goes to stderr
    return "hello world" # this goes to stdout
```

これらの標準出力は保存できます。以下に例を示します。

```
$ sam local invoke 1> stdout.log
...

$ cat stdout.log
"hello world"

$ sam local invoke 2> stderr.log
...

$ cat stderr.log
Invoking app.lambda_handler (python3.9)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.
Mounting /Users/.../sam-app/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated, inside runtime container
START RequestId: 0b46e646-3bdf-4b58-8beb-242d00912c46 Version: $LATEST
some log
END RequestId: 0b46e646-3bdf-4b58-8beb-242d00912c46
REPORT RequestId: 0b46e646-3bdf-4b58-8beb-242d00912c46  Init Duration: 0.91 ms
 Duration: 589.19 ms Billed Duration: 590 ms Memory Size: 128 MB Max Memory Used: 128
 MB
```

これらの標準出力を使用して、ローカル開発プロセスをさらに自動化できます。

オプション

カスタムイベントを渡して Lambda 関数を呼び出す

イベントを Lambda 関数に渡すには、`--event` オプションを使用します。以下に例を示します。

```
$ sam local invoke --event events/s3.json S3JsonLoggerFunction
```

sam local generate-event サブコマンドを使用してイベントを作成できます。詳細については、「を使用したテストの概要 sam local generate-event」を参照してください。

Lambda 関数を呼び出すときに環境変数を渡す

Lambda 関数で環境変数を使用する場合は、ローカルテスト中に --env-vars オプションを使用して環境変数を渡すことができます。これは、クラウドで既にデプロイされているアプリケーション内のサービスを使用して、Lambda 関数をローカルでテストするための優れた方法です。以下に例を示します。

```
$ sam local invoke --env-vars locals.json
```

テンプレートまたは関数を指定する

AWS SAM CLI が参照するテンプレートを指定するには、--template オプションを使用します。AWS SAM CLI は、その AWS SAM テンプレートとそれがポイントするリソースのみをロードします。

ネストされたアプリケーションまたはスタックの関数を呼び出すには、関数の論理 ID とともに、アプリケーションまたはスタックの論理 ID を指定します。以下に例を示します。

```
$ sam local invoke StackLogicalId/FunctionLogicalId
```

Terraform プロジェクトから Lambda 関数をテストする

--hook-name オプションを使用して、Terraform プロジェクトから Lambda 関数をローカルでテストします。詳細については、「の使用 AWS SAM CLI with Terraform ローカルデバッグとテスト用」を参照してください。

以下に例を示します。

```
$ sam local invoke --hook-name terraform --beta-features
```

ベストプラクティス

アプリケーションに .aws-sam を実行している sam build ディレクトリがある場合は、関数コードを更新するたびに必ず sam build を実行してください。その後、更新された関数コードをローカルでテストするために sam local invoke を実行します。

ローカルテストは、クラウドにデプロイする前に迅速な開発とテストを行うための優れたソリューションです。ただし、ローカルテストでは、クラウド内のリソース間の許可など、すべてが検証されるわけではありません。可能な限り、アプリケーションをクラウドでテストします。クラウドテストのワークフローを高速化するために sam sync を使用することをお勧めします。

例

Amazon API Gateway サンプルイベントを生成し、それを使用してローカルで Lambda 関数を呼び出す

まず、API Gateway HTTP API イベントペイロードを生成し、events フォルダに保存します。

```
$ sam local generate-event apigateway http-api-proxy > events/apigateway_event.json
```

次に、イベントからパラメータ値を返すように Lambda 関数を変更します。

```
def lambda_handler(event, context):
    print("HelloWorldFunction invoked")
    return {
        "statusCode": 200,
        "body": json.dumps({
            "message": event['queryStringParameters']['parameter2'],
        }),
    }
```

その後、ローカルで Lambda 関数を呼び出し、カスタムイベントを提供します。

```
$ sam local invoke --event events/apigateway_event.json

Invoking app.lambda_handler (python3.9)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

Mounting /Users/...sam-app/.aws-sam/build/HelloWorldFunction as /var/task:ro,delegated,
 inside runtime container
START RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8 Version: $LATEST
some log
END RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8
REPORT RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8  Init Duration: 1.63 ms
 Duration: 564.07 ms       Billed Duration: 565 ms Memory Size: 128 MB     Max Memory
 Used: 128 MB
```

```
{"statusCode": 200, "body": "{\"message\": \"value\"}"}%
```

Lambda 関数をローカルで呼び出すときに環境変数を渡す

このアプリケーションには、Amazon DynamoDB テーブル名の環境変数を使用する Lambda 関数が含まれています。AWS SAM テンプレートで定義されている関数の例を次に示します。

```
AWSTemplateFormatVersion: 2010-09-09
Transform: AWS::Serverless-2016-10-31
...
Resources:
  getAllItemsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: src/get-all-items.getAllItemsHandler
      Description: get all items
      Policies:
        - DynamoDBReadPolicy:
            TableName: !Ref SampleTable
      Environment:
        Variables:
          SAMPLE_TABLE: !Ref SampleTable
...
```

Lambda 関数をクラウド内の DynamoDB テーブルとインタラクションさせながら、ローカルでテストしたいと考えています。これを実行するには、環境変数ファイルを作成し、プロジェクトのルートディレクトリに `locals.json` として保存します。ここで `SAMPLE_TABLE` に指定される値は、クラウド内の DynamoDB テーブルを参照します。

```
{
    "getAllItemsFunction": {
        "SAMPLE_TABLE": "dev-demo-SampleTable-1U991234LD5UM98"
    }
}
```

次に、`sam local invoke` を実行して、`--env-vars` オプションを使用して環境変数を渡します。

```
$ sam local invoke getAllItemsFunction --env-vars locals.json

Mounting /Users/...sam-app/.aws-sam/build/HelloWorldFunction as /var/task:ro,delegated,
 inside runtime container
```

```
START RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8 Version: $LATEST
some log
END RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8
REPORT RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8  Init Duration: 1.63 ms
 Duration: 564.07 ms       Billed Duration: 565 ms Memory Size: 128 MB     Max Memory
 Used: 128 MB
{"statusCode":200,"body":"{}"}
```

詳細はこちら

すべての `sam local invoke` オプションのリストについては、「sam local invoke」を参照してください。

`sam local` の使用のデモについては、「ローカル開発用の AWS SAM」を参照してください。YouTube での Serverless Land の SAM を使用したセッションのシリーズのローカル開発環境からの AWS クラウド リソースのテスト。

sam local start-api を使用したテストの概要

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local start-api` サブコマンドを使用して、AWS Lambda 関数をローカルで実行し、ローカル HTTP サーバーホストを通じてテストします。このタイプのテストは、Amazon API Gateway エンドポイントによって呼び出される Lambda 関数に役立ちます。

- AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください。
- `sam local start-api` コマンドオプションのリストについては、「sam local start-api」を参照してください。
- 一般的な開発ワークフローでの `sam local start-api` の使用例については、「ステップ 7: (オプション) アプリケーションをローカルでテストする」を参照してください。

前提条件

`sam local start-api` を使用するには、次を実行して AWS SAM CLI をインストールします。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

`sam local start-api` を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.
- でのデプロイの概要 AWS SAM.

sam local start-api の使用

`sam local start-api` を実行すると、AWS SAM CLI は現在の作業ディレクトリがプロジェクトのルートディレクトリであると想定します。AWS SAM CLI は最初に `.aws-sam` サブフォルダ内の `template.[yaml|yml]` ファイルを検索します。見つからない場合、AWS SAM CLI は現在の作業ディレクトリ内で `template.[yaml|yml]` ファイルを探します。

ローカル HTTP サーバーを起動するには

1. プロジェクトのルートディレクトリから次のコマンドを実行します。

```
$ sam local start-api <options>
```

2. AWS SAM CLI は、Lambda 関数をローカル Docker コンテナに構築します。その後、HTTP サーバーエンドポイントのローカルアドレスを出力します。以下に例を示します。

```
$ sam local start-api

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

Mounting /Users/.../sam-app/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated, inside runtime container
Containers Initialization is done.
Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET]
You can now browse to the above endpoints to invoke your functions. You do not
 need to restart/reload SAM CLI while working on your functions, changes will be
 reflected instantly/automatically. If you used sam build before running local
 commands, you will need to re-run sam build for the changes to be picked up. You
 only need to restart SAM CLI if you update your AWS SAM template
2023-04-12 14:41:05 WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:3000
```

3. Lambda 関数は、ブラウザまたはコマンドプロンプトを通じて呼び出すことができます。以下に例を示します。

```
sam-app$ curl http://127.0.0.1:3000/hello
{"message": "Hello world!"}%
```

4. Lambda 関数のコードを変更するときは、ローカル HTTP サーバーを更新するために次を考慮してください。

   - アプリケーションに .aws-sam ディレクトリがなく、関数がインタープリタ型言語を使用している場合、AWS SAM CLI は新しいコンテナを作成してホストすることで関数を自動的に更新します。
   - アプリケーションに .aws-sam ディレクトリがある場合は、関数を更新するために sam build を実行する必要があります。その後、関数をホストするために再度 sam local start-api を実行します。
   - 関数がコンパイル型言語を使用している場合、またはプロジェクトで複雑なパッケージ化サポートが必要な場合は、独自のビルドソリューションを実行して関数を更新します。その後、関数をホストするために再度 sam local start-api を実行します。

Lambda オーソライザーを使用する Lambda 関数

> **Note**
>
> この機能は AWS SAM CLI バージョン 1.80.0 の新機能です。アップグレードするには、AWS SAM CLI のアップグレード を参照してください。

Lambda オーソライザーを使用する Lambda 関数の場合、AWS SAM CLI は Lambda 関数エンドポイントを呼び出す前に Lambda オーソライザーを自動的に呼び出します。

Lambda オーソライザーを使用する関数のローカル HTTP サーバーを起動する例を次に示します。

```
$ sam local start-api
2023-04-17 15:02:13 Attaching import module proxy for analyzing dynamic imports

AWS SAM CLI does not guarantee 100% fidelity between authorizers locally
and authorizers deployed on AWS. Any application critical behavior should
be validated thoroughly before deploying to production.
```

```
Testing application behaviour against authorizers deployed on AWS can be done using the
 sam sync command.

Mounting HelloWorldFunction at http://127.0.0.1:3000/authorized-request [GET]
You can now browse to the above endpoints to invoke your functions. You do not need
 to restart/reload SAM CLI while working on your functions, changes will be reflected
 instantly/automatically. If you used sam build before running local commands, you will
 need to re-run sam build for the changes to be picked up. You only need to restart SAM
 CLI if you update your AWS SAM template
2023-04-17 15:02:13 WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:3000
2023-04-17 15:02:13 Press CTRL+C to quit
```

ローカル HTTP サーバー経由で Lambda 関数エンドポイントを呼び出すと、 AWS SAM CLI は最初に Lambda オーソライザーを呼び出します。認可が成功すると、AWS SAM CLI は Lambda 関数エンドポイントを呼び出します。以下に例を示します。

```
$ curl http://127.0.0.1:3000/authorized-request --header "header:my_token"
{"message": "from authorizer"}%

Invoking app.authorizer_handler (python3.8)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.8-rapid-x86_64.

Mounting /Users/.../sam-app/... as /var/task:ro,delegated, inside runtime container
START RequestId: 38d3b472-a2c8-4ea6-9a77-9b386989bef0 Version: $LATEST
END RequestId: 38d3b472-a2c8-4ea6-9a77-9b386989bef0
REPORT RequestId: 38d3b472-a2c8-4ea6-9a77-9b386989bef0    Init Duration: 1.08 ms
 Duration: 628.26 msBilled Duration: 629 ms    Memory Size: 128 MB    Max Memory Used:
 128 MB
Invoking app.request_handler (python3.8)
Using local image: public.ecr.aws/lambda/python:3.8-rapid-x86_64.

Mounting /Users/.../sam-app/... as /var/task:ro,delegated, inside runtime container
START RequestId: fdc12255-79a3-4365-97e9-9459d06446ff Version: $LATEST
END RequestId: fdc12255-79a3-4365-97e9-9459d06446ff
REPORT RequestId: fdc12255-79a3-4365-97e9-9459d06446ff    Init Duration: 0.95 ms
 Duration: 659.13 msBilled Duration: 660 ms    Memory Size: 128 MB    Max Memory Used:
 128 MB
No Content-Type given. Defaulting to 'application/json'.
```

```
2023-04-17 15:03:03 127.0.0.1 - - [17/Apr/2023 15:03:03] "GET /authorized-request
 HTTP/1.1" 200 -
```

オプション

コンテナを継続的に再利用してローカル関数の呼び出しを高速化する

デフォルトでは、ローカル HTTP サーバー経由で関数が呼び出されるたびに、AWS SAM CLI は新しいコンテナを作成します。関数呼び出しのためにコンテナを自動的に再利用する `--warm-containers` オプションを使用します。これは、AWS SAM CLI がローカル呼び出しのために Lambda 関数を準備するのにかかる時間を短縮します。`eager` または `lazy` 引数を指定することで、このオプションをさらにカスタマイズできます。

- `eager` – 起動時にすべての関数のコンテナがロードされ、呼び出し間で保持されます。
- `lazy` – 各関数が初めて呼び出される場合に限り、コンテナがロードされます。その後、追加の呼び出しのために永続化されます。

以下に例を示します。

```
$ sam local start-api --warm-containers eager
```

`--warm-containers` を使用して Lambda 関数コードを変更する場合:

- アプリケーションに `.aws-sam` ディレクトリがある場合は、`sam build` を実行してアプリケーションのビルドアーティファクト内の関数コードを更新します。
- コードの変更が検出されると、AWS SAM CLI は Lambda 関数コンテナを自動的にシャットダウンします。
- 関数を再度呼び出すと、AWS SAM CLI は新しいコンテナを自動的に作成します。

Lambda 関数に使用するコンテナイメージを指定する

デフォルトでは、AWS SAM CLI は Amazon Elastic Container Registry (Amazon ECR) の Lambda ベースイメージを使用して関数をローカルで呼び出します。カスタムコンテナイメージを参照するには、`--invoke-image` オプションを使用します。以下に例を示します。

```
$ sam local start-api --invoke-image public.ecr.aws/sam/emu-python3.8
```

カスタムコンテナイメージで使用する関数を指定できます。以下に例を示します。

```
$ sam local start-api --invoke-image Function1=amazon/aws/sam-cli-emulation-image-python3.8
```

ローカルでテストするテンプレートを指定する

AWS SAM CLI が参照するテンプレートを指定するには、--template オプションを使用します。AWS SAM CLI は、その AWS SAM テンプレートとそれがポイントするリソースのみをロードします。以下に例を示します。

```
$ sam local start-api --template myTemplate.yaml
```

Lambda 関数のホスト開発環境を指定する

デフォルトでは、sam local start-api サブコマンドは IP アドレス 127.0.0.1 の localhost を使用して HTTP サーバーを作成します。ローカル開発環境がローカルマシンから分離されている場合は、これらの値をカスタマイズできます。

--container-host オプションを使用してホストを指定します。以下に例を示します。

```
$ sam local start-api --container-host host.docker.internal
```

--container-host-interface オプションを使用して、コンテナポートがバインドするホストネットワークの IP アドレスを指定します。以下に例を示します。

```
$ sam local start-api --container-host-interface 0.0.0.0
```

ベストプラクティス

アプリケーションに .aws-sam を実行している sam build ディレクトリがある場合は、関数コードを更新するたびに必ず sam build を実行してください。その後、更新された関数コードをローカルでテストするために sam local start-api を実行します。

ローカルテストは、クラウドにデプロイする前に迅速な開発とテストを行うための優れたソリューションです。ただし、ローカルテストでは、クラウド内のリソース間の許可など、すべてが検証されるわけではありません。可能な限り、アプリケーションをクラウドでテストします。クラウドテストのワークフローを高速化するために sam sync を使用することをお勧めします。

詳細はこちら

すべての sam local start-api オプションのリストについては、「sam local start-api」を参照してください。

を使用したテストの概要 sam local start-lambda

を使用する AWS SAM CLI および を通じて Lambda AWS CLI 関数を呼び出すsam local start-lambdaサブコマンドSDKs。このコマンドは、Lambda をエミュレートするローカルエンドポイントを起動します。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- sam local start-lambda コマンドオプションのリストについては、「sam local start-lambda」を参照してください。

前提条件

を使用するにはsam local start-lambda、 をインストールします。 AWS SAM CLI 以下を完了します。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

sam local start-lambda を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.
- でのデプロイの概要 AWS SAM.

sam local start-lambda の使用

を実行するとsam local start-lambda、 AWS SAM CLI は、現在の作業ディレクトリがプロジェクトのルートディレクトリであることを前提としています。の AWS SAM CLI は、まず.aws-samサブフォルダ内のtemplate.[yaml|yml]ファイルを検索します。見つからない場合は、 AWS SAM CLI は、現在の作業ディレクトリ内のtemplate.[yaml|yml]ファイルを検索します。

sam local start-lambda を使用するには

1. プロジェクトのルートディレクトリから次のコマンドを実行します。

```
$ sam local start-lambda <options>
```

2. の AWS SAM CLI はローカルで Lambda 関数を構築します Docker コンテナ。その後、ローカルアドレスをHTTPサーバーエンドポイントに出力します。以下に例を示します。

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

Mounting /Users/.../sam-app/hello_world as /var/task:ro,delegated, inside runtime
 container
Containers Initialization is done.
Starting the Local Lambda Service. You can now invoke your Lambda Functions defined
 in your template through the endpoint.
2023-04-13 07:25:43 WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:3001
2023-04-13 07:25:43 Press CTRL+C to quit
```

3. AWS CLI または SDKsを使用して、Lambda 関数をローカルで呼び出します。

   AWS CLIを使用した例を次に示します。

```
$ aws lambda invoke --function-name "HelloWorldFunction" --endpoint-
url "http://127.0.0.1:3001" --no-verify-ssl out.txt

StatusCode: 200
(END)
```

   以下は、 を使用した例です。 AWS SDK を Python:

```
import boto3
from botocore.config import Config
from botocore import UNSIGNED

lambda_client = boto3.client('lambda',
                             endpoint_url="http://127.0.0.1:3001",
```

```
                                use_ssl=False,
                                verify=False,
                                config=Config(signature_version=UNSIGNED,
                                              read_timeout=1,
                                              retries={'max_attempts': 0}
                                              )
                               )
lambda_client.invoke(FunctionName="HelloWorldFunction")
```

オプション

テンプレートを指定する

のテンプレートを指定するには AWS SAM CLI を参照するには、 `--template`オプションを使用します。の AWS SAM CLI は、その AWS SAM テンプレートとそれが指しているリソースのみをロードします。以下に例を示します。

```
$ sam local start-lambda --template myTemplate.yaml
```

ベストプラクティス

アプリケーションに `.aws-sam` を実行している `sam build` ディレクトリがある場合は、関数コードを更新するたびに必ず `sam build` を実行してください。その後、更新された関数コードをローカルでテストするために `sam local start-lambda` を実行します。

ローカルテストは、クラウドにデプロイする前に迅速な開発とテストを行うための優れたソリューションです。ただし、ローカルテストでは、クラウド内のリソース間の許可など、すべてが検証されるわけではありません。可能な限り、アプリケーションをクラウドでテストします。クラウドテストのワークフローを高速化するために [sam sync を使用](#)することをお勧めします。

詳細

すべての `sam local start-lambda` オプションのリストについては、「[sam local start-lambda](#)」を参照してください。

を使用して Lambda 関数をローカルに呼び出す AWS SAM

クラウドでのテストまたはデプロイの前に Lambda 関数をローカルで呼び出すことには、まざまな利点があります。これにより、関数のロジックをより迅速にテストできます。最初にローカルでテス

トすることで、クラウドでのテスト時やデプロイ時に問題が見つかる可能性を低くでき、不要なコストを回避するのにも役立ちます。さらに、ローカルでテストすることでデバッグが楽になります。

Lambda 関数は、sam local invoke コマンドを使用して関数の論理 ID とイベントファイルを指定することによってローカルで呼び出せます。また、sam local invoke は `stdin` をイベントとして受け取ります。イベントの詳細については、AWS Lambda デベロッパーガイドの「イベント」を参照してください。さまざまな AWS サービスのイベントメッセージ形式の詳細については、 AWS Lambda デベロッパーガイドの「他の のサービス AWS Lambda で使用する」を参照してください。

> **Note**
>
> sam local invoke コマンドは AWS Command Line Interface （ AWS CLI) コマンド に対応しますaws lambda invoke。どちらのコマンドを使用しても Lambda 関数を呼び出すことができます。

sam local invoke コマンドは、呼び出す関数を含むプロジェクトディレクトリで実行する必要があります。

例:

```
# Invoking function with event file
$ sam local invoke "Ratings" -e event.json

# Invoking function with event via stdin
$ echo '{"message": "Hey, are you there?" }' | sam local invoke --event - "Ratings"

# For more options
$ sam local invoke --help
```

環境変数ファイル

テンプレートで定義されている値をオーバーライドする環境変数をローカルで宣言するには、次の手順を実行します。

1. 上書きする環境変数を含むJSONファイルを作成します。
2. `--env-vars` 引数を使用して、テンプレートで定義されている値をオーバーライドします。

環境変数の宣言

すべてのリソースにグローバルに適用する環境変数を宣言するには、次のような Parameters オブジェクトを指定します。

```
{
    "Parameters": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
        "STAGE": "dev"
    }
}
```

各リソースごとに別々の環境変数を宣言するには、以下のようにリソースごとにオブジェクトを指定します。

```
{
    "MyFunction1": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
    },
    "MyFunction2": {
        "TABLE_NAME": "localtable",
        "STAGE": "dev"
    }
}
```

各リソースのオブジェクトを指定する場合、以下の識別子を使用できます (優先順位の高いものから順にリストされています)。

1. `logical_id`
2. `function_id`
3. `function_name`
4. フルパス識別子

環境変数を宣言するための前述の両方の方法を単一のファイルで使用できます。その場合、特定のリソースに対して指定した環境変数がグローバル環境変数よりも優先されます。

環境変数を などのJSONファイルに保存しますenv.json。

環境変数の値のオーバーライド

JSON ファイルで定義されている環境変数を上書きするには、 invoke または start-api コマンドで `--env-vars`引数を使用します。以下に例を示します。

```
sam local invoke --env-vars env.json
```

レイヤー

アプリケーションにレイヤーが含まれている場合、ローカルホスト上のレイヤーの問題をデバッグする方法の詳細については、「AWS SAM で Lambda レイヤーを使用して効率を向上させる」を参照してください。

詳細

関数をローカルで呼び出す実践的な例については、 AWS SAM 「」の「モジュール 2 - ローカルで実行する」を参照してください。

を使用して API Gateway をローカルで実行する AWS SAM

Amazon API Gateway をローカルで実行すると、さまざまな利点があります。例えば、APIGateway をローカルで実行すると、 AWS クラウドにデプロイする前にAPIエンドポイントをローカルでテストできます。最初にローカルでテストすれば、多くの場合クラウドでのテストと開発を減らすことができ、コスト削減に役立ちます。さらに、ローカルでの実行はデバッグを容易にします。

HTTP リクエスト/レスポンス機能をテストするために使用できる API Gateway のローカルインスタンスを起動するには、 `sam local start-api` AWS SAM CLI コマンド。この機能にはホットリロードが搭載されているので、関数をすばやく開発して繰り返し実行することができます。

> **Note**
>
> 「ホットリロード」とは、変更されたファイルのみを更新し、アプリケーションの状態を維持することです。これに対して、「ライブリロード」では、アプリケーション全体が更新されるので、アプリケーションの状態が失われます。

`sam local start-api` コマンドを使用する手順については、「sam local start-api を使用したテストの概要」を参照してください。

デフォルトでは、 は AWS Lambda プロキシ統合 AWS SAM を使用し、 HttpApi および Apiリソースタイプの両方をサポートします。HttpApi リソースタイプのプロキシ統合の詳細については、API「ゲートウェイデベロッパーガイドHTTP」の「 の AWS Lambda プロキシ統合の使用APIs」を参照してください。Api リソースタイプとのプロキシ統合の詳細については、API「ゲートウェイデベロッパーガイド」の「ゲートウェイ Lambda プロキシ統合を理解する」を参照してください。 API

例:

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

AWS SAM は、 HttpApiまたは Apiイベントソースが定義されている AWS SAM テンプレート内の関数を自動的に検出します。次に、定義されたHTTPパスに関数をマウントします。

以下の Api 例では、Ratings 関数が GET リクエストの /ratings で ratings.py:handler() をマウントします。

```
Ratings:
  Type: AWS::Serverless::Function
  Properties:
    Handler: ratings.handler
    Runtime: python3.9
    Events:
      Api:
        Type: Api
        Properties:
          Path: /ratings
          Method: get
```

以下は、Api レスポンスの例です。

```
// Example of a Proxy Integration response
exports.handler = (event, context, callback) => {
    callback(null, {
        statusCode: 200,
        headers: { "x-custom-header" : "my custom header value" },
        body: "hello world"
    });
}
```

関数のコードを変更する場合は、sam build コマンドを実行して sam local start-api で変更を検出します。

環境変数ファイル

テンプレートで定義されている値をオーバーライドする環境変数をローカルで宣言するには、次の手順を実行します。

1. 上書きする環境変数を含むJSONファイルを作成します。
2. --env-vars 引数を使用して、テンプレートで定義されている値をオーバーライドします。

環境変数の宣言

すべてのリソースにグローバルに適用する環境変数を宣言するには、次のような Parameters オブジェクトを指定します。

```
{
    "Parameters": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
        "STAGE": "dev"
    }
}
```

各リソースごとに別々の環境変数を宣言するには、以下のようにリソースごとにオブジェクトを指定します。

```
{
    "MyFunction1": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
    },
    "MyFunction2": {
        "TABLE_NAME": "localtable",
        "STAGE": "dev"
    }
}
```

各リソースのオブジェクトを指定する場合、以下の識別子を使用できます (優先順位の高いものから順にリストされています)。

1. `logical_id`
2. `function_id`
3. `function_name`
4. フルパス識別子

環境変数を宣言するための前述の両方の方法を単一のファイルで使用できます。その場合、特定のリソースに対して指定した環境変数がグローバル環境変数よりも優先されます。

環境変数を などのJSONファイルに保存しますenv.json。

環境変数の値のオーバーライド

JSON ファイルで定義されている環境変数を上書きするには、 invoke または start-api コマンドで `--env-vars`引数を使用します。以下に例を示します。

```
$ sam local start-api --env-vars env.json
```

レイヤー

アプリケーションにレイヤーが含まれている場合、ローカルホスト上のレイヤーの問題をデバッグする方法の詳細については、「AWS SAM で Lambda レイヤーを使用して効率を向上させる」を参照してください。

sam remote test-event を使用したクラウドテストの概要

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) の `sam remote test-event` コマンドを使用して、AWS Lambda 関数の共有可能なテストイベントにアクセスし、管理します。

共有可能なテストイベントの詳細については、「AWS Lambda デベロッパーガイド」の「共有可能なテストイベント」を参照してください。

トピック

- AWS SAM CLI で sam remote test-event を使用するようにセットアップする
- sam remote test-event コマンドの使用
- 共有可能なテストイベントを使用する

- 共有可能なテストイベントを管理する

前提条件

`sam remote test-event` を使用するには、次を実行して AWS SAM CLI をインストールします。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

AWS SAM CLI が既にインストールされている場合は、最新バージョンの AWS SAM CLI にアップグレードすることをお勧めします。詳細については、「AWS SAM CLI のアップグレード」を参照してください。

`sam remote test-event` を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.
- でのデプロイの概要 AWS SAM.
- の使用の概要 sam sync 同期する AWS クラウド.

AWS SAM CLI で sam remote test-event を使用するようにセットアップする

次のセットアップステップを実行して、AWS SAM CLI `sam remote test-event` コマンドを使用できるようにします。

1. AWS アカウント を使用するように AWS SAM CLI を設定する - Lambda の共有可能なテストイベントに、同じ AWS アカウント内のユーザーがアクセスして管理できます。AWS アカウント を使用するように AWS SAM CLI を設定する方法については、「AWS SAM CLI の設定」を参照してください。
2. 共有可能なテストイベントのアクセス許可を設定する — 共有可能なテストイベントにアクセスして管理するには、適切なアクセス許可が必要です。詳細については、「AWS Lambda デベロッパーガイド」の「共有可能なテストイベント」を参照してください。

sam remote test-event コマンドの使用

AWS SAM CLI `sam remote test-event` コマンドには、共有可能なテストイベントへのアクセスと管理に使用できる以下のサブコマンドがあります。

- `delete` — Amazon EventBridge スキーマレジストリから、共有可能なテストイベントを削除します。
- `get` — EventBridge スキーマレジストリから共有可能なテストイベントを取得します。
- `list` — EventBridge スキーマレジストリにある関数用の既存の共有可能なテストイベントを一覧表示します。
- `put` — ローカルファイルのイベントを EventBridge スキーマレジストリに保存します。

AWS SAM CLI を使用してこれらのサブコマンドを一覧表示するには、以下を実行します。

```
$ sam remote test-event --help
```

共有可能なテストイベントを削除する

`delete` サブコマンドを以下と併せて使用することで、共有可能なテストイベントを削除できます。

- 削除する共有可能なテストイベントの名前を指定します。
- イベントに関連付けられた許容可能な Lambda 関数 ID を指定します。
- Lambda 関数の論理 ID を指定する場合は、Lambda 関数に関連付けられた AWS CloudFormation スタック名も指定する必要があります。

以下に例を示します。

```
$ sam remote test-event delete HelloWorldFunction --stack-name sam-app --name demo-
event
```

`delete` サブコマンドで使用するオプションのリストについては、「sam remote test-event delete」を参照してください。AWS SAM CLI から、以下の操作を実行することもできます。

```
$ sam remote test-event delete --help
```

共有可能なテストイベントを取得する

get サブコマンドを以下と併せて使用することで、EventBridge スキーマレジストリから共有可能なテストイベントを取得できます。

- 取得する共有可能なテストイベントの名前を指定します。
- イベントに関連付けられた許容可能な Lambda 関数 ID を指定します。
- Lambda 関数の論理 ID を指定する場合は、Lambda 関数に関連付けられた AWS CloudFormation スタック名も指定する必要があります。

以下は、sam-app スタックの HelloWorldFunction Lambda 関数に関連付けられた demo-event という名前の共有可能なテストイベントを取得する例です。このコマンドでは、イベントをコンソールに出力します。

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
```

共有可能なテストイベントを取得してローカルマシンに保存するには、--output-file オプションを使用してファイルパスと名前を指定します。以下は、現在の作業ディレクトリに demo-event を demo-event.json として保存する例です。

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
 --output-file demo-event.json
```

get サブコマンドで使用するオプションのリストについては、「sam remote test-event get」を参照してください。AWS SAM CLI から、以下の操作を実行することもできます。

```
$ sam remote test-event get --help
```

共有可能なテストイベントを一覧表示する

スキーマレジストリから、特定の Lambda 関数のすべての共有可能なテストイベントを一覧表示できます。list サブコマンドを以下と併せて使用します。

- イベントに関連付けられた Lambda 関数の許容可能な ID を指定します。
- Lambda 関数の論理 ID を指定する場合は、Lambda 関数に関連付けられた AWS CloudFormation スタック名も指定する必要があります。

以下は、sam-app スタックの HelloWorldFunction Lambda 関数に関連付けられたすべての共有可能なテストイベントのリストを取得する例です。

```
$ sam remote test-event list HelloWorldFunction --stack-name sam-app
```

list サブコマンドで使用するオプションのリストについては、「sam remote test-event list」を参照してください。AWS SAM CLI から、以下の操作を実行することもできます。

```
$ sam remote test-event list --help
```

共有可能なテストイベントを保存する

共有可能なテストイベントを EventBridge スキーマレジストリに保存できます。put サブコマンドを以下と併せて使用します。

- 共有可能なテストイベントに関連付けられた Lambda 関数の許容可能な ID を指定します。
- 共有可能なテストイベントの名前を指定します。
- アップロードするローカルイベントのファイルパスと名前を指定します。

以下は、ローカルの demo-event.json イベントを demo-event として保存し、sam-app スタックの HelloWorldFunction Lambda 関数に関連付ける例です。

```
$ sam remote test-event put HelloWorldFunction --stack-name sam-app --name demo-event
 --file demo-event.json
```

EventBridge スキーマレジストリに同じ名前の共有可能なテストイベントが存在する場合、AWS SAM CLI はそれを上書きしません。上書きするには、コマンドに --force オプションを追加します。

put サブコマンドで使用するオプションのリストについては、「sam remote test-event put」を参照してください。AWS SAM CLI から、以下の操作を実行することもできます。

```
$ sam remote test-event put --help
```

共有可能なテストイベントを使用する

共有可能なテストイベントを使用して、AWS クラウド で `sam remote invoke` コマンドを実行して Lambda 関数をテストします。詳細については、「共有可能なテストイベントをクラウド内の Lambda 関数に渡す」を参照してください。

共有可能なテストイベントを管理する

このトピックには、共有可能なテストイベントを管理および使用する方法の例が含まれています。

共有可能なテストイベントを取得、変更、使用する

EventBridge スキーマレジストリから共有可能なテストイベントを取得してローカルで変更し、そのローカルテストイベントを AWS クラウド の Lambda 関数で使用できます。以下に例を示します。

1. 共有可能なテストイベントを取得する — `sam remote test-event get` サブコマンドを使用して、特定の Lambda 関数の共有可能なテストイベントを取得し、ローカルに保存します。

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
 --output-file demo-event.json
```

2. 共有可能なテストイベントを変更する — 任意のテキストエディタを使用して、共有可能なテストイベントを変更します。
3. 共有可能なテストイベントを使用する — `sam remote invoke` コマンドを使用して、`--event-file` でイベントのファイルパスと名前を指定します。

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --event-file demo-
event.json
```

共有可能なテストイベントを取得し、変更、アップロード、使用する

EventBridge スキーマレジストリから共有可能なテストイベントを取得し、ローカルで変更してアップロードできます。その後、共有可能なテストイベントを AWS クラウド の Lambda 関数に直接渡すことができます。以下に例を示します。

1. 共有可能なテストイベントを取得する — `sam remote test-event get` サブコマンドを使用して、特定の Lambda 関数の共有可能なテストイベントを取得し、ローカルに保存します。

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
 --output-file demo-event.json
```

2. 共有可能なテストイベントを変更する — 任意のテキストエディタを使用して、共有可能なテストイベントを変更します。
3. 共有可能なテストイベントをアップロードする — `sam remote test-event put` サブコマンドを使用して、共有可能なテストイベントを EventBridge スキーマレジストリにアップロードして保存します。この例では、共有可能なテストの古いバージョンを上書きする `--force` オプションを使用しています。

```
$ sam remote test-event put HelloWorldFunction --stack-name sam-app --name demo-event
 --file demo-event.json --force
```

4. 共有可能なテストイベントを Lambda 関数に渡す — `sam remote invoke` コマンドを使用して、共有可能なテストイベントを AWS クラウド の Lambda 関数に直接渡します。

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --test-event-name demo-
event
```

を使用したクラウドでのテストの概要 sam remote invoke

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam remote invoke` コマンドを使用して、 でサポートされている AWS リソースとやり取りします AWS クラウド。`sam remote invoke` を使用して、次のリソースを呼び出します。

- Amazon Kinesis Data Streams — Kinesis Data Streams アプリケーションにデータレコードを送信します。
- AWS Lambda – イベントを呼び出して Lambda 関数に渡します。
- Amazon Simple Queue Service (Amazon SQS) — Amazon SQSキューにメッセージを送信します。
- AWS Step Functions – Step Functions ステートマシンを呼び出して実行を開始します。

の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？

一般的な開発ワークフローでの `sam remote invoke` の使用例については、「ステップ 5: で 関数を操作する AWS クラウド」を参照してください。

トピック

前提条件

を使用するにはsam remote invoke、 をインストールします。 AWS SAM CLI 以下を完了します。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

また、 の最新バージョンにアップグレードすることをお勧めします。 AWS SAM CLI。 詳細については、「」を参照してくださいAWS SAM CLI のアップグレード。

sam remote invoke を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.
- でのデプロイの概要 AWS SAM.
- の使用の概要 sam sync 同期する AWS クラウド.

sam remote invoke コマンドの使用

このコマンドを使用する前に、リソースが AWS クラウドにデプロイされている必要があります。

次のコマンド構造を使用し、プロジェクトのルート ディレクトリから実行します。

```
$ sam remote invoke <arguments> <options>
```

> **Note**
>
> このページでは、コマンドプロンプトで提供されるオプションについて説明します。オプションは、コマンドプロンプトで渡す代わりに、プロジェクトの設定ファイルで設定することもできます。詳細については、「プロジェクト設定を構成する」を参照してください。

`sam remote invoke` 引数とオプションの説明については、「sam remote invoke」を参照してください。

Kinesis Data Streams の使用

データレコードを Kinesis Data Streams アプリケーションに送信できます。の AWS SAM CLI はデータレコードを送信し、シャード ID とシーケンス番号を返します。以下に例を示します。

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event hello-world

 Putting record to Kinesis data stream KinesisStream

 Auto converting value 'hello-world' into JSON '"hello-world"'. If you don't want auto-
conversion, please provide
 a JSON string as event

 {
   "ShardId": "shardId-000000000000",
   "SequenceNumber": "49646251411914806775980850790050483811301135051202232322"
 }%
```

データレコードを送信するには

1. Kinesis Data Streams アプリケーションの引数としてリソース ID 値を指定します。有効なリソース の詳細についてはIDs、「リソース ID 」を参照してください。
2. Kinesis Data Streams アプリケーションに送信するイベントとしてデータレコードを指定します。イベントは、`--event` オプションを使用してコマンドラインで提供することも、`--event-file` を使用してファイルから提供することもできます。イベントを指定しない場合、AWS SAM CLI は空のイベントを送信します。

Lambda 関数を使用する

クラウド内の Lambda 関数を呼び出して、空のイベントを渡すか、コマンドラインまたはファイルからイベントを提供することができます。の AWS SAM CLI は Lambda 関数を呼び出し、そのレスポンスを返します。以下に例を示します。

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

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

Lambda 関数を呼び出すには

1. Lambda 関数の引数としてリソース ID の値を指定します。有効なリソース の詳細については IDs、「リソース ID 」を参照してください。
2. Lambda 関数に送信するイベントを指定します。イベントは、`--event` オプションを使用してコマンドラインで提供することも、`--event-file` を使用してファイルから提供することもできます。イベントを指定しない場合、 AWS SAM CLI は空のイベントを送信します。

レスポンスストリーミングが設定された Lambda 関数

`sam remote invoke` コマンドは、レスポンスをストリーミングするように設定された Lambda 関数をサポートします。 AWS SAM テンプレートの `FunctionUrlConfig`プロパティを使用してレスポンスをストリーミングするように Lambda 関数を設定できます。を使用する場合`sam remote invoke`、 AWS SAM CLI は Lambda 設定を自動的に検出し、レスポンスストリーミングで呼び出します。

例については、レスポンスをストリーミングするように設定された Lambda 関数を呼び出すを参照してください。

共有可能なテストイベントをクラウド内の Lambda 関数に渡す

共有可能なテストイベントは、同じ AWS アカウント内の他のユーザーと共有できるテストイベントです。詳細については、「AWS Lambda デベロッパーガイド」の「共有可能なテストイベント」を参照してください。

共有可能なテストイベントへのアクセスと管理

は、 AWS SAM CLI `sam remote test-event` 共有可能なテストイベントにアクセスして管理するための コマンド。例えば、`sam remote test-event` を使用して以下を実行できます。

- Amazon EventBridge スキーマレジストリから共有可能なテストイベントを取得します。
- 共有可能なテストイベントをローカルで変更し、 EventBridge スキーマレジストリにアップロードします。
- EventBridge スキーマレジストリから共有可能なテストイベントを削除します。

詳細については、「sam remote test-event を使用したクラウドテストの概要」を参照してください。

共有可能なテストイベントをクラウド内の Lambda 関数に渡す

EventBridge スキーマレジストリからクラウドの Lambda 関数に共有可能なテストイベントを渡すには、 `--test-event-name`オプションを使用して共有可能なテストイベントの名前を指定します。以下に例を示します。

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --test-event-name demo-event
```

共有可能なテストイベントをローカルに保存する場合は、`--event-file` オプションを使用してローカルテストイベントのファイルパスと名前を指定できます。以下に例を示します。

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --event-file demo-event.json
```

Amazon での の使用 SQS

Amazon SQSキューにメッセージを送信できます。の AWS SAM CLI 以下を返します。

- メッセージ ID
- MD5 メッセージ本文の
- レスポンスメタデータ

以下に例を示します。

```
$ sam remote invoke MySqsQueue --stack-name sqs-example -event hello

Sending message to SQS queue MySqsQueue

{
  "MD5OfMessageBody": "5d41402abc4b2a76b9719d911017c592",
  "MessageId": "05c7af65-9ae8-4014-ae28-809d6d8ec652"
}%
```

単一のメッセージを送信するには

1. Amazon SQSキューの引数としてリソース ID 値を指定します。有効なリソース の詳細についてはIDs、「リソース ID 」を参照してください。
2. Amazon SQSキューに送信するイベントを指定します。イベントは、`--event` オプションを使用してコマンドラインで提供することも、`--event-file` を使用してファイルから提供することもできます。イベントを指定しない場合、 AWS SAM CLI は空のイベントを送信します。

Step Functions で を使用する

Step Functions ステート マシンを呼び出して実行を開始できます。の AWS SAM CLI は、ステートマシンワークフローが完了するのを待ち、実行の最後のステップの出力を返します。以下に例を示します。

```
$ sam remote invoke HelloWorldStateMachine --stack-name state-machine-example --
event '{"is_developer": true}'

Invoking Step Function HelloWorldStateMachine

"Hello Developer World"%
```

ステートマシンを呼び出すには

1. Step Functions ステートマシンの引数としてリソース ID の値を指定します。有効なリソース の詳細についてはIDs、「リソース ID 」を参照してください。
2. ステートマシンに送信するイベントを指定します。イベントは、`--event` オプションを使用してコマンドラインで提供することも、`--event-file` を使用してファイルから提供することもできます。イベントを指定しない場合、 AWS SAM CLI は空のイベントを送信します。

sam リモート呼び出しコマンド オプションの使用

このセクションでは、sam remote invoke コマンドで使用できる主なオプションのいくつかを説明します。 オプションの完全なリストについては、「sam remote invoke」を参照してください。

リソースにイベントを渡します。

以下のオプションを使用して、クラウド内のリソースにイベントを渡します。

- --event — コマンドラインでイベントを渡します。
- --event-file — ファイルからイベントを渡します。

Lambda での例

--event を使用して、コマンドラインでイベントを文字列値として渡します。

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --event '{"message":
 "hello!"}'

Invoking Lambda Function HelloWorldFunction


START RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceab Version: $LATEST
END RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceab
REPORT RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceab  Duration: 16.41 ms      Billed
 Duration: 17 ms  Memory Size: 128 MB     Max Memory Used: 67 MB  Init Duration: 185.96
 ms
{"statusCode":200,"body":"{\"message\":\"hello!\"}"}%
```

--event-file を使用してファイルからのイベントを渡し、ファイルへのパスを指定します。

```
$ cat event.json

{"message": "hello from file"}%

$ sam remote invoke HelloWorldFunction --stack-name sam-app --event-file event.json


Invoking Lambda Function HelloWorldFunction


START RequestId: 3bc71f7d-153a-4b1e-8c9a-901d91b1bec9 Version: $LATEST
```

```
END RequestId: 3bc71f7d-153a-4b1e-8c9a-901d91b1bec9
REPORT RequestId: 3bc71f7d-153a-4b1e-8c9a-901d91b1bec9  Duration: 21.15 ms      Billed
 Duration: 22 ms  Memory Size: 128 MB     Max Memory Used: 67 MB
{"statusCode":200,"body":"{\"message\":\"hello from file\"}"}%
```

stdin を使用してイベントを渡します。

```
$ cat event.json

{"message": "hello from file"}%

$ cat event.json | sam remote invoke HelloWorldFunction --stack-name sam-app --event-
file -

Reading event from stdin (you can also pass it from file with --event-file)

Invoking Lambda Function HelloWorldFunction

START RequestId: 85ecc902-8ad0-4a2b-a8c8-9bb4f65f5a7a Version: $LATEST
END RequestId: 85ecc902-8ad0-4a2b-a8c8-9bb4f65f5a7a
REPORT RequestId: 85ecc902-8ad0-4a2b-a8c8-9bb4f65f5a7a  Duration: 1.36 ms       Billed
 Duration: 2 ms   Memory Size: 128 MB       Max Memory Used: 67 MB
{"statusCode":200,"body":"{\"message\":\"hello from file\"}"}%
```

を設定する AWS SAM CLI レスポンス出力

でサポートされているリソースを呼び出すとsam remote invoke、 AWS SAM CLI は、以下を含むレスポンスを返します。

- リクエストメタデータ – リクエストに関連するメタデータです。これには、リクエスト ID とリクエスト開始時間が含まれます。
- Resource Response — クラウド内で呼び出された後のリソースからのレスポンスです。

--output オプションを使用して、 AWS SAM CLI 出力レスポンス。以下のオプション値を使用できます。

- json – メタデータとリソースのレスポンスは で返されます。JSON 構造。レスポンスには、SDK 出力。
- text – メタデータがテキスト構造で返されます。リソースレスポンスは、リソースの出力形式で返されます。

以下は、json 出力の例です。

```
$ sam remote invoke --stack-name sam-app --output json

Invoking Lambda Function HelloWorldFunction



{
  "ResponseMetadata": {
    "RequestId": "3bdf9a30-776d-4a90-94a6-4cccc0fc7b41",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 19 Jun 2023 17:15:46 GMT",
      "content-type": "application/json",
      "content-length": "57",
      "connection": "keep-alive",
      "x-amzn-requestid": "3bdf9a30-776d-4a90-94a6-4cccc0fc7b41",
      "x-amzn-remapped-content-length": "0",
      "x-amz-executed-version": "$LATEST",
      "x-amz-log-result":
 "U1RBUlQgUmVxdWVzdElkOiAzYmRmOWEzMC03NzZkLTRhOTAtOTRhNi00Y2NjYzBmYzdiNDEgVmVyc2lvbjogJExBVEVTV
      "x-amzn-trace-id":
 "root=1-64908d42-17dab270273fcc6b527dd6b8;sampled=0;lineage=2301f8dc:0"
    },
    "RetryAttempts": 0
  },
  "StatusCode": 200,
  "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiAzYmRmOWEzMC03NzZkLTRhOTAtOTRhNi00Y2NjYzBmYzdiNDEgVmVyc2lvbjogJExBVEVTV
  "ExecutedVersion": "$LATEST",
  "Payload": "{\"statusCode\":200,\"body\":\"{\\\"message\\\":\\\"hello world\\\"}\"}"
}%
```

json 出力を指定すると、レスポンス全体が stdout に返されます。以下に例を示します。

```
$ sam remote invoke --stack-name sam-app --output json 1> stdout.log

Invoking Lambda Function HelloWorldFunction



$ cat stdout.log
```

```
{
  "ResponseMetadata": {
    "RequestId": "d30d280f-8188-4372-bc94-ce0f1603b6bb",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 19 Jun 2023 17:35:56 GMT",
      "content-type": "application/json",
      "content-length": "57",
      "connection": "keep-alive",
      "x-amzn-requestid": "d30d280f-8188-4372-bc94-ce0f1603b6bb",
      "x-amzn-remapped-content-length": "0",
      "x-amz-executed-version": "$LATEST",
      "x-amz-log-result":
 "U1RBUlQgUmVxdWVzdElkOiBkMzBkMjgwZi04MTg4LTQzNzItYmM5NC1jZTBmMTYwM2I2YmIgVmVyc2lvbjogJExBVEVTV
      "x-amzn-trace-id":
 "root=1-649091fc-771473c7778689627a6122b7;sampled=0;lineage=2301f8dc:0"
    },
    "RetryAttempts": 0
  },
  "StatusCode": 200,
  "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiBkMzBkMjgwZi04MTg4LTQzNzItYmM5NC1jZTBmMTYwM2I2YmIgVmVyc2lvbjogJExBVEVTV
  "ExecutedVersion": "$LATEST",
  "Payload": "{\"statusCode\":200,\"body\":\"{\\\"message\\\":\\\"hello world\\\"}\"}"
}%
```


以下は、text 出力の例です。

```
$ sam remote invoke --stack-name sam-app --output text

Invoking Lambda Function HelloWorldFunction


START RequestId: 4dbacc43-1ec6-47c2-982b-9dc4620144d6 Version: $LATEST
END RequestId: 4dbacc43-1ec6-47c2-982b-9dc4620144d6
REPORT RequestId: 4dbacc43-1ec6-47c2-982b-9dc4620144d6  Duration: 9.13 ms       Billed
 Duration: 10 ms  Memory Size: 128 MB     Max Memory Used: 67 MB  Init Duration: 165.50
 ms
{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

text 出力を指定すると、Lambda 関数ランタイム出力 (ログなど) が stderr に返されます。Lambda 関数ペイロードは stdout に返されます。以下に例を示します。

```
$ sam remote invoke --stack-name sam-app --output text 2> stderr.log

{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%



$ cat stderr.log

Invoking Lambda Function HelloWorldFunction
START RequestId: 82273c3b-aa3a-4d16-8f1c-1d2ad3ace891 Version: $LATEST
END RequestId: 82273c3b-aa3a-4d16-8f1c-1d2ad3ace891
REPORT RequestId: 82273c3b-aa3a-4d16-8f1c-1d2ad3ace891  Duration: 40.62 ms      Billed
 Duration: 41 ms  Memory Size: 128 MB     Max Memory Used: 68 MB

$ sam remote invoke --stack-name sam-app --output text 1> stdout.log

Invoking Lambda Function HelloWorldFunction


START RequestId: 74acaa9f-5b80-4a5c-b3b8-ffaccb84cbbd Version: $LATEST
END RequestId: 74acaa9f-5b80-4a5c-b3b8-ffaccb84cbbd
REPORT RequestId: 74acaa9f-5b80-4a5c-b3b8-ffaccb84cbbd  Duration: 2.31 ms       Billed
 Duration: 3 ms   Memory Size: 128 MB     Max Memory Used: 67 MB

$ cat stdout.log

{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

カスタマイズ Boto3 parameters

の場合`sam remote invoke`、 AWS SAM CLI は、 AWS SDK for Python (Boto3) を使用してクラウド内のリソースとやり取りします。`--parameter` オプションを使用してカスタマイズできます。Boto3 パラメータ。サポートされるカスタマイズ可能なパラメータのリストについては、「[--parameter]」を参照してください。

例

パラメータ値を検証し、許可を確認するための Lambda 関数の呼び出し

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --
parameter InvocationType="DryRun"
```

1 つのコマンドで **--parameter** オプションを複数回使用して、複数のパラメータを指定することができます。

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --
parameter InvocationType="Event" --parameter LogType="None"
```

その他のオプション

sam remote invoke オプションの完全なリストについては、「sam remote invoke」を参照してください。

プロジェクト設定ファイルを設定する

設定ファイルで sam remote invoke を設定するには、テーブルで remote_invoke を使用します。以下は、sam remote invoke コマンドのデフォルト値を設定する samconfig.toml ファイルの例です。

```
...
version =0.1

[default]
...
[default.remote_invoke.parameters]
stack_name = "cloud-app"
event = '{"message": "Hello!"}'
```

例

を使用する基本的な例についてはsam remote invoke、AWS Compute Blog のAWS SAM 「リモートでの関数のテスト AWS Lambda」を参照してください。

Kinesis Data Streams の例

基本的な例

ファイルから Kinesis Data Streams アプリケーションにデータレコードを送信します。Kinesis Data Streams アプリケーションは、リソース ID ARNに を提供することで識別されます。

```
$ sam remote invoke arn:aws:kinesis:us-west-2:01234567890:stream/kinesis-example-
KinesisStream-BgnLcAey4xUQ --event-file event.json
```

コマンドラインで指定されるイベントを Kinesis Data Streams アプリケーションに送信します。

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event hello-world

Putting record to Kinesis data stream KinesisStream

Auto converting value 'hello-world' into JSON '"hello-world"'. If you don't want auto-
conversion, please provide
a JSON string as event

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980903986194508740483329854174920706"
}%
```

Kinesis Data Streams アプリケーションの物理 ID を取得します。次に、コマンドラインにイベントを指定します。

```
$ sam list resources --stack-name kinesis-example --output json

[
  {
    "LogicalResourceId": "KinesisStream",
    "PhysicalResourceId": "kinesis-example-KinesisStream-ZgnLcQey4xUQ"
  }
]

$ sam remote invoke kinesis-example-KinesisStream-ZgnLcQey4xUQ --event hello

Putting record to Kinesis data stream KinesisStream

Auto converting value 'hello' into JSON '"hello"'. If you don't want auto-conversion,
 please provide a JSON
string as event

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980904340716841045751814812900261890"
}%
```

イベントとしてコマンドラインにJSON文字列を指定します。

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event '{"method":
 "GET", "body": ""}'

Putting record to Kinesis data stream KinesisStream

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980904492868617924990209230536441858"
}%
```

Kinesis Data Streams アプリケーションに空のイベントを送信します。

```
$ sam remote invoke KinesisStream --stack-name kinesis-example

Putting record to Kinesis data stream KinesisStream

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980904866469008589597168190416224258"
}%
```

を返す AWS SAM CLI JSON 形式のレスポンス：

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event '{"hello":
 "world"}' --output json

Putting record to Kinesis data stream KinesisStream

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980905078409420803696667195489648642",
  "ResponseMetadata": {
    "RequestId": "ebbbd307-3e9f-4431-b67c-f0715e9e353e",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "ebbbd307-3e9f-4431-b67c-f0715e9e353e",
      "x-amz-id-2": "Q3yBcgTwtPaQTV26IKclbECmZikUYOzKY+CzcxA84ZHgCkc5T2N/
ITWg6RPOQcWw8Gn0tNPcEJBEHyVVqboJAPgCritqsvCu",
      "date": "Thu, 09 Nov 2023 18:13:10 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "110"
    },
```

```
    "RetryAttempts": 0
  }
}%
```

JSON出力を stdout に戻します。

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event '{"hello":
 "world"}' --output json 1> stdout.log

Putting record to Kinesis data stream KinesisStream


$ cat stdout.log
{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980906397777867595039988349006774274",
  "ResponseMetadata": {
    "RequestId": "f4290006-d84b-b1cd-a9ee-28306eeb2939",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "f4290006-d84b-b1cd-a9ee-28306eeb2939",
      "x-amz-id-2": "npCqz
+IBKpoL4sQ1ClbUmxuJlbeA24Fx1UgpIrS6mm2NoIeV2qdZSN5AhNurdssykXajBrXaC9anMhj2eG/h7Hnbf
+bPuotU",
      "date": "Thu, 09 Nov 2023 18:33:26 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "110"
    },
    "RetryAttempts": 0
  }
}%
```

Lambda での例

基本的な例

をリソース ID ARNとして指定して Lambda 関数を呼び出します。

```
$ sam remote invoke arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-ohRFEn2RuAvp
```

論理 ID をリソース ID として指定して Lambda 関数を呼び出します。

また、 `--stack-name`オプションを使用して AWS CloudFormation スタック名を指定する必要があります。以下に例を示します。

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

アプリケーションに単一の Lambda 関数が含まれている場合、その論理 ID を指定する必要はありません。`--stack-name` オプションのみを指定できます。以下に例を示します。

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

物理 ID をリソース ID として指定して Lambda 関数を呼び出します。

を使用してデプロイすると、物理 ID が作成されます AWS CloudFormation。

```
$ sam remote invoke sam-app-HelloWorldFunction-TZvxQRFNv0k4
```

子スタックの Lambda 関数を呼び出します。

この例では、アプリケーションには次のディレクトリ構造が含まれています。

```
lambda-example
### childstack
#   ### function
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### template.yaml
### events
#   ### event.json
### samconfig.toml
### template.yaml
```

`childstack` の Lambda 関数を呼び出すには、以下を実行します。

```
$ sam remote invoke ChildStack/HelloWorldFunction --stack-name lambda-example

Invoking Lambda Function HelloWorldFunction

START RequestId: 207a864b-e67c-4307-8478-365b004d4bcd Version: $LATEST
END RequestId: 207a864b-e67c-4307-8478-365b004d4bcd
```

```
REPORT RequestId: 207a864b-e67c-4307-8478-365b004d4bcd  Duration: 1.27 ms       Billed
 Duration: 2 ms   Memory Size: 128 MB     Max Memory Used: 36 MB  Init Duration: 111.07
 ms
{"statusCode": 200, "body": "{\"message\": \"Hello\", \"received_event\": {}}"}%
```

レスポンスをストリーミングするように設定された Lambda 関数を呼び出す

この例では、 AWS SAM CLI は、レスポンスをストリーミングするように設定された Lambda 関数を含む新しいサーバーレスアプリケーションを初期化します。アプリケーションを にデプロイ AWS クラウド しsam remote invoke、 を使用してクラウド内の 関数とやり取りします。

まず、sam init コマンドを実行して新しいサーバーレスアプリケーションを作成することから始めます。Lambda Response Streaming クイックスタートテンプレートを選択し、アプリケーションに名前を付けますlambda-streaming-nodejs-app。

```
$ sam init

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

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

 Choose an AWS Quick Start application template
        1 - Hello World Example
        ...
        9 - Lambda Response Streaming
        ...
        15 - Machine Learning
 Template: 9

 Which runtime would you like to use?
        1 - go (provided.al2)
        2 - nodejs18.x
        3 - nodejs16.x
 Runtime: 2

 Based on your selections, the only Package type available is Zip.
 We will proceed to selecting the Package type as Zip.
```

```
 Based on your selections, the only dependency manager available is npm.
 We will proceed copying the template using npm.

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

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

 Project name [sam-app]: lambda-streaming-nodejs-app

     -----------------------
     Generating application:
     -----------------------
     Name: lambda-streaming-nodejs-app
     Runtime: nodejs18.x
     Architectures: x86_64
     Dependency Manager: npm
     Application Template: response-streaming
     Output Directory: .
     Configuration file: lambda-streaming-nodejs-app/samconfig.toml

     Next steps can be found in the README file at lambda-streaming-nodejs-app/
README.md


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

の AWS SAM CLI は、次の構造でプロジェクトを作成します。

```
lambda-streaming-nodejs-app
 ### README.md
 ### __tests__
 #   ### unit
 #       ### index.test.js
 ### package.json
 ### samconfig.toml
```

```
 ### src
 #   ### index.js
 ### template.yaml
```

以下は、Lambda 関数コードの例です。

```
exports.handler = awslambda.streamifyResponse(
   async (event, responseStream, context) => {
     const httpResponseMetadata = {
       statusCode: 200,
       headers: {
         "Content-Type": "text/html",
         "X-Custom-Header": "Example-Custom-Header"
       }
     };

     responseStream = awslambda.HttpResponseStream.from(responseStream,
 httpResponseMetadata);
     // It's recommended to use a `pipeline` over the `write` method for more complex
 use cases.
     // Learn more: https://docs.aws.amazon.com/lambda/latest/dg/configuration-
response-streaming.html
     responseStream.write("<html>");
     responseStream.write("<p>First write!</p>");

     responseStream.write("<h1>Streaming h1</h1>");
     await new Promise(r => setTimeout(r, 1000));
     responseStream.write("<h2>Streaming h2</h2>");
     await new Promise(r => setTimeout(r, 1000));
     responseStream.write("<h3>Streaming h3</h3>");
     await new Promise(r => setTimeout(r, 1000));

     // Long strings will be streamed
     const loremIpsum1 = "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 Quisque vitae mi tincidunt tellus ultricies dignissim id et diam. Morbi pharetra eu
 nisi et finibus. Vivamus diam nulla, vulputate et nisl cursus, pellentesque vehicula
 libero. Cras imperdiet lorem ante, non posuere dolor sollicitudin a. Vestibulum ipsum
 lacus, blandit nec augue id, lobortis dictum urna. Vestibulum ante ipsum primis in
 faucibus orci luctus et ultrices posuere cubilia curae; Morbi auctor orci eget tellus
 aliquam, non maximus massa porta. In diam ante, pulvinar aliquam nisl non, elementum
 hendrerit sapien. Vestibulum massa nunc, mattis non congue vitae, placerat in quam.
 Nam vulputate lectus metus, et dignissim erat varius a.";
     responseStream.write(`<p>${loremIpsum1}</p>`);
```

```
    await new Promise(r => setTimeout(r, 1000));

    responseStream.write("<p>DONE!</p>");
    responseStream.write("</html>");
    responseStream.end();
  }
);
```

以下は、template.yaml ファイルの例です。Lambda 関数のレスポンスストリーミングは、FunctionUrlConfig プロパティを使用して設定されます。

```
AWSTemplateFormatVersion: '2010-09-09'
 Transform: AWS::Serverless-2016-10-31

 Description: >
   Sample SAM Template for lambda-streaming-nodejs-app

 Resources:
   StreamingFunction:
     Type: AWS::Serverless::Function
     Properties:
       CodeUri: src/
       Handler: index.handler
       Runtime: nodejs18.x
       Architectures:
         - x86_64
       Timeout: 10
       FunctionUrlConfig:
         AuthType: AWS_IAM
         InvokeMode: RESPONSE_STREAM

 Outputs:
   StreamingFunction:
     Description: "Streaming Lambda Function ARN"
     Value: !GetAtt StreamingFunction.Arn
   StreamingFunctionURL:
     Description: "Streaming Lambda Function URL"
     Value: !GetAtt StreamingFunctionUrl.FunctionUrl
```

通常は、sam build および sam deploy --guided を使用して本番アプリケーションを構築し、デプロイすることができます。この例では、開発環境を想定し、アプリケーションを構築してデプロイするために sam sync コマンドを使用します。

> **Note**
>
> sam sync コマンドは開発環境に推奨されます。詳細については、「の使用の概要 sam sync 同期する AWS クラウド」を参照してください。

sam sync を実行する前に、プロジェクトが samconfig.toml ファイル内で正しく設定されていることを確認します。最も重要なのは、stack_name と watch の値を確認することです。設定ファイル内でこれらの値が指定されていれば、コマンドラインでそれらを指定する必要がありません。

```
version = 0.1

 [default]
 [default.global.parameters]
 stack_name = "lambda-streaming-nodejs-app"

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

 [default.validate.parameters]
 lint = true

 [default.deploy.parameters]
 capabilities = "CAPABILITY_IAM"
 confirm_changeset = true
 resolve_s3 = true
 s3_prefix = "lambda-streaming-nodejs-app"
 region = "us-west-2"
 image_repositories = []

 [default.package.parameters]
 resolve_s3 = true

 [default.sync.parameters]
 watch = true

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

 [default.local_start_lambda.parameters]
 warm_containers = "EAGER"
```

次に、sam sync を実行してアプリケーションを構築し、デプロイします。--watch オプションは設定ファイルで設定されるため、 AWS SAM CLI は、アプリケーションを構築し、アプリケーションをデプロイし、変更を監視します。

```
$ sam sync

 The SAM CLI will use the AWS Lambda, Amazon API Gateway, and AWS StepFunctions APIs to
 upload your code
 without

 performing a CloudFormation deployment. This will cause drift in your CloudFormation
 stack.
 **The sync command should only be used against a development stack**.



 Queued infra sync. Waiting for in progress code syncs to complete...

 Starting infra sync.

 Building codeuri:

 /Users/.../lambda-streaming-nodejs-app/src runtime: nodejs18.x metadata: {}
 architecture: x86_64 functions: StreamingFunction
 package.json file not found. Continuing the build without dependencies.

 Running NodejsNpmBuilder:CopySource


 Build Succeeded

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


        Deploying with following values
        ===============================
        Stack name                   : lambda-streaming-nodejs-app
        Region                       : us-west-2
        Disable rollback             : False
```

```
        Deployment s3 bucket         : aws-sam-cli-managed-default-samcliamzn-s3-demo-
bucket-1a4x26zbcdkqr
        Capabilities                 : ["CAPABILITY_NAMED_IAM",
 "CAPABILITY_AUTO_EXPAND"]
        Parameter overrides          : {}
        Signing Profiles             : null

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


 2023-06-20 12:11:16 - Waiting for stack create/update to complete

 CloudFormation events from stack operations (refresh every 0.5 seconds)
 ---------------------------------------------------------------------------------------------
 ResourceStatus            ResourceType              LogicalResourceId
 ResourceStatusReason
 ---------------------------------------------------------------------------------------------
 CREATE_IN_PROGRESS        AWS::CloudFormation::St   lambda-streaming-
 Transformation
                           ack                       nodejs-app
 succeeded
 CREATE_IN_PROGRESS        AWS::IAM::Role            StreamingFunctionRole     -

 CREATE_IN_PROGRESS        AWS::CloudFormation::St   AwsSamAutoDependencyLay   -

                           ack                       erNestedStack

 CREATE_IN_PROGRESS        AWS::IAM::Role            StreamingFunctionRole     Resource
 creation

 Initiated
 CREATE_IN_PROGRESS        AWS::CloudFormation::St   AwsSamAutoDependencyLay   Resource
 creation
                           ack                       erNestedStack
 Initiated
 CREATE_COMPLETE           AWS::IAM::Role            StreamingFunctionRole     -

 CREATE_COMPLETE           AWS::CloudFormation::St   AwsSamAutoDependencyLay   -

                           ack                       erNestedStack

 CREATE_IN_PROGRESS        AWS::Lambda::Function     StreamingFunction         -
```

```
 CREATE_IN_PROGRESS        AWS::Lambda::Function     StreamingFunction         Resource
 creation

 Initiated
 CREATE_COMPLETE           AWS::Lambda::Function     StreamingFunction         -

 CREATE_IN_PROGRESS        AWS::Lambda::Url          StreamingFunctionUrl      -

 CREATE_IN_PROGRESS        AWS::Lambda::Url          StreamingFunctionUrl      Resource
 creation

 Initiated
 CREATE_COMPLETE           AWS::Lambda::Url          StreamingFunctionUrl      -

 CREATE_COMPLETE           AWS::CloudFormation::St   lambda-streaming-         -

                           ack                       nodejs-app

 -----------------------------------------------------------------------------------------

 CloudFormation outputs from deployed stack
 -----------------------------------------------------------------------------------------
 Outputs

 -----------------------------------------------------------------------------------------
 Key                 StreamingFunction

 Description         Streaming Lambda Function ARN

 Value               arn:aws:lambda:us-west-2:012345678910:function:lambda-streaming-
nodejs-app-
 StreamingFunction-gUmhO833A0vZ


 Key                 StreamingFunctionURL

 Description         Streaming Lambda Function URL

 Value               https://wxgkcc2dyntgtrwhf2dgdcvylu0rnnof.lambda-url.us-
west-2.on.aws/
 -----------------------------------------------------------------------------------------
```

```
 Stack creation succeeded. Sync infra completed.



 Infra sync completed.
```

関数がクラウドにデプロイされたので、sam remote invoke を使用して関数とやり取りできるようになりました。の AWS SAM CLI は、関数がレスポンスストリーミング用に設定されていることを自動的に検出し、関数のストリーミングレスポンスをリアルタイムで出力し始めます。

```
$ sam remote invoke StreamingFunction

 Invoking Lambda Function StreamingFunction

 {"statusCode":200,"headers":{"Content-Type":"text/html","X-Custom-Header":"Example-
Custom-Header"}}<html><p>First write!</p><h1>Streaming h1</h1><h2>Streaming h2</
h2><h3>Streaming h3</h3><p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 Quisque vitae mi tincidunt tellus ultricies dignissim id et diam. Morbi pharetra eu
 nisi et finibus. Vivamus diam nulla, vulputate et nisl cursus, pellentesque vehicula
 libero. Cras imperdiet lorem ante, non posuere dolor sollicitudin a. Vestibulum ipsum
 lacus, blandit nec augue id, lobortis dictum urna. Vestibulum ante ipsum primis in
 faucibus orci luctus et ultrices posuere cubilia curae; Morbi auctor orci eget tellus
 aliquam, non maximus massa porta. In diam ante, pulvinar aliquam nisl non, elementum
 hendrerit sapien. Vestibulum massa nunc, mattis non congue vitae, placerat in quam.
 Nam vulputate lectus metus, et dignissim erat varius a.</p><p>DONE!</p></html>START
 RequestId: 1e4cdf04-60de-4769-b3a2-c1481982deb4 Version: $LATEST
 END RequestId: 1e4cdf04-60de-4769-b3a2-c1481982deb4
 REPORT RequestId: 1e4cdf04-60de-4769-b3a2-c1481982deb4  Duration: 4088.66 ms
 Billed Duration: 4089 ms        Memory Size: 128 MB     Max Memory Used: 68 MB  Init
 Duration: 168.45 ms
```

関数コードを変更すると、 AWS SAM CLI は、変更を即座に検出してデプロイします。以下は、AWS SAM CLI 関数コードに変更が加えられた後に出力されます。

```
Syncing Lambda Function StreamingFunction...

 Building codeuri:

 /Users/.../lambda-streaming-nodejs-app/src runtime: nodejs18.x metadata: {}
 architecture:
 x86_64 functions: StreamingFunction
```

```
package.json file not found. Continuing the build without dependencies.

Running NodejsNpmBuilder:CopySource

Finished syncing Lambda Function StreamingFunction.

Syncing Layer StreamingFunctione9cfe924DepLayer...

SyncFlow [Layer StreamingFunctione9cfe924DepLayer]: Skipping resource update as the

content didn't change

Finished syncing Layer StreamingFunctione9cfe924DepLayer.
```

これで、sam remote invoke を再度使用して、クラウド内の関数とやり取りし、変更をテストできるようになりました。

SQS 例

基本的な例

をリソース ID ARNとして指定して Amazon SQSキューを呼び出します。

```
$ sam remote invoke arn:aws:sqs:us-west-2:01234567890:sqs-example-4DonhBsjsW1b --
event '{"hello": "world"}' --output json

Sending message to SQS queue MySqsQueue

{
  "MD5OfMessageBody": "49dfdd54b01cbcd2d2ab5e9e5ee6b9b9",
  "MessageId": "4f464cdd-15ef-4b57-bd72-3ad225d80adc",
  "ResponseMetadata": {
    "RequestId": "95d39377-8323-5ef0-9223-ceb198bd09bd",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "95d39377-8323-5ef0-9223-ceb198bd09bd",
      "date": "Wed, 08 Nov 2023 23:27:26 GMT",
      "content-type": "application/x-amz-json-1.0",
      "content-length": "106",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
```

```
}%
```

Step Functions の例

基本的な例

物理 ID をリソース ID として指定してステートマシンを呼び出します。

まず、`sam list resources` を使用して物理 ID を取得します。

```
$ sam list resources --stack-name state-machine-example --output json

[
  {
    "LogicalResourceId": "HelloWorldStateMachine",
    "PhysicalResourceId": "arn:aws:states:us-
west-2:513423067560:stateMachine:HelloWorldStateMachine-z69tFEUx0F66"
  },
  {
    "LogicalResourceId": "HelloWorldStateMachineRole",
    "PhysicalResourceId": "simple-state-machine-HelloWorldStateMachineRole-
PduA0BDGuFXw"
  }
]
```

次に、物理 ID をリソース ID として使用してステートマシンを呼び出します。コマンドラインで `--event` オプションを指定してイベントを渡します。

```
$ sam remote invoke arn:aws:states:us-
west-2:01234567890:stateMachine:HelloWorldStateMachine-z69tFEUx0F66 --
event '{"is_developer": true}'

Invoking Step Function arn:aws:states:us-
west-2:01234567890:stateMachine:HelloWorldStateMachine-z69tFEUx0F66
"Hello Developer World"%
```

空のイベントを渡してステートマシンを呼び出します。

```
$ sam remote invoke HelloWorldStateMachine --stack-name state-machine-example

Invoking Step Function HelloWorldStateMachine
```

```
"Hello World"%
```

関連リンク

`sam remote invoke` および の使用に関連するドキュメントの場合 AWS SAM CLI、以下を参照してください。

- sam remote invoke
- AWS SAMCLI トラブルシューティング

AWS SAM を使用してローカル統合テストを自動化する

sam local invoke を使用したテストの概要 を使用するとコードを手動でテストできますが、AWS SAM では自動化した統合テストを使用してコードをテストすることもできます。統合テストは、開発サイクルの早い段階で問題を検出し、コードの品質を向上させ、コストを削減しながら時間を節約するのに役立ちます。

AWS SAM 内に自動化された統合テストを作成するには、AWS クラウドにデプロイする前に、まずローカルの Lambda 関数に対するテストを実行します。を使用したテストの概要 sam local start-lambda コマンドは、Lambda 呼び出しエンドポイントをエミュレートするローカルのエンドポイントを起動します。これは、自動化されたテストから呼び出すことができます。このエンドポイントは Lambda 呼び出しエンドポイントをエミュレートするので、テストの記述後は、ローカルの Lambda 関数、またはデプロイされた Lambda 関数にも、変更なしでこれらのテストを実行できます。CI/CD パイプライン内のデプロイされた AWS SAM スタックに対しても、同じテストを実行できます。

プロセスの仕組みは以下のとおりです。

1. ローカル Lambda エンドポイントを起動する。

   AWS SAM テンプレートが含まれるディレクトリで以下のコマンドを実行して、ローカル Lambda エンドポイントを起動します。

   ```
   sam local start-lambda
   ```

   このコマンドは、`http://127.0.0.1:3001` をエミュレートする AWS Lambda でローカルエンドポイントを起動します。自動化されたテストは、このローカル Lambda エンドポイントに対して実行できます。AWS CLI または SDK を使用してこのエンドポイントを呼び出すと、リクエストで指定された Lambda 関数がローカルで実行され、レスポンスが返されます。

2. ローカル Lambda エンドポイントに対して統合テストを実行する。

   統合テストでは、テストデータで Lambda 関数を呼び出し、レスポンスを待機して、レスポンスが期待どおりであることを確認するために AWS SDK を使用できます。統合テストをローカルで実行するには、前のステップで起動したローカル Lambda エンドポイントを呼び出すための Lambda Invoke API コールを送信するように AWS SDK を設定する必要があります。

   以下は、Python の例です (他の言語の AWS SDK にも同じような設定があります)。

```
import boto3
import botocore

# Set "running_locally" flag if you are running the integration test locally
running_locally = True

if running_locally:

    # Create Lambda SDK client to connect to appropriate Lambda endpoint
    lambda_client = boto3.client('lambda',
        region_name="us-west-2",
        endpoint_url="http://127.0.0.1:3001",
        use_ssl=False,
        verify=False,
        config=botocore.client.Config(
            signature_version=botocore.UNSIGNED,
            read_timeout=15,
            retries={'max_attempts': 0},
        )
    )
else:
    lambda_client = boto3.client('lambda')


# Invoke your Lambda function as you normally usually do. The function will run
# locally if it is configured to do so
response = lambda_client.invoke(FunctionName="HelloWorldFunction")

# Verify the response
assert response == "Hello World"
```

このコードは、`running_locally` を `False` に設定することによって、デプロイされた Lambda 関数のテストに使用できます。これにより、AWS クラウド内の AWS Lambda に接続するための AWS SDK がセットアップされます。

AWS SAM を使用してサンプルのイベントペイロードを生成する

Lambda 関数をテストするには、他の AWS サービスによってトリガーされた際に Lambda 関数が受け取るデータを模倣する、サンプルのイベントペイロードを生成およびカスタマイズします。これには、API Gateway、AWS CloudFormation、Amazon S3 などのサービスが含まれます。

サンプルのイベントペイロードを生成すると、ライブ環境で作業することなく、さまざまな入力を使用して Lambda 関数の動作をテストできます。このアプローチは、 関数をテストするために AWS サービスイベントのサンプルを手動で作成する場合と比較して時間も節約します。

サンプルイベントペイロードを生成できるサービスの完全なリストについては、以下のコマンドを使用します。

```
sam local generate-event --help
```

特定のサービスに使用できるオプションのリストについては、以下のコマンドを使用します。

```
sam local generate-event [SERVICE] --help
```

例:

```
#Generates the event from S3 when a new object is created
sam local generate-event s3 put

# Generates the event from S3 when an object is deleted
sam local generate-event s3 delete
```

AWS SAM を使用してサーバーレスアプリケーションをデバッグする

アプリケーションをテストすることで、見つかった問題をデバッグする準備が整います。AWS SAM コマンドラインインターフェイス (CLI) では、サーバーレスアプリケーションを AWS クラウドにアップロードする前にローカルでテストおよびデバッグできます。アプリケーションのデバッグでは、アプリケーション内の問題やエラーを特定して修正します。

AWS SAM を使用してステップスルーデバッグを実行できます。これは、コードを一度に 1 行または命令実行する方法です。AWS SAM CLI 内でデバッグモードを使って Lambda 関数をローカルに呼び出す場合は、それにデバッガーをアタッチすることができます。デバッガーを使用すると、コードを 1 行ずつステップスルーする、異なる変数の値を確認する、および他のアプリケーションと同じように問題を修正することができます。アプリケーションをパッケージ化してデプロイするステップを実行する前に、アプリケーションが期待どおりに動作しているかどうかを確認し、問題をデバッグして修正できます。

> **Note**
>
> アプリケーションに 1 つ、または複数のレイヤーが含まれている場合は、アプリケーションをローカルで実行してデバッグすると、レイヤーパッケージがダウンロードされ、ローカルホストにキャッシュされます。詳細については、「レイヤーがローカルにキャッシュされる方法」を参照してください。

トピック

- AWS SAM を使用して関数をローカルにデバッグする
- AWS SAM を使用したデバッグ時に複数のランタイム引数を渡す
- AWS CloudFormation Linter による AWS SAM アプリケーションの検証

AWS SAM を使用して関数をローカルにデバッグする

AWS SAM をさまざまな AWS ツールキットと共に使用して、サーバーレスアプリケーションのローカルでのテストとデバッグを実行できます。Lambda 関数のステップスルーデバッグを使用すると、アプリケーションの 1 行または命令の問題を、ローカル環境の時点で特定し修正できます。

Lambda 関数のローカルでのステップスルーデバッグ実行には、ブレークポイントの設定、変数の調査、関数コードを一度に 1 行ずつ実行する、などの手法が含まれます。ローカルステップスルーデバッグは、クラウドで発生する可能性のある問題の検出とトラブルシューティングを可能することによって、フィードバックループを強化します。

AWS Toolkits を使用したデバッグに加え、AWS SAM をデバッグモードで実行することもできます。詳細については、このセクションの各トピックを参照してください。

AWS Toolkit の使用

AWS Toolkit は統合開発環境 (IDE) プラグインで、ブレークポイントの設定、変数の調査、および関数コードの一度に 1 行ずつの実行といった一般的なデバッグタスクの多くを実行する機能を提供します。AWSToolkit は、AWS SAM を使用して構築されたサーバーレスアプリケーションの開発、デバッグ、およびデプロイを容易にします。これらは、統合開発環境 (IDE) に統合された Lambda 関数の構築、テスト、デバッグ、デプロイ、および呼び出しのエクスペリエンスを提供します。

AWS SAM で使用できる AWS Toolkit の詳細については、以下を参照してください。

- AWS Toolkit for Visual Studio Code
- AWS Cloud9
- AWS Toolkit for JetBrains

IDE とランタイムの異なる組み合わせで動作する AWS Toolkit にはさまざまなものがあります。以下の表は、AWS SAM アプリケーションのステップスルーデバッグをサポートする一般的な IDE/ランタイムの組み合わせのリストです。

IDE	ランタイム	AWS Toolkit	ステップスルーデバッグの手順
Visual Studio Code	• Node.js • Python • .NET • Java • Go	AWS Toolkit for Visual Studio Code	AWS Toolkit for Visual Studio Code ユーザーガイドの「Working with AWS サーバーレスアプリケーション」

IDE	ランタイム	AWS Toolkit	ステップスルーデバッグの手順
AWS Cloud9	• Node.js • Python	AWS Cloud9 (AWS Toolkit が有効化されたもの)[1]	AWS Cloud9 ユーザーガイドの「Working with AWS serverless applications using the AWS Toolkit」
WebStorm	Node.js	AWS Toolkit for JetBrains[2]	AWS Toolkit for JetBrains の「Running (invoking) or debugging a local function」
PyCharm	Python	AWS Toolkit for JetBrains[2]	AWS Toolkit for JetBrains の「Running (invoking) or debugging a local function」
Rider	.NET	AWS Toolkit for JetBrains[2]	AWS Toolkit for JetBrains の「Running (invoking) or debugging a local function」
IntelliJ	Java	AWS Toolkit for JetBrains[2]	AWS Toolkit for JetBrains の「Running (invoking) or debugging a local function」

IDE	ランタイム	AWS Toolkit	ステップスルーデバッグの手順
GoLand	Go	AWS Toolkit for JetBrains[2]	AWS Toolkit for JetBrains の「Running (invoking) or debugging a local function」

注意:

1. AWS SAM アプリケーションのステップスルーデバッグに AWS Cloud9 を使用するには、AWS Toolkit が有効化されている必要があります。詳細については、AWS Cloud9ユーザーガイドの「Enabling the AWS Toolkit」を参照してください。
2. AWS SAM アプリケーションのステップスルーデバッグに AWS Toolkit for JetBrains を使用するには、まず AWS Toolkit for JetBrains の「Installing the AWS Toolkit for JetBrains」に記載されている手順に従って、ツールキットをインストールして設定する必要があります。

AWS SAM のデバッグモードでのローカル実行

AWS Toolkit との統合に加えて、ptvsd または delve といったサードパーティーデバッガーにアタッチするために、AWS SAM を「デバッグモード」で実行することもできます。

デバッグモードで AWS SAM を実行するには、`--debug-port` または `-d` オプションを使用した sam local invoke コマンドまたは sam local start-api を使用できます。

以下はその例です。

```
# Invoke a function locally in debug mode on port 5858
sam local invoke -d 5858 <function logical id>

# Start local API Gateway in debug mode on port 5858
sam local start-api -d 5858
```

> **Note**
>
> `sam local start-api` を使用している場合は、ローカル API Gateway インスタンスがすべての Lambda 関数を公開します。ただし、単一のデバッグポートしか指定できないため、

デバッグできるのは一度に 1 つの関数のみです。AWS SAM CLI がポートにバインドする前に、API を呼び出す必要があります。そうすることで、デバッガーによる接続が可能になります。

AWS SAM を使用したデバッグ時に複数のランタイム引数を渡す

AWS SAM で追加のランタイム引数を渡すことで、問題の調査と変数のトラブルシューティングをより効果的に行えます。これにより、デバッグプロセスに制御と柔軟性が追加されるので、カスタマイズされたランタイム設定と環境において便利です。

関数のデバッグ時に追加のランタイム引数を渡すには、環境変数 DEBUGGER_ARGS を使用します。これは、AWS SAM CLI が関数を開始するために使用する Run Command に引数の文字列を直接渡します。

例えば、Python 関数の実行時に iKPdb のようなデバッガーをロードする場合は、`DEBUGGER_ARGS: -m ikpdb --ikpdb-port=5858 --ikpdb-working-directory=/var/task/ --ikpdb-client-working-directory=/myApp --ikpdb-address=0.0.0.0` のように渡すことができます。これは、指定した他の引数と共に、iKPdb を実行時にロードします。

この場合、完全な AWS SAM CLI コマンドは以下のようになります。

```
DEBUGGER_ARGS="-m ikpdb --ikpdb-port=5858 --ikpdb-working-directory=/var/task/ --ikpdb-client-working-directory=/myApp --ikpdb-address=0.0.0.0" echo {} | sam local invoke -d 5858 myFunction
```

デバッガー引数は、すべてのランタイムの関数に渡すことができます。

AWS CloudFormation Linter による AWS SAM アプリケーションの検証

AWS CloudFormation Linter (cfn-lint) は、AWS CloudFormation テンプレートの詳細な検証を行うために使用できるオープンソースのツールです。cfn-lint には、AWS CloudFormation リソース仕様に基づくルールが含まれています。cfn-lint を使用してリソースをこれらのルールと比較すると、エラー、警告、情報提供に関する詳細なメッセージを受け取ることができます。または、独自のカスタムルールを作成して検証することもできます。cfn-lint の詳細については、AWS CloudFormation GitHub リポジトリの「cfn-lint」を参照してください。

cfn-lint を使用して --lint オプションで sam validate を実行すると、AWS SAM コマンドラインインターフェイス (AWS SAM CLI) から AWS Serverless Application Model (AWS SAM) テンプレートを検証できます。

```
sam validate --lint
```

カスタムルールの作成や検証オプションの指定など、cfn-lint の動作をカスタマイズするには、設定ファイルを定義します。詳細については、AWS CloudFormation GitHub リポジトリの「cfn-lint」にある「Config File」(設定ファイル) を参照してください。sam validate --lint を実行すると、設定ファイルで定義されている cfn-lint の動作が適用されます。

例

AWS SAM テンプレートで cfn-lint 検証を実行する

```
sam validate --lint --template myTemplate.yaml
```

詳細はこちら

sam validate コマンドの詳細については、「sam validate」を参照してください。

AWS SAM を使用してアプリケーションとリソースをデプロイする

アプリケーションをデプロイすると、AWS クラウドに AWS リソースをプロビジョニングおよび設定でき、アプリケーションをクラウドで実行できます。AWS SAM では、基盤となるデプロイメカニズムとして AWS CloudFormation が使用されます。また AWS SAM では、sam build コマンドの実行時に作成したビルドアーティファクトがサーバーレスアプリケーションをデプロイするための標準入力として使用されます。

AWS SAM を使用すると、サーバーレスアプリケーションを手動でデプロイすることも、デプロイを自動化することもできます。デプロイを自動化するには、選択した継続的インテグレーションおよび継続的デプロイ (CI/CD) システムを使用して AWS SAM パイプラインを使用します。デプロイパイプラインは、サーバーレスアプリケーションの新しいバージョンをリリースするために実行されるステップの自動シーケンスです。

このセクションのトピックでは、自動デプロイと手動デプロイの両方に関するガイダンスが提供されます。アプリケーションを手動でデプロイするには、AWS SAM CLI コマンドを使用します。デプロイを自動化するには、このセクションのトピックを参照してください。トピックでは特に、パイプラインと CI/CD システムを使用したデプロイの自動化に関する詳細なコンテンツが提供されます。これには、スターターパイプラインの生成、自動化の設定、デプロイのトラブルシューティング、OpenID Connect (OIDC) ユーザー認証の使用、デプロイ時のローカルファイルのアップロードが含まれます。

トピック

- でのデプロイの概要 AWS SAM
- AWS SAM を使用してアプリケーションをデプロイするためのオプション
- CI/CD システムとパイプラインを使用した AWS SAM によるデプロイ
- の使用の概要 sam sync 同期する AWS クラウド

でのデプロイの概要 AWS SAM

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam deploy` コマンドを使用して、サーバーレスアプリケーションを にデプロイします AWS クラウド。

- の概要 AWS SAM CLI「AWS SAMCLI とは？」を参照してください。

- `sam deploy` コマンドオプションのリストについては、「sam deploy」を参照してください。
- 一般的な開発ワークフローでの `sam deploy` の使用例については、「ステップ 3: アプリケーションを にデプロイする AWS クラウド」を参照してください。

トピック

前提条件

を使用するには`sam deploy`、 をインストールします。 AWS SAM CLI 以下を完了します。

- AWS SAM 前提条件.
- AWS SAM CLI のインストール.

`sam deploy` を使用する前に、次の基本を理解しておくことをお勧めします。

- AWS SAM CLI の設定.
- AWS SAM でアプリケーションを作成する.
- AWS SAM を使用した構築の概要.

sam deploy を使用したアプリケーションのデプロイ

サーバーレスアプリケーションを初めてデプロイする場合は、`--guided` オプションを使用します。の AWS SAM CLI は、アプリケーションのデプロイ設定を設定するインタラクティブフローをガイドします。

インタラクティブフローを使用してアプリケーションをデプロイするには

1. プロジェクトのルートディレクトリに移動します。これは AWS SAM テンプレートと同じ場所です。

```
$ cd sam-app
```

2. 次のコマンドを実行します。

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

3. インタラクティブフローでは、 AWS SAM CLI は、アプリケーションのデプロイ設定を設定するオプションをプロンプトします。

   括弧 ([ ]) はデフォルト値を示します。回答を空白のままにすると、デフォルト値が選択されます。デフォルト値は、次の設定ファイルから取得されます。

   - ~/.aws/config – 一般的な AWS アカウント設定。
   - ~/.aws/credentials – AWS アカウントの認証情報。
   - *<project>*/samconfig.toml – プロジェクトの設定ファイル。

   に応答して値を指定する AWS SAM CLI プロンプト。例えば、はいに **y**、いいえに **n**、または文字列値を入力できます。

   の AWS SAM CLI は、プロジェクトのsamconfig.tomlファイルに応答を書き込みます。以降のデプロイでは、これらの設定値で sam deploy を使用してデプロイできます。これらの値を再設定するには、sam deploy --guided を再度使用するか、設定ファイルを直接変更します。

   以下は、その出力例です。

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

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

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

        Setting default arguments for 'sam deploy'
```

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

4. 次に、 AWS SAM CLI はアプリケーションを にデプロイします AWS クラウド。デプロイ中、進行状況がコマンドプロンプトに表示されます。デプロイの主な段階は次のとおりです。

   - .zip ファイルアーカイブとしてパッケージ化された AWS Lambda 関数を持つアプリケーションの場合、 AWS SAM CLI パッケージを圧縮して Amazon Simple Storage Service (Amazon S3) バケットにアップロードします。必要に応じて、 AWS SAM CLI は新しいバケットを作成します。
   - Lambda 関数パッケージをコンテナイメージとして使用するアプリケーションの場合、 AWS SAM CLI はイメージを Amazon Elastic Container Registry (Amazon) にアップロードしますECR。必要に応じて、 AWS SAM CLI は新しいリポジトリを作成します。
   - の AWS SAM CLI は AWS CloudFormation 変更セットを作成し、アプリケーションをスタック AWS CloudFormation として にデプロイします。
   - の AWS SAM CLI Lambda 関数の新しいCodeUri値を使用してデプロイされた AWS SAM テンプレートを変更します。

   以下は、 AWS SAM CLI デプロイ出力：

```
        Looking for resources needed for deployment:

        Managed S3 bucket: aws-sam-cli-managed-default-samcliamzn-s3-demo-source-
bucket-1a4x26zbcdkqr
```

```
        A different default S3 bucket can be set in samconfig.toml and auto
 resolution of buckets turned off by setting resolve_s3=False

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

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

        Uploading to sam-app-zip/da3c598813f1c2151579b73ad788cac8  262144 / 619839
 (42.29%)Uploading to sam-app-zip/da3c598813f1c2151579b73ad788cac8  524288 / 619839
  (84.58%)Uploading to sam-app-zip/da3c598813f1c2151579b73ad788cac8  619839 /
 619839  (100.00%)

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

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

        Uploading to sam-app-zip/be84c20f868068e4dc4a2c11966edf2d.template  1212 /
 1212  (100.00%)


Waiting for changeset to be created..

CloudFormation stack changeset
-----------------------------------------------------------------------------------------
```

```
Operation                 LogicalResourceId         ResourceType
 Replacement
-------------------------------------------------------------------------------------
+ Add                     HelloWorldFunctionHell    AWS::Lambda::Permissio    N/A

                          oWorldPermissionProd      n

+ Add                     HelloWorldFunctionRole    AWS::IAM::Role            N/A

+ Add                     HelloWorldFunction        AWS::Lambda::Function     N/A

+ Add                     ServerlessRestApiDeplo    AWS::ApiGateway::Deplo    N/A

                          yment47fc2d5f9d           yment

+ Add                     ServerlessRestApiProdS    AWS::ApiGateway::Stage    N/A

                          tage

+ Add                     ServerlessRestApi         AWS::ApiGateway::RestA    N/A

                                                    pi

-------------------------------------------------------------------------------------


Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680559234/d9f58a77-98bc-41cd-
b9f4-433a5a450d7a


Previewing CloudFormation changeset before deployment
======================================================
Deploy this changeset? [y/N]: y

2023-04-03 12:00:50 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
-------------------------------------------------------------------------------------
ResourceStatus            ResourceType              LogicalResourceId
 ResourceStatusReason
-------------------------------------------------------------------------------------
CREATE_IN_PROGRESS        AWS::IAM::Role            HelloWorldFunctionRole    -
```

```
CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   Resource
 creation

 Initiated
CREATE_COMPLETE          AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       Resource
 creation

 Initiated
CREATE_COMPLETE          AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        -

                         pi

CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        Resource
 creation
                         pi
 Initiated
CREATE_COMPLETE          AWS::ApiGateway::RestA   ServerlessRestApi        -

                         pi

CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   -

                         n                        oWorldPermissionProd

CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -

                         yment                    yment47fc2d5f9d

CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   Resource
 creation
                         n                        oWorldPermissionProd
 Initiated
CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   Resource
 creation
                         yment                    yment47fc2d5f9d
 Initiated
CREATE_COMPLETE          AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -
```

```
                         yment                     yment47fc2d5f9d

CREATE_IN_PROGRESS       AWS::ApiGateway::Stage    ServerlessRestApiProdS   -

                                                   tage

CREATE_IN_PROGRESS       AWS::ApiGateway::Stage    ServerlessRestApiProdS   Resource
 creation
                                                   tage
 Initiated
CREATE_COMPLETE          AWS::ApiGateway::Stage    ServerlessRestApiProdS   -

                                                   tage

CREATE_COMPLETE          AWS::Lambda::Permissio    HelloWorldFunctionHell   -

                         n                         oWorldPermissionProd

CREATE_COMPLETE          AWS::CloudFormation::S    sam-app-zip              -

                         tack

-------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-------------------------------------------------------------------------------------------
Outputs

-------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/sam-app-zip-

HelloWorldFunctionRole-11ZOGSCG28H0M


Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World
 function
Value               https://njzfhdmls0.execute-api.us-west-2.amazonaws.com/Prod/
hello/
```

```
Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN

Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-

HelloWorldFunction-XPqNX4TBu7qn

-----------------------------------------------------------------------------------------------


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

5. デプロイされたアプリケーションを表示するには、次を実行します。

   1. URL https://console.aws.amazon.com/cloudformation を使用して AWS CloudFormation コンソールを直接開きます。
   2. [スタック] を選択します。
   3. アプリケーション名でスタックを識別し、それを選択します。

デプロイ前に変更を検証する

を設定できます。 AWS SAM CLI は AWS CloudFormation 、変更セットを表示し、デプロイする前に確認を求めます。

デプロイ前に変更を確認するには

1. `sam deploy --guided` 中、デプロイ前に **Y** と入力して変更を確認します。

   ```
   #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
   Confirm changes before deploy [Y/n]: Y
   ```

   あるいは、次のように `samconfig.toml` ファイルを変更することもできます。

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

2. デプロイ中、 は AWS SAM CLI は、デプロイ前に変更を確認するよう求められます。以下に例を示します。

```
Waiting for changeset to be created..

CloudFormation stack changeset
-----------------------------------------------------------------------------------------
Operation                LogicalResourceId        ResourceType
 Replacement
-----------------------------------------------------------------------------------------
+ Add                    HelloWorldFunctionHell   AWS::Lambda::Permissio   N/A

                         oWorldPermissionProd     n

+ Add                    HelloWorldFunctionRole   AWS::IAM::Role           N/A

+ Add                    HelloWorldFunction       AWS::Lambda::Function    N/A

+ Add                    ServerlessRestApiDeplo   AWS::ApiGateway::Deplo   N/A

                         yment47fc2d5f9d          yment

+ Add                    ServerlessRestApiProdS   AWS::ApiGateway::Stage   N/A

                         tage

+ Add                    ServerlessRestApi        AWS::ApiGateway::RestA   N/A

                                                  pi

-----------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680559234/d9f58a77-98bc-41cd-
b9f4-433a5a450d7a
Previewing CloudFormation changeset before deployment
======================================================
Deploy this changeset? [y/N]: y
```

デプロイ中に追加パラメータを指定する

デプロイ時に設定する追加のパラメータ値を指定できます。これを実行するには、デプロイ中に AWS SAM テンプレートを変更し、パラメータ値を設定します。

追加のパラメータを指定するには

1. AWS SAM テンプレートParametersのセクションを変更します。以下に例を示します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Globals:
...
Parameters:
  DomainName:
    Type: String
    Default: example
    Description: Domain name
```

2. sam deploy --guided を実行します。出力例を次に示します。

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

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

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

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app-zip]: ENTER
        AWS Region [us-west-2]: ENTER
        Parameter DomainName [example]: ENTER
```

Lambda 関数のコード署名を設定する

デプロイ時に Lambda 関数のコード署名を設定できます。これを行うには、 AWS SAM テンプレートを変更し、デプロイ中にコード署名を設定します。

コード署名を設定するには

1. AWS SAM テンプレートCodeSigningConfigArnで を指定します。以下に例を示します。

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.7
      CodeSigningConfigArn: arn:aws:lambda:us-east-1:111122223333:code-signing-
config:csc-12e12345db1234567
```

2. sam deploy --guided を実行します。の AWS SAM CLI は、コード署名を設定するよう求められます。以下は、その出力例です。

```
#Found code signing configurations in your function definitions
Do you want to sign your code? [Y/n]: ENTER
#Please provide signing profile details for the following functions & layers
#Signing profile details for function 'HelloWorld'
Signing Profile Name:
Signing Profile Owner Account ID (optional):
#Signing profile details for layer 'MyLayer', which is used by functions
 {'HelloWorld'}
Signing Profile Name:
Signing Profile Owner Account ID (optional):
```

ベストプラクティス

- を使用する場合sam deploy、 AWS SAM CLI は、 .aws-sam ディレクトリにあるアプリケーションのビルドアーティファクトをデプロイします。アプリケーションの元のファイルに変更を加える場合は、デプロイする前に sam build を実行して .aws-sam ディレクトリを更新します。
- アプリケーションを初めてデプロイする場合、sam deploy --guided を使用してデプロイ設定を構成します。以降のデプロイでは、構成済みの設定で sam deploy を使用してデプロイできます。

sam deploy のオプション

sam deploy 用に一般的に使用されるオプションを次に示します。すべてのオプションのリストについては、「sam deploy」を参照してください。

ガイド付きのインタラクティブフローを使用してアプリケーションをデプロイする

--guided オプションを使用して、インタラクティブフローを通じてアプリケーションのデプロイ設定を構成します。以下に例を示します。

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

アプリケーションのデプロイ設定はプロジェクトの samconfig.toml ファイルに保存されます。詳細については、「プロジェクト設定を構成する」を参照してください。

トラブルシューティング

をトラブルシューティングするには AWS SAM CLI「AWS SAMCLI トラブルシューティング」を参照してください。

例

.zip ファイルアーカイブとしてパッケージ化された Lambda 関数を含む Hello World アプリケーションをデプロイする

例については、Hello World アプリケーションのチュートリアルの「ステップ 3: アプリケーションをにデプロイする AWS クラウド」を参照してください。

コンテナイメージとしてパッケージ化された Lambda 関数を含む Hello World アプリケーションをデプロイする

まず、sam init を使用して Hello World アプリケーションを作成します。インタラクティブフロー中に、Python3.9 ランタイムと Image パッケージタイプを選択します。

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

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        ...
```

```
Template: 1

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

Which runtime would you like to use?
        1 - aot.dotnet7 (provided.al2)
        ...
        15 - nodejs12.x
        16 - python3.9
        17 - python3.8
        ...
Runtime: 16

What package type would you like to use?
        1 - Zip
        2 - Image
Package type: 2

Based on your selections, the only dependency manager available is pip.
We will proceed copying the template using pip.
...
Project name [sam-app]: ENTER

    -----------------------
    Generating application:
    -----------------------
    Name: sam-app
    Base Image: amazon/python3.9-base
    Architectures: x86_64
    Dependency Manager: pip
    Output Directory: .
    Configuration file: sam-app/samconfig.toml

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

次に、cd を実行してプロジェクトのルートディレクトリに移動し、sam build を実行します。の AWS SAM CLI を使用して Lambda 関数をローカルに構築する Docker.

```
sam-app $ sam build
Building codeuri: /Users/.../sam-app runtime: None metadata: {'Dockerfile':
 'Dockerfile', 'DockerContext': '/Users/.../sam-app/hello_world', 'DockerTag':
 'python3.9-v1'} architecture: x86_64 functions: HelloWorldFunction
```

```
Building image for HelloWorldFunction function
Setting DockerBuildArgs: {} for HelloWorldFunction function
Step 1/5 : FROM public.ecr.aws/lambda/python:3.9
 ---> 0a5e3da309aa
Step 2/5 : COPY requirements.txt ./
 ---> abc4e82e85f9
Step 3/5 : RUN python3.9 -m pip install -r requirements.txt -t .
 ---> [Warning] The requested image's platform (linux/amd64) does not match the
 detected host platform (linux/arm64/v8) and no specific platform was requested
 ---> Running in 43845e7aa22d
Collecting requests
  Downloading requests-2.28.2-py3-none-any.whl (62 kB)
     ######################################## 62.8/62.8 KB 829.5 kB/s eta 0:00:00
Collecting idna<4,>=2.5
  Downloading idna-3.4-py3-none-any.whl (61 kB)
     ######################################## 61.5/61.5 KB 2.4 MB/s eta 0:00:00
Collecting charset-normalizer<4,>=2
  Downloading charset_normalizer-3.1.0-cp39-cp39-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (199 kB)
     ######################################## 199.2/199.2 KB 2.1 MB/s eta 0:00:00
Collecting certifi>=2017.4.17
  Downloading certifi-2022.12.7-py3-none-any.whl (155 kB)
     ######################################## 155.3/155.3 KB 10.2 MB/s eta 0:00:00
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.15-py2.py3-none-any.whl (140 kB)
     ######################################## 140.9/140.9 KB 9.1 MB/s eta 0:00:00
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests
Successfully installed certifi-2022.12.7 charset-normalizer-3.1.0 idna-3.4
 requests-2.28.2 urllib3-1.26.15
Removing intermediate container 43845e7aa22d
 ---> cab8ace899ce
Step 4/5 : COPY app.py ./
 ---> 4146f3cd69f2
Step 5/5 : CMD ["app.lambda_handler"]
 ---> [Warning] The requested image's platform (linux/amd64) does not match the
 detected host platform (linux/arm64/v8) and no specific platform was requested
 ---> Running in f4131ddffb31
Removing intermediate container f4131ddffb31
 ---> d2f5180b2154
Successfully built d2f5180b2154
Successfully tagged helloworldfunction:python3.9-v1


Build Succeeded
```

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

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

次に、アプリケーションをデプロイするために sam deploy --guided を実行します。の AWS SAM CLI は、デプロイ設定の設定をガイドします。次に、 AWS SAM CLI はアプリケーションを にデプロイします AWS クラウド。

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

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

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

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

        Looking for resources needed for deployment:
```

```
        Managed S3 bucket: aws-sam-cli-managed-default-samcliamzn-s3-demo-source-
bucket-1a4x26zbcdkqr
        A different default S3 bucket can be set in samconfig.toml and auto resolution
 of buckets turned off by setting resolve_s3=False

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

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

e95fc5e75742: Pushed
d8df51e7bdd7: Pushed
b1d0d7e0b34a: Pushed
0071317b94d8: Pushed
d98f98baf147: Pushed
2d244e0816c6: Pushed
eb2eeb1ebe42: Pushed
a5ca065a3279: Pushed
fe9e144829c9: Pushed
helloworldfunction-d2f5180b2154-python3.9-v1: digest:
 sha256:cceb71401b47dc3007a7a1e1f2e0baf162999e0e6841d15954745ecc0c447533 size: 2206

        Deploying with following values
        ===============================
        Stack name                   : sam-app
        Region                       : us-west-2
        Confirm changeset            : True
        Disable rollback             : False
        Deployment image repository  :
                                       {
                                           "HelloWorldFunction":
 "012345678910.dkr.ecr.us-west-2.amazonaws.com/samapp7427b055/
helloworldfunction19d43fc4repo"
                                       }
        Deployment s3 bucket         : aws-sam-cli-managed-default-samcliamzn-s3-demo-
source-bucket-1a4x26zbcdkqr
```

```
        Capabilities                 : ["CAPABILITY_IAM"]
        Parameter overrides          : {}
        Signing Profiles             : {}

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

HelloWorldFunction may not have authorization defined.
        Uploading to sam-app/682ad27c7cf7a17c7f77a1688b0844f2.template  1328 / 1328
 (100.00%)


Waiting for changeset to be created..

CloudFormation stack changeset
-------------------------------------------------------------------------------------------------
Operation                LogicalResourceId        ResourceType             Replacement

-------------------------------------------------------------------------------------------------
+ Add                    HelloWorldFunctionHell   AWS::Lambda::Permissio   N/A

                         oWorldPermissionProd     n

+ Add                    HelloWorldFunctionRole   AWS::IAM::Role           N/A

+ Add                    HelloWorldFunction       AWS::Lambda::Function    N/A

+ Add                    ServerlessRestApiDeplo   AWS::ApiGateway::Deplo   N/A

                         yment47fc2d5f9d          yment

+ Add                    ServerlessRestApiProdS   AWS::ApiGateway::Stage   N/A

                         tage

+ Add                    ServerlessRestApi        AWS::ApiGateway::RestA   N/A

                                                  pi

-------------------------------------------------------------------------------------------------
```

```
Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680634124/0ffd4faf-2e2b-487e-
b9e0-9116e8299ac4


Previewing CloudFormation changeset before deployment
======================================================
Deploy this changeset? [y/N]: y

2023-04-04 08:49:15 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
---------------------------------------------------------------------------------------------------
ResourceStatus           ResourceType             LogicalResourceId                          
 ResourceStatusReason
---------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS       AWS::CloudFormation::S   sam-app                 User
 Initiated
                         tack

CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   Resource
 creation
                                                                           Initiated

CREATE_COMPLETE          AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       Resource
 creation
                                                                           Initiated

CREATE_COMPLETE          AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        -

                         pi

CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        Resource
 creation
                         pi                                                Initiated
```

```
CREATE_COMPLETE          AWS::ApiGateway::RestA   ServerlessRestApi        -
                         pi
CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   -
                         n                        oWorldPermissionProd
CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -
                         yment                    yment47fc2d5f9d
CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   Resource
 creation
                         n                        oWorldPermissionProd     Initiated
CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   Resource
 creation
                         yment                    yment47fc2d5f9d          Initiated
CREATE_COMPLETE          AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -
                         yment                    yment47fc2d5f9d
CREATE_IN_PROGRESS       AWS::ApiGateway::Stage   ServerlessRestApiProdS   -
                                                  tage
CREATE_IN_PROGRESS       AWS::ApiGateway::Stage   ServerlessRestApiProdS   Resource
 creation
                                                  tage                     Initiated
CREATE_COMPLETE          AWS::ApiGateway::Stage   ServerlessRestApiProdS   -
                                                  tage
CREATE_COMPLETE          AWS::Lambda::Permissio   HelloWorldFunctionHell   -
                         n                        oWorldPermissionProd
CREATE_COMPLETE          AWS::CloudFormation::S   sam-app                  -
                         tack
```

```
---------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
---------------------------------------------------------------------------------------------------
Outputs

---------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/sam-app-HelloWorldFunctionRole-
JFML1JOKHJ71

Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World function

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


Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN

Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-
kyg6Y2iNRUPg

---------------------------------------------------------------------------------------------------


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

詳細

の使用の詳細については、 AWS SAM CLI `sam deploy` コマンドについては、以下を参照してください。

- 完全な AWS SAM ワークショップ: モジュール 3 - 手動でデプロイする – を使用してサーバーレスアプリケーションを構築、パッケージ化、デプロイする方法について説明します。 AWS SAM CLI.

AWS SAM を使用してアプリケーションをデプロイするためのオプション

AWS SAM ではアプリケーションのデプロイを手動で行いますが、デプロイを自動化することもできます。アプリケーションを手動でデプロイするには AWS SAM CLI を使用します。デプロイを自動化するには、パイプラインと、継続的な統合/継続的なデプロイ (CI/CD) を使用します。このセクションのトピックでは、両方のアプローチについての情報を提供しています。

トピック

- 手動によるデプロイのための AWS SAM CLI の使用法
- CI/CD システムとパイプラインを使用したデプロイ
- 段階的なデプロイ
- AWS SAM CLI を使用したデプロイのトラブルシューティング
- 詳細はこちら

手動によるデプロイのための AWS SAM CLI の使用法

サーバーレスアプリケーションをローカルで開発してテストしたら、sam deploy コマンドを使用してアプリケーションをデプロイできます。

AWS SAM によるプロンプトを用いたデプロイガイドを使用するには、--guided フラグを指定します。このフラグを指定すると、sam deploy コマンドがアプリケーションアーティファクトを zip で圧縮し、それらを Amazon Simple Storage Service (Amazon S3) (.zip ファイルアーカイブの場合) または Amazon Elastic Container Registry (Amazon ECR) (コンテナイメージの場合) のいずれかにアップロードします。その後、コマンドがアプリケーションを AWS クラウドにデプロイします。

例:

```
# Deploy an application using prompts:
sam deploy --guided
```

CI/CD システムとパイプラインを使用したデプロイ

AWS SAM は、パイプラインと継続的な統合/継続的なデプロイ (CI/CD) システムを使用したデプロイの自動化に役立ちます。パイプラインを作成し、サーバーレスアプリケーションの CI/CD タスクを簡素化するために AWS SAM を使用します。複数の CI/CD システムが AWS SAM のビルドコンテ

ナイメージをサポートしており、さらに AWS SAM では、デプロイに関する AWS のベストプラクティスをカプセル化する、複数の CI/CD システム用のデフォルトパイプラインテンプレートもいくつか提供しています。

詳細については、「CI/CD システムとパイプラインを使用した AWS SAM によるデプロイ」を参照してください。

段階的なデプロイ

AWS SAM アプリケーションを一度にデプロイするのではなく、段階的にデプロイする場合は、AWS CodeDeploy が提供するデプロイ設定を指定できます。詳細については、AWS CodeDeploy ユーザーガイドの「Working with deployment configurations in CodeDeploy」を参照してください。

段階的なデプロイのための AWS SAM アプリケーションの設定については、「AWS SAM を使用したサーバーレスアプリケーションの段階的なデプロイ」を参照してください。

AWS SAM CLI を使用したデプロイのトラブルシューティング

AWS SAM CLI エラー:「セキュリティの制約に準拠していません」

sam deploy --guided の実行時に、`HelloWorldFunction may not have authorization defined, Is this okay? [y/N]` という質問のプロンプトが表示されます。このプロンプトに「**N**」(デフォルトのレスポンス) と答えた場合、以下のエラーが表示されます。

```
Error: Security Constraints Not Satisfied
```

このプロンプトは、デプロイしようとしているアプリケーションに、認可なしで設定された Amazon API Gateway API が存在する可能性があることを知らせています。このプロンプトに「**N**」と答えることによって、この状態は望ましくないと伝えることになります。

この問題を解決するには、以下のオプションがあります。

- 認可を使用してアプリケーションを設定する。認可の設定については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。
- この質問に「**Y**」と答えて、認可なしで設定された API Gateway API 含むアプリケーションのデプロイを許容することを示します。

詳細はこちら

サーバーレスアプリケーションをデプロイする実践的な例については、「包括的な AWS SAM ワークショップ」の次のセクションを参照してください:

- モジュール 3 - 手動でデプロイする – AWS SAM CLI を使用してサーバーレスアプリケーションを構築、パッケージ化、デプロイする方法を学びます。
- モジュール 4 - CI/CD – 継続的インテグレーションおよびデリバリー (CI/CD) パイプラインを作成することで、構築、パッケージ化、デプロイのフェーズを自動化する方法を学びます。

CI/CD システムとパイプラインを使用した AWS SAM によるデプロイ

AWS SAM は、組織が希望する CI/CD システム用のパイプラインを作成するために役立つことから、デプロイ頻度の短縮、変更のリードタイムの短縮、およびデプロイエラーの低減などの CI/CD のメリットを最小限の労力で実現できます。

AWS SAM は、ビルドコンテナイメージを活用して、サーバーレスアプリケーションの CI/CD タスクを簡素化します。AWS SAM が提供するイメージには、AWS SAM CLI、およびサポートされている多数の AWS Lambda ランタイム用のビルドツールが含まれています。これは、AWS SAM CLI を使用したサーバーレスアプリケーションの構築とパッケージ化を容易にします。これらのイメージは、チームが CI/CD システム用に独自のイメージを作成して管理する必要性も軽減します。AWS SAM ビルドコンテナイメージの詳細については、「のイメージリポジトリ AWS SAM」を参照してください。

AWS SAM ビルドコンテナイメージは、複数の CI/CD システムでサポートされています。どの CI/CD システムを使用するかの判断は、いくつかの要因に左右されます。これには、アプリケーションが単一のランタイムを使用するのか、それとも複数のランタイムを使用するのか、またはアプリケーションがコンテナイメージ内で構築されるのか、それともホストマシン (仮想マシン (VM) またはベアメタルホスト) で直接構築されるのか、などがあります。

AWS SAM は、デプロイに関する AWS のベストプラクティスをカプセル化する、複数の CI/CD システム用のデフォルトパイプラインテンプレートの一式も提供します。これらのデフォルトパイプラインテンプレートは標準の JSON/YAML パイプライン設定形式を使用し、組み込まれたベストプラクティスは、マルチアカウントおよびマルチリージョンのデプロイを実行し、パイプラインがインフラストラクチャに意図しない変更を行えないことを実証するために役立ちます。

サーバーレスアプリケーションをデプロイするための AWS SAM の使用には、1) AWS SAM CLI コマンドを使用するように既存のパイプラインを変更する、または 2) 独自のアプリケーションの開始点として使用できる CI/CD パイプラインの設定例を生成するという 2 つの主なオプションがあります。

トピック

パイプラインとは

パイプラインは、アプリケーションの新しいバージョンをリリースするために実行されるステップの自動シーケンスです。AWS SAM では、アプリケーションのデプロイに AWS CodePipeline、Jenkins、GitLab CI/CD、および GitHub Actions などの一般的な CI/CD システムを多数使用できます。

パイプラインテンプレートには、マルチアカウントおよびマルチリージョンのデプロイに役立つ AWS デプロイのベストプラクティスが含まれています。開発や本番稼働などの AWS 環境は通常、異なる AWS アカウントに存在します。これにより開発チームは、インフラストラクチャに意図しない変更を加えることなく、安全なデプロイパイプラインを設定できます。

また、独自のカスタムパイプラインテンプレートを提供して、開発チーム全体のパイプラインを標準化することもできます。

デプロイ時にローカルファイル AWS SAM をアップロードする方法

アプリケーションを にデプロイする場合 AWS クラウド、 は、ローカルファイルを Amazon Simple Storage Service (Amazon S3) などのアクセス可能な AWS サービスに最初にアップロード AWS CloudFormation する必要があります。これにより、 AWS SAM テンプレートが参照するローカルファイルが取り込まれます。この要件を満たすには、 AWS SAM CLI `sam deploy` または `sam package` コマンドを使用する場合、 は以下を実行します。

1. ローカルファイルをアクセス可能な AWS サービスに自動的にアップロードします。

2. 新しいファイルパスを参照するようにアプリケーションテンプレートが自動的に更新されます。

トピック

- デモ: を使用する AWS SAM CLI Lambda 関数コードをアップロードするには
- 対応するユースケース
- 詳細

デモ: を使用する AWS SAM CLI Lambda 関数コードをアップロードするには

このデモでは、Lambda 関数の.zip パッケージタイプを使用して Hello World サンプルアプリケーションを初期化します。当社では、 AWS SAM CLI Lambda 関数コードを自動的に Amazon S3 にアップロードし、アプリケーションテンプレートで新しいパスを参照します。

まず、`sam init` を実行してHello World アプリケーションを初期化します。

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

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        ...
Template: 1

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

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

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

Project name [sam-app]: demo

    -----------------------
```

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

...
```

Lambda 関数コードは、プロジェクトの hello_world のサブディレクトリにまとめられています。

```
demo
### README.md
### hello_world
#   ### __init__.py
#   ### app.py
#   ### requirements.txt
### template.yaml
### tests
```

AWS SAM テンプレート内では、 CodeUriプロパティを使用して Lambda 関数コードへのローカルパスを参照します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # More info about Function Resource:
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.9
      ...
```

次に、sam build を実行してアプリケーションを構築し、デプロイの準備をします。

```
$ sam build
Starting Build use cache
Manifest file is changed (new hash: 3298f13049d19cffaa37ca931dd4d421) or dependency
 folder (.aws-sam/deps/7896875f-9bcc-4350-8adb-2c1d543627a1) is missing for
 (HelloWorldFunction), downloading dependencies and copying/building source
Building codeuri: /Users/.../demo/hello_world runtime: python3.9 metadata: {}
 architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CleanUp
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource
Running PythonPipBuilder:CopySource

Build Succeeded

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

次に、アプリケーションをデプロイするために sam deploy --guided を実行します。

```
$ sam deploy --guided

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

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

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

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

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

File with same data already exists at demo/da3c598813f1c2151579b73ad788cac8, skipping
 upload

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

Initiating deployment
=====================
...
Waiting for changeset to be created..
CloudFormation stack changeset
-----------------------------------------------------------------------------------------
Operation                LogicalResourceId        ResourceType             Replacement

-----------------------------------------------------------------------------------------
+ Add                    HelloWorldFunctionHell   AWS::Lambda::Permissio   N/A

                         oWorldPermissionProd     n

+ Add                    HelloWorldFunctionRole   AWS::IAM::Role           N/A

...
```

```
------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680906292/1164338d-72e7-4593-a372-
f2b3e67f542f

2023-04-07 12:24:58 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
------------------------------------------------------------------------------------------------
ResourceStatus           ResourceType             LogicalResourceId
 ResourceStatusReason
------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   Resource
 creation
                                                                           Initiated

...
------------------------------------------------------------------------------------------------
CloudFormation outputs from deployed stack
------------------------------------------------------------------------------------------------
Outputs

------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/demo-HelloWorldFunctionRole-
VQ4CU7UY7S2K

Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World function

Value               https://satnon55e9.execute-api.us-west-2.amazonaws.com/Prod/hello/


Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN
```

```
Value                arn:aws:lambda:us-west-2:012345678910:function:demo-

HelloWorldFunction-G14inKTmSQvK

------------------------------------------------------------------------------------------------
Successfully created/updated stack - demo in us-west-2
```

デプロイ中、 は AWS SAM CLI は Lambda 関数コードを Amazon S3 に自動的にアップロードし、テンプレートを更新します。 AWS CloudFormation コンソールで変更されたテンプレートは、Amazon S3 バケットパスを反映しています。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://aws-sam-cli-managed-default-samcliamzn-s3-demo-source-
bucket-1a4x26zbcdkqr/demo/da3c598813f1c2151579b73ad788cac8
      Handler: app.lambda_handler
      ...
```

対応するユースケース

の AWS SAM CLI は、多数のファイルタイプ、 AWS CloudFormation リソースタイプ、 AWS CloudFormation マクロに対してこのプロセスを自動的に促進できます。

ファイルタイプ

アプリケーションファイルと Docker イメージがサポートされています。

AWS CloudFormation リソースタイプ

以下は、サポートされているリソースタイプとそれらのプロパティのリストです。

リソース	プロパティ
AWS::ApiGateway::RestApi	BodyS3Location
AWS::ApiGatewayV2::Api	BodyS3Location

リソース	プロパティ
AWS::AppSync:FunctionConfiguration	CodeS3Location RequestMappingTemplateS3Location ResponseMappingTemplateS3Location
AWS::AppSync::GraphQLSchema	DefinitionS3Location
AWS::AppSync::Resolver	CodeS3Location RequestMappingTemplateS3Location ResponseMappingTemplateS3Location
AWS::CloudFormation::ModuleVersion	ModulePackage
AWS::CloudFormation::ResourceVersion	SchemaHandlerPackage
AWS::ECR::Repository	RepositoryName
AWS::ElasticBeanstalk::ApplicationVersion	SourceBundle
AWS::Glue::Job	Command.ScriptLocation
AWS::Lambda::Function	Code Code.ImageUri
AWS::Lambda::LayerVersion	Content
AWS::Serverless::Api	DefinitionUri

リソース	プロパティ
AWS::Serverless::Function	CodeUri ImageUri
AWS::Serverless::GraphQLApi	SchemaUri Function.CodeUri Resolver.CodeUri
AWS::Serverless::HttpApi	DefinitionUri
AWS::Serverless::LayerVersion	ContentUri
AWS::Serverless::StateMachine	DefinitionUri
AWS::StepFunctions::StateMachine	DefinitionS3Location

AWS CloudFormation マクロ

AWS::Include 変換マクロを使用して参照されるファイルはサポートされています。

詳細

AWS::Include 変換の詳細については、「 ユーザーガイド」の AWS::Include 「変換」を参照してください。 AWS CloudFormation

AWS SAM テンプレートでAWS::Include変換を使用する例については、「Serverless Land でパターンAPIHTTPAPI化するゲートウェイSQS」を参照してください。

AWS SAM を使用してスターター CI/CD パイプラインを生成する

デプロイを自動化する準備ができたら、AWS SAM のスターターパイプラインテンプレートのいずれかを使用して、使用する CI/CD システムのデプロイパイプラインを生成できます。デプロイパイプラインは、サーバーレスアプリケーションのデプロイを自動化するために設定および使用するものです。スターターパイプラインテンプレートは、サーバーレスアプリケーションのデプロイパイプラインをすばやくセットアップできるように事前設定されています。

スターターパイプラインテンプレートを使用すると、sam pipeline init コマンドを使用して数分でパイプラインを生成できます。

スターターパイプラインテンプレートは CI/CD システムの使い慣れた JSON/YAML 構文を使用し、複数のアカウントとリージョンにまたがるアーティファクトの管理、およびアプリケーションのデプロイに必要な最小限の許可の使用などのベストプラクティスを組み込みます。AWS SAM CLI は現在、AWS CodePipeline、Jenkins、GitLab CI/CD、GitHub Actions、および Bitbucket Pipelines 向けのスターター CI/CD パイプライン設定の生成をサポートしています。

以下は、スターターパイプライン設定を生成するために実行する必要があるおおまかなタスクです。

1. インフラストラクチャリソースを作成する - パイプラインには、必要な許可を持つ IAM ユーザーとロール、Amazon S3 バケット、およびオプションの Amazon ECR リポジトリなどの特定の AWS リソースが必要です。
2. Git リポジトリを CI/CD システムに接続する - CI/CD システムは、どの Git リポジトリがパイプラインの実行をトリガーするのかを知る必要があります。使用している Git リポジトリと CI/CD システムの組み合わせによっては、このステップが必要ない場合があります。
3. パイプライン設定を生成する - このステップは、2 つのデプロイステージが含まれるスターターパイプライン設定を生成します。
4. パイプライン設定を Git リポジトリにコミットする - このステップは、CI/CD システムがパイプライン設定を認識しており、変更のコミット時に実行されることを確実にするために必要です。

スターターパイプライン設定を生成して Git リポジトリにコミットしたら、そのリポジトリにコード変更がコミットされるたびにパイプラインがトリガーされ、自動的に実行されます。

これらのステップの順序と各ステップの詳細は、CI/CD システムに応じて異なります。

- AWS CodePipeline を使用している場合は、「AWS SAM での AWS CodePipeline 向けのスターターパイプラインの生成」を参照してください。
- Jenkins、GitLab CI/CD、GitHub Actions、または Bitbucket Pipelines を使用している場合は、AWS SAM を使用して Jenkins、GitLab CI/CD、GitHub Actions、Bitbucket Pipelines 向けのスターターパイプラインを生成するを参照してください。

AWS SAM での AWS CodePipeline 向けのスターターパイプラインの生成

AWS CodePipeline 向けのスターターパイプライン設定を生成するには、以下のタスクをこの順序どおりに実行します。

1. インフラストラクチャリソースを作成する
2. パイプライン設定を生成する
3. パイプライン設定を Git にコミットする
4. Git リポジトリを CI/CD システムに接続する

> **Note**
>
> 以下の手順は、`sam pipeline bootstrap` と `sam pipeline init` の 2 つの AWS SAM CLI コマンドを活用します。2 つのコマンドを使用する理由は、管理者 (IAM ユーザーやロールといったインフラストラクチャ AWS リソースをセットアップする許可が必要なユーザー) がデベロッパー (個々のパイプラインをセットアップする許可だけが必要で、インフラストラクチャ AWS リソースをセットアップする許可は必要ないユーザー) よりも多くの許可を持つユースケースを処理するためです。

ステップ 1: インフラストラクチャリソースを作成する

AWS SAM を使用するパイプライには、必要な許可を持つ IAM ユーザーとロール、Amazon S3 バケット、およびオプションの Amazon ECR リポジトリといった特定の AWS リソースが必要です。パイプラインのデプロイステージごとに、一連のインフラストラクチャリソースが必要です。

このセットアップには、以下のコマンドを実行できます。

```
sam pipeline bootstrap
```

> **Note**
>
> 上記のコマンドは、パイプラインのデプロイステージごとに実行します。

ステップ 2: パイプライン設定を生成する

パイプライン設定を生成するには、以下のコマンドを実行します。

```
sam pipeline init
```

ステップ 3: パイプライン設定を Git リポジトリにコミットする

このステップは、CI/CD システムがパイプライン設定を認識しており、変更のコミット時に実行されることを確実にするために必要です。

ステップ 4 : Git リポジトリを CI/CD システムに接続する

AWS CodePipeline の場合は、この時点で以下のコマンドを実行して接続を確立できます。

```
sam deploy -t codepipeline.yaml --stack-name <pipeline-stack-name> --
capabilities=CAPABILITY_IAM --region <region-X>
```

GitHub または Bitbucket を使用している場合は、上記の sam deploy コマンドを実行してから、デベロッパーツールコンソールユーザーガイドの保留中の接続の更新トピックにある接続を完了するにはに記載されている手順を実行して接続を確立します。それに加えて、sam deploy コマンドの出力からの CodeStarConnectionArn のコピーを保存しておきます。main 以外のブランチで AWS CodePipeline を使用する場合に必要となるからです。

その他ブランチの設定

デフォルトで、AWS CodePipeline は AWS SAM で main ブランチを使用します。main 以外のブランチを使用する場合は、sam deploy コマンドを再度実行する必要があります。使用している Git リポジトリによっては、CodeStarConnectionArn も提供する必要が生じる場合があることに注意してください。

```
# For GitHub and Bitbucket
sam deploy -t codepipeline.yaml --stack-name <feature-pipeline-stack-name> --
capabilities=CAPABILITY_IAM --parameter-overrides="FeatureGitBranch=<branch-name>
 CodeStarConnectionArn=<codestar-connection-arn>"

# For AWS CodeCommit
sam deploy -t codepipeline.yaml --stack-name <feature-pipeline-stack-name> --
capabilities=CAPABILITY_IAM --parameter-overrides="FeatureGitBranch=<branch-name>"
```

詳細はこちら

CI/CD パイプラインを設定する実践的な例については、「包括的な AWS SAM ワークショップ」の「AWS CodePipeline で CI/CD を使用する」を参照してください。

AWS SAM を使用して Jenkins、GitLab CI/CD、GitHub Actions、Bitbucket Pipelines 向けのスターターパイプラインを生成する

Jenkins、GitLab CI/CD、GitHub Actions、Bitbucket Pipelines 向けのスターターパイプライン設定を生成するには、以下のタスクをこの順序どおりに実行します。

1. インフラストラクチャリソースを作成する
2. Git リポジトリを CI/CD システムに接続する
3. 認証情報オブジェクトを作成する
4. パイプライン設定を生成する
5. パイプライン設定を Git リポジトリにコミットする

> **Note**
>
> 以下の手順は、`sam pipeline bootstrap` と `sam pipeline init` の 2 つの AWS SAM CLI コマンドを活用します。2 つのコマンドを使用する理由は、管理者 (IAM ユーザーやロールといったインフラストラクチャ AWS リソースをセットアップする許可が必要なユーザー) がデベロッパー (個々のパイプラインをセットアップする許可だけが必要で、インフラストラクチャ AWS リソースをセットアップする許可は必要ないユーザー) よりも多くの許可を持つユースケースを処理するためです。

ステップ 1: インフラストラクチャリソースを作成する

AWS SAM を使用するパイプライには、必要な許可を持つ IAM ユーザーとロール、Amazon S3 バケット、およびオプションの Amazon ECR リポジトリといった特定の AWS リソースが必要です。パイプラインのデプロイステージごとに、一連のインフラストラクチャリソースが必要です。

このセットアップには、以下のコマンドを実行できます。

```
sam pipeline bootstrap
```

> **Note**
>
> 上記のコマンドは、パイプラインのデプロイステージごとに実行します。

後続のステップで必要になるため、パイプラインのデプロイステージごとにパイプラインユーザーの AWS 認証情報 (キー ID とシークレットキー) をキャプチャする必要があります。

ステップ 2: Git リポジトリを CI/CD システムに接続する

構築とデプロイのために CI/CD システムがアプリケーションソースコードにアクセスできるようにするには、Git リポジトリを CI/CD システムに接続する必要があります。

> **Note**
>
> 以下の組み合わせのいずれかを使用している場合は接続が自動的に行われるため、このステップを省略できます。
>
> 1. GitHub Actions と GitHub リポジトリ
> 2. GitLab CI/CD と GitLab リポジトリ
> 3. Bitbucket リポジトリを持つ Bitbucket Pipelines

Git リポジトリを CI/CD システムに接続するには、以下のいずれかを実行します。

- Jenkins を使用している場合は、「ブランチソースの追加」について Jenkins ドキュメントを参照してください。
- GitLab CI/CD と Git リポジトリ (GitLab 以外) を使用している場合は、「外部リポジトリの接続」について GitLab ドキュメントを参照してください。

ステップ 3: 認証情報オブジェクトを作成する

各 CI/CD システムには、CI/CD システムが Git リポジトリにアクセスするために必要な認証情報を管理するための独自の方法があります。

必要な認証情報オブジェクトを作成するには、以下のいずれかを実行します。

- Jenkins を使用している場合は、キー ID とシークレットキーの両方を保存する単一の「認証情報」を作成します。Building a Jenkins Pipeline with AWS SAM ブログの「Configure Jenkins」セクションにある手順に従ってください。次のステップでは「認証情報 ID」が必要になります。
- GitLab CI/CD を使用している場合は、2 つの「保護された変数」(キー ID とシークレットキーごとに 1 つずつ) を作成します。GitLab ドキュメントにある手順に従ってください。次のステップでは 2 つの「変数キー」が必要になります。

- GitHub Actions を使用している場合は、2 つの「暗号化された秘密」(キーとシークレットキーごとに 1 つずつ) を作成します。GitHub ドキュメントにある手順に従ってください。次のステップでは 2 つの「シークレット名」が必要になります。
- Bitbucket Pipelines を使用している場合は、2 つの「セキュア変数」(キー ID とシークレットキーごとに 1 つずつ) を作成します。変数とシークレットにある手順に従ってください。次のステップでは 2 つの「シークレット名」が必要になります。

ステップ 4: パイプライン設定を生成する

パイプライン設定を生成するには、以下のコマンドを実行します。前のステップで作成した認証情報オブジェクトを入力する必要があります。

```
sam pipeline init
```

ステップ 5: パイプライン設定を Git リポジトリにコミットする

このステップは、CI/CD システムがパイプライン設定を認識しており、変更のコミット時に実行されることを確実にするために必要です。

詳細はこちら

GitHub Actions を使用して CI/CD パイプラインを設定する実践的な例については、「包括的な AWS SAM ワークショップ」の「GitHub を使用した CI/CD」を参照してください。

AWS SAM でスターターパイプラインをカスタマイズする方法

CI/CD 管理者は、組織内のデベロッパーがパイプライン構成を作成するために使用できる、スターターパイプラインテンプレートおよび関連するガイド付きプロンプトをカスタマイズできます。

AWS SAM CLI は、スターターテンプレートの作成時に Cookiecutter テンプレートを使用します。クッキーカッターテンプレートの詳細については、Cookiecutter を参照してください。

また、`sam pipeline init` コマンドを使用してパイプライン設定を作成するときに、AWS SAM CLI がユーザに表示するプロンプトをカスタマイズすることもできます。ユーザプロンプトをカスタマイズするには、次の操作を行います。

1. **`questions.json`** ファイルの作成 – `questions.json` ファイルはプロジェクトリポジトリのルートにある必要があります。これは、`cookiecutter.json` ファイルと同じディレクトリです。`questions.json` ファイルのスキーマを表示するには、「questions.json.schema」を参照

してください。questions.json ファイルの例を表示するには、「questions.json」を参照してください。

2. 質問キーをクッキーカッターの名前でマッピングする – questions.json ファイル内の各オブジェクトには cookieCutter テンプレートの名前と一致するキーが必要です。このキーマッチングは、AWS SAM CLI が、ユーザプロンプトのレスポンスを Cookie Cutter テンプレートにマッピングする方法です。このキーマッチングの例については、このトピックの後半の ファイルの例 セクション。
3. **metadata.json** ファイルの作成 – パイプラインが metadata.json ファイルで持つステージ数を宣言します。ステージ数は、sam pipeline init コマンドに、インフラストラクチャリソースを作成するステージ数に関する情報を求めるステージ数、または --bootstrap オプションの場合はステージ数を指示します。metadata.json 2 つのステージを持つパイプラインを宣言する ファイルの例については、metadata.json を参照してください。

プロジェクトの例

以下に、Cookiecutter テンプレート、questions.json ファイル、および metadata.json ファイルをそれぞれ含むプロジェクトの例を示します。

- Jenkins の例: 2 段階の Jenkins パイプラインテンプレート
- CodePipeline の例: 2 段階の CodePipeline パイプラインテンプレート

ファイルの例

次の一連のファイルは、questions.json ファイルの質問が Cookiecutter テンプレートファイルのエントリにどのように関連付けられているかを示しています。これらの例はファイルスニペットであり、完全なファイルではないことに注意してください。完全なファイルの例については、このトピックの前半の プロジェクトの例 セクションを参照してください。

例 **questions.json**:

```
{
  "questions": [{
    "key": "intro",
    "question": "\nThis template configures a pipeline that deploys a serverless
 application to a testing and a production stage.\n",
    "kind": "info"
  }, {
    "key": "pipeline_user_jenkins_credential_id",
```

```
    "question": "What is the Jenkins credential ID (via Jenkins plugin \"aws-
credentials\") for pipeline user access key?",
    "isRequired": true
  }, {
    "key": "sam_template",
    "question": "What is the template file path?",
    "default": "template.yaml"
  }, {
    ...
```

例 **cookiecutter.json**:

```
{
  "outputDir": "aws-sam-pipeline",
  "pipeline_user_jenkins_credential_id": "",
  "sam_template": "",
    ...
```

例 **Jenkinsfile**:

```
pipeline {
  agent any
  environment {
    PIPELINE_USER_CREDENTIAL_ID =
 '{{cookiecutter.pipeline_user_jenkins_credential_id}}'
    SAM_TEMPLATE = '{{cookiecutter.sam_template}}'
    ...
```

AWS SAM アプリケーションのデプロイを自動化する

AWS SAM の場合、AWS SAM アプリケーションのデプロイを自動化する方法は、使用している CI/CD システムによって異なります。このため、このセクションでは、AWS SAM のビルドコンテナイメージでサーバーレスアプリケーションの構築を自動化する、さまざまな CI/CD システムの設定例を示しています。これらのコンテナイメージにより、AWS SAM CLI を使用したサーバーレスアプリケーションを簡単に構築およびパッケージ化できます。

既存の CI/CD パイプラインが AWS SAM を使用してサーバーレスアプリケーションをデプロイするための手順は、使用している CI/CD システムによって若干異なります。

以下のトピックでは、AWS SAM ビルドコンテナイメージ内でサーバーレスアプリケーションを構築するように CI/CD システムを設定する例を説明します。

トピック

- AWS CodePipeline を使用した AWS SAM によるデプロイ
- Bitbucket Pipelines を使用した AWS SAM によるデプロイ
- AWS SAM でのデプロイのための Jenkins の使用
- AWS SAM でのデプロイのための GitLab CI/CD の使用
- GitHub アクションを使用して AWS SAM でデプロイする

AWS CodePipeline を使用した AWS SAM によるデプロイ

AWS SAM アプリケーションの構築とデプロイを自動化するように AWS CodePipeline パイプラインを設定するには、AWS CloudFormation テンプレートと `buildspec.yml` ファイルに以下を実行する行が含まれている必要があります。

1. 使用可能なイメージから必要なランタイムがあるビルドコンテナイメージを参照する。以下の例は、`public.ecr.aws/sam/build-nodejs20.x` ビルドコンテナイメージを使用します。
2. 必要な AWS SAM コマンドラインインターフェイス (CLI) コマンドを実行するようにパイプラインステージを設定する。以下の例は、sam build と sam deploy (必要なオプション付き) の 2 つの AWS SAM CLI コマンドを実行します。

この例では、`runtime: nodejs20.x` の AWS SAM テンプレートファイルですべての関数とレイヤーが宣言されていることを前提としています。

AWS CloudFormation テンプレートスニペット:

```
  CodeBuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: public.ecr.aws/sam/build-nodejs20.x
        Type: LINUX_CONTAINER
      ...
```

`buildspec.yml` スニペット:

```
version: 0.2
phases:
```

```
  build:
    commands:
      - sam build
      - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

異なるランタイムに利用できる Amazon Elastic Container Registry (Amazon ECR) ビルドコンテナイメージのリストについては、「のイメージリポジトリ AWS SAM」を参照してください。

Bitbucket Pipelines を使用した AWS SAM によるデプロイ

AWS SAM アプリケーションの構築とデプロイを自動化するように Bitbucket Pipeline を設定するには、`bitbucket-pipelines.yml` に以下を実行する行が含まれている必要があります。

1. 使用可能なイメージから必要なランタイムがあるビルドコンテナイメージを参照する。以下の例は、`public.ecr.aws/sam/build-nodejs20.x` ビルドコンテナイメージを使用します。
2. 必要な AWS SAM コマンドラインインターフェイス (CLI) コマンドを実行するようにパイプラインステージを設定する。以下の例は、sam build と sam deploy (必要なオプション付き) の 2 つの AWS SAM CLI コマンドを実行します。

この例では、`runtime: nodejs20.x` の AWS SAM テンプレートファイルですべての関数とレイヤーが宣言されていることを前提としています。

```
image: public.ecr.aws/sam/build-nodejs20.x

pipelines:
  branches:
    main: # branch name
      - step:
          name: Build and Package
          script:
            - sam build
            - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

異なるランタイムに利用できる Amazon Elastic Container Registry (Amazon ECR) ビルドコンテナイメージのリストについては、「のイメージリポジトリ AWS SAM」を参照してください。

AWS SAM でのデプロイのための Jenkins の使用

AWS SAM アプリケーションの構築とデプロイを自動化するように Jenkins パイプラインを設定するには、`Jenkinsfile` に以下を実行する行が含まれている必要があります。

1. 使用可能なイメージから必要なランタイムがあるビルドコンテナイメージを参照する。以下の例は、public.ecr.aws/sam/build-nodejs20.x ビルドコンテナイメージを使用します。
2. 必要な AWS SAM コマンドラインインターフェイス (CLI) コマンドを実行するようにパイプラインステージを設定する。以下の例は、sam build と sam deploy (必要なオプション付き) の 2 つの AWS SAM CLI コマンドを実行します。

この例では、runtime: nodejs20.x の AWS SAM テンプレートファイルですべての関数とレイヤーが宣言されていることを前提としています。

```
pipeline {
    agent { docker { image 'public.ecr.aws/sam/build-nodejs20.x' } }
    stages {
        stage('build') {
            steps {
                sh 'sam build'
                sh 'sam deploy --no-confirm-changeset --no-fail-on-empty-changeset'
            }
        }
    }
}
```

異なるランタイムに利用できる Amazon Elastic Container Registry (Amazon ECR) ビルドコンテナイメージのリストについては、「のイメージリポジトリ AWS SAM」を参照してください。

AWS SAM でのデプロイのための GitLab CI/CD の使用

AWS SAM アプリケーションの構築とデプロイを自動化するように GitLab パイプラインを設定するには、gitlab-ci.yml に以下を実行する行が含まれている必要があります。

1. 使用可能なイメージから必要なランタイムがあるビルドコンテナイメージを参照する。以下の例は、public.ecr.aws/sam/build-nodejs20.x ビルドコンテナイメージを使用します。
2. 必要な AWS SAM コマンドラインインターフェイス (CLI) コマンドを実行するようにパイプラインステージを設定する。以下の例は、sam build と sam deploy (必要なオプション付き) の 2 つの AWS SAM CLI コマンドを実行します。

この例では、runtime: nodejs20.x の AWS SAM テンプレートファイルですべての関数とレイヤーが宣言されていることを前提としています。

```
image: public.ecr.aws/sam/build-nodejs20.x
```

```
deploy:
  script:
    - sam build
    - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

異なるランタイムに利用できる Amazon Elastic Container Registry (Amazon ECR) ビルドコンテナイメージのリストについては、「のイメージリポジトリ AWS SAM」を参照してください。

GitHub アクションを使用して AWS SAM でデプロイする

AWS SAM アプリケーションの構築とデプロイを自動化するように GitHub パイプラインを設定するには、まず AWS SAM コマンドラインインターフェイス (CLI) をホストにインストールする必要があります。このセットアップには、GitHub ワークフローの GitHub Actions を使用できます。

以下の GitHub ワークフロー例は、一連の GitHub アクションを使用して Ubuntu ホストをセットアップしてから、AWS SAM CLI コマンドを実行して AWS SAM アプリケーションの構築とデプロイを行います。

```
on:
  push:
    branches:
      - main
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v3
      - uses: aws-actions/setup-sam@v2
      - uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-2
      - run: sam build --use-container
      - run: sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

異なるランタイムに利用できる Amazon Elastic Container Registry (Amazon ECR) ビルドコンテナイメージのリストについては、「のイメージリポジトリ AWS SAM」を参照してください。

AWS SAM パイプラインを使用した OIDC 認証の使用方法

AWS Serverless Application Model (AWS SAM) は、Bitbucket、GitHub Actions、および GitLab の継続的インテグレーションおよび継続的デリバリー (CI/CD) のプラットフォームに対して OpenID Connect (OIDC) ユーザー認証をサポートしています。このサポートにより、これらのプラットフォームのいずれかから認証された CI/CD ユーザーアカウントを使用して、サーバーレスアプリケーションパイプラインを管理できます。このサポートがないと、複数のAWS Identity and Access Management (IAM) ユーザーを作成して管理し、AWS SAM パイプラインへのアクセスを制御する必要が生じます。

AWS SAM パイプラインでの OIDC の設定

`sam pipeline bootstrap` 設定プロセス中に、以下を実行して AWS SAM パイプラインで OIDC を設定します。

1. ID プロバイダーの選択を求められたら、OIDC を選択します。
2. 次に、サポートされている OIDC プロバイダーを選択します。
3. OIDC プロバイダーの URL を入力します (**`https://`** から入力します)。

> **Note**
>
> AWS SAM は、`AWS::IAM::OIDCProvider` リソースタイプを生成するときにこの URL を参照します。

4. 次に、プロンプトに従い、選択したプラットフォームへのアクセスに必要な CI/CD プラットフォーム情報を入力します。これらの詳細はプラットフォームによって異なり、次のものが含まれる場合があります。
   - OIDC クライアント ID。
   - コードリポジトリ名または一意の識別子 (UUID)。
   - リポジトリに関連付けられたグループまたは組織名。
   - コードリポジトリが属する GitHub 組織。
   - GitHub リポジトリ名。
   - デプロイ元のブランチ。
5. AWS SAM は、入力した OIDC 設定の概要を表示します。設定の番号を入力して編集するか、Enter を押して続行します。

6. 入力した OIDC 接続をサポートするために必要なリソースの作成を確認するメッセージが表示されたら、Y を押して続行します。

AWS SAM は、パイプライン実行ロールを引き受ける指定された設定で `AWS::IAM::OIDCProvider` AWS CloudFormation リソースを生成します。この AWS CloudFormation リソースタイプの詳細については、AWS CloudFormation ユーザーガイドの「AWS::IAM::OIDCProvider」を参照してください。

> **Note**
>
> ID プロバイダー (IdP) リソースが AWS アカウント に既に存在する場合は、AWS SAM は、新しいリソースを作成する代わりにそのリソースを参照します。

例

以下は、AWS SAM パイプラインで OIDC を設定する例です。

```
Select a permissions provider:
    1 - IAM (default)
    2 - OpenID Connect (OIDC)
Choice (1, 2): 2
Select an OIDC provider:
    1 - GitHub Actions
    2 - GitLab
    3 - Bitbucket
Choice (1, 2, 3): 1
Enter the URL of the OIDC provider [https://token.actions.githubusercontent.com]:
Enter the OIDC client ID (sometimes called audience) [sts.amazonaws.com]:
Enter the GitHub organization that the code repository belongs to. If there is no
 organization enter your username instead: my-org
Enter GitHub repository name: testing
Enter the name of the branch that deployments will occur from [main]:

[3] Reference application build resources
Enter the pipeline execution role ARN if you have previously created one, or we will
 create one for you []:
Enter the CloudFormation execution role ARN if you have previously created one, or we
 will create one for you []:
```

```
Please enter the artifact bucket ARN for your Lambda function. If you do not have a
 bucket, we will create one for you []:
Does your application contain any IMAGE type Lambda functions? [y/N]:

[4] Summary
Below is the summary of the answers:
    1 - Account: 123456
    2 - Stage configuration name: dev
    3 - Region: us-east-1
    4 - OIDC identity provider URL: https://token.actions.githubusercontent.com
    5 - OIDC client ID: sts.amazonaws.com
    6 - GitHub organization: my-org
    7 - GitHub repository: testing
    8 - Deployment branch: main
    9 - Pipeline execution role: [to be created]
    10 - CloudFormation execution role: [to be created]
    11 - Artifacts bucket: [to be created]
    12 - ECR image repository: [skipped]
Press enter to confirm the values above, or select an item to edit the value:

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

詳細はこちら

AWS SAM パイプラインで OIDC を使用する方法の詳細については、「sam pipeline bootstrap」を参照してください。

の使用の概要 sam sync 同期する AWS クラウド

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam sync` コマンドには、ローカルアプリケーションの変更を にすばやく同期するためのオプションが用意されています AWS クラウド。アプリケーションを開発する際に次の目的で `sam sync` を使用します。

1. ローカルの変更を自動的に検出して に同期します AWS クラウド。
2. AWS クラウドに同期するローカル変更をカスタマイズする。
3. テストと検証のためにクラウドでアプリケーションを準備する。

sam sync を使用すると、テストや検証のためにローカルの変更をクラウドに同期するのにかかる時間を短縮する、迅速な開発ワークフローを作成できます。

> **Note**
>
> sam sync コマンドは開発環境に推奨されます。本番環境の場合は、sam deploy を使用するか、または継続的インテグレーションおよびデリバリー (CI/CD) パイプラインを設定することをお勧めします。詳細については、「AWS SAM を使用してアプリケーションとリソースをデプロイする」を参照してください。

sam sync コマンドは の一部です。 AWS SAM Accelerate. AWS SAM Accelerate には、 でサーバーレスアプリケーションを開発およびテストするエクスペリエンスを高速化するために使用できるツールが用意されています AWS クラウド。

トピック

ローカルの変更を自動的に検出して に同期する AWS クラウド

アプリケーションと AWS クラウドの同期を開始する --watch オプションを指定して sam sync を実行します。これは次を実行します。

1. アプリケーションを構築する – このプロセスは、sam build コマンドの使用と似ています。
2. アプリケーションをデプロイする – AWS SAM CLI は、デフォルト設定 AWS CloudFormation を使用してアプリケーションを にデプロイします。次のデフォルト値が使用されます。
   a. AWS .aws ユーザーフォルダにある認証情報と一般的な設定。
   b. アプリケーションの samconfig.toml ファイルにあるアプリケーションデプロイ設定。

   デフォルト値が見つからない場合は、 AWS SAM CLI が通知し、同期プロセスを終了します。

3. ローカルの変更を監視する – AWS SAM CLI は実行中のままで、アプリケーションに対するローカルの変更を監視します。これが `--watch` オプションによって提供されるものです。

   このオプションは、デフォルトでオンになっている場合があります。デフォルト値については、アプリケーションの `samconfig.toml` ファイルを参照してください。ファイルの例を次に示します。

   ```
   ...
   [default.sync]
   [default.sync.parameters]
   watch = true
   ...
   ```

4. へのローカル変更の同期 AWS クラウド — ローカル変更を行うと、 AWS SAM CLI は、これらの変更を検出し AWS クラウド 、利用可能な最も迅速な方法で に同期します。変更の種類に応じて、次のような状況が発生する可能性があります。

   a. 更新されたリソースが AWS サービス をサポートしている場合APIs、 AWS SAM CLI は、変更をデプロイするために使用します。これにより、 AWS クラウド内のリソースを更新するための迅速な同期が行われます。

   b. 更新されたリソースが AWS サービス をサポートしていない場合はAPIs、 AWS SAM CLI は AWS CloudFormation デプロイを実行します。これにより、 AWS クラウド内のアプリケーション全体が更新されます。迅速さは劣りますが、デプロイを手動で開始する必要がなくなります。

`sam sync` コマンドは でアプリケーションを自動的に更新するため AWS クラウド、開発環境にのみ推奨されます。`sam sync` を実行すると、次を確認するよう求められます。

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

Confirm that you are synchronizing a development stack.

Enter Y to proceed with the command, or enter N to cancel:
 [Y/n]: ENTER
```

に同期されるローカル変更をカスタマイズする AWS クラウド

AWS クラウドに同期するローカル変更をカスタマイズするオプションを指定します。これにより、テストと検証のためにクラウドでローカルの変更が表示されるまでにかかる時間を短縮できます。

例えば、 AWS Lambda 関数コードなどのコード変更のみを同期する--codeオプションを指定します。開発中、特に Lambda コードに注力している場合、これを使用することで、テストと検証のために変更をすぐにクラウドに取り込むことができます。以下に例を示します。

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

特定の Lambda 関数またはレイヤーのコード変更のみを同期するには、--resource-id オプションを使用します。以下に例を示します。

```
$ sam sync --code --resource-id HelloWorldFunction --resource-id HelloWorldLayer
```

テストと検証のためにクラウドでアプリケーションを準備する

sam sync コマンドは、 AWS クラウドでアプリケーションを更新するために利用できる最も迅速な方法を自動的に見つけます。これにより、開発とクラウドテストのワークフローが高速化されます。AWS サービスを使用することでAPIs、サポートされているリソースを迅速に開発、同期、テストできます。実践的な例については、「The Complete AWS SAM Workshop」の「モジュール 6 - AWS SAM Accelerate」を参照してください。

sam sync コマンドのオプション

sam sync コマンドの変更に使用できる主なオプションの一部を次に示します。すべてのオプションのリストについては、「sam sync」を参照してください。

1 回限りの AWS CloudFormation デプロイを実行する

自動同期をオフにする --no-watch オプションを使用します。以下に例を示します。

```
$ sam sync --no-watch
```

の AWS SAM CLI は 1 回限りの AWS CloudFormation デプロイを実行します。このコマンドは、sam build および sam deploy コマンドによって実行されるアクションをグループ化します。

初期 AWS CloudFormation デプロイをスキップする

sam sync 実行のたびに AWS CloudFormation デプロイが必要かどうかをカスタマイズできます。

- `sam sync` を実行するたびに AWS CloudFormation デプロイを要求する `--no-skip-deploy-sync` を指定します。これにより、ローカルインフラストラクチャが に同期され AWS CloudFormation、ドリフトを防ぐことができます。このオプションを使用すると、開発およびテストのワークフローにさらに時間がかかります。
- を指定`--skip-deploy-sync`して AWS CloudFormation デプロイをオプションにします。の AWS SAM CLI は、ローカル AWS SAM テンプレートをデプロイされた AWS CloudFormation テンプレートと比較し、変更が検出されない場合、最初の AWS CloudFormation デプロイをスキップします。 AWS CloudFormation デプロイをスキップすると、ローカルの変更を に同期する際の時間を短縮できます AWS クラウド。

  変更が検出されない場合、 AWS SAM CLI は引き続き、以下のシナリオで AWS CloudFormation デプロイを実行します。

  - 前回の AWS CloudFormation デプロイから 7 日以上経過している場合。
  - 多数の Lambda 関数コードの変更が検出された場合、 AWS CloudFormation デプロイをアプリケーションを更新する最も迅速な方法にします。

以下に例を示します。

```
$ sam sync --skip-deploy-sync
```

ネストされたスタックからリソースを同期する

ネストされたスタックからリソースを同期するには

1. `--stack-name` を使用してルートスタックを指定します。
2. 次の形式を使用して、ネストされたスタック内のリソースを識別します: *nestedStackId/resourceId*。
3. `--resource-id` を使用して、ネストされたスタックにリソースを提供します。

   以下に例を示します。

   ```
   $ sam sync --code --stack-name sam-app --resource-id myNestedStack/
   HelloWorldFunction
   ```

ネストしたアプリケーションの作成方法については、「AWS SAM 内で、ネストされたアプリケーションを使用して、コードとリソースを再使用する」を参照してください。

更新する特定の AWS CloudFormation スタックを指定する

更新する特定の AWS CloudFormation スタックを指定するには、 `--stack-name`オプションを指定します。以下に例を示します。

```
$ sam sync --stack-name dev-sam-app
```

ソースフォルダにプロジェクトを構築することでビルド時間を短縮する

サポートされているランタイムとビルドメソッドについては、`--build-in-source` オプションを使用してプロジェクトをソースフォルダに直接構築できます。デフォルトでは、 AWS SAM CLI は、ソースコードとプロジェクトファイルのコピーを含む一時ディレクトリに構築されます。では`--build-in-source`、 AWS SAM CLI はソースフォルダに直接ビルドします。これにより、ファイルを一時ディレクトリにコピーする必要がなくなるため、ビルドプロセスが高速化されます。

サポートされているランタイムとビルドメソッドのリストについては、「 `--build-in-source`」を参照してください。

同期を開始しないファイルとフォルダを指定する

更新時に同期を開始しないファイルまたはフォルダを指定するには `--watch-exclude` オプションを使用します。このオプションの詳細については、「`--watch-exclude`」を参照してください。

`HelloWorldFunction` 関数に関連付けられた `package-lock.json` ファイルを除外する例を次に示します。

```
$ sam sync --watch --watch-exclude HelloWorldFunction=package-lock.json
```

このコマンドを実行すると、 AWS SAM CLI は同期プロセスを開始します。これには以下が含まれます。

- `sam build` を実行して関数を構築し、デプロイに向けてアプリケーションを準備します。
- アプリケーションをデプロイするには `sam deploy` を実行します。
- アプリケーションに対する変更を監視します。

`package-lock.json` ファイルを変更すると、 AWS SAM CLI は同期を開始しません。別のファイルが更新されると、 AWS SAM CLI は、 `package-lock.json` ファイルを含む同期を開始します。

子スタックの Lambda 関数を指定する例を次に示します。

```
$ sam sync --watch --watch-exclude ChildStackA/MyFunction=database.sqlite3
```

トラブルシューティング

をトラブルシューティングするには AWS SAM CLI「AWS SAMCLI トラブルシューティング」を参照してください。

例

sam sync を使用して Hello World アプリケーションを更新する

この例では、サンプルの Hello World アプリケーションを初期化することから始めます。このアプリケーションの詳細については、「チュートリアル: を使用して Hello World アプリケーションをデプロイする AWS SAM」を参照してください。

sam sync を実行すると、構築とデプロイのプロセスが開始されます。

```
$ sam sync

The SAM CLI will use the AWS Lambda, Amazon API Gateway, and AWS StepFunctions APIs to
 upload your code without
performing a CloudFormation deployment. This will cause drift in your CloudFormation
 stack.
**The sync command should only be used against a development stack**.

Confirm that you are synchronizing a development stack.

Enter Y to proceed with the command, or enter N to cancel:
 [Y/n]:
Queued infra sync. Waiting for in progress code syncs to complete...
Starting infra sync.
Manifest file is changed (new hash: 3298f13049d19cffaa37ca931dd4d421) or dependency
 folder (.aws-sam/deps/0663e6fe-a888-4efb-b908-e2344261e9c7) is missing for
 (HelloWorldFunction), downloading dependencies and copying/building source
Building codeuri: /Users/.../Demo/sync/sam-app/hello_world runtime: python3.9 metadata:
 {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CleanUp
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource

Build Succeeded
```

```
Successfully packaged artifacts and wrote output template to file /var/
folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpx_5t4u3f.
Execute the following command to deploy the packaged template
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpx_5t4u3f
 --stack-name <YOUR STACK NAME>


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

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


2023-03-17 11:17:19 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 0.5 seconds)
---------------------------------------------------------------------------------------
ResourceStatus                       ResourceType
 LogicalResourceId                    ResourceStatusReason
---------------------------------------------------------------------------------------
CREATE_IN_PROGRESS                   AWS::CloudFormation::Stack          sam-app
                      Transformation succeeded
CREATE_IN_PROGRESS                   AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt    -
                                                                          ack
CREATE_IN_PROGRESS                   AWS::IAM::Role
 HelloWorldFunctionRole               -
CREATE_IN_PROGRESS                   AWS::IAM::Role
 HelloWorldFunctionRole               Resource creation Initiated
CREATE_IN_PROGRESS                   AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt    Resource creation Initiated
                                                                          ack
CREATE_COMPLETE                      AWS::IAM::Role
 HelloWorldFunctionRole               -
```

```
CREATE_COMPLETE                      AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                  ack
CREATE_IN_PROGRESS                   AWS::Lambda::Function
 HelloWorldFunction                   -
CREATE_IN_PROGRESS                   AWS::Lambda::Function
 HelloWorldFunction                   Resource creation Initiated
CREATE_COMPLETE                      AWS::Lambda::Function
 HelloWorldFunction                   -
CREATE_IN_PROGRESS                   AWS::ApiGateway::RestApi
 ServerlessRestApi                    -
CREATE_IN_PROGRESS                   AWS::ApiGateway::RestApi
 ServerlessRestApi                    Resource creation Initiated
CREATE_COMPLETE                      AWS::ApiGateway::RestApi
 ServerlessRestApi                    -
CREATE_IN_PROGRESS                   AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                  5f9d
CREATE_IN_PROGRESS                   AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                  ssionProd
CREATE_IN_PROGRESS                   AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   Resource creation Initiated
                                                                  ssionProd
CREATE_IN_PROGRESS                   AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   Resource creation Initiated
                                                                  5f9d
CREATE_COMPLETE                      AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                  5f9d
CREATE_IN_PROGRESS                   AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
CREATE_IN_PROGRESS                   AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          Resource creation Initiated
CREATE_COMPLETE                      AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
CREATE_COMPLETE                      AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                  ssionProd
CREATE_COMPLETE                      AWS::CloudFormation::Stack          sam-app
                     -
-------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
```

```
---------------------------------------------------------------------------------------------------
Outputs
---------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole
Description         Implicit IAM Role created for Hello World function
Value               arn:aws:iam::012345678910:role/sam-app-HelloWorldFunctionRole-
BUFVMO2PJIYF

Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World function
Value               https://pcrx5gdaof.execute-api.us-west-2.amazonaws.com/Prod/hello/

Key                 HelloWorldFunction
Description         Hello World Lambda Function ARN
Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-2PlN6TPTQoco
---------------------------------------------------------------------------------------------------
Stack creation succeeded. Sync infra completed.

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

デプロイが完了したら、HelloWorldFunction コードを変更します。の AWS SAM CLI はこの変更を検出し、アプリケーションを に同期します AWS クラウド。は AWS サービス AWS Lambda をサポートしているためAPIs、クイック同期が実行されます。

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

次に、アプリケーションの AWS SAM テンプレートでAPIエンドポイントを変更します。/hello を /helloworld に変更します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    ...
```

```
    Properties:
      ...
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /helloworld
            Method: get
```

Amazon API Gateway リソースは AWS サービス をサポートしていないためAPI、 AWS SAM CLI は AWS CloudFormation デプロイを自動的に実行します。以下は、その出力例です。

```
Queued infra sync. Waiting for in progress code syncs to complete...
Starting infra sync.
Manifest is not changed for (HelloWorldFunction), running incremental build
Building codeuri: /Users/.../Demo/sync/sam-app/hello_world runtime: python3.9 metadata:
 {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CopySource

Build Succeeded

Successfully packaged artifacts and wrote output template to file /var/
folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpuabo0jb9.
Execute the following command to deploy the packaged template
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpuabo0jb9
 --stack-name <YOUR STACK NAME>


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

Initiating deployment
=====================
```

```
2023-03-17 14:41:18 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 0.5 seconds)
---------------------------------------------------------------------------------------------------
ResourceStatus                      ResourceType
 LogicalResourceId                   ResourceStatusReason
---------------------------------------------------------------------------------------------------
UPDATE_IN_PROGRESS                  AWS::CloudFormation::Stack          sam-app
                     Transformation succeeded
UPDATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                         ack
UPDATE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                         ack
UPDATE_IN_PROGRESS                  AWS::ApiGateway::RestApi
 ServerlessRestApi                   -
UPDATE_COMPLETE                     AWS::ApiGateway::RestApi
 ServerlessRestApi                   -
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment8cf30e   -
                                                                         d3cd
UPDATE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   Requested update requires the
                                                                         ssionProd
                     creation of a new physical

                     resource; hence creating one.
UPDATE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   Resource creation Initiated
                                                                         ssionProd
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment8cf30e   Resource creation Initiated
                                                                         d3cd
CREATE_COMPLETE                     AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment8cf30e   -
                                                                         d3cd
UPDATE_IN_PROGRESS                  AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
UPDATE_COMPLETE                     AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
UPDATE_COMPLETE                     AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                         ssionProd
```

```
UPDATE_COMPLETE_CLEANUP_IN_PROGRE   AWS::CloudFormation::Stack          sam-app
                      -
SS
DELETE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                       ssionProd
DELETE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                       5f9d
DELETE_COMPLETE                     AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                       5f9d
UPDATE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                       ack
DELETE_COMPLETE                     AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                       ssionProd
UPDATE_COMPLETE                     AWS::CloudFormation::Stack          sam-app
                      -
-------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-------------------------------------------------------------------------------------------------
Outputs
-------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole
Description         Implicit IAM Role created for Hello World function
Value               arn:aws:iam::012345678910:role/sam-app-HelloWorldFunctionRole-
BUFVMO2PJIYF

Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World function
Value               https://pcrx5gdaof.execute-api.us-west-2.amazonaws.com/Prod/hello/

Key                 HelloWorldFunction
Description         Hello World Lambda Function ARN
Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-2PlN6TPTQoco
-------------------------------------------------------------------------------------------------


Stack update succeeded. Sync infra completed.
```

```
Infra sync completed.
```

詳細

すべての sam sync オプションの説明については、「sam sync」を参照してください。

AWS SAM を使用してサーバーレスアプリケーションをモニタリングする

サーバーレスアプリケーションをデプロイしたら、それをモニタリングしてオペレーションに関するインサイトを提供し、異常を検出できます。これはトラブルシューティングに役立ちます。このセクションでは、サーバーレスアプリケーションをモニタリングする方法について説明します。これには、異常を検出した際に通知するように Amazon CloudWatch を設定する方法に関する情報が含まれます。また、エラーの強調表示やログの表示、フィルタリング、フェッチ、テーリングログなど、ログの操作に関する情報も提供されます。

トピック

AWS SAM サーバーレスアプリケーションをモニタリングするための CloudWatch Application Insights の使用

Amazon CloudWatch Application Insights は、アプリケーション内の AWS リソースを監視して潜在的な問題を特定するのに役立ちます。AWS リソースデータを分析して問題の兆候を探し、それらを視覚化する自動ダッシュボードを構築できます。CloudWatch Application Insights を AWS Serverless Application Model (AWS SAM) アプリケーションで使用するように設定できます。CloudWatch Application Insights の詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Application Insights」を参照してください。

トピック

Configuring CloudWatch Application Insights と AWS SAM

AWS SAM コマンドラインインターフェイス (AWS SAM CLI) または AWS SAM テンプレートを使用して、AWS SAM アプリケーションの CloudWatch Application Insights を設定します。

AWS SAM CLI による設定

sam init でアプリケーションを初期化するときは、インタラクティブフローまたは --application-insights オプションを使用して CloudWatch Application Insights を有効にします。

AWS SAM CLI のインタラクティブフローで CloudWatch Application Insights を有効にするには、プロンプトが表示されたら **y** と入力します。

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

--application-insights オプションで CloudWatch Application Insights を有効にするには、次の操作を行います。

```
sam init --application-insights
```

sam init コマンドを使用する場合の詳細については、「sam init」を参照してください。

AWS SAM テンプレートによる設定

AWS SAM テンプレートで `AWS::ResourceGroups::Group` と `AWS::ApplicationInsights::Application` リソースを定義して、CloudWatch Application Insights を有効にします。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  ApplicationResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
        - Ref: AWS::StackName
      ResourceQuery:
        Type: CLOUDFORMATION_STACK_1_0
  ApplicationInsightsMonitoring:
    Type: AWS::ApplicationInsights::Application
    Properties:
```

```
      ResourceGroupName:
        Fn::Join:
          - ''
          - - ApplicationInsights-SAM-
          - Ref: AWS::StackName
        AutoConfigurationEnabled: 'true'
    DependsOn: ApplicationResourceGroup
```

- `AWS::ResourceGroups::Group` — 大量のリソースのタスクを一度に管理および自動化するために、AWS リソースを整理するグループを作成します。ここでは、CloudWatch Application Insights で使用するリソースグループを作成します。このリソースタイプの詳細については、「AWS CloudFormation ユーザーガイド」の「AWS::ResourceGroups::Group」を参照してください。
- `AWS::ApplicationInsights::Application` — リソースグループの CloudWatch Application Insights を設定します。このリソースタイプの詳細については、「AWS CloudFormation ユーザーガイド」の「AWS::ApplicationInsights::Application」を参照してください。

どちらのリソースも、アプリケーションのデプロイ時に自動的に AWS CloudFormation に渡されます。AWS SAM テンプレートの AWS CloudFormation 構文を使用して、CloudWatch Application Insights をさらに設定できます。詳細については、「Amazon CloudWatch ユーザーガイド」の「AWS CloudFormation テンプレートの使用」を参照してください。

sam init --application-insights コマンドを使用すると、これらのリソースは両方とも AWS SAM テンプレートに自動的に生成されます。生成されたテンプレートの例を示します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  sam-app-test

  Sample SAM Template for sam-app-test

# More info about Globals: https://github.com/awslabs/serverless-application-model/
blob/master/docs/globals.rst
Globals:
  Function:
    Timeout: 3
    MemorySize: 128

Resources:
```

```
  HelloWorldFunction:
    Type: AWS::Serverless::Function # More info about Function Resource:
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.9
      Architectures:
      - x86_64
      Events:
        HelloWorld:
          Type: Api # More info about API Event Source: https://github.com/awslabs/
serverless-application-model/blob/master/versions/2016-10-31.md#api
          Properties:
            Path: /hello
            Method: get

  ApplicationResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
        - Ref: AWS::StackName
      ResourceQuery:
       Type: CLOUDFORMATION_STACK_1_0
  ApplicationInsightsMonitoring:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
        - Ref: AWS::StackName
      AutoConfigurationEnabled: 'true'
    DependsOn: ApplicationResourceGroup

Outputs:
  # ServerlessRestApi is an implicit API created out of Events key under
 Serverless::Function
  # Find out more about other implicit resources you can reference within SAM
```

```
  # https://github.com/awslabs/serverless-application-model/blob/master/docs/internals/
generated_resources.rst#api
  HelloWorldApi:
    Description: API Gateway endpoint URL for Prod stage for Hello World function
    Value: !Sub "https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaws.com/
Prod/hello/"
  HelloWorldFunction:
    Description: Hello World Lambda Function ARN
    Value: !GetAtt HelloWorldFunction.Arn
  HelloWorldFunctionIamRole:
    Description: Implicit IAM Role created for Hello World function
    Value: !GetAtt HelloWorldFunctionRole.Arn
```

次のステップ

CloudWatch Application Insights を設定したら、sam build を使用してアプリケーションを構築し、sam deploy を使用してデプロイします。CloudWatch Application Insights がサポートするリソースはすべてモニタリング対象に設定されます。

- サポートされているリソースのリストについては、「Amazon CloudWatch ユーザーガイド」の「サポートされているログとメトリクス」を参照してください。
- CloudWatch Application Insights へのアクセス方法の詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch Application Insights へのアクセス」を参照してください。

AWS SAM でのログの使用

トラブルシューティングをシンプル化するため、AWS SAM CLI には `sam logs` というコマンドがあります。このコマンドを使用すると、コマンドラインから Lambda 関数によって生成されたログを取得できます。

> **Note**
>
> `sam logs` コマンドは、AWS Lambda を使用してデプロイするものだけでなく、すべての AWS SAM 関数で機能します。

AWS CloudFormation スタックによるログの取得

関数が AWS CloudFormation スタックの一部である場合は、関数の論理 ID を使用してログを取得できます。

```
sam logs -n HelloWorldFunction --stack-name mystack
```

Lambda 関数名によるログの取得

または、関数の名前を使用してログを取得することもできます。

```
sam logs -n mystack-HelloWorldFunction-1FJ8PD
```

ログ終端の表示

`--tail` オプションを追加して、新しいログを待機し、到着するたびにそれらを表示します。これは、デプロイ中、または実稼働問題のトラブルシューティング時に役立ちます。

```
sam logs -n HelloWorldFunction --stack-name mystack --tail
```

特定時間範囲のログの表示

`-s` および `-e` オプションを使用して、特定の時間範囲のログを表示できます。

```
sam logs -n HelloWorldFunction --stack-name mystack -s '10min ago' -e '2min ago'
```

ログのフィルタリング

`--filter` オプションを使用して、ログイベントの語句、フレーズ、または値が一致するログをすばやく検索します。

```
sam logs -n HelloWorldFunction --stack-name mystack --filter "error"
```

出力では、AWS SAM CLI が「error」という単語のすべての出現箇所に下線を追加するので、ログ出力内のフィルターキーワードを簡単に見つけることができます。

エラーの強調表示

Lambda 関数がクラッシュまたはタイムアウトすると、AWS SAM CLI がタイムアウトメッセージを赤色でハイライトします。これは、ログ出力の巨大なストリーム内で、タイムアウトしている特定の実行を簡単に見つけるために役立ちます。

JSON の整形出力

ログメッセージが JSON 文字列で出力表示される場合、AWS SAM CLI は、JSON を視覚的に解析して理解できるように、JSON を自動的に整形出力します。

AWS SAM リファレンス

このセクションには、AWS SAM リファレンスマテリアルが含まれています。これには、AWS SAM CLI コマンドに関するリファレンス情報などの AWS SAM CLI リファレンスマテリアル、および設定、バージョン管理、トラブルシューティング情報などの AWS SAM CLI についての追加情報が含まれています。さらに、コネクタ、イメージリポジトリ、デプロイに関するリファレンス情報など、AWS SAM の仕様と AWS SAM テンプレートに関するリファレンス情報が含まれています。

AWS SAM の仕様と AWS SAM テンプレート

AWS SAM の仕様は、Apache 2.0 ライセンスに基づくオープンソース仕様です。AWS SAM の仕様の最新バージョンは、AWS SAM プロジェクトと AWS SAM テンプレート で入手できます。AWS SAM の仕様には、サーバーレスアプリケーションの関数、イベント、API、設定、およびアクセス許可の定義に使用する簡略化された省略構文があります。

sam init コマンドの実行時に作成されるフォルダとファイルである AWS SAM アプリケーションプロジェクトディレクトリを介して AWS SAM の仕様を操作します。このディレクトリには、AWS リソースを定義する重要なファイルである AWS SAM テンプレートが含まれています。AWS SAM テンプレートは AWS CloudFormation テンプレートの拡張です。AWS CloudFormation テンプレートの完全なリファレンスについては、「AWS CloudFormation ユーザーガイド」の「テンプレートリファレンス」を参照してください。

AWS SAM CLI コマンドリファレンス

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) は、AWS SAM テンプレートやサポートされているサードパーティの統合と併用することで、サーバーレスアプリケーションを構築し、実行できるコマンドラインツールです。

AWS SAM CLI コマンドは、サーバーレスアプリケーションの開発、テスト、および AWS クラウドへのデプロイに使用できます。以下は、AWS SAM CLI コマンドの例です。

- `sam init` - AWS SAM CLI を初めて使用するユーザーの場合は、パラメータを指定せずに `sam init` コマンドを実行して、Hello World アプリケーションを作成することができます。このコマンドは、選択した言語での事前設定済みの AWS SAM テンプレートとサンプルアプリケーションコードを生成します。
- `sam local invoke` および `sam local start-api` - AWS クラウド にアプリケーションコードをデプロイする前に、これらのコマンドを使用してコードをローカルでテストします。

- `sam logs` - このコマンドを使用して、Lambda 関数によって生成されたログを取得します。これは、アプリケーションを AWS クラウド にデプロイした後でアプリケーションをテストおよびデバッグするために役立ちます。
- `sam package` - このコマンドを使用して、アプリケーションコードと依存関係をデプロイパッケージにバンドルします。デプロイパッケージは、アプリケーションを AWS クラウド にアップロードするために必要です。
- `sam deploy` - このコマンドを使用して、サーバーレスアプリケーションを AWS クラウド にデプロイします。このコマンドは、AWS SAM テンプレートに定義されている AWS リソースを作成し、許可とその他の設定を行います。

AWS SAM CLI のインストール手順については、「AWS SAM CLI のインストール」を参照してください。

AWS SAM ポリシーテンプレート

AWS SAM では、ポリシーテンプレートのリストから選択して、AWS Lambda 関数のアクセス許可の範囲をアプリケーションが使用するリソースに絞り込むことができます。使用可能なポリシーテンプレートのリストについては、「ポリシーテンプレート表」を参照してください。ポリシーテンプレートと AWS SAM の一般的な情報については、「AWS SAMポリシーテンプレート」を参照してください。

トピック

- AWS SAM プロジェクトと AWS SAM テンプレート
- AWS SAM CLI コマンドリファレンス
- AWS SAM CLI　設定ファイル
- AWS SAM コネクタリファレンス
- AWS SAMポリシーテンプレート
- のイメージリポジトリ AWS SAM
- AWS SAM CLI でのテレメトリ
- AWS SAM テンプレート内でリソースアクセスを設定および管理する

AWS SAM CLI コマンドリファレンス

このセクションでは、AWS SAM CLI コマンドのリファレンス情報を示します。これには、使用についての詳細、各コマンドで使用できるさまざまなオプションの包括的なリスト、および追加的な情報が含まれています。対応する情報がある場合、追加の情報には引数、環境変数、イベントなどの詳細が含まれます。詳細については、各コマンドを参照してください。AWS SAM CLI のインストール手順については、「AWS SAM CLI のインストール」を参照してください。

トピック

sam build

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam build` コマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメント AWS SAM CLI `sam build` コマンドについては、「」を参照してくださいAWS SAM を使用した構築の概要。

`sam build` コマンドは、ローカルテストや AWS クラウドへのデプロイなど、開発者ワークフローの後続のステップに備えてアプリケーションを準備します。

使用方法

```
$ sam build <arguments> <options>
```

引数

リソース ID

オプション。AWS SAM テンプレートで宣言された 1 つのリソースを構築する AWS SAM ように に指示します。指定されたリソースのビルドアーティファクトは、ワークフローの後続コマンド (つまり、`sam package` と `sam deploy`) に使用できる唯一のアーティファクトになります。

オプション

`--base-dir, -s` *DIRECTORY*

関数またはレイヤーのソースコードへの相対パスを、このディレクトリを基準にして解決します。このオプションは、ソースコードフォルダへの相対パスの解決方法を変更したい場合に使用します。デフォルトで、相対パスは AWS SAM テンプレートの場所を基準にして解決されます。

このオプションは、構築しているルートアプリケーションまたはスタックのリソースに加えて、ネストされたアプリケーションまたはスタックにも適用されます。

このオプションは、以下のリソースタイプとプロパティに適用されます。

- リソースタイプ: `AWS::Serverless::Function` プロパティ: `CodeUri`
- リソースタイプ: `AWS::Serverless::Function` リソース属性: `Metadata` エントリ: `DockerContext`
- リソースタイプ: `AWS::Serverless::LayerVersion` プロパティ: `ContentUri`
- リソースタイプ: `AWS::Lambda::Function` プロパティ: `Code`

- リソースタイプ: AWS::Lambda::LayerVersion プロパティ: Content

`--beta-features | --no-beta-features`

ベータ機能を許可または拒否します。

`--build-dir, -b` *DIRECTORY*

ビルドアーティファクトが保存されているディレクトリへのパスです。このオプションを使用すると、このディレクトリとそのコンテンツのすべてが削除されます。

`--build-image` *TEXT*

ビルド用にプルするコンテナイメージURIの 。デフォルトでは、 は Amazon ECR Public からコンテナイメージを AWS SAM プルします。このオプションは、別の場所からイメージをプルするために使用します。

このオプションは複数回指定できます。このオプションの各インスタンスには、文字列またはキーバリューペアを使用できます。文字列を指定すると、アプリケーション内のすべてのリソースに使用するコンテナイメージURIの になります。例えば、sam build --use-container --build-image amazon/aws-sam-cli-build-image-python3.8 と指定します。キーと値のペアを指定する場合、キーはリソース名であり、値はそのリソースに使用するコンテナイメージURIの です。例えば、 ですsam build --use-container --build-image Function1=amazon/aws-sam-cli-build-image-python3.8。キーバリューペアを使用すると、異なるリソースに異なるコンテナイメージを指定できます。

このオプションは、--use-container オプションが指定されている場合のみに適用され、指定されていない場合はエラーが発生します。

`--build-in-source | --no-build-in-source`

ソースフォルダにプロジェクトを直接構築するには --build-in-source を指定します。

--build-in-source オプションは、次のランタイムとビルドメソッドをサポートします:

- ランタイム – すべて Node.js ランタイムは、 sam init --runtimeオプションでサポートされています。
- ビルドメソッド – Makefile、esbuild。

--build-in-source オプションは、次のオプションとは互換性がありません:

- --hook-name
- --use-container

デフォルト: --no-build-in-source

--cached | --no-cached

キャッシュされたビルドを有効または無効にします。このオプションを使用して、以前の builds から変更されていないビルドアーティファクトを再利用します。 AWS SAM は、プロジェクトディレクトリ内のファイルを変更したかどうかを評価します。デフォルトでは、ビルドはキャッシュされません。この --no-cached オプションが呼び出されると、samcofig.toml の cached = true 設定が上書きされます。

> **Note**
>
> AWS SAM は、特定のバージョンが提供されていない場合、プロジェクトが依存するサードパーティーモジュールが変更されたかどうかを評価しません。例えば、Python 関数にエントリ を持つrequirements.txtファイルが含まれておりrequests=1.x、最新のリクエストモジュールのバージョンが から 1.1に変わった場合1.2、キャッシュされていないビルドを実行するまで、 AWS SAM は最新バージョンをプルしません。

--cache-dir

--cached が指定されている場合にキャッシュアーティファクトが保存されるディレクトリです。デフォルトのキャッシュディレクトリは .aws-sam/cache です。

--config-env *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

--config-file *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

--container-env-var, -e *TEXT*

ビルドコンテナに渡す環境変数です。このオプションは複数回指定できます。このオプションの各インスタンスは、キーバリューペアを使用します。このペアのキーはリソースと環境変数で、値は環境変数の値です。例: --container-env-var Function1.GITHUB_TOKEN=TOKEN1 --container-env-var Function2.GITHUB_TOKEN=TOKEN2。

このオプションは、--use-container オプションが指定されている場合のみに適用され、指定されていない場合はエラーが発生します。

--container-env-var-file, -ef *PATH*

コンテナの環境変数の値を含むJSONファイルのパスとファイル名。コンテナ環境変数ファイルの詳細については、「コンテナ環境変数ファイル」を参照してください。

このオプションは、--use-container オプションが指定されている場合のみに適用され、指定されていない場合はエラーが発生します。

--debug

デバッグログを有効にして、 AWS SAM CLI は、タイムスタンプを表示するために と を生成します。

--docker-network *TEXT*

既存の の名前または ID を指定します。Docker Lambda が使用するネットワーク Docker コンテナは、デフォルトのブリッジネットワークとともに に接続する必要があります。指定しない場合、Lambda コンテナはデフォルトのブリッジにのみ接続します。Docker ネットワーク。

--exclude, -x

sam build から除外するリソースの名前。例えば、テンプレートに Function1、Function2、Function3 が含まれていて、sam build --exclude Function2 を実行する場合、Function1 および Function3 だけが構築されます。

--help

このメッセージを表示して終了します。

--hook-name *TEXT*

拡張に使用されるフックの名前 AWS SAM CLI 機能。

許容値: terraform。

--manifest , -m *PATH*

デフォルトの代わりに使用する、カスタム依存関係のマニフェストファイル (package.json など) へのパスです。

`--no-use-container`

IDE ツールキットを使用してデフォルトの動作を設定できるオプション。`sam build --no-use-container` を使用して、Docker コンテナの代わりにローカルマシンでビルドを実行することもできます。

`--parallel`

並列ビルドを有効にします。このオプションを使用して、 AWS SAM テンプレートの関数とレイヤーを並行して構築します。デフォルトで、関数とレイヤーは順番に構築されます。

`--parameter-overrides`

(オプション) キーと値のペアとしてエンコードされた AWS CloudFormation パラメータオーバーライドを含む文字列。 AWS Command Line Interface () と同じ形式を使用しますAWS CLI。例えば、「`ParameterKey=KeyPairName, ParameterValue=MyKey ParameterKey=InstanceType, ParameterValue=t1.micro`」などです。このオプションは `--hook-name` と互換性がありません。

`--profile` *`TEXT`*

認証情報を取得する AWS 認証情報ファイルからの特定のプロファイル。

`--region` *`TEXT`*

デプロイ AWS リージョン 先の 。例えば、us-east-1 などです。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--skip-prepare-infra`

インフラストラクチャに変更が加えられていない場合は、準備段階をスキップします。`--hook-name` オプションと合わせて使用します。

`--skip-pull-image`

コマンドが Lambda ランタイム用の最新 Docker イメージのプルダウンをスキップするべきかどうかを指定します。

`--template-file, --template, -t` *`PATH`*

AWS SAM テンプレートファイル のパスとファイル名`[default: template.[yaml|yml]]`。このオプションは `--hook-name` と互換性がありません。

`--terraform-project-root-path`

を含む最上位ディレクトリへの相対パスまたは絶対パス Terraform 設定ファイルまたは関数のソースコード。これらのファイルが を含むディレクトリの外部にある場合 Terraform ルートモジュールでは、このオプションを使用して絶対パスまたは相対パスを指定します。このオプションは `--hook-name` を `terraform` に設定する必要があります。

`--use-container, -u`

関数がネイティブにコンパイルされた依存関係を持つパッケージに依存する場合は、このオプションを使用して、Lambda に似た Docker コンテナ内で関数を構築します。

例

`sam build` サブコマンドの使用での、詳細な例と詳しいチュートリアルについては、「AWS SAM を使用した構築の概要」を参照してください。

sam delete

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam delete` コマンド。

の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？

`sam delete` コマンドは、 AWS CloudFormation スタック、Amazon S3 と Amazon にパッケージ化およびデプロイされたアーティファクトECR、および AWS SAM テンプレートファイルを削除することで、 AWS SAM アプリケーションを削除します。

このコマンドは、Amazon ECRコンパニオンスタックがデプロイされているかどうかもチェックし、デプロイされている場合は、そのスタックと Amazon ECR リポジトリの削除をユーザーに促します。`--no-prompts` が指定されている場合、コンパニオンスタックと Amazon ECR リポジトリはデフォルトで削除されます。

使用方法

```
$ sam delete <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は `default` です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある `samconfig.toml` です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログ記録をオンにして、 AWS SAM CLI は と を生成してタイムスタンプを表示します。

`--help`

このメッセージを表示して終了します。

`--no-prompts`

非インタラクティブモードで AWS SAM 動作させるには、このオプションを指定します。スタック名は、`--stack-name` オプションとともに、または設定 `toml` ファイルで指定する必要があります。

`--profile` *TEXT*

認証情報を取得する AWS 認証情報ファイルからの特定のプロファイル。

`--region` *TEXT*

デプロイ先の AWS リージョン。例えば、us-east-1 などです。

`--s3-bucket`

削除する Amazon S3 バケットのパス。

`--s3-prefix`

削除する Amazon S3 バケットのプレフィックス。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--stack-name` *TEXT*

削除する AWS CloudFormation スタックの名前。

例

次のコマンドは、スタック MY-STACK を削除します。

```
$ sam delete --stack-name MY-STACK
```

次のコマンドは、スタック MY-STACK と S3 バケット amzn-s3-demo-bucket を削除します。

```
$ sam delete \
    --stack-name MyStack \
    --s3-bucket MySAMBucket
```

sam deploy

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam deploy` コマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメントについては、 AWS SAM CLI `sam deploy` コマンド、「」を参照してくださいでのデプロイの概要 AWS SAM。

`sam deploy` コマンドは、 AWS クラウド を使用してアプリケーションを にデプロイします AWS CloudFormation。

使用方法

```
$ <environment variables> sam deploy <options>
```

環境変数

`SAM_CLI_POLL_DELAY`

シェルに 秒の値でSAM_CLI_POLL_DELAY環境変数を設定し、 が AWS SAM AWS CloudFormation スタック状態CLIをチェックする頻度を設定します。これは、 からスロットリ

ングを表示する場合に役立ちます AWS CloudFormation。この env 変数は、 の実行中に行われるdescribe_stackAPIコールのポーリングに使用されますsam deploy。

この変数の例を次に示します。

```
$ SAM_CLI_POLL_DELAY=5 sam deploy
```

オプション

`--capabilities LIST`

特定のスタックの作成 AWS CloudFormation を許可するために指定する必要がある機能のリスト。一部のスタックテンプレートには AWS アカウント、新しい AWS Identity and Access Management （ IAM) ユーザーを作成するなど、 のアクセス許可に影響するリソースが含まれている場合があります。このようなスタックについては、このパラメータを指定することによって、それらの機能を明示的に承認する必要があります。有効な値は、CAPABILITY_IAM と CAPABILITY_NAMED_IAM のみです。IAM リソースがある場合は、どちらの機能も指定できます。カスタム名を持つIAMリソースがある場合は、 を指定する必要がありますCAPABILITY_NAMED_IAM。このオプションを指定しない場合は、オペレーションが InsufficientCapabilities エラーを返します。

ネストされたアプリケーションを含むアプリケーションをデプロイする場合は、CAPABILITY_AUTO_EXPAND を使用して、アプリケーションにネストされたアプリケーションが含まれることを承認する必要があります。詳細については、「ネストされたアプリケーションのデプロイ」を参照してください。

`--config-env TEXT`

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は default です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file PATH`

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある samconfig.toml です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--confirm-changeset | --no-confirm-changeset`

が AWS SAM CLI は、計算された変更セットをデプロイします。

`--debug`

デバッグログを有効にして、 AWS SAM CLI は と を生成してタイムスタンプを表示します。

`--disable-rollback | --no-disable-rollback`

デプロイ中にエラーが発生した場合に AWS CloudFormation スタックをロールバックするかどうかを指定します。デフォルトでは、デプロイ中にエラーが発生した場合、 AWS CloudFormation スタックは最後の安定状態に戻ります。`--disable-rollback` を指定した状態でデプロイ中にエラーが発生した場合、エラーが発生する前に作成または更新されたリソースはロールバックされません。

`--fail-on-empty-changeset | --no-fail-on-empty-changeset`

スタックに対して行う変更がない場合に 0 以外の終了コードを返すかどうかを指定します。デフォルトの動作では、ゼロ以外の終了コードが返されます。

`--force-upload`

このオプションを指定して、アーティファクトが Amazon S3 バケット内の既存のアーティファクトと一致する場合でも、それらをアップロードします。一致するアーティファクトは上書きされます。

`--guided, -g`

このオプションを指定すると、 AWS SAM CLI プロンプトを使用してデプロイをガイドします。

`--help`

このメッセージを表示して終了します。

`--image-repositories` *`TEXT`*

Amazon ECRリポジトリ への関数のマッピングURI。論理 ID で関数を参照します。以下に例を示します。

```
$ sam deploy --image-repositories Function1=123456789012.dkr.ecr.us-
east-1.amazonaws.com/my-repo
```

このオプションは 1 つのコマンドで複数回指定できます。

`--image-repository` *`TEXT`*

このコマンドが関数のイメージをアップロードする Amazon ECRリポジトリの名前。このオプションは、`Image` パッケージタイプで宣言された関数に必要です。

`--kms-key-id` *TEXT*

Amazon S3 バケットに保存されているアーティファクトの暗号化に使用される AWS Key Management Service （ AWS KMS) キーの ID。このオプションを指定しない場合、 は Amazon S3-managed暗号化キー AWS SAM を使用します。

`--metadata`

テンプレートで参照されるすべてのアーティファクトにアタッチするメタデータのマップです。

`--no-execute-changeset`

変更セットを適用するかどうかを示します。このオプションは、変更セットを適用する前にスタックの変更を表示したい場合に指定します。このコマンドは、 AWS CloudFormation 変更セットを作成してから、その変更セットを適用せずに終了します。変更セットを適用するには、このオプションを指定せずに同じコマンドを実行します。

`--no-progressbar`

Amazon S3 へのアーティファクトのアップロード時に、進行状況バーを表示しません。

`--notification-arns` *LIST*

スタックARNs AWS CloudFormation に関連付ける Amazon Simple Notification Service (Amazon SNS) トピックのリスト。

`--on-failure [ROLLBACK | DELETE | DO_NOTHING]`

スタックの作成が失敗されたときに実行するアクションを指定します。

以下のオプションが利用できます。

- `ROLLBACK` — スタックを以前の既知の正常な状態にロールバックします。
- `DELETE` — 以前の既知の正常な状態が存在する場合、スタックを以前の既知の状態にロールバックします。以前の既知の正常な状態が存在しない場合は、スタックを削除します。
- `DO_NOTHING` — スタックのロールバックおよび削除を行いません。この結果は `--disable-rollback` の結果と同じです。

デフォルトの動作は `ROLLBACK` です。

> **Note**
>
> `--disable-rollback` オプションまたは `--on-failure` オプションを指定できますが、両方を指定することはできません。

`--parameter-overrides` *LIST*

キーと値のペアとしてエンコードされた AWS CloudFormation パラメータを含む文字列。() と同じ形式を使用します AWS Command Line Interface AWS CLI。の AWS SAM CLI 形式は明示的なキーと値のキーワードで、各オーバーライドはスペースで区切られます。ここでは、以下の 2 つの例を示します。

```
$ sam deploy --parameter-overrides ParameterKey=value1,ParameterValue=value2
```

```
$ sam deploy --parameter-overrides ParameterKey=value1,ParameterValue=value2
 ParameterKey=hello,ParameterValue=world ParameterKey=apple,ParameterValue=banana
```

`--profile` *TEXT*

認証情報を取得する AWS 認証情報ファイルからの特定のプロファイル。

`--region` *TEXT*

デプロイ AWS リージョン 先の 。例えば、us-east-1 などです。

`--resolve-image-repos`

ガイドなしデプロイのパッケージ化とデプロイに使用する Amazon ECR リポジトリを自動的に作成します。このオプションは、PackageType: Image が指定された関数とレイヤーにのみ適用されます。--guided オプションを指定すると、 AWS SAM CLI は を無視します--resolve-image-repos。

> **Note**
>
> このオプションを使用して関数またはレイヤーの Amazon ECR リポジトリ AWS SAM を自動的に作成し、後で AWS SAM テンプレートからそれらの関数またはレイヤーを削除すると、対応する Amazon ECR リポジトリは自動的に削除されます。

`--resolve-s3`

ガイドなしデプロイのパッケージ化とデプロイに使用する Amazon S3 バケットを自動的に作成します。--guided オプションを指定すると、 は AWS SAM CLIを無視します--resolve-s3。--s3-bucket と --resolve-s3 の両方のオプションを指定するとエラーが発生します。

`--role-arn` *TEXT*

変更セットを適用するときに AWS CloudFormation 引き受けるIAMロールの Amazon リソースネーム (ARN) 。

`--s3-bucket` *TEXT*

このコマンドが AWS CloudFormation テンプレートをアップロードする Amazon S3 バケットの名前。テンプレートが 51,200 バイトより大きい場合は、`--s3-bucket` オプションまたは `--resolve-s3` オプションは必須です。`--s3-bucket` と `--resolve-s3` の両方のオプションを指定するとエラーが発生します。

`--s3-prefix` *TEXT*

Amazon S3 バケットにアップロードされるアーティファクト名に追加されたプレフィックスです。プレフィックス名は、Amazon S3 バケットのパス名 (フォルダ名) です。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--signing-profiles` *LIST*

デプロイパッケージに署名するための署名プロファイルのリストです。このオプションはキーバリューペアのリストを使用します。このペアのキーは署名する関数またはレイヤーの名前、値は署名プロファイルで、オプションのプロファイル所有者は：で区切られます。例えば、`FunctionNameToSign=SigningProfileName1 LayerNameToSign=SigningProfileName2:SigningProfileOwner` と指定します。

`--stack-name` *TEXT*

(必須) デプロイ先の AWS CloudFormation スタックの名前。既存のスタックを指定すると、コマンドはスタックを更新します。新しいスタックを指定すると、コマンドはスタックを作成します。

`--tags` *LIST*

作成または更新されたスタックに関連付けるタグのリスト。 AWS CloudFormation は、これらのタグをサポートしているスタック内のリソースにも伝達します。

`--template-file, --template, -t` *PATH*

AWS SAM テンプレートがあるパスとファイル名。

> **Note**
>
> このオプションを指定すると、 は、テンプレートと、それが指しているローカルリソースのみを AWS SAM デプロイします。

`--use-json`

AWS CloudFormation テンプレートJSONの出力。デフォルトの出力は ですYAML。

例

`sam deploy` サブコマンドの使用での、詳細な例と詳しいチュートリアルについては、「でのデプロイの概要 AWS SAM」を参照してください。

sam init

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam init` コマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメント AWS SAM CLI `sam init` コマンドについては、「」を参照してくださいAWS SAM でアプリケーションを作成する。

`sam init` コマンドには、新しいサーバーレス アプリケーションを初期化するためのオプションがあります。

使用方法

```
$ sam init <options>
```

オプション

`--app-template` *TEXT*

使用するマネージドアプリケーションテンプレートの識別子です。よくわからない場合は、インタラクティブなワークフローのオプションを指定せずに `sam init` を呼び出します。

このパラメータは、--no-interactive が指定され、--location が指定されていない場合に必要です。

このパラメータは でのみ使用できます。 AWS SAM CLI バージョン 0.30.0 以降。それより前のバージョンでこのパラメータを指定すると、エラーが発生します。

--application-insights | --no-application-insights

アプリケーションの Amazon CloudWatch Application Insights モニタリングを有効にします。詳細については、「AWS SAM サーバーレスアプリケーションをモニタリングするための CloudWatch Application Insights の使用」を参照してください。

デフォルトのオプションは --no-application-insights です。

--architecture, -a *[x86_64 | arm64]*

アプリケーションの Lambda 関数の命令セットアーキテクチャ。x86_64 または arm64 のいずれかを指定します。

--base-image *[amazon/dotnet8-base | amazon/dotnet6-base | amazon/java21-base | amazon/java17-base | amazon/java11-base | amazon/nodejs22.x-base | amazon/nodejs20.x-base | amazon/nodejs18.x-base | amazon/nodejs16.x-base | amazon/python3.13-base | amazon/python3.12-base | amazon/python3.11-base | amazon/python3.10-base | amazon/python3.9-base | amazon/python3.8-base | amazon/ruby3.3-base | amazon/ruby3.2-base]*

アプリケーションのベースイメージです。このオプションは、パッケージタイプが Image の場合のみに適用されます。

このパラメータは、--no-interactive が指定されている、--package-type が Image として指定されている、および --location が指定されていない場合に必要です。

--config-env *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

--config-file *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログを有効にして、 AWS SAM CLI は を生成し、 はタイムスタンプを表示します。

`--dependency-manager, -d` *`[ gradle | mod | maven | bundler | npm | cli-package | pip ]`*

Lambda ランタイムの依存関係マネージャーです。

`--extra-content`

テンプレートの cookiecutter.json 設定にあるカスタムパラメータ ({"customParam1": "customValue1", "customParam2":"customValue2"} など) を上書きします。

`--help, -h`

このメッセージを表示して終了します。

`--location, -l` *`TEXT`*

テンプレートまたはアプリケーションの場所 (Git、Mercurial、HTTP/HTTPS、.zip ファイル、パス) 。

このパラメータは、`--no-interactive` が指定されていて、`--runtime`、`--name`、および `--app-template` が提供されていない場合に必要です。

Git リポジトリの場合は、リポジトリのルートの場所を使用する必要があります。

ローカルパスの場合、テンプレートは.zip ファイルまたは Cookiecutter 形式にする必要があります。

`--name, -n` *`TEXT`*

ディレクトリとして生成されるプロジェクトの名前です。

このパラメータは、`--no-interactive` が指定され、`--location` が指定されていない場合に必要です。

`--no-input`

Cookiecutter プロンプトを無効にし、テンプレート設定で定義されている vcfdefault 値を受け入れます。

`--no-interactive`

init パラメータのインタラクティブなプロンプトを無効にし、必要な値が欠落している場合は失敗します。

`--output-dir, -o` *PATH*

初期化されたアプリケーションが出力される場所です。

`--package-type` *[Zip | Image]*

サンプルアプリケーションのパッケージタイプです。Zip は .zip ファイルアーカイブを作成し、Image はコンテナイメージを作成します。

`--runtime, -r` *[dotnet8 | dotnet6 | java21 | java17 | java11 | nodejs22.x | nodejs20.x | nodejs18.x | nodejs16.x | python3.13 | python3.12 | python3.11 | python3.10 | python3.9 | python3.8 | ruby3.3 | ruby3.2]*

アプリケーションの Lambda ランタイムです。このオプションは、パッケージタイプが Zip の場合のみに適用されます。

このパラメータは、`--no-interactive` が指定されている、`--package-type` が Zip として指定されている、および `--location` が指定されていない場合に必要です。

`--save-params`

コマンドラインで指定したパラメータ AWS SAM を設定ファイルに保存します。

`--tracing | --no-tracing`

Lambda 関数の AWS X-Ray トレースを有効にします。

例

`sam init` サブコマンドの使用での、詳細な例と詳しいチュートリアルについては、「AWS SAM でアプリケーションを作成する」を参照してください。

sam list

このページでは、AWS Serverless Application Model コマンドラインインタフェイス (AWS SAM CLI) の `sam list` コマンドのリファレンス情報を提供します。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください

`sam list` コマンドは、サーバーレスアプリケーションのリソースとサーバーレスアプリケーションの状態に関する重要な情報を出力します。デプロイの前後に sam list を使用して、ローカルおよびクラウドの開発をサポートします。

使用方法

```
$ sam list <options> <subcommand>
```

オプション

`--help, -h`

このメッセージを表示して終了します。

サブコマンド

`endpoints`

AWS CloudFormation スタックのクラウドエンドポイントとローカルエンドポイントのリストを表示します。詳細については、「sam list endpoints」を参照してください。

`resources`

デプロイ時に AWS CloudFormation で作成された AWS Serverless Application Model (AWS SAM) テンプレート内のリソースを表示します。詳細については、「sam list resources」を参照してください。

`stack-outputs`

AWS SAM または AWS CloudFormation テンプレートからの AWS CloudFormation スタックの出力を表示します。詳細については、「sam list stack-outputs」を参照してください。

sam list endpoints

このページでは、AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) の `sam list endpoints` サブコマンドのリファレンス情報を提供します。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください

`sam list endpoints` サブコマンドは、AWS CloudFormation スタックのクラウド エンドポイントとローカル エンドポイントのリストを表示します。sam local および sam sync コマンドを使用して、これらのリソースを操作できます。

AWS Lambda および Amazon API Gateway リソースタイプは、このコマンドでサポートされています。

> **Note**
>
> カスタムドメインは、Amazon API Gateway リソース用に設定されている場合にサポートされます。このコマンドは、デフォルトのエンドポイントの代わりにカスタムドメインを出力します。

使用方法

```
$ sam list endpoints <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。

デフォルト値: `default`

設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *TEXT*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。

デフォルト値: 現在の作業ディレクトリ内の `samconfig.toml`。

設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログ記録をオンにして、AWS SAM CLI によって生成されたデバッグメッセージをタイムスタンプ付きで出力します。

`--help, -h`

このメッセージを表示して終了します。

`--output [json|table]`

結果を出力する形式を指定します。

デフォルト値: `table`

`--profile` *TEXT*

認証情報ファイルから特定のプロファイルを選択して、AWS 認証情報を取得します。

`--region` *TEXT*

サービスの AWS リージョンを設定します。例えば、`us-east-1` と指定します。

`--save-params`

コマンドラインで指定したパラメータを AWS SAM 設定ファイルに保存します。

`--stack-name` *TEXT*

デプロイされた AWS CloudFormation スタックの名前。スタック名は、アプリケーションの `samconfig.toml` ファイルまたは指定された設定ファイルで見つけることができます。

このオプションが指定されていない場合、テンプレートで定義されているローカルリソースが表示されます。

`--template-file, --template, -t` *PATH*

AWS SAM テンプレートファイル。

デフォルト値: `template.[yaml|yml|json]`

例

`test-stack` という名前の AWS CloudFormation スタックからデプロイされたリソースエンドポイントの出力を json 形式で表示します。

```
$ sam list endpoints --stack-name test-stack --output json

[
  {
    "LogicalResourceId": "HelloWorldFunction",
    "PhysicalResourceId": "sam-app-test-list-HelloWorldFunction-H85Y7yIV7ZLq",
    "CloudEndpoint": "https://zt55oi7kbljxjmcoahsj3cknwu0rposq.lambda-url.us-
east-1.on.aws/",
    "Methods": "-"
  },
  {
    "LogicalResourceId": "ServerlessRestApi",
    "PhysicalResourceId": "uj80uoe2o2",
    "CloudEndpoint": [
```

```
      "https://uj80uoe2o2.execute-api.us-east-1.amazonaws.com/Prod",
      "https://uj80uoe2o2.execute-api.us-east-1.amazonaws.com/Stage"
    ],
    "Methods": [
      "/hello['get']"
    ]
  }
]
```

sam list resources

このページでは、AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) の `sam list resources` サブコマンドのリファレンス情報を提供します。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください

`sam list resources` サブコマンドは、デプロイ時に AWS SAM 変換によって AWS CloudFormation で作成された AWS Serverless Application Model (AWS SAM) テンプレート内のリソースを表示します。

デプロイ前に AWS SAM テンプレートとともに sam list resources を使用して、作成されるリソースを確認します。デプロイされたリソースを含む統合リストを表示するには、AWS CloudFormation スタック名を指定します。

> **Note**
>
> AWS SAM テンプレートからリソースのリストを生成するために、テンプレートのローカル変換が実行されます。このリストには、特定のリージョン内など、条件付きでデプロイされるリソースが含まれます。

使用方法

```
$ sam list resources <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。

デフォルト値: `default`

設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *TEXT*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。

デフォルト値: 現在の作業ディレクトリ内の `samconfig.toml`。

設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログ記録をオンにして、AWS SAM CLI によって生成されたデバッグメッセージをタイムスタンプ付きで出力します。

`--help, -h`

このメッセージを表示して終了します。

`--output [json|table]`

結果を出力する形式を指定します。

デフォルト値: `table`

`--profile` *TEXT*

認証情報ファイルから特定のプロファイルを選択して、AWS 認証情報を取得します。

`--region` *TEXT*

サービスの AWS リージョンを設定します。例えば、`us-east-1` と指定します。

`--save-params`

コマンドラインで指定したパラメータを AWS SAM 設定ファイルに保存します。

`--stack-name` *TEXT*

デプロイされた AWS CloudFormation スタックの名前。スタック名は、アプリケーションの `samconfig.toml` ファイルまたは指定された設定ファイルで見つけることができます。

指定されると、テンプレートのリソース論理 ID が AWS CloudFormation の対応する物理 ID にマッピングされます。物理 ID の詳細については、「AWS CloudFormation ユーザーガイド」の「リソースフィールド」を参照してください。

このオプションが指定されていない場合、テンプレートで定義されているローカルリソースが表示されます。

`--template-file, --template, -t` *PATH*

AWS SAM テンプレートファイル。

デフォルト値: `template.[yaml|yml|json]`

例

AWS SAM テンプレートからのローカルリソースと、`test-stack` という名前の AWS CloudFormation スタックからデプロイされたリソースの出力を表形式で表示します。ローカルテンプレートと同じディレクトリから実行します。

```
$ sam list resources --stack-name test-stack --output table

-------------------------------------------------------------------------------------------------
Logical ID                                                      Physical ID
-------------------------------------------------------------------------------------------------
HelloWorldFunction                                              sam-app-test-list-
HelloWorldFunction-H85Y7yIV7ZLq
HelloWorldFunctionHelloWorldPermissionProd                      sam-app-test-list-

 HelloWorldFunctionHelloWorldPermissionProd-1QH7CPOCBL2IK
HelloWorldFunctionRole                                          sam-app-test-list-
HelloWorldFunctionRole-SRJDMJ6F7F41
ServerlessRestApi                                               uj80uoe2o2
ServerlessRestApiDeployment47fc2d5f9d                           pncw5f
ServerlessRestApiProdStage                                      Prod
ServerlessRestApiDeploymentf5716dc08b                           -
-------------------------------------------------------------------------------------------------
```

sam list stack-outputs

このページでは、AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) の `sam list stack-outputs` サブコマンドのリファレンス情報を提供します。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください

`sam list stack-outputs` サブコマンドは、AWS Serverless Application Model (AWS SAM) または AWS CloudFormation テンプレートからの AWS CloudFormation スタックの出力を表示しま

す。Outputs の詳細については、「AWS CloudFormation ユーザーガイド」の「出力」を参照してください。

使用方法

```
$ sam list stack-outputs <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。

デフォルト値: `default`

設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *TEXT*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。

デフォルト値: 現在の作業ディレクトリ内の `samconfig.toml`。

設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログ記録をオンにして、AWS SAM CLI によって生成されたデバッグメッセージをタイムスタンプ付きで出力します。

`--help, -h`

このメッセージを表示して終了します。

`--output [json|table]`

結果を出力する形式を指定します。

デフォルト値: `table`

`--profile` *TEXT*

認証情報ファイルから特定のプロファイルを選択して、AWS 認証情報を取得します。

`--region` *TEXT*

サービスの AWS リージョンを設定します。例えば、`us-east-1` と指定します。

`--save-params`

コマンドラインで指定したパラメータを AWS SAM 設定ファイルに保存します。

`--stack-name` *TEXT*

デプロイされた AWS CloudFormation スタックの名前。スタック名は、アプリケーションの `samconfig.toml` ファイルまたは指定された設定ファイルで見つけることができます。

このオプションは必須です。

例

`test-stack` という名前の AWS CloudFormation スタック内のリソースの出力を表形式で表示します。

```
$ sam list stack-outputs --stack-name test-stack --output table

-------------------------------------------------------------------------------------------------
OutputKey                                 OutputValue
 Description
-------------------------------------------------------------------------------------------------
HelloWorldFunctionIamRole                 arn:aws:iam::account-number:role/sam-
 Implicit IAM Role created for Hello
                                          app-test-list-HelloWorldFunctionRole-    World
 function
                                          SRJDMJ6F7F41
HelloWorldApi                             https://uj80uoe2o2.execute-api.us-       API
 Gateway endpoint URL for Prod
                                          east-1.amazonaws.com/Prod/hello/         stage
 for Hello World function
HelloWorldFunction                        arn:aws:lambda:us-                       Hello
 World Lambda Function ARN
                                          east-1:account-number:function:sam-app-
                                          test-list-
                                          HelloWorldFunction-H85Y7yIV7ZLq
-------------------------------------------------------------------------------------------------
```

sam local generate-event

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local generate-event`サブコマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメント AWS SAM CLI `sam local generate-event` コマンド、「」を参照してください[を使用したテストの概要 sam local generate-event](#)。

`sam local generate-event` サブコマンドは、サポートされている AWS のサービスのイベントペイロードサンプルを生成します。

使用方法

```
$ sam local generate-event <options> <service> <event> <event-options>
```

オプション

`--config-env TEXT`

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file PATH`

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある `samconfig.toml` です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--help`

このメッセージを表示して終了します。

サービス

サポートされているサービスのリストを表示するには、次を実行します。

```
$ sam local generate-event
```

[Event] (イベント)

各サービスについて生成できるサポート対象のイベントのリストを表示するには、次を実行します。

```
$ sam local generate-event <service>
```

イベントオプション

変更できるサポート対象のイベントオプションのリストを表示するには、次を実行します。

```
$ sam local generate-event <service> <event> --help
```

例

`sam local generate-event` サブコマンドの使用例については、「サンプルイベントを生成する」を参照してください。

sam local invoke

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local invoke`サブコマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメントについては、 AWS SAM CLI `sam local invoke` サブコマンド、「」を参照してくださいsam local invoke を使用したテストの概要。

`sam local invoke` サブコマンドは、 AWS Lambda 関数の 1 回限りの呼び出しをローカルで開始します。

使用方法

```
$ sam local invoke <arguments> <options>
```

> **Note**
>
> AWS SAM テンプレートに複数の関数が定義されている場合は、呼び出す関数の論理 ID を指定します。

引数

リソース ID

呼び出す Lambda 関数の ID。

この引数はオプションです。アプリケーションに単一の Lambda 関数が含まれている場合、AWS SAM CLIはそれを呼び出します。アプリケーションに複数の関数が含まれている場合は、呼び出す関数の ID を指定します。

有効な値 : リソースの論理 ID またはリソース ARN。

オプション

`--add-host` *`LIST`*

Docker コンテナのホストファイルへの IP アドレスマッピングに、ホスト名を渡します。このパラメータは、複数回渡すことができます。

Example

例: `--add-host` *`example.com:127.0.0.1`*

`--beta-features | --no-beta-features`

ベータ機能を許可または拒否します。

`--config-env` *`TEXT`*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *`PATH`*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「`samconfig.toml`」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--container-env-vars`

(オプション) ローカルでデバッグするときに、環境変数を Lambda 関数イメージコンテナに渡します。

`--container-host` *`TEXT`*

ローカルでエミュレートされた Lambda コンテナのホストです。デフォルト値は `localhost` です。を実行する場合 AWS SAM CLI macOS の Docker コンテナで、 を指定できますhost.docker.internal。コンテナを とは異なるホストで実行する場合 AWS SAM CLIでは、リモートホストの IP アドレスを指定できます。

`--container-host-interface` *TEXT*

コンテナポートがバインドする必要があるホストネットワークインターフェイスの IP アドレスです。デフォルト値は `127.0.0.1` です。`0.0.0.0` を使用して、すべてのインターフェイスにバインドします。

`--debug`

デバッグログ記録を有効にして、 AWS SAM CLI は を生成し、 はタイムスタンプを表示します。

`--debug-args` *TEXT*

デバッガーに渡す追加の引数です。

`--debug-port, -d` *TEXT*

これを指定すると、Lambda 関数コンテナがデバッグモードで起動され、このポートがローカルホストに公開されます。

`--debugger-path` *TEXT*

Lambda コンテナにマウントされたデバッガーへのホストパスです。

`--docker-network` *TEXT*

デフォルトのブリッジネットワークと共に、Lambda Docker コンテナが接続される必要がある既存の Docker ネットワークの名前または ID です。これを指定しない場合、Lambda コンテナはデフォルトのブリッジ Docker ネットワークだけに接続します。

`--docker-volume-basedir, -v` *TEXT*

AWS SAM ファイルが存在するベースディレクトリの場所。Docker がリモートマシンで実行されている場合は、Docker マシンに AWS SAM ファイルが存在するパスをマウントし、この値をリモートマシンに合わせて変更する必要があります。

`--env-vars, -n` *PATH*

Lambda 関数の環境変数の値を含むJSONファイル。環境変数ファイルの詳細については、「環境変数ファイル」を参照してください。

`--event, -e` *PATH*

呼び出されたときに Lambda 関数に渡されるイベントデータを含むJSONファイル。このオプションを指定しない場合、イベントは想定されません。JSON から入力するには`stdin`、値 '-'

を渡す必要があります。さまざまな AWS サービスのイベントメッセージ形式の詳細については、AWS Lambda 「 デベロッパーガイド」の「他の のサービスの使用」を参照してください。

`--force-image-build`

が AWS SAM CLI は、レイヤーで Lambda 関数を呼び出すために使用されるイメージを再構築する必要があります。

`--help`

このメッセージを表示して終了します。

`--hook-name TEXT`

拡張に使用されるフックの名前 AWS SAM CLI 機能。

許容値: `terraform`。

`--invoke-image TEXT`

ローカル関数の呼び出しに使用するコンテナイメージURIの 。デフォルトでは、 は Amazon ECR Public (にリストされている) からコンテナイメージを AWS SAM プルしますのイメージリポジトリ AWS SAM。このオプションは、別の場所からイメージをプルするために使用します。

例えば、`sam local invoke MyFunction --invoke-image amazon/aws-sam-cli-emulation-image-python3.8` と指定します。

`--layer-cache-basedir DIRECTORY`

テンプレートで使用するレイヤーがダウンロードされるベースディレクトリの場所を指定します。

`--log-file, -l TEXT`

ランタイムログを送信するログファイルです。

`--no-event`

空のイベントを使用して関数を呼び出します。

`--parameter-overrides`

キーと値のペアとしてエンコードされた AWS CloudFormation パラメータを含む文字列。() と同じ形式を使用します AWS Command Line Interface AWS CLI。の AWS SAM CLI 形式は明示的なキーと値のキーワードで、各オーバーライドはスペースで区切られます。ここでは、以下の 2 つの例を示します。

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--profile` *TEXT*

認証情報を取得する AWS 認証情報ファイルからの特定のプロファイル。

`--region` *TEXT*

デプロイ先の AWS リージョン。例えば、us-east-1 などです。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--shutdown`

シャットダウン動作の拡張機能処理をテストするために、呼び出し完了後にシャットダウンイベントをエミュレートします。

`--skip-prepare-infra`

インフラストラクチャに変更が加えられていない場合は、準備段階をスキップします。`--hook-name` オプションと合わせて使用します。

`--skip-pull-image`

デフォルトでは、 AWS SAM CLI は Lambda の最新のリモートランタイム環境をチェックし、ローカルイメージを自動的に更新して同期を維持します。

最新の のプルダウンをスキップするには、このオプションを指定します。Docker Lambda ランタイム環境のイメージ。

`--template, -t` *PATH*

AWS SAM テンプレートファイル。

このオプションは `--hook-name` と互換性がありません。

> **Note**
>
> このオプションを指定すると、 は、参照先のテンプレートとローカルリソースのみを AWS SAM ロードします。

`--terraform-plan-file`

ローカルへの相対パスまたは絶対パス Terraform を使用する際の計画ファイル AWS SAM CLI with Terraform Cloud。 このオプションでは、 `--hook-name`を に設定する必要がありま す`terraform`。

例

次の例では、`s3.json` イベントを使用して Lambda 関数をローカルで呼び出すことで、生成されたイベントをローカルテストに使用します。

```
$ sam local invoke --event events/s3.json S3JsonLoggerFunction
```

sam local start-api

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local start-api`サブコマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメントについては、 AWS SAM CLI `sam local start-api` サブコマンドについては、「」を参照してくださいsam local start-api を使用したテストの概要。

`sam local start-api` サブコマンドは、ローカルHTTPサーバーホストを介してテストするために AWS Lambda 関数をローカルで実行します。

使用方法

```
$ sam local start-api <options>
```

オプション

`--add-host` *`LIST`*

Docker コンテナのホストファイルへの IP アドレスマッピングに、ホスト名を渡します。このパラメータは、複数回渡すことができます。

Example

例: `--add-host` *`example.com:127.0.0.1`*

`--beta-features | --no-beta-features`

ベータ機能を許可または拒否します。

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--container-env-vars`

オプション。ローカルでデバッグする時に、環境変数をイメージコンテナに渡します。

`--container-host` *TEXT*

ローカルでエミュレートされた Lambda コンテナのホストです。デフォルト値は `localhost` です。を実行する場合 AWS SAM CLI macOS の Docker コンテナで、 を指定できますhost.docker.internal。コンテナを とは異なるホストで実行する場合 AWS SAM CLIでは、リモートホストの IP アドレスを指定できます。

`--container-host-interface` *TEXT*

コンテナポートがバインドする必要があるホストネットワークインターフェイスの IP アドレスです。デフォルト値は `127.0.0.1` です。`0.0.0.0` を使用して、すべてのインターフェイスにバインドします。

`--debug`

デバッグログ記録を有効にして、 によって生成されたデバッグメッセージを印刷します AWS SAM CLI タイムスタンプを表示します。

`--debug-args` *TEXT*

デバッガーに渡される追加の引数です。

`--debug-function`

オプション。`--warm-containers` が指定されているときにデバッグオプションを適用する Lambda 関数を指定します。このパラメータは、`--debug-port`、`--debugger-path`、および `--debug-args` に適用されます。

`--debug-port, -d` *TEXT*

これを指定すると、Lambda 関数コンテナがデバッグモードで起動され、このポートがローカルホストに公開されます。

`--debugger-path` *TEXT*

Lambda コンテナにマウントされるデバッガーへのホストパスです。

`--docker-network` *TEXT*

デフォルトのブリッジネットワークと共に、Lambda Docker コンテナが接続される必要がある既存の Docker ネットワークの名前または ID です。これを指定しない場合、Lambda コンテナはデフォルトのブリッジ Docker ネットワークのみに接続します。

`--docker-volume-basedir, -v` *TEXT*

AWS SAM ファイルが存在するベースディレクトリの場所。Docker がリモートマシンで実行されている場合は、Docker マシンに AWS SAM ファイルが存在するパスをマウントし、この値をリモートマシンに合わせて変更する必要があります。

`--env-vars, -n` *PATH*

Lambda 関数の環境変数の値を含むJSONファイル。

`--force-image-build`

以下を指定します。 AWS SAM CLI は、レイヤーを使用して関数を呼び出すために使用されるイメージを再構築します。

`--help`

このメッセージを表示して終了します。

`--hook-name` *TEXT*

拡張に使用されるフックの名前 AWS SAM CLI 機能。

許容値: `terraform`。

`--host` *TEXT*

バインド先のローカルホスト名または IP アドレス (デフォルト:「127.0.0.1」)。

`--invoke-image` *TEXT*

Lambda 関数に使用するコンテナイメージURIの 。デフォルトでは、 は Amazon ECR Public からコンテナイメージ AWS SAM をプルします。このオプションは、別の場所からイメージをプルするために使用します。

このオプションは複数回指定できます。このオプションの各インスタンスには、文字列またはキーバリューペアを使用できます。文字列を指定すると、アプリケーションのすべての関数に使用するコンテナイメージURIの になります。例えば、`sam local start-api --invoke-image public.ecr.aws/sam/emu-python3.8` と指定します。キーと値のペアを指定すると、キーはリソース名、値はそのリソースに使用するコンテナイメージURIの になります。例えば、`sam local start-api --invoke-image public.ecr.aws/sam/emu-python3.8 --invoke-image Function1=amazon/aws-sam-cli-emulation-image-python3.8` です。キーバリューペアを使用すると、異なるリソースに異なるコンテナイメージを指定できます。

`--layer-cache-basedir` *DIRECTORY*

テンプレートで使用されるレイヤーがダウンロードされる場所である basedir を指定します。

`--log-file, -l` *TEXT*

ランタイムログを送信するログファイルです。

`--parameter-overrides`

キーと値のペアとしてエンコードされた AWS CloudFormation パラメータを含む文字列。() と同じ形式を使用します AWS Command Line Interface AWS CLI。の AWS SAM CLI 形式は明示的なキーと値のキーワードで、各オーバーライドはスペースで区切られます。ここでは、以下の 2 つの例を示します。

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--port, -p` *INTEGER*

リッスンするローカルポート番号 (デフォルト:「3000」)。

`--profile` *TEXT*

認証情報を取得する AWS 認証情報ファイルからの特定のプロファイル。

`--region` *TEXT*

デプロイ先の AWS リージョン。例えば、us-east-1 などです。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--shutdown`

シャットダウン動作の拡張機能処理をテストするために、呼び出し完了後にシャットダウンイベントをエミュレートします。

`--skip-prepare-infra`

インフラストラクチャに変更が加えられていない場合は、準備段階をスキップします。`--hook-name` オプションと合わせて使用します。

`--skip-pull-image`

CLI が Lambda ランタイムの最新の Docker イメージのプルダウンをスキップするかどうかを指定します。

`--ssl-cert-file` *PATH*

SSL 証明書ファイルへのパス (デフォルト: None)。このオプションを使用する場合は、`--ssl-key-file` オプションも使用する必要があります。

`--ssl-key-file` *PATH*

SSL キーファイルへのパス (デフォルト: なし) 。このオプションを使用する場合は、`--ssl-cert-file` オプションも使用する必要があります。

`--static-dir, -s` *TEXT*

このディレクトリにある静的アセット (などCSS/JavaScript/HTML) ファイルは、 に表示されます/。

`--template, -t` *PATH*

AWS SAM テンプレートファイル。

> **Note**
>
> このオプションを指定すると、 は、参照先のテンプレートとローカルリソースのみを AWS SAM ロードします。

`--terraform-plan-file`

ローカルへの相対パスまたは絶対パス Terraform を使用する際の計画ファイル AWS SAM CLI with Terraform Cloud。 このオプションでは、 `--hook-name`を に設定する必要がありますterraform。

`--warm-containers [EAGER | LAZY]`

オプション。方法を指定します。 AWS SAM CLI は、各関数のコンテナを管理します。

以下の 2 つのオプションを使用できます。

EAGER: 起動時にすべての関数のコンテナがロードされ、呼び出し間で保持されます。

LAZY: 各関数が初めて呼び出される場合に限り、コンテナがロードされます。これらのコンテナは、追加の呼び出し用に保持されます。

例

次の例では、ローカルサーバーを起動し、 を介してアプリケーションをテストできますAPI。このコマンドを機能させるには、アプリケーションがインストール済みで、Docker が実行中である必要があります。

```
$ sam local start-api --port 3000
```

sam local start-lambda

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local start-lambda`サブコマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメント AWS SAM CLI `sam local start-lambda` サブコマンド、「」を参照してください[を使用したテストの概要 sam local start-lambda](#)。

`sam local start-lambda` サブコマンドは、 AWS Lambdaをエミュレートするローカルエンドポイントを起動します。

使用方法

```
$ sam local start-lambda <options>
```

オプション

`--add-host` *`LIST`*

Docker コンテナのホストファイルへの IP アドレスマッピングに、ホスト名を渡します。このパラメータは、複数回渡すことができます。

Example

例: `--add-host` *`example.com:127.0.0.1`*

`--beta-features | --no-beta-features`

ベータ機能を許可または拒否します。

`--config-env` *`TEXT`*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *`PATH`*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--container-env-vars`

オプション。ローカルでデバッグする時に、環境変数をイメージコンテナに渡します。

`--container-host` *`TEXT`*

ローカルでエミュレートされた Lambda コンテナのホストです。デフォルト値は `localhost` です。を実行する場合 AWS SAM CLI macOS の Docker コンテナで、 を指定できますhost.docker.internal。コンテナを とは異なるホストで実行する場合 AWS SAM CLIでは、リモートホストの IP アドレスを指定できます。

`--container-host-interface` *`TEXT`*

コンテナポートがバインドする必要があるホストネットワークインターフェイスの IP アドレスです。デフォルト値は `127.0.0.1` です。`0.0.0.0` を使用して、すべてのインターフェイスにバインドします。

`--debug`

デバッグログ記録を有効にして、 によって生成されたデバッグメッセージを印刷します。 AWS SAM CLI タイムスタンプを表示します。

`--debug-args` *TEXT*

デバッガーに渡される追加の引数です。

`--debug-function`

オプション。`--warm-containers` が指定されているときにデバッグオプションを適用する Lambda 関数を指定します。このパラメータは、`--debug-port`、`--debugger-path`、および `--debug-args` に適用されます。

`--debug-port, -d` *TEXT*

指定すると、Lambda 関数コンテナをデバッグモードで起動し、このポートをローカルホストに公開します。

`--debugger-path` *TEXT*

Lambda コンテナにマウントされるデバッガーへのホストパス。

`--docker-network` *TEXT*

デフォルトのブリッジネットワークと共に、Lambda Docker コンテナが接続される必要がある既存の Docker ネットワークの名前または ID です。これを指定すると、Lambda コンテナはデフォルトのブリッジ Docker ネットワークにのみ接続します。

`--docker-volume-basedir, -v` *TEXT*

AWS SAM ファイルが存在するベースディレクトリの場所。Docker がリモートマシンで実行されている場合は、Docker マシン上の AWS SAM ファイルが存在するパスをマウントし、この値をリモートマシンに合わせて変更する必要があります。

`--env-vars, -n` *PATH*

Lambda 関数の環境変数の値を含むJSONファイル。

`--force-image-build`

が CLI は、レイヤーで関数を呼び出すために使用されるイメージを再構築します。

`--help`

このメッセージを表示して終了します。

`--hook-name` *TEXT*

拡張に使用されるフックの名前 AWS SAM CLI 機能。

許容値: `terraform`。

`--host` *TEXT*

バインド先のローカルホスト名または IP アドレス (デフォルト:「127.0.0.1」)。

`--invoke-image` *TEXT*

ローカル関数の呼び出しに使用するコンテナイメージURIの 。デフォルトでは、 は Amazon ECR Public からコンテナイメージ AWS SAM をプルします。このオプションは、別の場所からイメージをプルするために使用します。

例えば、`sam local start-lambda MyFunction --invoke-image amazon/aws-sam-cli-emulation-image-python3.8` と指定します。

`--layer-cache-basedir` *DIRECTORY*

テンプレートで使用されるレイヤーがダウンロードされる場所である basedir を指定します。

`--log-file, -l` *TEXT*

ランタイムログを送信するログファイルです。

`--parameter-overrides`

キーと値のペアとしてエンコードされた AWS CloudFormation パラメータを含む文字列。() と同じ形式を使用します AWS Command Line Interface AWS CLI。の AWS SAM CLI 形式は明示的なキーと値のキーワードで、各オーバーライドはスペースで区切られます。ここでは、以下の 2 つの例を示します。

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--port, -p` *INTEGER*

リッスンするローカルポート番号 (デフォルト:「3001」)。

`--profile` *TEXT*

認証情報ファイルから AWS 認証情報を取得する特定のプロファイル。

`--region` *TEXT*

デプロイ先の AWS リージョン。例えば、us-east-1 などです。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--shutdown`

シャットダウン動作の拡張機能処理をテストするために、呼び出し完了後にシャットダウンイベントをエミュレートします。

`--skip-prepare-infra`

インフラストラクチャに変更が加えられていない場合は、準備段階をスキップします。`--hook-name` オプションと合わせて使用します。

`--skip-pull-image`

が CLI は、Lambda ランタイムの最新の Docker イメージのプルダウンをスキップします。

`--template, -t` *PATH*

AWS SAM テンプレートファイル。

> **Note**
>
> このオプションを指定すると、 は、参照先のテンプレートとローカルリソースのみを AWS SAM ロードします。このオプションは `--hook-name` と互換性がありません。

`--terraform-plan-file`

ローカルへの相対パスまたは絶対パス Terraform を使用する際の計画ファイル AWS SAM CLI with Terraform Cloud。 このオプションでは、 `--hook-name`を に設定する必要がありますterraform。

`--warm-containers` *[EAGER | LAZY]*

オプション。方法を指定します。 AWS SAM CLI は、各関数のコンテナを管理します。

以下の 2 つのオプションを使用できます。

- EAGER: 起動時にすべての関数のコンテナがロードされ、呼び出し間で保持されます。
- LAZY: 各関数が初めて呼び出される場合に限り、コンテナがロードされます。これらのコンテナは、追加の呼び出し用に保持されます。

例

`sam local start-lambda` サブコマンドの使用での、詳細な例と詳しいチュートリアルについては、「を使用したテストの概要 sam local start-lambda」を参照してください。

sam logs

このページでは、AWS Serverless Application Model コマンドラインインタフェイス (AWS SAM CLI) の `sam logs` コマンドのリファレンス情報を提供します。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください。

`sam logs` コマンドは、AWS Lambda 関数によって生成されたログを取得します。

使用方法

```
$ sam logs <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--cw-log-group` *LIST*

指定した CloudWatch Logs ロググループからのログを含めます。`name` とともにこのオプションを指定した場合、AWS SAM は、その名前のリソースからのログに加えて、指定されたロググループのログも含めます。

`--debug`

デバッグロギングをオンにして、AWS SAM CLI が生成するデバッグメッセージを出力表示し、タイムスタンプを表示します。

`---end-time, e` *TEXT*

この時刻までのログを取得します。時刻には、「5mins ago」、「tomorrow」などの相対的な値、または「2018-01-01 10:10:10」のような形式化されたタイムスタンプにすることができます。

`--filter` *TEXT*

式を指定して、ログイベントの用語、フレーズ、または値に一致するログをすばやく検索できるようにします。これは、シンプルなキーワード (「error」など)、または Amazon CloudWatch Logs でサポートされるパターンにすることができます。構文については、Amazon CloudWatch Logs ドキュメントを参照してください。

`--help`

このメッセージを表示して終了します。

`--include-traces`

ログ出力に X-Ray トレースを含めます。

`--name, -n` *TEXT*

ログを取得するリソースの名前。このリソースが AWS CloudFormation スタックの一部である場合、これは AWS CloudFormation/AWS SAM テンプレート内の関数リソースの論理 ID であることが可能です。パラメータを再度繰り返すことで、複数の名前を指定できます。リソースがネストされたスタックにある場合、そのリソースからログをプルするために、ネストされたスタック名の前に名前を付加できます (NestedStackLogicalId/ResourceLogicalId)。リソース名が指定されていない場合、指定されたスタックがスキャンされ、サポートされているすべてのリソースのログ情報が取得されます。このオプションを指定しない場合、AWS SAM は、指定したスタック内のすべてのリソースのログを取得します。以下のリソースタイプがサポートされています。

- `AWS::Serverless::Function`
- `AWS::Lambda::Function`
- `AWS::Serverless::Api`
- `AWS::ApiGateway::RestApi`
- `AWS::Serverless::HttpApi`
- `AWS::ApiGatewayV2::Api`
- `AWS::Serverless::StateMachine`
- `AWS::StepFunctions::StateMachine`

`--output` *TEXT*

ログの出力形式を指定します。フォーマットされたログを印刷するには、`text` を指定します。ログを JSON として印刷するには、`json` を指定します。

`--profile` *TEXT*

AWS 認証情報を取得する、認証情報ファイルから特定のプロファイルです。

`--region` *TEXT*

デプロイ先の AWS リージョンです。例えば、us-east-1 などです。

`--save-params`

コマンドラインで指定したパラメータを AWS SAM 設定ファイルに保存します。

`--stack-name` *TEXT*

このリソースが一部を成す AWS CloudFormation スタックの名前です。

`--start-time, -s` *TEXT*

この時刻以降のログを取得します。時刻には、「5mins ago」、「yesterday」などの相対的な値、または「2018-01-01 10:10:10」のような形式化されたタイムスタンプにすることができます。デフォルトは「10mins ago」です。

`--tail, -t`

ログ出力を tail します。これにより、終了時間引数は無視され、ログが使用可能になった時点で引き続き取得されます。

例

関数が AWS CloudFormation スタックの一部である場合、スタック名を指定するときに、関数の論理 ID を使用してログを取得できます。

```
$ sam logs -n HelloWorldFunction --stack-name myStack
```

-s (--start-time) と-e (--end-time) オプションを使用して、特定の時間範囲のログを表示します。

```
$ sam logs -n HelloWorldFunction --stack-name myStack -s '10min ago' -e '2min ago'
```

`--tail` オプションを追加して、新しいログを待機し、到着するたびにそれらを表示することもできます。

```
$ sam logs -n HelloWorldFunction --stack-name myStack --tail
```

--filter オプションを使用して、ログイベントの語句、フレーズ、または値が一致するログをすばやく検索します。

```
$ sam logs -n HelloWorldFunction --stack-name myStack --filter "error"
```

子スタック内のリソースのログを表示します。

```
$ sam logs --stack-name myStack -n childStack/HelloWorldFunction
```

アプリケーションでサポートされているすべてのリソースのログを追跡します。

```
$ sam logs --stack-name sam-app --tail
```

アプリケーション内の特定の Lambda 関数および API Gateway API のログを取得します。

```
$ sam logs --stack-name sam-app --name HelloWorldFunction --name HelloWorldRestApi
```

アプリケーション内のサポートされているすべてのリソースのログを取得し、さらに指定したロググループからログを取得します。

```
$ sam logs --cw-log-group /aws/lambda/myfunction-123 --cw-log-group /aws/lambda/
myfunction-456
```

sam package

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) は AWS SAM アプリケーションをパッケージ化します。

このコマンドは、コードと依存関係の.zipファイルを作成し、そのファイルを Amazon Simple Storage Service (Amazon S3) にアップロードします。Amazon S3 に保存されているすべてのファイルの暗号化 AWS SAM を有効にします。次に、 AWS SAM テンプレートのコピーが返され、ローカルアーティファクトへの参照を、コマンドがアーティファクトをアップロードした Amazon S3 の場所に置き換えます。

デフォルトでは、このコマンドを使用すると、 AWS SAM CLI は、現在の作業ディレクトリがプロジェクトのルートディレクトリであることを前提としています。の AWS SAM CLI まず、 sam build コマンドを使用して構築され、.aws-samサブフォルダにあり、 という名前のテンプレートファイ

ルを見つけようとしますtemplate.yaml。次に、 AWS SAM CLI は、 または という名前のテンプレートファイルを現在の作業ディレクトリtemplate.yamltemplate.ymlで検索しようとします。--template オプションを指定すると、 AWS SAM CLIのデフォルトの動作は上書きされ、その AWS SAM テンプレートとそれが指すローカルリソースのみをパッケージ化します。

> **Note**
>
> sam deploy が sam package の機能を暗黙的に実行するようになりました。sam deploy コマンドを直接使用して、アプリケーションをパッケージ化およびデプロイできます。

使用方法

```
$ sam package <arguments> <options>
```

引数

リソース ID

パッケージ化する Lambda 関数の ID です。

この引数はオプションです。アプリケーションに単一の Lambda 関数が含まれている場合、AWS SAM CLIはそれをパッケージ化します。アプリケーションに複数の関数が含まれている場合は、関数の ID を 1 つ指定してパッケージ化します。

有効な値 : リソースの論理 ID またはリソース ARN。

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログ記録を有効にして、 によって生成されたデバッグメッセージを印刷します。 AWS SAM CLI タイムスタンプを表示します。

`--force-upload`

Amazon S3 バケット内の既存のファイルを上書きします。このフラグを指定して、アーティファクトが Amazon S3 バケット内の既存のアーティファクトと一致する場合でも、それらをアップロードします。

`--help`

このメッセージを表示して終了します。

`--image-repository` *TEXT*

このコマンドが関数URIの画像をアップロードする Amazon Elastic Container Registry (Amazon ECR) リポジトリの 。`Image` パッケージタイプで宣言された関数に必要です。

`--kms-key-id` *TEXT*

Amazon S3 バケットに保存されているアーティファクトの暗号化に使用される AWS Key Management Service （ AWS KMS) キーの ID。このオプションが指定されていない場合、 は Amazon S3-managed暗号化キー AWS SAM を使用します。

`--metadata`

(オプション) テンプレートで参照されるすべてのアーティファクトにアタッチするメタデータのマップです。

`--no-progressbar`

Amazon S3 へのアーティファクトのアップロード時に、進行状況バーを表示しません。

`--output-template-file` *PATH*

コマンドがパッケージ化されたテンプレートを書き込むファイルへのパス。パスを指定しない場合、コマンドはテンプレートを標準出力に書き込みます。

`--profile` *TEXT*

認証情報ファイルから AWS 認証情報を取得する特定のプロファイル。

`--region` *TEXT*

デプロイ先の AWS リージョン。例えば、us-east-1 などです。

`--resolve-s3`

パッケージ化に使用する Amazon S3 バケットを自動的に作成します。`--s3-bucket` および `--resolve-s3` オプションの両方を指定すると、エラーが発生します。

`--s3-bucket` *TEXT*

このコマンドがアーティファクトをアップロードする Amazon S3 バケットの名前です。アーティファクトが 51,200 バイトより大きい場合には、`--s3-bucket` または `--resolve-s3` オプションは必須です。`--s3-bucket` および `--resolve-s3` オプションの両方を指定すると、エラーが発生します。

`--s3-prefix` *TEXT*

Amazon S3 バケットにアップロードされるアーティファクト名に追加されたプレフィックスです。プレフィックス名は、Amazon S3 バケットのパス名 (フォルダ名) です。これは、`Zip` パッケージタイプで宣言された関数のみに適用されます。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--signing-profiles` *LIST*

(オプション) デプロイパッケージに署名するための署名プロファイルのリストです。このパラメータはキーバリューペアのリストを使用します。このペアのキーは署名する関数またはレイヤーの名前、値は署名プロファイルで、オプションのプロファイル所有者は：で区切られます。例えば、`FunctionNameToSign=SigningProfileName1 LayerNameToSign=SigningProfileName2:SigningProfileOwner` と指定します。

`--template-file, --template, -t` *PATH*

AWS SAM テンプレートがあるパスとファイル名。

> **Note**
>
> このオプションを指定すると、 は、参照先のテンプレートとローカルリソースのみを AWS SAM パッケージ化します。

`--use-json`

AWS CloudFormation テンプレートJSONの出力。デフォルトでは、YAML が使用されます。

例

次の例では、Lambda 関数と CodeDeploy アプリケーションのアーティファクトを作成してパッケージ化します。アーティファクトは Amazon S3 バケットにアップロードされます。コマンドの出力は、`package.yml` という新しいファイルです。

```
$ sam package \
  --template-file template.yml \
  --output-template-file package.yml \
  --s3-bucket amzn-s3-demo-bucket
```

sam pipeline bootstrap

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam local pipeline bootstrap`サブコマンド。

の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？

`sam pipeline bootstrap` サブコマンドは、CI/CD システムに接続するために必要な AWS インフラストラクチャリソースを生成します。このステップは、sam pipeline init コマンドを実行する前に、パイプラインのデプロイステージごとに実行する必要があります。

このサブコマンドは、次の AWS インフラストラクチャリソースを設定します。

- パイプラインのアクセス許可を設定するオプション:
  - CI/CD システムと共有されるアクセスキー ID とシークレットキーアクセス認証情報を持つパイプラインIAMユーザー。

    > **Note**
    >
    > アクセスキーを定期的にローテーションすることをお勧めします。詳細については、「 ユーザーガイド」の「長期的な認証情報を必要とするユースケースのアクセスキーを定期的にローテーションする」を参照してください。 IAM
  - を通じてサポートされている CI/CD プラットフォームOIDC。パイプラインOIDCでの AWS SAM の使用の概要については、「」を参照してくださいAWS SAM パイプラインを使用した OIDC 認証の使用方法。
- AWS SAM アプリケーションをデプロイ AWS CloudFormation するために が引き受ける AWS CloudFormation 実行IAMロール。

- AWS SAM アーティファクトを保持する Amazon S3 バケット。
- オプションで、コンテナECRイメージ Lambda デプロイパッケージを保持する Amazon イメージリポジトリ (パッケージタイプ のリソースがある場合Image) 。

使用方法

```
$ sam pipeline bootstrap <options>
```

オプション

`--bitbucket-repo-uuid` *TEXT*

Bitbucket リポジトリUUIDの 。このオプションは、アクセス許可に Bitbucket を使用することOIDCに固有のものです。

> **Note**
>
> この値は、https://bitbucket.org/ で確認できます。*workspace*/*repository*/admin/addon/admin/pipelines/openid-connect

`--bucket` *TEXT*

AWS SAM アーティファクトを保持する Amazon S3 バケットARNの 。

`--cicd-provider` *TEXT*

AWS SAM パイプラインの CI/CD プラットフォーム。

`--cloudformation-execution-role` *TEXT*

アプリケーションのスタックをデプロイ AWS CloudFormation するときに引き受けるIAMロールARNの 。独自のロールを使用したい場合にのみ提供します。提供されない場合は、このコマンドによって新しいロールが作成されます。

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトのパラメータ値を指定する環境名です。デフォルト値は **`default`** です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある samconfig.toml です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--confirm-changeset | --no-confirm-changeset`

リソースのデプロイを確認するプロンプトが表示されます。

`--create-image-repository | --no-create-image-repository`

何も指定されていない場合は、Amazon ECRイメージリポジトリを作成するかどうかを指定します。Amazon ECRリポジトリには、Lambda 関数のコンテナイメージ、または のパッケージタイプを持つレイヤーが保持されますImage。デフォルト: `--no-create-image-repository`。

`--debug`

デバッグログ記録を有効にし、 AWS SAM CLI は を生成し、 はタイムスタンプを表示します。

`--deployment-branch` *TEXT*

デプロイ元のブランチの名前です。このオプションは、アクセス許可に GitHub アクションを使用することOIDCに固有のものです。

`--github-org` *TEXT*

リポジトリが属する GitHub 組織。組織が存在しない場合は、リポジトリ所有者のユーザー名を入力します。このオプションは、アクセス許可に アクションを使用すること GitHubOIDCに固有のものです。

`--github-repo` *TEXT*

デプロイが行われる GitHub リポジトリの名前。このオプションは、アクセス許可に GitHub アクションを使用することOIDCに固有のものです。

`--gitlab-group` *TEXT*

リポジトリが属する GitLab グループ。このオプションは、 アクセス許可に を使用すること GitLab OIDCに固有です。

`--gitlab-project` *TEXT*

GitLab プロジェクト名。このオプションは、 アクセス許可に を使用すること GitLab OIDCに固有です。

`--help, -h`

このメッセージを表示して終了します。

`--image-repository` *`TEXT`*

Lambda 関数ARNのコンテナイメージを保持する Amazon ECRイメージリポジトリ、またはのパッケージタイプを持つレイヤーの Image。提供されている場合は、`--create-image-repository` オプションが無視されます。提供されておらず、`--create-image-repository` が指定されている場合は、コマンドによって作成されます。

`--interactive | --no-interactive`

bootstrap パラメータのインタラクティブなプロンプトを無効にし、必要なパラメータが欠落している場合は失敗します。デフォルト値は `--interactive` です。このコマンドの必須パラメータは `--stage` のみです。

> **Note**
>
> `--no-interactive` が とともに指定されている場合は`--use-oidc-provider`、OIDCプロバイダーに必要なすべてのパラメータを含める必要があります。

`--oidc-client-id` *`TEXT`*

OIDC プロバイダーで使用するように設定されたクライアント ID。

`--oidc-provider` *`[github-actions | gitlab | bitbucket-pipelines]`*

アクセスOIDC許可に使用される CI/CD プロバイダーの名前。 GitLab、 GitHub、および Bitbucket がサポートされています。

`--oidc-provider-url` *`TEXT`*

OIDC プロバイダーURLの 。値は **`https://`** で始める必要があります。

`--permissions-provider` *`[oidc | iam]`*

パイプラインの実行ロールを取得するアクセス許可プロバイダーを選択します。デフォルト値は **`iam`** です。

`--pipeline-execution-role` *TEXT*

このステージで運用するためにパイプラインユーザーが引き受けるIAMロールARNの 。独自のロールを使用したい場合にのみ提供します。提供されない場合は、このコマンドによって新しいロールが作成されます。

`--pipeline-user` *TEXT*

CI/CD システムと共有されているアクセスキー ID とシークレットアクセスキーを持つIAMユーザーの Amazon リソースネーム (ARN) 。これは、このIAMユーザーに、対応する AWS アカウントにアクセスするアクセス許可を付与するために使用されます。指定しない場合、 コマンドはアクセスキー ID とシークレットアクセスキー認証情報とともにIAMユーザーを作成します。

`--profile` *TEXT*

認証情報を取得する AWS 認証情報ファイルからの特定のプロファイル。

`--region` *TEXT*

デプロイ先の AWS リージョン。例えば、`us-east-1` と指定します。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--stage` *TEXT*

対応するデプロイメントステージの名前です。これは、作成された AWS インフラストラクチャリソースのサフィックスとして使用されます。

トラブルシューティング

エラー: 必須パラメータがありません

`--no-interactive` が `--use-oidc-provider` と一緒に指定され、必須パラメータのいずれかが指定されていない場合、欠落しているパラメータの説明とともにこのエラーメッセージが表示されます。

例

次の例では、CI/CD システムの作成に必要な AWS リソースを作成し、デバッグログ記録を有効にし、 によって生成されたデバッグメッセージを出力します。 AWS SAM CLI: イベントを使用して

Lambda 関数をローカルで呼び出すことで、生成されたs3.jsonイベントをローカルテストに使用します。

```
$ sam pipeline bootstrap --debug
```

sam pipeline init

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) sam local pipeline initサブコマンド。

の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？

sam pipeline init サブコマンドは、CI/CD システムが を使用してサーバーレスアプリケーションをデプロイするために使用できるパイプライン設定ファイルを生成します AWS SAM。

sam pipeline init を使用する前に、パイプラインの各ステージに必要なリソースをブートストラップする必要があります。これは、セットアップと設定ファイルの生成プロセス全体でガイドされるように sam pipeline init --bootstrap を実行することで可能になります。または、sam pipeline bootstrap コマンド使用して以前作成したリソースを参照します。

使用方法

```
$ sam pipeline init <options>
```

オプション

`--bootstrap`

必要な AWS インフラストラクチャリソースの作成をユーザーに説明するインタラクティブモードを有効にします。

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は default です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *TEXT*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトのルートディレクトリにある samconfig.toml です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログ記録を有効にして、 AWS SAM CLI は を生成し、 はタイムスタンプを表示します。

`--help, -h`

このメッセージを表示して終了します。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

例

次の例は、 `--bootstrap`オプションを使用して、必要な AWS インフラストラクチャリソースの作成を順を追って説明するインタラクティブモードを順を追って説明する方法を示しています。

```
$ sam pipeline init --bootstrap
```

sam publish

このページでは、AWS Serverless Application Model コマンドラインインタフェイス (AWS SAM CLI) の `sam publish` コマンドのリファレンス情報を提供します。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください

`sam publish` コマンドは、AWS SAM アプリケーションを AWS Serverless Application Repository に公開します。このコマンドは、パッケージ化された AWS SAM テンプレートを受け取り、指定された AWS リージョンにアプリケーションを公開します。

`sam publish` コマンドは、公開に必要なアプリケーションメタデータを包含する `Metadata` セクションが AWS SAM テンプレートに含まれていることを期待します。`Metadata` セクションでは、`LicenseUrl` および `ReadmeUrl` プロパティがローカルファイルではなく、Amazon Simple Storage Service (Amazon S3) バケットを参照する必要があります。AWS SAM テンプレートの `Metadata` セクションに関する詳細については、「AWS SAM CLI を使用してアプリケーションを公開する」を参照してください。

デフォルトで、`sam publish` はアプリケーションをプライベートとして作成します。他の AWS アカウントによるアプリケーションの表示とデプロイを許可する前に、アプリケーションを共有する

必要があります。アプリケーションの共有については、AWS Serverless Application Repository デベロッパーガイドの「AWS Serverless Application Repository Resource-Based Policy Examples」を参照してください。

> **Note**
>
> `sam publish` は現在、ローカルに指定されているネストされたアプリケーションの公開をサポートしていません。アプリケーションにネストされたアプリケーションが含まれている場合は、親アプリケーションを公開する前に、それらを個別に AWS Serverless Application Repository に公開する必要があります。

使用方法

```
$ sam publish <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「`samconfig.toml`」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグロギングをオンにして、AWS SAM CLI が生成するメッセージを出力表示し、タイムスタンプを表示します。

`--help`

このメッセージを表示して終了します。

`--profile` *TEXT*

AWS 認証情報を取得する、認証情報ファイルから特定のプロファイルです。

`--region` *TEXT*

デプロイ先の AWS リージョンです。例えば、us-east-1 などです。

`--save-params`

コマンドラインで指定したパラメータを AWS SAM 設定ファイルに保存します。

`--semantic-version` *TEXT*

(オプション) このオプションを使用して、テンプレートファイルの `Metadata` セクションにある `SemanticVersion` プロパティを上書きするアプリケーションのセマンティックバージョンを提供します。セマンティックバージョニングの詳細については、「Semantic Versioning specification を参照してください。

`--template, -t` *PATH*

AWS SAM テンプレートファイル `[default: template.[yaml|yml]]` のパスです。

例

アプリケーションを発行するには

```
$ sam publish --template packaged.yaml --region us-east-1
```

sam remote invoke

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam remote invoke` コマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメントについては、 AWS SAM CLI `sam remote invoke` コマンド、「」を参照してください[を使用したクラウドでのテストの概要 sam remote invoke。

`sam remote invoke` コマンドは、 AWS クラウドでサポートされているリソースを呼び出します。

使用方法

```
$ sam remote invoke <arguments> <options>
```

引数

リソース ID

呼び出すサポート対象リソースの ID です。

この引数には、次の値を使用できます。

- Amazon リソースネーム (ARN) – リソースARNの 。

  > **Tip**
  >
  > `sam list stack-outputs --stack-name <stack-name>` を使用して、 リソースARNの を取得します。

- 論理 ID – リソースの論理 ID。また、 `--stack-name`オプションを使用して AWS CloudFormation スタック名を指定する必要があります。
- 物理 ID — リソースの物理 ID。この ID は、 を使用してリソースをデプロイするときに作成されます AWS CloudFormation。

  > **Tip**
  >
  > `sam list resources --stack-name <stack-name>` を使用してリソースの物理 ID を取得します。

ARN または物理 ID を指定する場合 :

ARN または物理 ID を指定する場合は、スタック名を指定しないでください。`--stack-name` オプションを使用してスタック名を指定するか、スタック名が設定ファイルで定義されている場合、 AWS SAM CLI は、 AWS CloudFormation スタックからの論理 ID 値としてリソース ID を自動的に処理します。

リソース ID を指定しない場合:

リソース ID を指定せず、 `--stack-name`オプションでスタック名を指定した場合、 AWS SAM CLIは、次の口ジックを使用して AWS CloudFormation スタック内のリソースを自動的に呼び出そうとします。

1. の AWS SAM CLI は、次の順序でリソースタイプを識別し、スタックでリソースタイプが見つかったら次のステップに移動します。

   a. Lambda

b. Step Functions

c. Amazon SQS

d. Kinesis Data Streams

2. リソースタイプにスタック内に 1 つのリソースがある場合、 AWS SAM CLI が呼び出します。リソースタイプの複数のリソースがスタックに存在する場合、 AWS SAM CLI はエラーを返します。

以下は、 AWS SAM CLI は以下を実行します。

- 2 つの Lambda 関数と Amazon SQSキューを含むスタック – AWS SAM CLI は Lambda リソースタイプを見つけ、スタックに複数の Lambda 関数が含まれているため、 と エラーを返します。
- Lambda 関数と 2 つの Amazon Kinesis Data Streams アプリケーションを含むスタック – AWS SAM CLI は、スタックに単一の Lambda リソースが含まれているため、Lambda 関数を見つけて呼び出します。
- 1 つの Amazon SQSキューと 2 つの Kinesis Data Streams アプリケーションを含むスタック – AWS SAM CLI は、スタックに単一の Amazon SQSキューが含まれているため、Amazon SQSキューを見つけて呼び出します。

オプション

`--beta-features | --no-beta-features`

ベータ機能を許可または拒否します。

`--config-env` *`TEXT`*

から使用する環境を指定する AWS SAM CLI 設定ファイル。

デフォルト: `default`

`--config-file` *`FILENAME`*

設定ファイルのパスとファイル名を指定します。

設定ファイルの詳細については、「AWS SAM CLI の設定」を参照してください。

デフォルト: プロジェクトディレクトリのルートにある `samconfig.toml`。

`--debug`

デバッグログの記録をアクティブ化します。これにより、 によって生成されたデバッグメッセージとタイムスタンプが出力されます。 AWS SAM CLI.

`--event, -e` *`TEXT`*

ターゲットリソースに送信するイベントです。

`--event-file` *`FILENAME`*

ターゲットリソースに送信するイベントが含まれるファイルへのパスです。

`--help, -h`

ヘルプメッセージを表示して終了します。

`--output` *`[ text | json ]`*

呼び出しの結果を特定の出力形式で出力します。

`json` – リクエストメタデータとリソースレスポンスは JSON構造で返されます。レスポンスにはフルSDK出力が含まれます。

`text` – リクエストメタデータがテキスト構造で返されます。リソースレスポンスは、呼び出されたリソースの出力形式で返されます。

`--parameter`

追加の Boto3 呼び出されるリソースに渡すことができるパラメータ。

Amazon Kinesis Data Streams

次の追加パラメータは、Kinesis データストリームにレコードを追加するために使用できます。

- `ExplicitHashKey='`*`string`*`'`
- `PartitionKey='`*`string`*`'`
- `SequenceNumberForOrdering='`*`string`*`'`
- `StreamARN='`*`string`*`'`

各パラメータの説明については、「Kinesis.Client.Put_Record」を参照してください。

AWS Lambda

以下の追加のパラメータは、Lambda リソースを呼び出して、バッファされたレスポンスを受け取るために使用できます。

- ClientContext='*base64-encoded string*'
- InvocationType='*[DryRun | Event | RequestResponse]*'
- LogType='*[None | Tail]*'
- Qualifier='*string*'

以下の追加パラメータは、レスポンスストリーミングで Lambda リソースを呼び出すために使用できます。

- ClientContext='*base64-encoded string*'
- InvocationType='*[DryRun | RequestResponse]*'
- LogType='*[None | Tail]*'
- Qualifier='*string*'

各パラメータの説明については、以下を参照してください。

- バッファされたレスポンスを使用する Lambda – Lambda.Client.Invoke
- レスポンスストリーミングを使用する Lambda – Lambda.Client.Invoke_with_Response_Stream

Amazon Simple Queue Service (Amazon SQS)

Amazon SQSキューにメッセージを送信するには、次の追加パラメータを使用できます。

- DelaySeconds=*integer*
- MessageAttributes='*json string*'
- MessageDeduplicationId='*string*'
- MessageGroupId='*string*'
- MessageSystemAttributes='*json string*'

各パラメータの説明については、SQS「.Client.send_message」を参照してください。

AWS Step Functions

次の追加パラメータは、ステートマシンの実行を開始するために使用できます。

- name='*string*'
- traceHeader='*string*'

各パラメータの説明については、SFN「.Client.start_execution」を参照してください。

`--profile` *TEXT*

認証情報ファイルから AWS 認証情報を取得する特定のプロファイル。

`--region` *TEXT*

リソース AWS リージョン の 。例えば、`us-east-1` と指定します。

`--stack-name` *TEXT*

リソースが属する AWS CloudFormation スタックの名前。

`--test-event-name` *NAME*

Lambda 関数に渡す共有可能なテストイベントの名前。

> **Note**
>
> このオプションは Lambda 関数のみをサポートします。

例

次の例では、 AWS クラウドでサポートされているリソースを呼び出し、デバッグログ記録をアクティブ化します。これにより、 によって生成されたデバッグメッセージとタイムスタンプが出力されます。 AWS SAM CLI:

```
$ sam remote invoke--debug
```

sam remote test-event

このページでは、AWS Serverless Application Model コマンドラインインタフェイス (AWS SAM CLI) の `sam remote test-event` コマンドのリファレンス情報を提供します。

- AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください
- AWS SAM CLI `sam remote test-event` コマンドの使用に関するドキュメントについては、「sam remote test-event を使用したクラウドテストの概要」を参照してください。

`sam remote test-event` コマンドは、Amazon EventBridge スキーマレジストリ内の共有可能なテストイベントを操作します。

使用方法

```
$ sam remote test-event <options> <subcommand>
```

オプション

`--help, -h`

ヘルプメッセージを表示して終了します。

サブコマンド

`delete`

EventBridge スキーマレジストリから、共有可能なテストイベントを削除します。詳細については、「sam remote test-event delete」を参照してください。

`get`

EventBridge スキーマレジストリから、共有可能なテストイベントを取得します。詳細については、「sam remote test-event get」を参照してください。

`list`

AWS Lambda 関数用の既存の共有可能なテストイベントを一覧表示します。詳細については、「sam remote test-event list」を参照してください。

`put`

ローカルファイルのイベント EventBridge スキーマレジストリに保存します。詳細については、「sam remote test-event put」を参照してください。

sam remote test-event delete

このページでは、AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) の `sam remote test-event delete` サブコマンドのリファレンス情報を提供します。

- AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください
- AWS SAM CLI `sam remote test-event` コマンドの使用に関するドキュメントについては、「sam remote test-event を使用したクラウドテストの概要」を参照してください。

`sam remote test-event delete` サブコマンドは、Amazon EventBridge スキーマレジストリから共有可能なテストイベントを削除します。

使用方法

```
$ sam remote test-event delete <arguments> <options>
```

引数

リソース ID

共有可能なテストイベントに関連付けられた AWS Lambda 関数の ID。

論理 ID を指定する場合は、`--stack-name` オプションを使用して Lambda 関数に関連付けられた AWS CloudFormation スタックの値も指定する必要があります。

有効な値: リソースの論理 ID またはリソース ARN。

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「`samconfig.toml`」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--help, -h`

ヘルプメッセージを表示して終了します。

`--name` *TEXT*

削除する共有可能なテストイベントの名前。

`--stack-name` *TEXT*

Lambda 関数に関連付けられた AWS CloudFormation スタックの名前。

このオプションは、Lambda 関数の論理 ID を引数として指定する場合に必要です。

sam remote test-event get

このページでは、AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) の `sam remote test-event get` サブコマンドのリファレンス情報を提供します。

- AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください
- AWS SAM CLI `sam remote test-event` コマンドの使用に関するドキュメントについては、「sam remote test-event を使用したクラウドテストの概要」を参照してください。

`sam remote test-event get` サブコマンドで、Amazon EventBridge スキーマレジストリから共有可能なテストイベントを取得します。

使用方法

```
$ sam remote test-event get <arguments> <options>
```

引数

リソース ID

取得する共有可能なテストイベントに関連付けられた AWS Lambda 関数の ID。

論理 ID を指定する場合は、`--stack-name` オプションを使用して Lambda 関数に関連付けられた AWS CloudFormation スタックの値も指定する必要があります。

有効な値: リソースの論理 ID またはリソース ARN。

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「`samconfig.toml`」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--help, -h`

ヘルプメッセージを表示して終了します。

`--name` *TEXT*

取得する共有可能なテストイベントの名前。

`--output-file` *FILENAME*

イベントをローカルマシンに保存する際に指定するファイルパスと名前。

このオプションを指定しない場合、AWS SAM CLI は共有可能なテストイベントの内容をコンソールに出力します。

`--stack-name` *TEXT*

Lambda 関数に関連付けられた AWS CloudFormation スタックの名前。

このオプションは、Lambda 関数の論理 ID を引数として指定する場合に必要です。

sam remote test-event list

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam remote test-event list`サブコマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメントについては、 AWS SAM CLI `sam remote test-event` コマンド、「」を参照してくださいsam remote test-event を使用したクラウドテストの概要。

`sam remote test-event list` サブコマンドは、Amazon EventBridge スキーマレジストリから特定の AWS Lambda 関数の既存の共有可能なテストイベントを一覧表示します。

使用方法

```
$ sam remote test-event list <arguments> <options>
```

引数

リソース ID

共有可能なテストイベントに関連付けられた Lambda 関数の ID。

論理 ID を指定する場合は、 `--stack-name`オプションを使用して Lambda 関数に関連付けられた AWS CloudFormation スタックの値も指定する必要があります。

有効な値 : リソースの論理 ID またはリソース ARN.

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「`samconfig.toml`」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--help, -h`

ヘルプメッセージを表示して終了します。

`--stack-name` *TEXT*

Lambda 関数に関連付けられた AWS CloudFormation スタックの名前。

このオプションは、Lambda 関数の論理 ID を引数として指定する場合に必要です。

例

このコマンドの使用例については、「共有可能なテストイベントを一覧表示する」を参照してください。

sam remote test-event put

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam remote test-event put`サブコマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメントについては、 AWS SAM CLI `sam remote test-event` コマンド、「」を参照してくださいsam remote test-event を使用したクラウドテストの概要。

`sam remote test-event put` サブコマンドは、共有可能なテストイベントをローカルマシンから Amazon EventBridge スキーマレジストリに保存します。

使用方法

```
$ sam remote test-event put <arguments> <options>
```

引数

リソース ID

共有可能なテストイベントに関連付けられた AWS Lambda 関数の ID。

論理 ID を指定する場合は、 `--stack-name`オプションを使用して Lambda 関数に関連付けられた AWS CloudFormation スタックの値も指定する必要があります。

有効な値 : リソースの論理 ID またはリソース ARN.

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「`samconfig.toml`」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--file` *FILENAME*

ローカルマシン上のイベントのファイルパスと名前。

`stdin` から読み取るファイル名の値として `-` を指定します。

このオプションは必須です。

`--force, -f`

共有可能なテストイベントを同じ名前で上書きします。

`--help, -h`

ヘルプメッセージを表示して終了します。

`--name` *TEXT*

共有可能なテストイベントを保存する際に指定する名前。

EventBridge スキーマレジストリに同じ名前の共有可能なテストイベントが存在する場合、 AWS SAM CLI は上書きしません。上書きするには、`--force` オプションを追加します。

`--output-file` *FILENAME*

イベントをローカルマシンに保存する際に指定するファイルパスと名前。

このオプションを指定しない場合、 AWS SAM CLI は共有可能なテストイベントの内容をコンソールに出力します。

`--stack-name` *TEXT*

Lambda 関数に関連付けられた AWS CloudFormation スタックの名前。

このオプションは、Lambda 関数の論理 ID を引数として指定する場合に必要です。

例

このコマンドの使用例については、「共有可能なテストイベントを保存する」を参照してください。

sam sync

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam sync` コマンド。

- の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？
- の使用に関するドキュメント AWS SAM CLI「AWS SAMCLI」を参照してください。

`sam sync` コマンドは、ローカル アプリケーションの変更を AWS クラウドに同期します。

使用方法

```
$ sam sync <options>
```

オプション

`--base-dir, -s` *`DIRECTORY`*

関数またはレイヤーのソースコードへの相対パスを、このディレクトリを基準にして解決します。このオプションは、ソースコードフォルダの相対パスの解決方法を変更する場合に使用します。デフォルトでは、相対パスは AWS SAM テンプレートの場所に関して解決されます。

このオプションは、構築しているルートアプリケーションまたはスタックのリソースに加えて、ネストされたアプリケーションまたはスタックにも適用されます。このオプションは、以下のリソースタイプとプロパティにも適用されます。

- リソースタイプ: `AWS::Serverless::Function` プロパティ: `CodeUri`
- リソースタイプ: `AWS::Serverless::Function` リソース属性: `Metadata` エントリ: `DockerContext`
- リソースタイプ: `AWS::Serverless::LayerVersion` プロパティ: `ContentUri`
- リソースタイプ: `AWS::Lambda::Function` プロパティ: `Code`
- リソースタイプ: `AWS::Lambda::LayerVersion` プロパティ: `Content`

`--build-image` *`TEXT`*

アプリケーションの構築時に使用するコンテナイメージURIの 。デフォルトでは、 は Amazon Elastic Container Registry (Amazon ECR) Public URIのコンテナイメージリポジトリ AWS SAM を使用します。別の画像を使用するには、このオプションを指定します。

このオプションは、1 つのコマンドで複数回使用できます。各オプションには、文字列またはキーと値のペアを指定できます。

- 文字列 – アプリケーション内のすべてのリソースが使用するコンテナイメージURIの を指定します。以下に例を示します。

```
$ sam sync --build-image amazon/aws-sam-cli-build-image-python3.8
```

- キーと値のペア – リソース名をキーとして指定し、そのリソースURIで使用するコンテナイメージを値として指定します。この形式を使用して、アプリケーション内のリソースURIごとに異なるコンテナイメージを指定します。以下に例を示します。

```
$ sam sync --build-image Function1=amazon/aws-sam-cli-build-image-python3.8
```

このオプションは、`--use-container` オプションが指定されている場合のみに適用され、指定されていない場合はエラーが発生します。

`--build-in-source | --no-build-in-source`

ソースフォルダにプロジェクトを直接構築するには `--build-in-source` を指定します。

`--build-in-source` オプションは、次のランタイムとビルドメソッドをサポートします:

- ランタイム – すべて Node.js ランタイムは、 `sam init --runtime`オプションでサポートされています。
- ビルドメソッド – `Makefile`、`esbuild`。

`--build-in-source` オプションは、次のオプションとは互換性がありません:

- `--use-container`

デフォルト: `--no-build-in-source`

`--capabilities` *`LIST`*

が特定のスタック AWS CloudFormation を作成できるように指定する機能のリスト。一部のスタックテンプレートには、 のアクセス許可に影響を与える可能性のあるリソースが含まれている場合があります AWS アカウント。たとえば、new AWS Identity and Access Management (IAM) ユーザーを作成します。デフォルト値を上書きするには、このオプションを指定します。有効な値には次のようなものがあります。

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM
- CAPABILITY_RESOURCE_POLICY
- CAPABILITY_AUTO_EXPAND

デフォルト: `CAPABILITY_NAMED_IAM` および `CAPABILITY_AUTO_EXPAND`

`--code`

デフォルトでは、 はアプリケーション内のすべてのリソースを AWS SAM 同期します。以下を含むコードリソースのみを同期するには、このオプションを指定します。

- `AWS::Serverless::Function`
- `AWS::Lambda::Function`
- `AWS::Serverless::LayerVersion`
- `AWS::Lambda::LayerVersion`
- `AWS::Serverless::Api`
- `AWS::ApiGateway::RestApi`

- AWS::Serverless::HttpApi
- AWS::ApiGatewayV2::Api
- AWS::Serverless::StateMachine
- AWS::StepFunctions::StateMachine

コードリソースを同期するために、 はデプロイスルーではなく AWS サービスAPIsを直接 AWS SAM 使用します AWS CloudFormation。 AWS CloudFormation スタックを更新するには、 sam sync --watchまたは を実行しますsam deploy。

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--dependency-layer | --no-dependency-layer`

同期処理を高速化するために個々の関数の依存関係を別のレイヤーに分離するかどうかを指定します。

デフォルト: `--dependency-layer`

`--image-repository` *TEXT*

このコマンドが関数のイメージをアップロードする Amazon Elastic Container Registry (Amazon ECR) リポジトリの名前。Image パッケージタイプで宣言された関数に必要です。

`--image-repositories` *TEXT*

関数の Amazon ECRリポジトリ へのマッピングURI。論理 ID で関数を参照します。以下に例を示します。

```
$ sam sync --image-repositories Function1=123456789012.dkr.ecr.us-
east-1.amazonaws.com/my-repo
```

このオプションは 1 つのコマンドで複数回指定できます。

`--kms-key-id` *TEXT*

Amazon S3 バケットに保管されているアーティファクトを暗号化するために使用される AWS Key Management Service（ AWS KMS) キーの ID。 Amazon S3 このオプションを指定しない場合、 は Amazon S3-managed暗号化キー AWS SAM を使用します。

`--metadata`

テンプレートで参照するすべてのアーティファクトにアタッチするメタデータのマップです。

`--notification-arns` *LIST*

ARNs がスタック AWS CloudFormation に関連付ける Amazon Simple Notification Service (Amazon SNS) トピックのリスト。

`--no-use-container`

IDE ツールキットを使用してデフォルトの動作を設定できるオプション。

`--parameter-overrides`

キーと値のペアとしてエンコードされた AWS CloudFormation パラメータオーバーライドを含む文字列。 AWS Command Line Interface () と同じ形式を使用しますAWS CLI。 AWS SAM CLI 形式は明示的なキーと値のキーワードで、各オーバーライドはスペースで区切られます。ここでは、以下の 2 つの例を示します。

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--resource` *TEXT*

同期するリソースタイプを指定します。複数のリソースを同期する場合、このオプションを複数回指定できます。このオプションは、`--code` でサポートされています。値は、`--code` にリストされたリソースの 1 つである必要があります。例えば、`--resource AWS::Serverless::Function --resource AWS::Serverless::LayerVersion` と指定します。

`--resource-id` *TEXT*

同期するリソース ID を指定します。複数のリソースを同期する場合、このオプションを複数回指定できます。このオプションは、`--code` でサポートされています。例えば、`--resource-id Function1 --resource-id Function2` と指定します。

`--role-arn` *TEXT*

変更セットを適用するときに が AWS CloudFormation 引き受ける IAMロールの Amazon リソースネーム (ARN) 。

`--s3-bucket` *TEXT*

このコマンドが AWS CloudFormation テンプレートをアップロードする Amazon Simple Storage Service (Amazon S3) バケットの名前。テンプレートが 51,200 バイトより大きい場合は、`--s3-bucket` または `--resolve-s3` オプションは必須です。`--s3-bucket` と `--resolve-s3` の両方のオプションを指定するとエラーが発生します。

`--s3-prefix` *TEXT*

Amazon S3 バケットにアップロードするアーティファクト名に追加されたプレフィックスです。プレフィックス名は、Amazon S3 バケットのパス名 (フォルダ名) です。これは、`Zip` パッケージタイプで宣言された関数にのみ適用されます。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--skip-deploy-sync | --no-skip-deploy-sync`

必要がない場合は、初期インフラストラクチャ同期をスキップするように `--skip-deploy-sync` を指定します。 AWS SAM CLI は、ローカル AWS SAM テンプレートをデプロイされた AWS CloudFormation テンプレートと比較し、変更が検出された場合にのみデプロイを実行します。

が実行されるたびに AWS CloudFormation デプロイを実行する`--no-skip-deploy-sync`ように指定`sam sync`します。

詳細については、「初期 AWS CloudFormation デプロイをスキップする」を参照してください。

デフォルト: `--skip-deploy-sync`

`--stack-name` *TEXT*

アプリケーションの AWS CloudFormation スタックの名前。

このオプションは必須です。

`--tags` *LIST*

作成または更新されたスタックに関連付けるタグのリスト。 は、これらのタグをサポートするスタック内のリソース AWS CloudFormation にも伝播します。

`--template-file, --template, -t` *PATH*

AWS SAM テンプレートが配置されているパスとファイル名。

> **Note**
>
> このオプションを指定すると、 はテンプレートとそれが指すローカルリソースのみを AWS SAM デプロイします。

`--use-container, -u`

関数がネイティブにコンパイルされた依存関係を持つパッケージに依存している場合は、このオプションを使用して のような 内に関数を構築します AWS Lambda。Docker コンテナ。

> **Note**
>
> 現在、このオプションには `--dependency-layer` との互換性がありません。`--use-container` で を使用する場合`--dependency-layer`、 AWS SAM CLI から通知され、に進みます`--no-dependency-layer`。

`--watch`

ローカルアプリケーションの変更を監視し、 に自動的に同期するプロセスを開始します AWS クラウド。デフォルトでは、このオプションを指定すると、 は更新時にアプリケーション内のすべてのリソースを AWS SAM 同期します。このオプションを使用すると、 は初期 AWS CloudFormation デプロイ AWS SAM を実行します。次に、 AWS サービス AWS SAM を使用してコードリソースAPIsを更新します。 AWS SAM テンプレートを更新すると AWS CloudFormation 、 AWS SAM は を使用してインフラストラクチャリソースを更新します。

`--watch-exclude` *TEXT*

ファイルに変更がないかどうかを確認するための監視対象からファイルまたはフォルダを除外します。このオプションを使用するには、`--watch` も指定する必要があります。

このオプションは key-value ペアを受け取ります。

- キー – アプリケーション内の Lambda 関数の論理 ID。
- 値 – 除外する関連付けられたファイル名またはフォルダ。

`--watch-exclude` オプションで指定されたファイルまたはフォルダを更新すると、 AWS SAM CLI は同期を開始しません。ただし、他のファイルまたはフォルダの更新によって同期が開始されると、これらのファイルまたはフォルダはその同期に含まれます。

このオプションは、1 つのコマンドで複数回指定できます。

例

このコマンドの使用例については、「sam sync コマンドのオプション」を参照してください。

sam traces

このページでは、AWS Serverless Application Model コマンドラインインタフェイス (AWS SAM CLI) の `sam traces` コマンドのリファレンス情報を提供します。

AWS SAM CLI の概要については、「AWS SAMCLI とは？」を参照してください

`sam traces` コマンドは、AWS X-Ray で AWS アカウント 内の AWS リージョン トレースを取得します。

使用方法

```
$ sam traces <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「`samconfig.toml`」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--end-time` *TEXT*

この時刻までのトレースを取得します。時刻には、「5mins ago」、「tomorrow」などの相対的な値、または「2018-01-01 10:10:10」のような形式化されたタイムスタンプにすることができます。

`--output` *TEXT*

ログの出力形式を指定します。フォーマットされたログを印刷するには、`text` を指定します。ログを JSON として印刷するには、`json` を指定します。

`--save-params`

コマンドラインで指定したパラメータを AWS SAM 設定ファイルに保存します。

`--start-time` *TEXT*

この時刻以降のトレースを取得します。時刻には、「5mins ago」、「yesterday」などの相対的な値、または「2018-01-01 10:10:10」のような形式化されたタイムスタンプにすることができます。デフォルトは「10mins ago」です。

`--tail`

トレース出力の末尾を表示します。これにより、終了時間引数は無視され、トレースが使用可能になった時点で引き続き取得されます。

`--trace-id` *TEXT*

X-Ray トレースの固有の識別子。

例

次のコマンドを実行して、X-Ray トレースを ID で取得します。

```
$ sam traces --trace-id tracing-id-1 --trace-id tracing-id-2
```

次のコマンドを実行して、X-Ray トレースが使用可能になったら末尾を表示します。

```
$ sam traces --tail
```

sam validate

このページでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) `sam validate` コマンド。

の概要 AWS SAM CLI、「」を参照してください。 AWS SAMCLI とは？

`sam validate` コマンドは、 AWS SAM テンプレートファイルが有効かどうかを確認します。

使用方法

```
$ sam validate <options>
```

オプション

`--config-env` *TEXT*

使用する設定ファイル内のデフォルトパラメータ値を指定する環境名です。デフォルト値は「default」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--config-file` *PATH*

使用するデフォルトのパラメータ値が含まれる設定ファイルのパスとファイル名です。デフォルト値は、プロジェクトディレクトリのルートにある「samconfig.toml」です。設定ファイルの詳細については、「AWS SAM CLI　設定ファイル」を参照してください。

`--debug`

デバッグログ記録を有効にして、 によって生成されたデバッグメッセージを印刷します。 AWS SAM CLI タイムスタンプを表示します。

`--lint`

cfn-lint を通じてテンプレートのリンティング検証を実行します。`cfnlintrc` 設定ファイルを作成して、追加のパラメータを指定します。詳細については、 AWS CloudFormation GitHub リポジトリの「cfn-lint」を参照してください。

`--profile` *TEXT*

認証情報ファイルから AWS 認証情報を取得する特定のプロファイル。

`--region` *TEXT*

デプロイ先の AWS リージョン。例えば、us-east-1 などです。

`--save-params`

コマンドラインで指定したパラメータを設定 AWS SAM ファイルに保存します。

`--template-file, --template, -t` *PATH*

AWS SAM テンプレートファイル。デフォルト値は `template.[yaml|yml]` です。

テンプレートが現在の作業ディレクトリにあり、`template.[yaml|yml|json]` という名前が付けられている場合、このオプションは必須ではありません。

sam build を実行したばかりの場合は、このオプションは不要です。

例

テンプレート検証のための、このコマンドの使用例については「AWS SAM テンプレートファイルを検証する」を参照してください。

cfn-lint でこのコマンドを使用する例については、「AWS CloudFormation Linter による AWS SAM アプリケーションの検証」を参照してください。

AWS SAM CLI の管理

このセクションでは、AWS SAM CLI のバージョンを管理およびカスタマイズする方法について説明します。ここには、プロジェクトレベルの設定ファイルを使用した、AWS SAM CLIコマンドのパラメータ値の設定方法に関する情報が含まれています。また、AWS SAM CLI のさまざまなバージョンの管理、AWS SAM がユーザーに代わって AWS サービスを呼び出せるようにする AWS 認証情報の設定、および AWS SAM をカスタマイズするさまざまな方法に関する情報も含まれています。このセクションの最後では、一般的な AWS SAM のトラブルシューティングを取り扱います。

トピック

- AWS SAM CLI　設定ファイル
- AWS SAM CLI バージョンの管理
- AWS 認証情報のセットアップ
- AWS SAM CLI でのテレメトリ
- AWS SAMCLI トラブルシューティング

AWS SAM CLI　設定ファイル

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) は、設定に使用できるプロジェクトレベルの設定ファイルをサポートします。 AWS SAM CLI コマンドパラメータ値。

設定ファイルの作成と使用に関するドキュメントについては、「AWS SAM CLI の設定」を参照してください。

トピック

デフォルトの設定ファイルの設定

AWS SAM では、次のデフォルト設定ファイル設定が使用されます。

- [Name] (名前) – `samconfig`。
- 場所 — プロジェクトのルートにあります。これは `template.yaml` ファイルと同じ場所です。
- 形式 – `TOML`。詳細については、TOML「」の「」を参照してください。 TOML ドキュメント 。

以下に、デフォルトの設定ファイル名と場所を含むプロジェクト構成の例を示します。

```
sam-app
### README.md
### __init__.py
### events
### hello_world
### samconfig.toml
### template.yaml
### tests
```

`samconfig.toml` ファイルの例を次に示します。

```
...
version = 0.1
```

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

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

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

[default.sync.parameters]
watch = true

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

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

サポートされている設定ファイル形式

サポートされている形式は、TOML および [YAML|YML] です。次の基本構文を参照してください。

TOML

```
version = 0.1
[environment]
[environment.command]
[environment.command.parameters]
option = parameter value
```

YAML

```
version: 0.1
environment:
  command:
    parameters:
```

```
    option: parameter value
```

設定ファイルの指定

デフォルトでは、 AWS SAM CLI は、次の順序で設定ファイルを検索します。

1. カスタム設定ファイル – `--config-file`オプションを使用してファイル名と場所を指定する場合、 AWS SAM CLI は最初にこのファイルを検索します。
2. デフォルトの **`samconfig.toml`** ファイル — これはプロジェクトのルートにあるデフォルトの設定ファイル名と形式です。カスタム設定ファイルを指定しない場合、 AWS SAM CLI 次に、このファイルを検索します。
3. **`samconfig.[yaml|yml]`** file – プロジェクトのルートに `samconfig.toml` が存在しない場合、AWS SAM CLI は、このファイルを検索します。

以下に、`--config-file` オプションを使用してカスタム設定ファイルを指定する例を示します。

```
$ sam deploy --config-file myconfig.yaml
```

> **Note**
>
> `--config-file` パラメータは AWS SAM テンプレートファイルの場所を基準にする必要があります。 AWS SAM CLI は、設定が適用されるコンテキストを決定する必要があります。`samconfig.toml` ファイルは、 のバージョンの設定を管理します。 AWS SAM CLI、は、`samconfig.toml`ファイルの相対フォルダ内の `template.yaml` ファイル (またはオーバーライドされた設定ファイルパラメータ) CLIを検索します。

設定ファイルの基本

環境

環境とは、一意の構成設定一式を含む名前付き識別子です。1 つの AWS SAM アプリケーションに複数の環境を設定できます。

デフォルトの環境名は `default` です。

を使用する AWS SAM CLI `--config-env` 使用する環境を指定するオプション。

Command

コマンドは です。 AWS SAM CLI パラメータ値を指定する コマンド。

すべてのコマンドについてパラメータ値を指定するには、global 識別子を使用します。

を参照する場合 AWS SAM CLI コマンド、スペース ( ) とハイフン (-) をアンダースコア () に置き換えます_。以下の例を参照してください。

- build
- local_invoke
- local_start_api

パラメータ

パラメータは、キーと値のペアとして指定されます。

- キーは です。 AWS SAM CLI コマンドオプション名。
- 値は指定する値です。

キーを指定するときは、長い形式のコマンドオプション名を使用し、ハイフン (-) をアンダースコア (_) に置き換えます。次に例を示します。

- region
- stack_name
- template_file

パラメータ値のルール

TOML

- ブール値は true または false です。例えば、confirm_changeset = true と指定します。
- 文字列値の場合、引用符 ("") を使用します。例えば、region = "us-west-2" と指定します。
- リスト値の場合、引用符 ("") を使用し、各値をスペース ( ) で区切ります。例: capabilities = "CAPABILITY_IAM CAPABILITY_NAMED_IAM"。

- 値にキーと値のペアのリストが含まれる場合、ペアはスペースで区切られ ( )、各ペアの値はエンコードされた引用符 (\" \") で囲まれます。例えば、`tags = "project=\"my-application\" stage=\"production\""` と指定します。
- 複数回指定できるパラメータ値の場合、値は引数の配列になります。例: `image_repositories = ["my-function-1=image-repo-1", "my-function-2=image-repo-2"]`。

YAML

- ブール値は `true` または `false` です。例えば、`confirm_changeset: true` と指定します。
- 単一の文字列値を含むエントリの場合、引用符 ("") はオプションです。例えば、`region: us-west-2` と指定します。これには、1 つの文字列として提供される複数のキーと値のペアを含むエントリも含まれます。以下に例を示します。

```
$ sam deploy --tags "foo=bar hello=world"
```

```
default:
  deploy:
    parameters:
      tags: foo=bar hello=world
```

- 値のリストを含むエントリや、1 つのコマンドで複数回使用できるエントリの場合は、それらを文字列のリストとして指定します。

  以下に例を示します。

```
$ sam remote invoke --parameter "InvocationType=Event" --parameter "LogType=None"
```

```
default:
  remote_invoke:
    parameter:
    - InvocationType=Event
    - LogType=None
```

設定の優先順位

値を設定する場合、次の優先順位が適用されます。

- コマンドラインで指定したパラメータ値は、設定ファイルおよびテンプレートファイルの Parameters セクション内の対応する値よりも優先されます。
- --parameter-overrides オプションがコマンドラインまたは設定ファイルで parameter_overrides キーとともに使用される場合、その値はテンプレートファイルの Parameters セクションの値よりも優先されます。
- 設定ファイルでは、特定のコマンド用に指定されたエントリがグローバルエントリより優先されます。次の例では、sam deploy コマンドはスタック名 my-app-stack を使用します。

  TOML

  ```
  [default.global.parameters]
  stack_name = "common-stack"

  [default.deploy.parameters]
  stack_name = "my-app-stack"
  ```

  YAML

  ```
  default:
    global:
      parameters:
        stack_name: common-stack
    deploy:
      parameters:
        stack_name: my-app-stack
  ```

設定ファイルの作成と変更

設定ファイルの作成

sam init を使用してアプリケーションを作成すると、デフォルトの samconfig.toml ファイルが作成されます。設定ファイルを手動で作成することもできます。

設定ファイルの変更

設定ファイルは手動で変更できます。また、 AWS SAM CLI インタラクティブフロー、設定された値は括弧 () で表示されます[ ]。これらの値を変更すると、 AWS SAM CLI は設定ファイルを更新します。

sam deploy --guided コマンドを使用するインタラクティブフローの例を次に示します。

```
$ sam deploy --guided

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

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

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

設定ファイルを変更する場合、 AWS SAM CLI は次のようにグローバル値を処理します。

- パラメータ値が設定ファイルの globalセクションに存在する場合、 AWS SAM CLI は、特定のコマンドセクションに値を書き込みません。
- パラメータ値が セクションglobalと特定のコマンドセクションの両方に存在する場合、 AWS SAM CLI は、グローバル値に有利な特定のエントリを削除します。

AWS SAM CLI バージョンの管理

アップグレード、ダウングレード、アンインストールを通じて、AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) のバージョンを管理します。オプションで、AWS SAM CLI のナイトリービルドをダウンロードしてインストールできます。

トピック

- AWS SAM CLI のアップグレード
- AWS SAM CLI のアンインストール

- Homebrew を使用する AWS SAM CLI の管理から移行する
- AWS SAM CLI ナイトリービルドを管理する
- pip を使用した仮想環境への AWS SAM CLI のインストール
- Homebrew で AWS SAM CLI を管理する
- トラブルシューティング

AWS SAM CLI のアップグレード

Linux

Linux で AWS SAM CLI をアップグレードするには、「AWS SAM CLIのインストール」のインストール手順に従ってください。ただし、次のように install コマンドに `--update` オプションを追加します。

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

macOS

AWS SAM CLI は、インストールしたときと同じ方法でアップグレードする必要があります。AWS SAM CLI のインストールとアップグレードには、パッケージインストーラーの使用をおすすめします。

パッケージインストーラを使用して AWS SAM CLI をアップグレードするには、最新のパッケージバージョンをインストールします。手順については、AWS SAM CLIのインストール を参照してください。

Windows

AWS SAM CLI をアップグレードするには、Windows のインストール手順を AWS SAM CLI のインストール でもう一度繰り返します。

AWS SAM CLI のアンインストール

Linux

AWS SAM CLI を Linux でアンインストールするには、以下のコマンドを実行して、シンボリックリンクとインストールディレクトリを削除する必要があります。

1. シンボリックリンクとインストールパスを見つけます。

- which コマンドを使用して、シンボリックリンクを検索します。

```
which sam
```

出力には、AWS SAM バイナリがある場所のパスが表示されます。以下はその例です。

```
 /usr/local/bin/sam
```

- ls コマンドを使用して、シンボリックリンクがポイントするディレクトリを検索します。

```
ls -l /usr/local/bin/sam
```

以下の例では、インストールディレクトリが /usr/local/aws-sam-cli になっています。

```
 lrwxrwxrwx 1 ec2-user ec2-user 49 Oct 22 09:49 /usr/local/bin/sam -> /usr/local/
aws-sam-cli/current/bin/sam
```

2. シンボリックリンクを削除します。

```
sudo rm /usr/local/bin/sam
```

3. インストールディレクトリを削除します。

```
sudo rm -rf /usr/local/aws-sam-cli
```

macOS

AWS SAM CLI をインストールしたときと同じ方法でアンインストールします。AWS SAM CLI のインストールには、パッケージインストーラーの使用をおすすめします。

パッケージインストーラーを使用して AWS SAM CLI をインストールした場合は、次の手順でアンインストールします。

AWS SAM CLI をアンインストールするには

1. 以下を変更および実行して、AWS SAM CLI プログラムを削除します。

```
$ sudo rm -rf /path-to/aws-sam-cli
```

a. *sudo* — ユーザーが AWS SAM CLI プログラムがインストールされた場所への書き込み権限を持っている場合、sudo は必須ではありません。それ以外の場合、sudo が必要です。

b. *path-to* — AWS SAM CLI プログラムをインストールした場所へのパス。デフォルトの場所は `/usr/local` です。

2. 以下を変更および実行して、AWS SAM CLI $PATH を削除します。

```
$ sudo rm -rf /path-to-symlink-directory/sam
```

a. *sudo* — ユーザーが $PATH への書き込み権限を持っている場合、sudo は必須ではありません。それ以外の場合、sudo が必要です。

b. *path-to-symlink-directory* — $PATH の環境変数。デフォルトの場所は `/usr/local/bin` です。

3. 以下を実行して AWS SAM CLI がアンインストールされていることを確認します。

```
$ sam --version
command not found: sam
```

Windows

Windows の設定を使用して AWS SAM CLI をアンインストールするには、これらの手順を実行してください。

1. [スタート] メニューから、[プログラムの追加と削除] を検索します。
2. [AWS SAM Command Line Interface] という名前の結果を選択し、[Uninstall] (アンインストール) を選択してアンインストーラを起動します 。
3. AWS SAM CLI をアンインストールすることを確認します。

Homebrew を使用する AWS SAM CLI の管理から移行する

Homebrew で AWS SAM CLI のインストールとアップグレードを行っている場合、AWS の使用をおすすめします。サポートされている方法に切り替えるには、以下の手順に従ってください。

Homebrew の使用からの移行方法

1. Homebrew でインストールした AWS SAM CLI をアンインストールする のインストラクションに従い、Homebrew 管理のバージョンをアンインストールします。

2. AWS SAM CLI のインストール のインストラクションに従い、AWS SAM CLI をサポートされている方法でインストールします。

AWS SAM CLI ナイトリービルドを管理する

AWS SAM CLI ナイトリービルドをダウンロードしてインストールできます。ナイトリービルドには、実稼働バージョンよりも安定性が低い可能性があるプレリリースバージョンの AWS SAM CLI コードが含まれています。インストールされると、`sam-nightly` コマンドを使用したナイトリービルドを使用できるようになります。AWS SAM CLI の実稼働ビルドとナイトリービルドは、両バージョンを同時にインストールして使用することが可能です。

> **Note**
>
> ナイトリービルドには、プレリリースバージョンのビルドイメージは含まれません。そのため、--use-container オプションを使用してサーバーレスアプリケーションを構築すると、ビルドイメージの最新の実稼働バージョンが使用されます。

AWS SAM CLI ナイトリービルドのインストール

AWS SAM CLI ナイトリービルドをインストールするには、以下の手順に従ってください。

Linux

パッケージインストーラーを使用して、Linux x86_64 プラットフォームの AWS SAM CLI ナイトリービルドバージョンをインストールできます。

AWS SAM CLI ナイトリービルドをインストールするには

1. aws-sam-cli GitHub リポジトリ内の sam-cli-nightly から、パッケージインストーラーをダウンロードします。
2. AWS SAM CLI をインストールするの手順に従い、ナイトリービルドパッケージをインストールします。

macOS

ナイトリービルドパッケージインストーラーを使用して、macOS の AWS SAM CLI ナイトリービルドバージョンをインストールできます。

AWS SAM CLI ナイトリービルドをインストールするには

1. aws-sam-cli GitHub リポジトリ内の sam-cli-nightly から、お使いのプラットフォーム用パッケージインストーラーをダウンロードします。
2. AWS SAM CLI をインストールするの手順に従い、ナイトリービルドパッケージをインストールします。

Windows

ナイトリービルドバージョンの AWS SAM CLI は、こちらのダウンロードリンクから利用できます: AWS SAM CLI ナイトリービルド。Windows でナイトリービルドをインストールするには、「AWS SAM CLI のインストール」と同じ手順を実行しますが、代わりにナイトリービルドのダウンロードリンクを使用します。

ナイトリービルドバージョンがインストールされていることを確認するには、sam-nightly --version コマンドを実行します。このコマンドの出力は `1.X.Y.dev<YYYYMMDDHHmm>` 形式になります。以下はその例です。

```
SAM CLI, version 1.20.0.dev202103151200
```

Homebrew からパッケージインストーラーに移行する

Homebrew で AWS SAM CLI ナイトリービルドのインストールとアップグレードを行っていて、パッケージインストーラーの使用に移行する場合は、この手順に従ってください。

Homebrew からパッケージインストーラーに移行するには

1. Homebrew でインストールした AWS SAM CLI ナイトリービルドをアンインストールします。

   ```
   $ brew uninstall aws-sam-cli-nightly
   ```

2. 以下を実行して AWS SAM CLI ナイトリービルドがアンインストールされていることを確認します。

   ```
   $ sam-nightly --version
   zsh: command not found: sam-nightly
   ```

3. 前のセクションの手順に従って、AWS SAM CLI ナイトリービルドをインストールします。

pip を使用した仮想環境への AWS SAM CLI のインストール

AWS SAM CLI をインストールするには、ネイティブパッケージインストーラーを使用することをお勧めします。pip を使用する必要がある場合は、AWS SAM CLI を仮想環境にインストールすることをお勧めします。これにより、クリーンなインストール環境と、隔離された環境 (エラーが発生した場合) を利用できます。

> **Note**
>
> 2023 年 10 月 24 日をもって、AWS SAM CLI は Python 3.7 のサポートを終了します。詳細については、「AWS SAM CLI による Python 3.7 のサポートの終了」を参照してください。

仮想環境に AWS SAM CLI をインストールするには

1. 任意の開始ディレクトリから仮想環境を作成し、名前を付けます。

   Linux / macOS

   ```
   $ mkdir project
   $ cd project
   $ python3 -m venv venv
   ```

   Windows

   ```
   > mkdir project
   > cd project
   > py -3 -m venv venv
   ```

2. 仮想環境をアクティブ化する

   Linux / macOS

   ```
   $ . venv/bin/activate
   ```

   プロンプトが変わり、仮想環境がアクティブであることが示されます。

   ```
   (venv) $
   ```

Windows

```
> venv\Scripts\activate
```

プロンプトが変わり、仮想環境がアクティブであることが示されます。

```
(venv) >
```

3. 仮想環境に AWS SAM CLI をインストールします。

```
(venv) $ pip install --upgrade aws-sam-cli
```

4. AWS SAM CLI が正しくインストールされたことを確認します。

```
(venv) $ sam --version
SAM CLI, version 1.94.0
```

5. `deactivate` コマンドを使用して、仮想環境を終了できます。新しいセッションを開始するたびに、環境を再度アクティブ化する必要があります。

Homebrew で AWS SAM CLI を管理する

> **Note**
>
> 2023 年 9 月以降、AWS では AWS で管理される AWS SAM CLI (`aws/tap/aws-sam-cli`) 用 Homebrew インストーラーのメンテナンスを行いません。Homebrew を引き続き使用するには、コミュニティ管理のインストーラーを使用します (`aws-sam-cli`)。2023 年 9 月以降、`aws/tap/aws-sam-cli` を参照する Homebrew は `aws-sam-cli` にリダイレクトします。
> 私たちがサポートしているインストールとアップグレードの使用をおすすめします。

Homebrew を使用した AWS SAM CLI のインストール

> **Note**
>
> このインストラクションでは、コミュニティが管理する AWS SAM CLI Homebrew インストーラーを使用します。より詳細なサポートについては、homebrew-core リポジトリをご覧ください。

AWS SAM CLI をインストールするには

1. 下記を実行します。

```
$ brew install aws-sam-cli
```

2. インストールを確認します。

```
$ sam --version
```

AWS SAM CLI が正常にインストールされると、以下のような出力が表示されます。

```
SAM CLI, version 1.94.0
```

Homebrew を使用した AWS SAM CLI のアップグレード

Homebrew を使用して AWS SAM CLI をアップグレードするには、以下のコマンドを実行します。

```
$ brew upgrade aws-sam-cli
```

Homebrew でインストールした AWS SAM CLI をアンインストールする

AWS SAM CLI を Homebrew でインストールしていた場合は、次の手順に従ってアンインストールします。

AWS SAM CLI をアンインストールするには

1. 下記を実行します。

```
$ brew uninstall aws-sam-cli
```

2. 以下を実行して AWS SAM CLI がアンインストールされていることを確認します。

```
$ sam --version
command not found: sam
```

コミュニティ管理の Homebrew インストーラーに移行する

AWS 管理の Homebrew インストーラー (aws/tap/aws-sam-cli) を使用していて、Homebrew を引き続き使用する場合は、コミュニティ管理の Homebrew インストーラー (aws-sam-cli) への移行をおすすめします。

1 つのコマンドで切り替えるには、以下を実行します。

```
$ brew uninstall aws-sam-cli && brew untap aws/tap && brew cleanup aws/tap && brew
 update && brew install aws-sam-cli
```

以下の手順に従って、各コマンドを個別に実行してください。

コミュニティ管理の Homebrew インストーラーに移行するには

1. AWS 管理の AWS SAM CLI Homebrew バージョンをアンインストールします:

```
$ brew uninstall aws-sam-cli
```

2. AWS SAM CLI がアンインストールされたことを確認します。

```
$ which sam
sam not found
```

3. AWS 管理の AWS SAM CLI を削除します。以下をタップします:

```
$ brew untap aws/tap
```

次のようなエラーが表示された場合は、--force オプションを選択し、もう一度試してください。

```
Error: Refusing to untap aws/tap because it contains the following installed
 formulae or casks:
aws-sam-cli-nightly
```

4. AWS 管理のインストーラーのキャッシュファイルを削除します:

```
$ brew cleanup aws/tap
```

5. Homebrew およびすべてのフォーミュラを更新します:

```
$ brew update
```

6. コミュニティ管理の AWS SAM CLI をインストールします:

```
$ brew install aws-sam-cli
```

7. AWS SAM CLI のインストールに成功したことを確認します。

```
$ sam --version
SAM CLI, version 1.94.0
```

トラブルシューティング

AWS SAM CLI のインストールまたは使用中にエラーが発生した場合は、「AWS SAMCLI トラブルシューティング」を参照してください。

AWS 認証情報のセットアップ

AWS SAM コマンドラインインターフェイス (CLI) では、CLI がユーザーに代わって AWS のサービスを呼び出すことができるように、AWS 認証情報を設定する必要があります。例えば、AWS SAM CLI は Amazon S3 と AWS CloudFormation を呼び出します。

AWS のツール (AWS SDK の 1 つ、または AWS CLI など) での作業のために、AWS 認証情報がすでに設定されている場合もありますが、まだ設定されていない場合は、このトピックが AWS 認証情報を設定するために推奨されるアプローチを説明します。

AWS 認証情報を設定するには、設定する IAM ユーザーのアクセスキー ID とシークレットアクセスキーが必要です。アクセスキー ID とシークレットアクセスキーについては、IAM ユーザーガイドの「IAM ユーザーのアクセスキーの管理」を参照してください。

次に、AWS CLI がインストール済みかどうかを判断します。その後、以下のいずれかのセクションに記載されている手順を実行します。

AWS CLI の使用

AWS CLI がインストール済みの場合は、`aws configure` コマンドを実行して、プロンプトに従います。

```
$ aws configure
AWS Access Key ID [None]: your_access_key_id
AWS Secret Access Key [None]: your_secret_access_key
Default region name [None]:
Default output format [None]:
```

`aws configure` コマンドの詳細については、AWS Command Line Interface ユーザーガイドで AWS CLI の素早い設定を参照してください。

AWS CLI を使用しない場合

AWS CLI がインストールされていない場合は、認証情報ファイルを作成する、または環境変数を設定することができます。

- 認証情報ファイル - ローカルシステムの AWS 認証情報ファイルに認証情報を設定できます。このファイルは、以下の場所のいずれかにある必要があります。
  - Linux または macOS の `~/.aws/credentials`
  - Windows の `C:\Users\USERNAME\.aws\credentials`

  このファイルには以下の形式の行が含まれている必要があります。

  ```
  [default]
  aws_access_key_id = your_access_key_id
  aws_secret_access_key = your_secret_access_key
  ```

- 環境変数 - `AWS_ACCESS_KEY_ID` と `AWS_SECRET_ACCESS_KEY` の環境変数を設定できます。

  Linux または macOS でこれらの変数を設定するには、export コマンドを使用します。

  ```
  export AWS_ACCESS_KEY_ID=your_access_key_id
  export AWS_SECRET_ACCESS_KEY=your_secret_access_key
  ```

  Windows でこれらの変数を設定するには、set コマンドを使用します。

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

AWS SAM CLI でのテレメトリ

AWS では、お客様とのやりとりから学んだ情報に基づいてサービスを開発し、提供を開始します。お客様からのフィードバックを使用して、製品のイテレーションを行います。テレメトリは、お客様のニーズをより深く理解し、問題を診断し、カスタマーエクスペリエンスを向上させる機能を提供するのに役立つ追加情報です。

AWS SAM コマンド ラインインターフェイス (CLI) は、一般的な使用状況メトリクス、システムおよび環境の情報、エラーなどのテレメトリを収集します。収集されるテレメトリのタイプの詳細については、「収集される情報のタイプ」を参照してください。

AWS SAM CLI は、ユーザー名や E メールアドレスなどの個人情報は収集しません。また、プロジェクトレベルの機密情報も抽出しません。

お客様は、テレメトリを有効にするかどうかを制御し、いつでも設定を変更できます。テレメトリが有効化されたままの場合、AWS SAM CLI はテレメトリデータをバックグラウンドで送信します。この際、お客様とのやり取りは不要です。

セッションのテレメトリの無効化

macOS および Linux オペレーティングシステムでは、単一セッションのテレメトリを無効にできます。現在のセッションのテレメトリを無効にするには、以下のコマンドを実行して環境変数 `SAM_CLI_TELEMETRY` を `false` に設定します。新しいターミナルまたはセッションに対して、このコマンドを繰り返します。

```
export SAM_CLI_TELEMETRY=0
```

すべてのセッションでのプロファイルのテレメトリの無効化

オペレーティングシステムで AWS SAM CLI を実行している場合は、以下のコマンドを実行して全セッションのテレメトリを無効にします。

Linux でテレメトリを無効にするには

1. 以下を実行します:

```
echo "export SAM_CLI_TELEMETRY=0" >>~/.profile
```

2. 以下を実行します:

```
source ~/.profile
```

macOS でテレメトリを無効にするには

1. 以下を実行します:

```
echo "export SAM_CLI_TELEMETRY=0" >>~/.profile
```

2. 以下を実行します:

```
source ~/.profile
```

Windows でテレメトリを無効にするには

次のコマンドを使用して、ターミナルウィンドウの存続期間中、一時的に環境変数を設定できます。

コマンドプロンプトを使用する場合:

```
set SAM_CLI_TELEMETRY=0
```

PowerShell を使用する場合:

```
$env:SAM_CLI_TELEMETRY=0
```

コマンドプロンプトまたは PowerShell のいずれかで環境変数を永続的に設定するには、次のコマンドを使用します。

```
setx SAM_CLI_TELEMETRY 0
```

> **Note**
>
> ターミナルを閉じて再度開くまでは、変更内容は有効になりません。

収集される情報のタイプ

- 使用状況の情報 - お客様が実行する汎用コマンドとサブコマンド。
- エラーと診断情報 - 終了コード、内部例外名、Docker への接続時の失敗など、お客様が実行するコマンドのステータスと継続時間。
- システムと環境情報 - Python のバージョン、オペレーティングシステム (Windows、Linux、または macOS)、AWS SAM CLI が実行する環境 (AWS CodeBuild、AWS IDE ツールキット、ターミナルなど)、および使用属性のハッシュ値。

詳細はこちら

AWS SAM CLI で収集されるテレメトリデータは、AWS データプライバシーポリシーに準拠します。詳細については、次を参照してください:

- AWS のサービスの使用条件
- データプライバシーに関するよくある質問

AWS SAMCLI トラブルシューティング

このセクションでは、AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) を使用、インストール、管理する際に、エラーメッセージをトラブルシューティングする方法の詳細を提供します。

トピック

- トラブルシューティング
- エラーメッセージ
- 警告メッセージ

トラブルシューティング

AWS SAM CLI 関連のトラブルシューティングのガイダンスについては、「インストールエラーのトラブルシューティング」を参照してください。

エラーメッセージ

Curl エラー: 「curl: (6) Could not resolve: ...」

API Gateway エンドポイントを呼び出そうとしているときに、以下のエラーが表示されます。

```
curl: (6) Could not resolve: endpointdomain (Domain name not found)
```

これは、無効なドメインにリクエストを送信しようとしたことを意味します。このエラーは、サーバーレスアプリケーションが正常にデプロイされなかった場合、または curl コマンドに入力ミスがある場合に発生します。AWS CloudFormation コンソールまたは AWS CLI を使用して、アプリケーションが正常にデプロイされたこと、および curl コマンドが正しいことを確認します。

エラー: 指定されたスタック名で正確なリソース情報を見つけることができません

単一の Lambda 関数リソースが含まれるアプリケーションで `sam remote invoke` コマンドを実行するときに、以下のエラーが表示されます。

```
Error: Can't find exact resource information with given <stack-name>. Please provide
 full resource ARN or --stack-name to resolve the ambiguity.
```

考えられる原因: **`--stack-name`** オプションが指定されていない。

関数 ARN が引数として指定されていない場合、`sam remote invoke` コマンドには `--stack-name` オプションを指定する必要があります。

解決策: **`--stack-name`** オプションを指定する。

以下に例を示します。

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

Invoking Lambda Function HelloWorldFunction



START RequestId: 40593abb-e1ad-4d99-87bd-ac032e364e82 Version: $LATEST
END RequestId: 40593abb-e1ad-4d99-87bd-ac032e364e82
REPORT RequestId: 40593abb-e1ad-4d99-87bd-ac032e364e82  Duration: 11.31 ms
  Billed Duration: 12 ms  Memory Size: 128 MB     Max Memory Used: 67 MB  Init
 Duration: 171.71 ms
```

```
{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

エラー: スタック名からリソース情報を見つけることができません

`sam remote invoke` コマンドを実行して Lambda 関数 ARN を引数として渡すときに、以下のエラーが表示されます。

```
Error: Can't find resource information from stack name (<stack-name>) and resource id
 (<function-id>)
```

考えられる原因: **`samconfig.toml`** ファイルにスタック名値が定義されている。

AWS SAM CLI はまず、スタック名について `samconfig.toml` ファイルをチェックします。指定されている場合、引数は論理 ID 値として渡されます。

解決策: その代わりに関数の論理 ID を渡します。

関数の ARN の代わりに、関数の論理 ID を引数として渡すことができます。

解決方法: 設定ファイルからスタック名値を削除します。

設定ファイルからスタック名値を削除することができます。そうすることで、AWS SAM CLI が関数 ARN を論理 ID 値として渡さないようにします。

設定ファイルを変更した後で `sam build` を実行します。

エラー:「マネージドリソースの作成に失敗しました。認証情報が見つかりません」

sam deploy コマンドの実行時に、以下のエラーが表示されます。

```
Error: Failed to create managed resources: Unable to locate credentials
```

これは、AWS 認証情報をセットアップして、AWS SAM CLI が AWS のサービスの呼び出せるようにしていないことを意味します。これを修正するには、AWS 認証情報をセットアップする必要があります。詳細については、「AWS 認証情報のセットアップ」を参照してください。

エラー: Windows の FileNotFoundError

Windows 上の AWS SAM CLI でコマンドを実行する場合に、以下のエラーが表示されることがあります。

```
Error: FileNotFoundError
```

考えられる原因: AWS SAM CLI が Windows の最大パス制限を超えるファイルパスを操作している可能性があります。

解決策: この問題を解決するには、新しい長いパスの動作を有効にする必要があります。これを行うには、「Windows アプリの開発に関するドキュメント」の「Windows 10、バージョン 1607 以降で長いパスを有効にする」を参照してください。

エラー: pip の依存関係リゾルバー...

エラーの例:

```
ERROR: pip's dependency resolver does not currently take into account all the packages
 that are installed. This behaviour is the source of the following dependency
 conflicts.
aws-sam-cli 1.58.0 requires aws-sam-translator==1.51.0, but you have aws-sam-translator
 1.58.0 which is incompatible.
aws-sam-cli 1.58.0 requires typing-extensions==3.10.0.0, but you have typing-extensions
 4.4.0 which is incompatible.
```

考えられる原因: pip を使用してパッケージをインストールする場合、パッケージ間の依存関係が競合する可能性があります。

aws-sam-cli パッケージの各バージョンは、aws-sam-translator パッケージのバージョンに依存します。例えば、aws-sam-cli v1.58.0 は aws-sam-translator v1.51.0 に依存する可能性があります。

pip を使用して AWS SAM CLI をインストールし、その後、aws-sam-translator のより新しいバージョンに依存する別のパッケージをインストールすると、次の事象が発生します。

- aws-sam-translator のより新しいバージョンがインストールされます。
- aws-sam-cli の現在のバージョンと aws-sam-translator のより新しいバージョンには互換性がない可能性があります。
- AWS SAM CLI を使用すると、依存関係リゾルバーエラーが発生します。

解決方法

1. AWS SAM CLI ネイティブパッケージインストーラーを使用します。

a. pip を使用して AWS SAM CLI をアンインストールする 手順については、AWS SAM CLI のアンインストール を参照してください。
b. ネイティブパッケージインストーラーを使用して AWS SAM CLI をインストールします。手順については、AWS SAM CLI のインストール を参照してください。
c. 必要に応じて、ネイティブパッケージインストーラーを使用して AWS SAM CLI をアップグレードします。手順については、AWS SAM CLI のアップグレード を参照してください。

2. pip を使用する必要がある場合は、AWS SAM CLI を仮想環境にインストールすることをお勧めします。これにより、クリーンなインストール環境と、隔離された環境 (エラーが発生した場合) を利用できます。手順については、pip を使用した仮想環境への AWS SAM CLI のインストール を参照してください。

エラー: 「remote」というコマンドはありません

`sam remote invoke` コマンドの実行時に、以下のエラーが表示されます。

```
$ sam remote invoke ...
2023-06-20 08:15:07 Command remote not available
Usage: sam [OPTIONS] COMMAND [ARGS]...
Try 'sam -h' for help.

Error: No such command 'remote'.
```

考えられる原因: AWS SAM CLI のバージョンが古くなっている。

AWS SAM CLI `sam remote invoke` コマンドは、AWS SAM CLI バージョン 1.88.0 でリリースされたものです。バージョンは、`sam --version` コマンドを実行することで確認できます。

解決策: AWS SAM CLI を最新バージョンにアップグレードする。

手順については、AWS SAM CLI のアップグレード を参照してください。

エラー: AWS SAM プロジェクトをローカルで実行するには Docker が必要です。インストールしましたか?

sam local start-api コマンドの実行時に、以下のエラーが表示されます。

```
Error: Running AWS SAM projects locally requires Docker. Have you got it installed?
```

これは、Docker が正しくインストールされていないことを意味します。アプリケーションをローカルでテストするには Docker が必要です。これを修正するには、開発ホスト用の Docker をインストールする手順を実行します。詳細については、「Docker のインストール」を参照してください。

エラー:「セキュリティの制約に準拠していません」

sam deploy --guided の実行時に、*Function* may not have authorization defined, Is this okay? [y/N] という質問のプロンプトが表示されます。このプロンプトに「**N**」(デフォルトのレスポンス) と答えた場合、以下のエラーが表示されます。

```
Error: Security Constraints Not Satisfied
```

このプロンプトは、デプロイしようとしているアプリケーションに、認可なしで設定された、パブリックにアクセス可能な Amazon API Gateway API が存在する可能性があることを知らせています。このプロンプトに「**N**」と答えることによって、この状態は望ましくないと伝えることになります。

この問題を解決するには、以下のオプションがあります。

- 認可を使用してアプリケーションを設定する。認可の設定については、「AWS SAM テンプレートを使用して API アクセスを制御する」を参照してください。
- 認可なしで API エンドポイントにパブリックにアクセスできるようにすることが目的の場合は、デプロイを再起動し、デプロイすることに問題がない旨の意思表示をするために、この質問に **Y** で応答します。

メッセージ: 認証トークンがありません

API Gateway エンドポイントを呼び出そうとしているときに、以下のエラーが表示されます。

```
{"message":"Missing Authentication Token"}
```

これは、正しいドメインにリクエストを送信しようとしたものの、URI を認識できないことを意味します。この問題を解決するには、完全な URL を確認し、正しい URL で curl コマンドを更新します。

警告メッセージ

警告: AWS では AWS SAM 用Homebrew インストーラーのメンテナンスは行われなくなりました...

Homebrew で AWS SAM CLI をインストールする際、以下の警告メッセージが表示されます。

```
Warning: ... AWS will no longer maintain the Homebrew installer for AWS SAM (aws/tap/
aws-sam-cli).
    For AWS supported installations, use the first party installers ...
```

考えられる原因: AWS が Homebrew サポートのメンテナンスを終了した。

2023 年 9 月以降、AWS では AWS SAM CLI 用 Homebrew インストーラーのメンテナンスを行いません。

解決策: AWS でサポートされているインストール方法。

- AWS でサポートされているインストール方法は、AWS SAM CLI のインストール に記載してあります。

解決策: Homebrew を引き続き使用するには、コミュニティ管理のインストーラーを使用します。

- ご自身の判断で、コミュニティ管理の Homebrew インストーラーを使用してください。手順については、Homebrew で AWS SAM CLI を管理する を参照してください。

AWS SAM コネクタリファレンス

このセクションには、AWS Serverless Application Model (AWS SAM) コネクタリソースタイプに関するリファレンス情報が含まれています。コネクタの概要については、「AWS SAM コネクタによるリソースに対するアクセス許可の管理」を参照してください。

コネクタに対してサポートされている送信元リソースと送信先リソースのタイプ

`AWS::Serverless::Connector` リソースタイプは、送信元リソースと送信先リソース間で選択された数の接続をサポートします。AWS SAM テンプレートでコネクタを設定するときは、次の表を使用して、サポートされている接続と、ソースと宛先のリソースタイプごとに定義する必要のあるプロパティを参照してください。テンプレートでのコネクタの設定の詳細については、「AWS::Serverless::Connector」を参照してください。

ソースリソースと送信先リソースの両方について、同じテンプレート内で定義されている場合は、`Id` プロパティを使用します。オプションで、`Qualifier` を追加して、定義したリソースの範

囲を絞り込むことができます。リソースが同じテンプレート内にない場合は、サポートされているプロパティの組み合わせを使用してください。

新しい接続を要求するには、GitHub の serverless-application-model AWS リポジトリで新しい問題を送信してください。

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::ApiGateway::RestApi	AWS::Lambda::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type
AWS::ApiGateway::RestApi	AWS::Serverless::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type
AWS::ApiGatewayV2::Api	AWS::Lambda::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type
AWS::ApiGatewayV2::Api	AWS::Serverless::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type
AWS::AppSync::DataSource	AWS::DynamoDB::Table	Read	Id または RoleName および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::AppSync::DataSource	AWS::DynamoDB::Table	Write	Id または RoleName および Type	Id または Arn および Type
AWS::AppSync::DataSource	AWS::Events::EventBus	Write	Id または RoleName および Type	Id または Arn および Type
AWS::AppSync::DataSource	AWS::Lambda::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::AppSync::DataSource	AWS::Serverless::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::AppSync::DataSource	AWS::Serverless::SimpleTable	Read	Id または RoleName および Type	Id または Arn および Type
AWS::AppSync::DataSource	AWS::Serverless::SimpleTable	Write	Id または RoleName および Type	Id または Arn および Type
AWS::AppSync::GraphQLApi	AWS::Lambda::Function	Write	Id または ResourceId および Type	Id または Arn および Type
AWS::AppSync::GraphQLApi	AWS::Serverless::Function	Write	Id または ResourceId および Type	Id または Arn および Type
AWS::DynamoDB::Table	AWS::Lambda::Function	Read	Id または Arn および Type	Id または RoleName および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::DynamoDB::Table	AWS::Serverless::Function	Read	Id または Arn および Type	Id または RoleName および Type
AWS::Events::Rule	AWS::Events::EventBus	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Events::Rule	AWS::Lambda::Function	Write	Id または Arn および Type	Id または Arn および Type
AWS::Events::Rule	AWS::Serverless::Function	Write	Id または Arn および Type	Id または Arn および Type
AWS::Events::Rule	AWS::Serverless::StateMachine	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Events::Rule	AWS::SNS::Topic	Write	Id または Arn および Type	Id または Arn および Type
AWS::Events::Rule	AWS::SQS::Queue	Write	Id または Arn および Type	Id または Arn、QueueUrl、および Type
AWS::Events::Rule	AWS::StepFunctions::StateMachine	Write	Id または RoleName および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::Lambda::Function	AWS::DynamoDB::Table	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::Events::EventBus	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::Lambda::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::Location::PlaceIndex	Read	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::S3::Bucket	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::Serverless::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::Serverless::SimpleTable	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::Serverless::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::Lambda::Function	AWS::SNS::Topic	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::SQS::Queue	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Lambda::Function	AWS::StepFunctions::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type
AWS::S3::Bucket	AWS::Lambda::Function	Write	Id または Arn および Type	Id または Arn および Type
AWS::S3::Bucket	AWS::Serverless::Function	Write	Id または Arn および Type	Id または Arn および Type
AWS::Serverless::Api	AWS::Lambda::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type
AWS::Serverless::Api	AWS::Serverless::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::Serverless::Function	AWS::DynamoDB::Table	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::Function	AWS::Events::EventBus	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::Function	AWS::Lambda::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::Function	AWS::S3::Bucket	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::Function	AWS::Serverless::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::Function	AWS::Serverless::SimpleTable	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::Function	AWS::Serverless::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type
AWS::Serverless::Function	AWS::SNS::Topic	Write	Id または RoleName および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::Serverless::Function	AWS::SQS::Queue	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::Function	AWS::StepFunctions::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type
AWS::Serverless::HttpApi	AWS::Lambda::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type
AWS::Serverless::HttpApi	AWS::Serverless::Function	Write	Id または Qualifier 、ResourceId 、および Type	Id または Arn および Type
AWS::Serverless::SimpleTable	AWS::Lambda::Function	Read	Id または Arn および Type	Id または RoleName および Type
AWS::Serverless::SimpleTable	AWS::Serverless::Function	Read	Id または Arn および Type	Id または RoleName および Type
AWS::Serverless::StateMachine	AWS::DynamoDB::Table	Read, Write	Id または RoleName および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::Serverless::StateMachine	AWS::Events::EventBus	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::StateMachine	AWS::Lambda::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::StateMachine	AWS::S3::Bucket	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::StateMachine	AWS::Serverless::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::StateMachine	AWS::Serverless::SimpleTable	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::StateMachine	AWS::Serverless::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type
AWS::Serverless::StateMachine	AWS::SNS::Topic	Write	Id または RoleName および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::Serverless::StateMachine	AWS::SQS::Queue	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Serverless::StateMachine	AWS::StepFunctions::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type
AWS::SNS::Topic	AWS::Lambda::Function	Write	Id または Arn および Type	Id または Arn および Type
AWS::SNS::Topic	AWS::Serverless::Function	Write	Id または Arn および Type	Id または Arn および Type
AWS::SNS::Topic	AWS::SQS::Queue	Write	Id または Arn および Type	Id または Arn、QueueUrl、および Type
AWS::SQS::Queue	AWS::Lambda::Function	Read, Write	Id または Arn および Type	Id または RoleName および Type
AWS::SQS::Queue	AWS::Serverless::Function	Read, Write	Id または Arn および Type	Id または RoleName および Type
AWS::StepFunctions::StateMachine	AWS::DynamoDB::Table	Read, Write	Id または RoleName および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::StepFunctions::StateMachine	AWS::Events::EventBus	Write	Id または RoleName および Type	Id または Arn および Type
AWS::StepFunctions::StateMachine	AWS::Lambda::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::StepFunctions::StateMachine	AWS::S3::Bucket	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::StepFunctions::StateMachine	AWS::Serverless::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::StepFunctions::StateMachine	AWS::Serverless::SimpleTable	Read, Write	Id または RoleName および Type	Id または Arn および Type
AWS::StepFunctions::StateMachine	AWS::Serverless::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type
AWS::StepFunctions::StateMachine	AWS::SNS::Topic	Write	Id または RoleName および Type	Id または Arn および Type

ソースタイプ	送信先タイプ	アクセス許可	ソースプロパティ	送信先プロパティ
AWS::StepFunctions::StateMachine	AWS::SQS::Queue	Write	Id または RoleName および Type	Id または Arn および Type
AWS::StepFunctions::StateMachine	AWS::StepFunctions::StateMachine	Read, Write	Id または RoleName および Type	Id または Arn、Name、および Type

コネクタによって作成された IAM ポリシー

このセクションでは、コネクタの使用時に AWS SAM によって作成される AWS Identity and Access Management (IAM) ポリシーについて説明します。

AWS::DynamoDB::Table ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:ListStreams"
      ],
      "Resource": [
        "%{Source.Arn}/stream/*"
```

```
      ]
    }
  ]
}
```

AWS::Events::Rule ~ AWS::SNS::Topic

ポリシータイプ

AWS::SNS::Topic にアタッチされた AWS::SNS::TopicPolicy。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Resource": "%{Destination.Arn}",
      "Action": "sns:Publish",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "%{Source.Arn}"
        }
      }
    }
  ]
}
```

AWS::Events::Rule ~ AWS::Events::EventBus

ポリシータイプ

AWS::Events::Rule ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Events::Rule ~ AWS::StepFunctions::StateMachine

ポリシータイプ

AWS::Events::Rule ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Events::Rule ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function にアタッチされた AWS::Lambda::Permission。

アクセスカテゴリ

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "events.amazonaws.com",
  "SourceArn": "%{Source.Arn}"
}
```

AWS::Events::Rule ~ AWS::SQS::Queue

ポリシータイプ

AWS::SQS::Queue にアタッチされた AWS::SQS::QueuePolicy。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Resource": "%{Destination.Arn}",
      "Action": "sqs:SendMessage",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "%{Source.Arn}"
        }
      }
    }
  ]
}
```

AWS::Lambda::Function ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeAsync",
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Lambda::Function ~ AWS::S3::Bucket

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectAcl",
        "s3:GetObjectLegalHold",
        "s3:GetObjectRetention",
        "s3:GetObjectTorrent",
        "s3:GetObjectVersion",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionForReplication",
        "s3:GetObjectVersionTorrent",
```

```
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion",
        "s3:PutObject",
        "s3:PutObjectLegalHold",
        "s3:PutObjectRetention",
        "s3:RestoreObject"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
  ]
}
```

AWS::Lambda::Function ~ AWS::DynamoDB::Table

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:BatchGetItem",
        "dynamodb:ConditionCheckItem",
        "dynamodb:PartiQLSelect"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:PartiQLDelete",
        "dynamodb:PartiQLInsert",
        "dynamodb:PartiQLUpdate"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
```

```
}
```

AWS::Lambda::Function ~ AWS::SQS::Queue

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:GetQueueAttributes"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:DeleteMessage",
        "sqs:SendMessage",
        "sqs:ChangeMessageVisibility",
        "sqs:PurgeQueue"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
```

```
  ]
}
```

AWS::Lambda::Function ~ AWS::SNS::Topic

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Lambda::Function ~ AWS::StepFunctions::StateMachine

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution",
```

```
        "states:StartSyncExecution"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:StopExecution"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
      ]
    }
  ]
}
```

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeStateMachine",
        "states:ListExecutions"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:DescribeStateMachineForExecution",
        "states:GetExecutionHistory"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
```

```
      ]
    }
  ]
}
```

AWS::Lambda::Function ~ AWS::Events::EventBus

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Lambda::Function ~ AWS::Location::PlaceIndex

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "geo:DescribePlaceIndex",
        "geo:GetPlace",
        "geo:SearchPlaceIndexForPosition",
        "geo:SearchPlaceIndexForSuggestions",
        "geo:SearchPlaceIndexForText"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::ApiGatewayV2::Api ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function にアタッチされた AWS::Lambda::Permission。

アクセスカテゴリ

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "apigateway.amazonaws.com",
  "SourceArn": "arn:${AWS::Partition}:execute-api:${AWS::Region}:${AWS::AccountId}:
%{Source.ResourceId}/%{Source.Qualifier}"
}
```

AWS::ApiGateway::RestApi ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function にアタッチされた AWS::Lambda::Permission。

アクセスカテゴリ

Write

```
{
```

```
  "Action": "lambda:InvokeFunction",
  "Principal": "apigateway.amazonaws.com",
  "SourceArn": "arn:${AWS::Partition}:execute-api:${AWS::Region}:${AWS::AccountId}:
%{Source.ResourceId}/%{Source.Qualifier}"
}
```

AWS::SNS::Topic ~ AWS::SQS::Queue

ポリシータイプ

AWS::SQS::Queue にアタッチされた AWS::SQS::QueuePolicy。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Resource": "%{Destination.Arn}",
      "Action": "sqs:SendMessage",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "%{Source.Arn}"
        }
      }
    }
  ]
}
```

AWS::SNS::Topic ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function にアタッチされた AWS::Lambda::Permission。

アクセスカテゴリ

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "sns.amazonaws.com",
  "SourceArn": "%{Source.Arn}"
}
```

AWS::SQS::Queue ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:DeleteMessage"
      ],
      "Resource": [
        "%{Source.Arn}"
      ]
    }
  ]
}
```

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:GetQueueAttributes"
      ],
      "Resource": [
        "%{Source.Arn}"
```

```
      ]
    }
  ]
}
```

AWS::S3::Bucket ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function にアタッチされた AWS::Lambda::Permission。

アクセスカテゴリ

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "s3.amazonaws.com",
  "SourceArn": "%{Source.Arn}",
  "SourceAccount": "${AWS::AccountId}"
}
```

AWS::StepFunctions::StateMachine ~ AWS::Lambda::Function

ポリシータイプ

AWS::StepFunctions::StateMachine ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeAsync",
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "%{Destination.Arn}"
```

```
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine ~ AWS::SNS::Topic

ポリシータイプ

AWS::StepFunctions::StateMachine ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine ~ AWS::SQS::Queue

ポリシータイプ

AWS::StepFunctions::StateMachine ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "sqs:SendMessage"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine ~ AWS::S3::Bucket

ポリシータイプ

AWS::StepFunctions::StateMachine ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectAcl",
        "s3:GetObjectLegalHold",
        "s3:GetObjectRetention",
        "s3:GetObjectTorrent",
        "s3:GetObjectVersion",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionForReplication",
        "s3:GetObjectVersionTorrent",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
```

```
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion",
        "s3:PutObject",
        "s3:PutObjectLegalHold",
        "s3:PutObjectRetention",
        "s3:RestoreObject"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine ~ AWS::DynamoDB::Table

ポリシータイプ

AWS::StepFunctions::StateMachine ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:Query",
```

```
        "dynamodb:Scan",
        "dynamodb:BatchGetItem",
        "dynamodb:ConditionCheckItem",
        "dynamodb:PartiQLSelect"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:PartiQLDelete",
        "dynamodb:PartiQLInsert",
        "dynamodb:PartiQLUpdate"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine ~ AWS::StepFunctions::StateMachine

ポリシータイプ

AWS::StepFunctions::StateMachine ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:${AWS::Partition}:events:${AWS::Region}:${AWS::AccountId}:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "states:StopExecution"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule"
      ],
      "Resource": [
        "arn:${AWS::Partition}:events:${AWS::Region}:${AWS::AccountId}:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine ~ AWS::Events::EventBus

ポリシータイプ

AWS::StepFunctions::StateMachine ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
```

```
}
```

AWS::AppSync::DataSource ~AWS::DynamoDB::Table

ポリシータイプ

AWS::AppSync::DataSource ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:BatchGetItem",
        "dynamodb:ConditionCheckItem",
        "dynamodb:PartiQLSelect"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem",
```

```
        "dynamodb:PartiQLDelete",
        "dynamodb:PartiQLInsert",
        "dynamodb:PartiQLUpdate"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

AWS::AppSync::DataSource ~ AWS::Lambda::Function

ポリシータイプ

AWS::AppSync::DataSource ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeAsync",
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}:*"
      ]
    }
  ]
}
```

AWS::AppSync::DataSource ~ AWS::Events::EventBus

ポリシータイプ

AWS::AppSync::DataSource ロールにアタッチされたカスタマー管理ポリシー。

アクセスカテゴリ

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::AppSync::GraphQLApi ~ AWS::Lambda::Function

ポリシータイプ

AWS::Lambda::Function にアタッチされた AWS::Lambda::Permission。

アクセスカテゴリ

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "appsync.amazonaws.com",
  "SourceArn": "arn:${AWS::Partition}:appsync:${AWS::Region}:${AWS::AccountId}:apis/
%{Source.ResourceId}"
}
```

で使用する Docker のインストール AWS SAM CLI

Docker は、マシンでコンテナを実行するアプリケーションです。で Docker AWS SAM は、 のようなローカル環境をコンテナ AWS Lambda として提供して、サーバーレスアプリケーションを構築、テスト、デバッグできます。

> **Note**
>
> Docker は、アプリケーションをローカルでテストし、 `--use-container`オプションを使用してデプロイパッケージを構築する場合にのみ必要です。

トピック

インストール Docker

以下の手順に従って をインストールします。Docker オペレーティングシステムで。

Linux

Docker は、次のようなほとんどの最新の Linux ディストリビューションを含む、さまざまなオペレーティングシステムで使用できます。CentOS, Debian および Ubuntu。 のインストールの詳細については、Docker 特定のオペレーティングシステムで、Docker Docs ウェブサイトの「Get Docker」を参照してください。

をインストールするには Docker Amazon Linux 2 または Amazon Linux 2023 の場合

1. インスタンスでインストールされているパッケージとパッケージキャッシュを更新します。

   ```
   $ sudo yum update -y
   ```

2. 最新の をインストールする Docker Community Edition パッケージ。

   - Amazon Linux 2 の場合は、以下を実行します。

     ```
     $ sudo amazon-linux-extras install docker
     ```

   - Amazon Linux 2023 の場合は、以下を実行します。

     ```
     $ sudo yum install -y docker
     ```

3. を起動する Docker サービス。

```
$ sudo service docker start
```

4. docker グループec2-userへの の追加により、 を実行できます。Docker を使用しないコマンドsudo。

```
$ sudo usermod -a -G docker ec2-user
```

5. ログアウトしてから再度ログインして、新しい docker グループ許可を取得します。これを行うには、現在のSSHターミナルウィンドウを閉じて、新しいインスタンスに再接続します。新しいSSHセッションには、適切なdockerグループアクセス許可が必要です。
6. ec2-user が sudo を使用せずに Docker コマンドを実行できることを確認します。

```
$ docker ps
```

Docker がインストール済みで実行中であることを確認する、以下の出力が表示されます。

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

> **Note**
>
> Linux では、ホストマシンとは異なる命令セットアーキテクチャを使用して Lambda 関数を構築および実行するには、Docker。 例えば、x86_64マシンでarm64関数を実行するには、次のコマンドを実行して、Docker デーモン: `docker run --rm --privileged multiarch/qemu-user-static --reset -p yes`。

のインストールで問題が発生した場合 Docker「 インストールエラーのトラブルシューティング」を参照してください。または、Docker Docs ウェブサイトの Post-installation steps for Linux にある「Troubleshooting」セクションを参照してください。

macOS

> **Note**
>
> Docker デスクトップは正式にサポートされていますが、 で始まる AWS SAM CLI バージョン 1.47.0 では、 を使用する限り、代替 を使用できます。Docker ランタイム。

1. インストール Docker

   の AWS SAM CLI は をサポートします Docker macOS Sierra 10.12 以降で実行されています。のインストール方法 Docker「インストール」を参照してください。Docker 上の Desktop for Mac Docker Docs ウェブサイト。
2. 共有ドライブの構成

   の AWS SAM CLI では、プロジェクトディレクトリ、または親ディレクトリが共有ドライブにリストされている必要があります。これにはファイル共有が必要になる場合があります。詳細については、「」の「ボリュームのマウントにはファイル共有のトラブルシューティングトピックが必要」を参照してください。 Docker ドキュメント 。
3. インストールの検証

   後 Docker がインストールされています。動作していることを確認します。また、Docker コマンドラインからの コマンド (例: `docker ps`) 。コンテナをインストール、フェッチ、プルする必要はありません。 AWS SAM CLI は、必要に応じて自動的にこれを実行します。

のインストールで問題が発生した場合 Docker、トラブルシューティングのヒントの詳細については、「」の「トラブルシューティングと診断」セクションを参照してください。Docker Docs ウェブサイト。

Windows

> **Note**
>
> AWS SAM が正式にサポート Docker デスクトップ。ただし、 で始まる AWS SAM CLI バージョン 1.47.0 では、 を使用する限り、代替 を使用できます。Docker ランタイム。

1. インストール Docker.

Docker Desktop は、最新の Windows オペレーティングシステムをサポートしています。Windows のレガシーバージョンでは、Docker Toolbox を使用できます。正しい Windows のバージョンを選択する Docker インストール手順：

- をインストールするには Docker Windows 10 の場合は、「インストール」を参照してください。Docker 上の Windows 用デスクトップ Docker Docs ウェブサイト。
- をインストールするには Docker Windows の以前のバージョンについては、「」を参照してください。Docker のツールボックス Docker Toolbox GitHub リポジトリ。

2. 共有ドライブを設定します。

   の AWS SAM CLI では、プロジェクトディレクトリ、または親ディレクトリが共有ドライブにリストされている必要があります。場合によっては、 のドライブを共有する必要があります。Docker 正しく機能します。

3. インストールを確認します。

   後 Docker がインストールされています。動作していることを確認します。また、Docker コマンドラインからの コマンド (例: docker ps) 。コンテナをインストール、フェッチ、プルする必要はありません。 AWS SAM CLI は、必要に応じて自動的にこれを実行します。

のインストールで問題が発生した場合 Docker、トラブルシューティングのヒントの詳細については、「」の「トラブルシューティングと診断」セクションを参照してください。Docker Docs ウェブサイト。

次のステップ

のインストール方法 AWS SAM CLI「AWS SAM CLI のインストール」を参照してください。

のイメージリポジトリ AWS SAM

AWS SAM は、ビルドコンテナイメージを活用して、サーバーレスアプリケーションの継続的インテグレーションと継続的デリバリー (CI/CD) タスクを簡素化します。 AWS SAM が提供するイメージには、サポートされている多数の AWS Lambda ランタイム用の AWS SAM コマンドラインインターフェイス (CLI) とビルドツールが含まれています。これにより、 を使用してサーバーレスアプリケーションを簡単に構築およびパッケージ化できます。 AWS SAM CLI。 これらのイメージを CI/CD システムで使用して、 AWS SAM アプリケーションの構築とデプロイを自動化できます。例については、「CI/CD システムとパイプラインを使用したデプロイ」を参照してください。

AWS SAM ビルドコンテナイメージURIsには、 のバージョン がタグ付けされます。 AWS SAM CLI そのイメージに が含まれています。タグなしの を指定するとURI、最新バージョンが使用されます。例えば、`public.ecr.aws/sam/build-nodejs20.x` は最新のイメージを使用しますが、ただし、 はCLIバージョン 1.24.1 を含む AWS SAM イメージ`public.ecr.aws/sam/build-nodejs20.x:1.24.1`を使用します。

のバージョン 1.33.0 以降 AWS SAM CLIでは、 `x86_64`と `arm64` コンテナイメージの両方がサポートされているランタイムで使用できます。詳細については、AWS Lambda デベロッパーガイドの「Lambda ランタイム」を参照してください。

> **Note**
>
> のバージョン 1.22.0 以前 AWS SAM CLI、 は が使用するデフォルトのリポジトリ DockerHub でした AWS SAM CLI がコンテナイメージをプルしました。バージョン 1.22.0 以降、デフォルトのリポジトリは Amazon Elastic Container Registry Public (Amazon ECR Public) に変更されました。現在のデフォルト以外のリポジトリからコンテナイメージをプルするには、--build-image オプションが指定された sam build コマンドを使用できます。このトピックの最後にある例は、 DockerHub リポジトリイメージを使用してアプリケーションを構築する方法を示しています。

イメージリポジトリ URIs

次の表は、サーバーレスアプリケーションの構築とパッケージ化に使用できる Amazon ECR Public ビルドコンテナイメージURIsの一覧です AWS SAM。

> **Note**
>
> Amazon ECR Public の置き換え DockerHub で始まる AWS SAM CLI バージョン 1.22.0。の以前のバージョンを使用している場合 AWS SAM CLI、アップグレードすることをお勧めします。

ランタイム	Amazon ECR パブリック
カスタムランタイム (AL2023)	public.ecr.aws/sam/build-provided.al2023
カスタムランタイム (AL2)	public.ecr.aws/sam/build-provided.al2

ランタイム	Amazon ECR パブリック
カスタムランタイム	public.ecr.aws/sam/build-provided
Java 21	public.ecr.aws/sam/build-java21
Java 17	public.ecr.aws/sam/build-java17
Java 11	public.ecr.aws/sam/build-java11
Java 8	public.ecr.aws/sam/build-java8
。NET 8	public.ecr.aws/sam/build-dotnet8
。NET 7	public.ecr.aws/sam/build-dotnet7
。NET 6	public.ecr.aws/sam/build-dotnet6
Node.js 22	public.ecr.aws/sam/build-nodejs22.x
Node.js 20	public.ecr.aws/sam/build-nodejs20.x
Node.js 18	public.ecr.aws/sam/build-nodejs18.x
Node.js 16	public.ecr.aws/sam/build-nodejs16.x
Python 3.13	public.ecr.aws/sam/build-python3.13
Python 3.12	public.ecr.aws/sam/build-python3.12
Python 3.11	public.ecr.aws/sam/build-python3.11
Python 3.10	public.ecr.aws/sam/build-python3.10
Python 3.9	public.ecr.aws/sam/build-python3.9
Python 3.8	public.ecr.aws/sam/build-python3.8
Ruby 3.3	public.ecr.aws/sam/build-ruby3.3
Ruby 3.2	public.ecr.aws/sam/build-ruby3.2

例

次の 2 つのコマンド例では、 DockerHub リポジトリのコンテナイメージを使用してアプリケーションを構築します。

の構築 Node.js 22 Amazon からプルされたコンテナイメージを使用する アプリケーションECR：

```
$ sam build --use-container --build-image public.ecr.aws/sam/build-nodejs22.x
```

を使用して関数リソースを構築する Python 3.13 Amazon からプルされたコンテナイメージECR：

```
$ sam build --use-container --build-image Function1=public.ecr.aws/sam/build-python3.13
```

AWS SAM を使用したサーバーレスアプリケーションの段階的なデプロイ

AWS Serverless Application Model (AWS SAM) には CodeDeploy が組み込まれているため、AWS Lambda を段階的にデプロイできます。数行設定するだけで、AWS SAM が以下を実行します。

- Lambda 関数の新しいバージョンをデプロイし、新しいバージョンをポイントするエイリアスを自動的に作成する。
- 新しいバージョンが期待どおりに動作していることを確認するまで、カスタマートラフィックを新しいバージョンに段階的に移行する。更新が正しく動作しない場合は、変更をロールバックできます。
- トラフィック前およびトラフィック後のテスト関数を定義して、新しくデプロイされたコードが正しく設定されており、アプリケーションが期待どおりに動作していることを確認する。
- CloudWatch アラームがトリガーされた場合にデプロイを自動的にロールバックする。

> **Note**
>
> AWS SAM テンプレートを通じて段階的なデプロイを有効にすると、CodeDeploy リソースが自動的に作成されます。CodeDeploy リソースは、AWS Management Console で直接表示できます。

例

以下の例では、カスタマーを新しくデプロイされたバージョンの Lambda 関数に段階的に移行させるための CodeDeploy の使用を示します。

```
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs20.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live

    DeploymentPreference:
      Type: Canary10Percent10Minutes
      Alarms:
        # A list of alarms that you want to monitor
        - !Ref AliasErrorMetricGreaterThanZeroAlarm
        - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
      Hooks:
        # Validation Lambda functions that are run before & after traffic shifting
        PreTraffic: !Ref PreTrafficLambdaFunction
        PostTraffic: !Ref PostTrafficLambdaFunction
```

AWS SAM テンプレートに対するこれらの修正は、以下を実行します。

- `AutoPublishAlias`: このプロパティを追加し、エイリアス名を指定することで、AWS SAM が以下を実行します。
  - Lambda 関数の Amazon S3 URI に対する変更に基づいて、新しいコードがデプロイされていることを検出する。
  - 最新のコードでその関数の更新バージョンを作成し、発行する。
  - ユーザーが提供する名前でエイリアスを作成し (エイリアスが既に存在する場合を除く)、Lambda 関数の更新バージョンをポイントする。これを活用するには、関数の呼び出しがエイリアス修飾子を使用する必要があります。Lambda 関数のバージョニングとエイリアスになじみがない場合は、AWS Lambda 関数のバージョニングとエイリアスについて参照してください。
- `Deployment Preference Type`: 上記の例では、カスタマートラフィックの 10% が直ちに新しいバージョンに移行され、10 分後にすべてのトラフィックが新しいバージョンに移行されます。ただし、トラフィック前またはトラフィック後のテストが失敗した場合、または CloudWatch ア

ラームがトリガーされた場合は、CodeDeploy がデプロイをロールバックします。次の方法で、バージョン間のトラフィックの移行方法を指定できます。

- `Canary`: トラフィックは 2 回の増分で移行されます。事前定義された Canary オプションから選択できます。このオプションは、最初の増分で更新された Lambda 関数バージョンに移行されるトラフィックの割合と、2 番目の増分で残りのトラフィックが移行されるまでの間隔を分単位で指定します。
- `Linear`: トラフィックは、毎回同じ間隔 (分) の等しい増分で移行します。増分ごとに移行されるトラフィックの割合と、増分間の間隔 (分) を指定する事前定義された Linear オプションから選択できます。
- `AllAtOnce`: すべてのトラフィックは元の Lambda 関数から最新バージョンの Lambda 関数に一度に移行されます。

以下の表は、この例で使用したもの以外で利用可能なその他のトラフィック移行オプションの概要です。

デプロイプリファレンスのタイプ
Canary10Percent30Minutes
Canary10Percent5Minutes
Canary10Percent10Minutes
Canary10Percent15Minutes
Linear10PercentEvery10Minutes
Linear10PercentEvery1Minute
Linear10PercentEvery2Minutes
Linear10PercentEvery3Minutes
AllAtOnce

- `Alarms`: デプロイで発生したエラーによってトリガーされる CloudWatch アラームです。エラーが発生すると、デプロイは自動的にロールバックされます。例えば、デプロイしている更新されたコードがアプリケーション内にエラーを生じさせている場合です。別の例は、AWS Lambda、または指定した カスタム CloudWatch メトリクスがアラームのしきい値を超えた場合です。

- Hooks: トラフィックが新しいバージョンに移行を開始する前、および移行が完了した後でチェックを実行する、トラフィック前とトラフィック後のテスト関数です。
  - PreTraffic: トラフィックの移行を開始する前に、CodeDeploy が pre-traffic hook Lambda 関数を呼び出します。この Lambda 関数は、CodeDeploy にコールバックして成功したか失敗したかを伝える必要があります。関数が失敗すると移行が中止され、AWS CloudFormation に失敗が報告されます。関数が成功すると、CodeDeploy はトラフィックの移行に進みます。
  - PostTraffic: トラフィックの移行が完了した後で、CodeDeploy が post-traffic hook Lambda 関数を呼び出します。pre-traffic hook と同様に、この関数は CodeDeploy にコールバックして成功したか失敗したかを伝える必要があります。post-traffic hook を使用して、統合テストやその他の検証アクションを実行します。

  詳細については、「SAM Reference to Safe Deployments」を参照してください。

初めて Lambda 関数を段階的にデプロイする

Lambda 関数を段階的にデプロイする場合、CodeDeploy は、トラフィックの移行元である、以前デプロイされた関数バージョンが必要です。したがって、最初のデプロイは次の 2 つのステップで実行する必要があります。

- ステップ 1: Lambda 関数をデプロイし、AutoPublishAlias を使用してエイリアスを自動的に作成します。
- ステップ 2: DeploymentPreference を使用して段階的にデプロイします。

最初の段階的デプロイを 2 つのステップで実行すると、トラフィックの移行元である、以前の Lambda 関数バージョンが CodeDeploy に与えられます。

ステップ 1: Lambda 関数をデプロイする

```
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs20.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live
```

ステップ 2: 段階的にデプロイする

```
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs20.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live

    DeploymentPreference:
      Type: Canary10Percent10Minutes
      Alarms:
        # A list of alarms that you want to monitor
        - !Ref AliasErrorMetricGreaterThanZeroAlarm
        - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
      Hooks:
        # Validation Lambda functions that are run before and after traffic shifting
        PreTraffic: !Ref PreTrafficLambdaFunction
        PostTraffic: !Ref PostTrafficLambdaFunction
```

詳細はこちら

段階的なデプロイを設定する実践的な例については、「包括的な AWS SAM ワークショップ」の「モジュール 5 - canary デプロイ」を参照してください。

AWS SAM に関する重要なリファレンスノート

このセクションには、AWS Serverless Application Model (AWS SAM) に関する重要な注意事項と発表が含まれています。

トピック

- 2023 年の重要な注意事項

2023 年の重要な注意事項

2023 年 10 月

AWS SAM CLI による Python 3.7 のサポートの終了

公開日: 2023 年 10 月 20 日

Python 3.7 は、2023 年 6 月にサポート終了のステータスになりました。AWS SAM CLI は、2023 年 10 月 24 日に Python 3.7 のサポートを終了します。詳細については、aws-sam-cli GitHub リポジトリの「発表」を参照してください。

この変更は以下のユーザーに影響します。

- Python 3.7 を使用し、`pip` を介して AWS SAM CLI をインストールしたユーザー。
- `aws-sam-cli` をライブラリとして使用し、Python 3.7 でアプリケーションを構築するユーザー。

AWS SAM CLI を別の方法でインストールして管理している場合、影響はありません。

影響を受けるユーザーは、開発環境を Python 3.8 または最新のバージョンにアップグレードすることをお勧めします。

この変更は Python 3.7 AWS Lambda ランタイム環境のサポートには影響しません。詳細については、「AWS Lambda デベロッパーガイド」の「ランタイムの非推奨化に関するポリシー」を参照してください。

AWS SAM のサンプルサーバーレスアプリケーション

このセクションでは、DynamoDB イベントを処理するアプリケーションと Amazon S3 イベントを処理するアプリケーションの 2 つの例を示します。それぞれの例では、アプリケーションを作成する段階的なプロセスについて説明します。また、どちらにもイベントソースと AWS リソースの設定に関する詳細が含まれ、開始する前に実行する必要がある事項を特定したうえで、アプリケーションの初期化、テスト、パッケージング、デプロイの手順を実行します。

トピック

AWS SAM を使用して DynamoDB イベントを処理する

このサンプルアプリケーションでは、概要とクイックスタートガイドで学習した内容に基づいて構築を行い、別のサンプルアプリケーションをインストールします。このアプリケーションは、DynamoDB テーブルイベントソースによって呼び出される Lambda 関数で構成されます。この Lambda 関数はきわめてシンプルで、イベントソースメッセージ経由で渡されたデータをログに記録します。

この演習は、Lambda 関数が呼び出されたときに関数に渡されるイベントソースメッセージを模倣する方法を説明します。

開始する前に

AWS SAM CLI のインストール に記載されている必要なセットアップが完了していることを確認してください。

ステップ 1: アプリケーションを初期化する

このセクションでは、AWS SAM テンプレートとアプリケーションコードで構成されるアプリケーションパッケージをダウンロードします。

アプリケーションを初期化する

1. AWS SAM CLI コマンドプロンプトで以下のコマンドを実行します。

```
sam init \
--location gh:aws-samples/cookiecutter-aws-sam-dynamodb-python \
--no-input
```

上記のコマンドにある gh: は、GitHub URL https://github.com/ に展開されることに注意してください。

2. コマンドで作成されたディレクトリの内容を確認します (dynamodb_event_reader/)。

   - template.yaml - Read DynamoDB アプリケーションが必要とする 2 つの AWS リソース (Lambda 関数と DynamoDB テーブル) を定義します。このテンプレートは、2 つのリソース間のマッピングも定義します。
   - read_dynamodb_event/ ディレクトリ - DynamoDB アプリケーションコードが含まれています。

ステップ 2: アプリケーションのローカルでテストする

ローカルテストでは、AWS SAM CLI を使用してサンプル DynamoDB イベントを生成し、Lambda 関数を呼び出します。

```
sam local generate-event dynamodb update | sam local invoke --event - ReadDynamoDBEvent
```

generate-event コマンドは、すべてのコンポーネントが AWS クラウドにデプロイされたときに作成されるメッセージのようなテストイベントソースメッセージを作成します。このイベントソースメッセージは、Lambda 関数 ReadDynamoDBEvent にパイプされます。

app.py のソースコードに基づいて、期待されるメッセージがコンソールに出力されることを確認します。

ステップ 3: アプリケーションをパッケージ化する

アプリケーションをローカルでテストしたら、AWS SAM CLI を使用してデプロイパッケージを作成します。このデプロイパッケージは、アプリケーションを AWS クラウドにデプロイするために使用します。

Lambda デプロイパッケージを作成する

1. パッケージ化されたコードを保存する場所に S3 バケットを作成します。既存の S3 バケットを使用する場合は、このステップをスキップします。

```
aws s3 mb s3://bucketname
```

2. コマンドプロンプトで以下の package CLI コマンドを実行して、デプロイパッケージを作成します。

```
sam package \
    --template-file template.yaml \
    --output-template-file packaged.yaml \
    --s3-bucket bucketname
```

次のステップでアプリケーションをデプロイするときは、新しいテンプレートファイルである packaged.yaml を指定します。

ステップ 4: アプリケーションをデプロイする

デプロイパッケージを作成したところで、これを使用してアプリケーションを AWS クラウドにデプロイします。その後、アプリケーションをテストします。

サーバーレスアプリケーションを AWS クラウドにデプロイする

- AWS SAM CLI で deploy CLI コマンドを使用して、テンプレートで定義したすべてのリソースをデプロイします。

```
sam deploy \
    --template-file packaged.yaml \
    --stack-name sam-app \
    --capabilities CAPABILITY_IAM \
    --region us-east-1
```

コマンドの --capabilities パラメータは、AWS CloudFormation が IAM ロールを作成することを可能にします。

AWS CloudFormation は、テンプレートで定義されている AWS リソースを作成します。これらのリソースの名前には、AWS CloudFormation コンソールからアクセスできます。

AWS クラウドでサーバーレスアプリケーションをテストする

1. DynamoDB コンソールを開きます。
2. 先ほど作成したテーブルにレコードを挿入します。
3. テーブルの [Metrics] (メトリクス) タブに移動し、[View all CloudWatch metrics] (すべての CloudWatch メトリクスを表示) をクリックします。CloudWatch コンソールで [Logs] (ログ) を選択して、ログ出力を表示できるようにします。

次のステップ

AWS SAM GitHub リポジトリには、ダウンロードして実験するための追加のサンプルアプリケーションが含まれています。このリポジトリにアクセスするには、「AWS SAM サンプルアプリケーション」を参照してください。

AWS SAM を使用して Amazon S3 イベントを処理する

このサンプルアプリケーションでは、概要とクイックスタートガイドで学習した内容に基づいて構築を行い、より複雑なアプリケーションをインストールします。このアプリケーションは、Amazon S3 オブジェクトアップロードイベントソースによって呼び出される Lambda 関数で構成されます。この演習は、AWS リソースにアクセスし、Lambda 関数を使用して AWS のサービスの呼び出しを行う方法を説明します。

このサンプルサーバーレスアプリケーションは、Amazon S3 での object-creation イベントを処理します。Amazon S3 は、バケットにアップロードされる各画像について object-created イベントを検出し、Lambda 関数を呼び出します。この Lambda 関数は、画像内のテキストを検出するために Amazon Rekognition 呼び出します。次に、Amazon Rekognition によって返された結果を DynamoDB テーブルに保存します。

> **Note**
>
> このサンプルアプリケーションでは、これまでの例とは少し異なる順序のステップを実行します。この例では、Lambda 関数をローカルでテストする前に、AWS リソースの作成と IAM 許可の設定を行う必要があるためです。ここでは、AWS CloudFormation を利用してリソースが作成され、ユーザーの許可が設定されます。それ以外の場合は、Lambda 関数をローカルでテストする前に、これを手動で行う必要があります。
> この例はより複雑であるため、これまでのサンプルアプリケーションのインストールに慣れてから実行するようにしてください。

開始する前に

AWS SAM CLI のインストール に記載されている必要なセットアップが完了していることを確認してください。

ステップ 1: アプリケーションを初期化する

このセクションでは、AWS SAM テンプレートとアプリケーションコードで構成されるサンプルアプリケーションをダウンロードします。

アプリケーションを初期化する

1. AWS SAM CLI コマンドプロンプトで以下のコマンドを実行します。

```
sam init \
--location https://github.com/aws-samples/cookiecutter-aws-sam-s3-rekognition-
dynamodb-python \
--no-input
```

2. コマンドで作成されたディレクトリの内容を確認します (aws_sam_ocr/)。

   - template.yaml - Amazon S3 アプリケーションが必要とする、Lambda 関数、Amazon S3 バケット、および DynamoDB テーブルの 3 つの AWS リソースを定義します。テンプレートは、これらのリソース間のマッピングと許可も定義します。
   - src/ ディレクトリ - Amazon S3 アプリケーションコードが含まれています。
   - SampleEvent.json - ローカルでのテストに使用されるサンプルイベントソースです。

ステップ 2: アプリケーションをパッケージ化する

このアプリケーションをローカルでテストする前に、AWS SAM CLI を使用してデプロイパッケージを作成する必要があります。このデプロイパッケージは、アプリケーションを AWS クラウドにデプロイする際に使用します。このデプロイは、アプリケーションをローカルでテストするために必要な AWS リソースと許可を作成します。

Lambda デプロイパッケージを作成する

1. パッケージ化されたコードを保存する場所に S3 バケットを作成します。既存の S3 バケットを使用する場合は、このステップをスキップします。

```
aws s3 mb s3://bucketname
```

2. コマンドプロンプトで以下の package CLI コマンドを実行して、デプロイパッケージを作成します。

```
sam package \
    --template-file template.yaml \
    --output-template-file packaged.yaml \
    --s3-bucket bucketname
```

次のステップでアプリケーションをデプロイするときは、新しいテンプレートファイルである packaged.yaml を指定します。

ステップ 3: アプリケーションをデプロイする

デプロイパッケージを作成したところで、これを使用してアプリケーションを AWS クラウドにデプロイします。デプロイ後、それを AWS クラウドで呼び出すことによってアプリケーションをテストします。

サーバーレスアプリケーションを AWS クラウドにデプロイする

- AWS SAM CLI で deploy コマンドを使用して、テンプレートで定義したすべてのリソースをデプロイします。

```
sam deploy \
    --template-file packaged.yaml \
    --stack-name aws-sam-ocr \
    --capabilities CAPABILITY_IAM \
    --region us-east-1
```

コマンドの --capabilities パラメータは、AWS CloudFormation が IAM ロールを作成することを可能にします。

AWS CloudFormation は、テンプレートで定義されている AWS リソースを作成します。これらのリソースの名前には、AWS CloudFormation コンソールからアクセスできます。

AWS クラウドでサーバーレスアプリケーションをテストする

1. このサンプルアプリケーション用に作成した Amazon S3 バケットに画像をアップロードします。
2. DynamoDB コンソールを開き、作成されたテーブルを見つけます。Amazon Rekognition によって返された結果のテーブルを確認します。
3. DynamoDB テーブルに、アップロードされた画像内で Amazon Rekognition が見つけたテキストが含まれる新しいレコードがあることを確認します。

ステップ 4: アプリケーションをローカルでテストする

アプリケーションをローカルでテストする前に、AWS CloudFormation よって作成された AWS リソースの名前を取得する必要があります。

- AWS CloudFormation から Amazon S3 のキー名とバケット名を取得します。オブジェクトキー、バケット名、およびバケット ARN の値を置き換えて、`SampleEvent.json` ファイルを変更します。
- DynamoDB テーブル名を取得します。この名前は、以下の `sam local invoke` コマンドで使用されます。

AWS SAM CLI を使用してサンプル Amazon S3 イベントを生成し、Lambda 関数を呼び出します。

```
TABLE_NAME=Table name obtained from AWS CloudFormation console sam local invoke --event
 SampleEvent.json
```

`TABLE_NAME=` の部分は、DynamoDB テーブル名を設定します。`--event` パラメータは、Lambda 関数に渡すテストイベントメッセージが含まれるファイルを指定します。

これで、期待される DynamoDB レコードが Amazon Rekognition によって返された結果に基づいて作成されたことを確認できるようになりました。

次のステップ

AWS SAM GitHub リポジトリには、ダウンロードして実験するための追加のサンプルアプリケーションが含まれています。このリポジトリにアクセスするには、「AWS SAM サンプルアプリケーション」を参照してください。

AWS SAM CLI Terraform　のサポート

このセクションでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) と Terraform プロジェクトと Terraform クラウド。

フィードバックを提供し、機能リクエストを送信するには、 GitHub 問題。

トピック

- の開始方法 Terraform のサポート AWS SAM CLI
- の使用 AWS SAM CLI with Terraform ローカルデバッグとテスト用
- の使用 AWS SAM CLI ローカルデバッグとテスト用の Serverless.tf を使用した
- AWS SAM CLI with Terraform リファレンス
- とは AWS SAM CLI のサポート Terraform?

の開始方法 Terraform のサポート AWS SAM CLI

このトピックでは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) と Terraform.

フィードバックを提供し、機能リクエストを送信するには、 GitHub 問題。

トピック

- AWS SAM CLI Terraform 前提条件
- の使用 AWS SAM CLI を使用した コマンド Terraform
- のセットアップ Terraform projects
- のセットアップ Terraform Cloud

AWS SAM CLI Terraform 前提条件

の使用を開始するには、すべての前提条件を満たす必要があります AWS SAM CLI を で使用する Terraform プロジェクト。

1. のインストールまたはアップグレード AWS SAM CLI

   があるかどうかを確認するには AWS SAM CLI がインストールされている場合は、以下を実行します。

```
$ sam --version
```

の場合 AWS SAM CLI は既にインストールされており、出力にバージョンが表示されます。最新バージョンにアップグレードするには、「AWS SAM CLI のアップグレード」 を参照してください。

のインストール手順については、 AWS SAM CLI とその前提条件については、「」を参照してくださいAWS SAM CLI のインストール。

2. インストール Terraform

   があるかどうかを確認するには Terraform がインストールされている場合は、以下を実行します。

   ```
   $ terraform -version
   ```

   をインストールするには Terraform「インストール」を参照してください。Terraform ()Terraform レジストリ。

3. インストール Docker ローカルテスト用

   AWS SAM CLI には が必要です Docker ローカルテスト用。をインストールするには Docker「で使用する Docker のインストール AWS SAM CLI」を参照してください。

の使用 AWS SAM CLI を使用した コマンド Terraform

サポートされている を実行する場合 AWS SAM CLI コマンド、 `--hook-name`オプションを使用して `terraform`値を指定します。以下に例を示します。

```
$ sam local invoke --hook-name terraform
```

このオプションは、 で設定できます。 AWS SAM CLI 設定ファイル。

```
hook_name = "terraform"
```

のセットアップ Terraform projects

を使用するには、このトピックのステップを完了します。 AWS SAM CLI with Terraform プロジェクト。

の外部で AWS Lambda アーティファクトを構築する場合、追加のセットアップは必要ありません。Terraform プロジェクト。の使用を開始するの使用 AWS SAM CLI with Terraform ローカルデバッグとテスト用には、「」を参照してください。 AWS SAM CLI.

内で Lambda アーティファクトを構築する場合 Terraform プロジェクトでは、以下を実行する必要があります。

1. インストール Python 3.8 以降
2. のインストール Make ツール。
3. 内で Lambda アーティファクトのビルドロジックを定義する Terraform プロジェクト。
4. に通知するsam metadataリソースを定義する AWS SAM CLI ビルドロジックの 。
5. を使用する AWS SAM CLI sam build Lambda アーティファクトを構築するための コマンド。

インストール Python 3.8 以降

Python で使用するには 3.8 以降が必要です AWS SAM CLI。 を実行するとsam build、 AWS SAM CLI makefiles は、 を含む を作成します。Python Lambda アーティファクトを構築するための コマンド。

インストール手順については、Python の初心者ガイドにある「Downloading Python」(Python のダウンロード) を参照してください。

次を実行して、Python 3.8 以降がマシンのパスに追加されていることを確認します。

```
$ python --version
```

出力に 3.8 以降のバージョンの Python が表示される必要があります。

のインストール Make tool

GNU Make は、プロジェクトの実行可能ファイルやその他のソース以外のファイルの生成を制御するツールです。 AWS SAM CLI makefiles は、Lambda アーティファクトを構築するためにこのツールに依存する を作成します。

がない場合は Make ローカルマシンに がインストールされている場合は、先に進む前にインストールします。

Windows の場合は、Chocolatey を使用してインストールできます。Chocolatey を使用してインストールするには、「Windows で Make をインストールして使用する方法」の「Chocolatey の使用」を参照してください。

Lambda アーティファクトのビルドロジックを定義する

null_resource を使用する Terraform Lambda ビルドロジックを定義する リソースタイプ。以下は、カスタムビルドスクリプトを使用して Lambda 関数を構築する例です。

```
resource "null_resource" "build_lambda_function" {
    triggers = {
        build_number = "${timestamp()}"
    }

    provisioner "local-exec" {
        command =  substr(pathexpand("~"), 0, 1) == "/"? "./
py_build.sh \"${local.lambda_src_path}\" \"${local.building_path}\"
 \"${local.lambda_code_filename}\" Function" : "powershell.exe -File .\\PyBuild.ps1
 ${local.lambda_src_path} ${local.building_path} ${local.lambda_code_filename}
 Function"
    }
}
```

の定義 sam metadata リソース

sam metadata リソースは null_resource Terraform を提供する リソースタイプ AWS SAM CLI Lambda アーティファクトを見つけるために必要な情報を入力します。プロジェクト内の Lambda 関数またはレイヤーごとに固有の sam metadata リソースが必要です。このリソースタイプの詳細については、「」の「null_resource」を参照してください。 Terraform レジストリ。

を定義するには sam metadata リソース

1. で始まるリソースに名前sam_metadata_を付けて、リソースを として識別します。sam metadata リソース。
2. リソースの triggers ブロック内で Lambda アーティファクトプロパティを定義します。
3. Lambda ビルドロジックを含む null_resource を depends_on 引数で指定します。

   以下はテンプレートの例です。

   ```
   resource "null_resource" "sam_metadata_..." {
   ```

```
  triggers = {
    resource_name = resource_name
    resource_type = resource_type
    original_source_code = original_source_code
    built_output_path = built_output_path
  }
  depends_on = [
    null_resource.build_lambda_function # ref to your build logic
  ]
}
```

以下に、sam metadata リソースの例を示します。

```
resource "null_resource" "sam_metadata_aws_lambda_function_publish_book_review" {
    triggers = {
        resource_name = "aws_lambda_function.publish_book_review"
        resource_type = "ZIP_LAMBDA_FUNCTION"
        original_source_code = "${local.lambda_src_path}"
        built_output_path = "${local.building_path}/${local.lambda_code_filename}"
    }
    depends_on = [
        null_resource.build_lambda_function
    ]
}
```

sam metadata リソースの内容は、Lambda リソースタイプ (関数またはレイヤー) とパッケージングタイプ (ZIP またはイメージ) によって異なります。SSML の詳細と例については、「sam metadata resource」を参照してください。

sam metadata リソースを設定し、サポートされている を使用する場合 AWS SAM CLI コマンド、 AWS SAM CLI は、 を実行する前にメタデータファイルを生成します。 AWS SAM CLI コマンド。このファイルを生成したら、 --skip-prepare-infraオプションを将来の で使用できます。AWS SAM CLI メタデータ生成プロセスをスキップして時間を節約する コマンド。このオプションは、新しい Lambda 関数や新しいAPIエンドポイントの作成など、インフラストラクチャの変更を行っていない場合にのみ使用してください。

を使用する AWS SAM CLI Lambda アーティファクトを構築するには

を使用する AWS SAM CLI sam build Lambda アーティファクトを構築するための コマンド。を実行するとsam build、 AWS SAM CLI は以下を実行します。

1. 内のsam metadataリソースを検索します。Terraform Lambda リソースについて学習し、見つけるための プロジェクト。
2. Lambda ビルドロジックを開始して Lambda アーティファクトを構築します。
3. を整理する .aws-sam ディレクトリを作成します。Terraform で使用する プロジェクト AWS SAM CLI sam local コマンド。

サムビルドでビルドするには

1. を含むディレクトリから Terraform ルートモジュールで、以下を実行します。

```
$ sam build --hook-name terraform
```

2. 特定の Lambda 関数または Layer を構築するには、以下を実行します。

```
$ sam build --hook-name terraform lambda-resource-id
```

Lambda リソース ID は、Lambda 関数名または完全 Terraform aws_lambda_function.list_books または などの リソースアドレスmodule.list_book_function.aws_lambda_function.this[0]。

関数のソースコードまたはその他の Terraform 設定ファイルは、Terraform ルートモジュールの場合は、場所を指定する必要があります。--terraform-project-root-path オプションを使用して、これらのファイルを含む最上位ディレクトリへの絶対パスまたは相対パスを指定します。以下に例を示します。

```
$ sam build --hook-name terraform --terraform-project-root-path ~/projects/terraform/demo
```

コンテナを使用したビルド

を実行する場合 AWS SAM CLI sam build コマンドでは、 AWS SAM CLI ローカル を使用してアプリケーションを構築するには Docker コンテナ。

> **Note**
>
> 必要なもの Docker がインストールされ、設定されています。手順については、で使用する Docker のインストール AWS SAM CLI を参照してください。

コンテナを使用してビルドするには

1. Dockerfile を含む を作成する Terraform, Python および Make ツール。Lambda 関数ランタイムも含める必要があります。

   次に Dockerfile の例を示します。

```
FROM public.ecr.aws/amazonlinux/amazonlinux:2

RUN yum -y update \
    && yum install -y unzip tar gzip bzip2-devel ed gcc gcc-c++ gcc-gfortran \
    less libcurl-devel openssl openssl-devel readline-devel xz-devel \
    zlib-devel glibc-static libcxx libcxx-devel llvm-toolset-7 zlib-static \
    && rm -rf /var/cache/yum

RUN yum -y install make \
    && yum -y install zip

RUN yum install -y yum-utils \
    && yum-config-manager --add-repo https://rpm.releases.hashicorp.com/
AmazonLinux/hashicorp.repo \
    && yum -y install terraform \
    && terraform --version

# AWS Lambda Builders
RUN amazon-linux-extras enable python3.8
RUN yum clean metadata && yum -y install python3.8
RUN curl -L get-pip.io | python3.8
RUN pip3 install aws-lambda-builders
RUN ln -s /usr/bin/python3.8 /usr/bin/python3
RUN python3 --version

VOLUME /project
WORKDIR /project

ENTRYPOINT ["sh"]
```

2. の使用docker build をビルドするには Docker イメージ。

   以下に例を示します。

```
$ docker build --tag terraform-build:v1 <path-to-directory-containing-Dockerfile>
```

3. を実行する AWS SAM CLI `sam build` コマンドと `--use-container`および `--build-image`オプション。

   以下に例を示します。

   ```
   $ sam build --use-container --build-image terraform-build:v1
   ```

次のステップ

の使用を開始するには AWS SAM CLI を で使用する Terraform プロジェクトについては、「」を参照してくださいの使用 AWS SAM CLI with Terraform ローカルデバッグとテスト用。

のセットアップ Terraform Cloud

を使用することをお勧めします。Terraform v1.6.0 以降。古いバージョンを使用している場合は、Terraform ローカルでファイルを計画します。ローカルプランファイルは、 AWS SAM CLI ローカルテストとデバッグを実行するために必要な情報を提供します。

ローカルプランファイルを生成するには

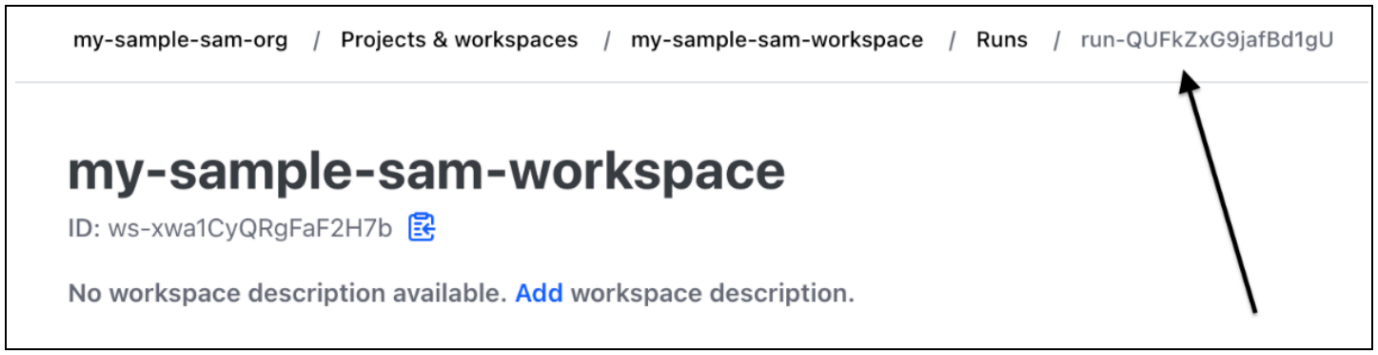
> **Note**
>
> これらのステップは には必要ありません。Terraform v1.6.0 以降。の使用を開始するには AWS SAM CLI with Terraform Cloud「の使用 AWS SAM CLI with Terraform」を参照してください。

1. API トークンの設定 – トークンのタイプは、アクセスレベルによって異なります。詳細については、 のAPI「トークン」を参照してください。 Terraform Cloud ドキュメント。
2. API トークン環境変数の設定 – コマンドラインの例を次に示します。

   ```
   $ export TOKEN="<api-token-value>"
   ```

3. 実行 ID を取得する – から Terraform Cloud コンソールで、 の実行 ID を見つけます。Terraform で使用する を実行する AWS SAM CLI.

   実行 ID は実行のブレッドクラムパスにあります。



4. プランファイルの取得 – APIトークンを使用してローカルプランファイルを取得します。コマンドからの出力例を次に示します。

```
curl \
   --header "Authorization: Bearer $TOKEN" \
   --header "Content-Type: application/vnd.api+json" \
   --location \
   https://app.terraform.io/api/v2/runs/<run ID>/plan/json-output \
   > custom_plan.json
```

これで、 を使用する準備ができました。 AWS SAM CLI with Terraform Cloud。 サポートされているを使用する場合 AWS SAM CLI コマンドで、 `--terraform-plan-file`オプションを使用してローカルプランファイルの名前とパスを指定します。以下に例を示します。

```
$ sam local invoke --hook-name terraform --terraform-plan-file custom-plan.json
```

以下は `sam local start-api` を使用したコマンドの例です。

```
$ sam local start-api --hook-name terraform --terraform-plan-file custom-plan.json
```

これらの例で使用できるサンプルアプリケーションについては、aws-samples の「api_gateway_v2_tf_cloud」を参照してください。GitHub リポジトリ。

次のステップ

の使用を開始するには AWS SAM CLI with Terraform Cloud「の使用 AWS SAM CLI with Terraform ローカルデバッグとテスト用」を参照してください。

の使用 AWS SAM CLI with Terraform ローカルデバッグとテスト用

このトピックでは、サポートされている AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) コマンドと Terraform プロジェクトと Terraform Cloud.

フィードバックを提供し、機能リクエストを送信するには、 GitHub 問題。

トピック

- を使用したローカルテスト sam local invoke
- を使用したローカルテスト sam local start-api
- を使用したローカルテスト sam local start-lambda
- Terraform 制限事項

を使用したローカルテスト sam local invoke

> **Note**
>
> を使用するには AWS SAM CLI をローカルでテストするには、Docker を組み込み、設定する必要があります。手順については、で使用する Docker のインストール AWS SAM CLI を参照してください。

以下の例では、イベントを渡して Lambda 関数をローカルでテストしています。

```
$ sam local invoke --hook-name terraform hello_world_function -e events/event.json -
```

このコマンドを使用する場合の詳細については、「sam local invoke を使用したテストの概要」を参照してください。

を使用したローカルテスト sam local start-api

`sam local start-api` で を使用するには Terraform、以下を実行します。

```
$ sam local start-api --hook-name terraform
```

以下に例を示します。

```
$ sam local start-api --hook-name terraform




Running Prepare Hook to prepare the current application


Executing prepare hook of hook "terraform"


Initializing Terraform application


...
Creating terraform plan and getting JSON output


....
Generating metadata file




Unresolvable attributes discovered in project, run terraform apply to resolve them.




Finished generating metadata file. Storing in...
Prepare hook completed and metadata file generated at: ...
Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET]


Mounting None at http://127.0.0.1:3000/hello [POST]
```

```
You can now browse to the above endpoints to invoke your functions. You do not need
 to restart/reload SAM CLI while working on your functions, changes will be reflected
 instantly/automatically. If you
used sam build before running local commands, you will need to re-run sam build for the
 changes to be picked up. You only need to restart SAM CLI if you update your AWS SAM
 template
2023-06-26 13:21:20  * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
```

このコマンドの詳細については、「sam local start-api を使用したテストの概要」を参照してください。

Lambda オーソライザーを使用する Lambda 関数

Lambda オーソライザーを使用するように設定された Lambda 関数の場合、 AWS SAM CLI は、Lambda 関数エンドポイントを呼び出す前に Lambda オーソライザーを自動的に呼び出します。

- この機能の詳細については、「」を参照してください。 AWS SAM CLI「Lambda オーソライザーを使用する Lambda 関数」を参照してください。
- での Lambda オーソライザーの使用の詳細については、「」を参照してください。Terraform、「」を参照してください。 Resource: aws_api_gateway_authorizer ()Terraform レジストリ。

を使用したローカルテスト sam local start-lambda

以下は、 AWS Command Line Interface () を使用して Lambda 関数をローカルでテストする例です AWS CLI。

1. を使用する AWS SAM CLI ローカルテスト環境を作成する場合：

```
$ sam local start-lambda --hook-name terraform hello_world_function
```

2. を使用して AWS CLI 関数をローカルで呼び出します。

```
$ aws lambda invoke --function-name hello_world_function --endpoint-
url http://127.0.0.1:3001/ response.json --cli-binary-format raw-in-base64-out --
payload file://events/event.json
```

このコマンドの詳細については、「を使用したテストの概要 sam local start-lambda」を参照してください。

Terraform 制限事項

以下は、 を使用する場合の制限です。 AWS SAM CLI with Terraform:

- 複数のレイヤーにリンクされている Lambda 関数。
- Terraform リソース間のリンクを定義するローカル変数。
- まだ作成されていない Lambda 関数の参照。これには、RESTAPIリソースの body 属性で定義されている関数が含まれます。

プロジェクトに新しいリソースが追加されたときに `terraform apply` を実行すると、このような制限を回避することができます。

の使用 AWS SAM CLI ローカルデバッグとテスト用の Serverless.tf を使用した

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) は、 AWS Lambda 関数とレイヤーのローカルデバッグとテストのために Serverless.tf モジュールで使用できます。以下 AWS SAM CLI コマンドがサポートされています。

- `sam build`
- `sam local invoke`
- `sam local start-api`
- `sam local start-lambda`

> **Note**
>
> Serverless.tf バージョン 4.6.0 以降のサポート AWS SAM CLI 統合。

の使用を開始するには AWS SAM CLI を Serverless.tf モジュールで使用して、最新バージョンの Serverless.tf に更新し、 AWS SAM CLI.

serverless.tf バージョン 6.0.0 以降では、`create_sam_metadata` パラメーターを `true` として設定する必要があります。これにより、 が生成するメタデータリソースが生成されます。 AWS SAM CLI `sam build` コマンドには が必要です。

詳細を確認するトピック Serverless.tf、「」を参照してくださいterraform-aws-lambda-module。

AWS SAM CLI with Terraform リファレンス

このセクションは、 AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) と Terraform ローカルデバッグとテスト用。

フィードバックを提供し、機能リクエストを送信するには、 GitHub 問題。

AWS SAM でサポートされている機能のリファレンス

のリファレンスドキュメント AWS SAM CLI での使用がサポートされている の機能 Terraform は次の場所にあります。

- sam build
- sam local invoke
- sam local start-api
- sam local start-lambda

Terraform 特定のリファレンス

の使用に固有のリファレンスドキュメント AWS SAM CLI with Terraform は次の場所にあります。

- sam metadata resource

sam metadata resource

このページには、 で使用されるsam metadata resourceリソースタイプのリファレンス情報が含まれています。Terraform プロジェクト。

- AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) と Terraform「とは AWS SAM CLI のサポート Terraform?」を参照してください。
- を使用するには AWS SAM CLI with Terraform「の使用 AWS SAM CLI with Terraform ローカルデバッグとテスト用」を参照してください。

トピック

- 引数
- 例

引数

引数	説明
built_output_path	AWS Lambda 関数のビルドされたアーティファクトへのパス。
docker_build_args	Docker ビルド引数JSONオブジェクトのデコードされた文字列。この引数はオプションです。
docker_context	Docker イメージのビルドコンテキストを含むディレクトリへのパス。
docker_file	Docker ファイルへのパス。このパスは、docker_context パスを起点とした相対パスです。 この引数はオプションです。デフォルト値は Dockerfile です。
docker_tag	作成された Docker イメージタグの値。この値はオプションです。
depends_on	Lambda 関数またはレイヤーのビルディングリソースへのパス。詳細については、「」の「 depends_on引数」を参照してください。Terraform レジストリ。
original_source_code	Lambda 関数が定義されている場所へのパス。この値は、文字列、文字列の配列、または文字列としてのデコードされたJSONオブジェクトです。 • 複数のコードパスはサポートされていないため、文字列の配列の場合は最初の値のみが使用されます。 • JSON オブジェクトの場合、 source_code_property も定義する必要があります。
resource_name	Lambda 関数の名前。
resource_type	Lambda 関数のパッケージタイプの形式。次の値が許容されます。

引数	説明
	• IMAGE_LAMBDA_FUNCTION • LAMBDA_LAYER • ZIP_LAMBDA_FUNCTION
source_code_property	JSON オブジェクト内の Lambda リソースコードへのパス。original_source_code がJSONオブジェクトの場合にこのプロパティを定義します。

例

ZIP パッケージタイプを使用して Lambda 関数を参照する sam メタデータリソース

```
# Lambda function resource
resource "aws_lambda_function" "tf_lambda_func" {
  filename = "${path.module}/python/hello-world.zip"
  handler = "index.lambda_handler"
  runtime = "python3.8"
  function_name = "function_example"
  role = aws_iam_role.iam_for_lambda.arn
  depends_on = [
    null_resource.build_lambda_function # function build logic
  ]
}

# sam metadata resource
resource "null_resource" "sam_metadata_function_example" {
  triggers = {
    resource_name = "aws_lambda_function.function_example"
    resource_type = "ZIP_LAMBDA_FUNCTION"
    original_source_code = "${path.module}/python"
    built_output_path = "${path.module}/building/function_example"
  }
  depends_on = [
    null_resource.build_lambda_function # function build logic
  ]
}
```

パッケージタイプがイメージの Lambda 関数 を参照する sam metadata resource

```
resource "null_resource" "sam_metadata_function {
  triggers = {
    resource_name = "aws_lambda_function.image_function"
    resource_type = "IMAGE_LAMBDA_FUNCTION"
    docker_context = local.lambda_src_path
    docker_file = "Dockerfile"
    docker_build_args = jsonencode(var.build_args)
    docker_tag = "latest"
  }
}
```

Lambda レイヤーを参照する sam metadata resource

```
resource "null_resource" "sam_metadata_layer1" {
  triggers = {
    resource_name = "aws_lambda_layer_version.layer"
    resource_type = "LAMBDA_LAYER"
    original_source_code = local.layer_src
    built_output_path = "${path.module}/${layer_build_path}"
  }
  depends_on = [null_resource.layer_build]
}
```

とは AWS SAM CLI のサポート Terraform?

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAM CLI) と Terraform プロジェクトまたは Terraform Cloud は、以下のローカルデバッグとテストを実行します。

- AWS Lambda 関数とレイヤー。
- Amazon API Gateway APIs。

の概要 Terraform「 とは」を参照してください。Terraformの？ HashiCorp Terraform ウェブサイト。

フィードバックを提供し、機能リクエストを送信するには、 GitHub 問題。

> **Note**
>
> の解析ステップの一部として AWS SAM CLIの統合、 AWS SAM CLI はユーザーコマンドを処理し、プロジェクトファイルとデータを生成します。コマンド出力は変更されませんが、特定の環境では、環境またはランナーにより出力で追加のログや情報が挿入される場合があります。

トピック

とは AWS SAM CLI?

AWS SAM CLI は、 AWS SAM テンプレートや、 などのサポートされているサードパーティー統合で使用できるコマンドラインツールです。Terraform、サーバーレスアプリケーションを構築して実行します。の概要 AWS SAM CLI「AWS SAMCLI とは？」を参照してください。

AWS SAM CLI は、 の次のコマンドをサポートしています。Terraform:

- `sam local invoke` – AWS Lambda 関数リソースの 1 回限りの呼び出しをローカルで開始します。このコマンドの詳細については、「sam local invoke を使用したテストの概要」を参照してください。
- `sam local start-api` – Lambda リソースをローカルで実行し、ローカルHTTPサーバーホストを介してテストします。このタイプのテストは、APIゲートウェイエンドポイントによって呼び出される Lambda 関数に役立ちます。このコマンドの詳細については、「sam local start-api を使用したテストの概要」を参照してください。
- `sam local start-lambda` – AWS Command Line Interface （ AWS CLI) または を使用して関数をローカルで呼び出すために、Lambda 関数のローカルエンドポイントを起動しますSDKs。このコマンドの詳細については、「を使用したテストの概要 sam local start-lambda」を参照してください。

の使用方法 AWS SAM CLI with Terraform?

コア Terraform ワークフローは、書き込み、計画、適用の 3 つのステージで構成されます。で AWS SAM CLI のサポート Terraformでは、 AWS SAM CLI `sam local` を引き続き使用する場合の一連のコマンド Terraform アプリケーションを管理する ワークフロー AWS。一般的には、以下の操作を実行する必要があります。

- 書き込み — を使用してインフラストラクチャをコードとして作成する Terraform.
- テストとデバッグ — を使用する AWS SAM CLI は、アプリケーションをローカルでテストおよびデバッグします。
- 計画 — 適用前に変更をプレビューします。
- 適用 — インフラストラクチャをプロビジョニングします。

の使用例 AWS SAM CLI with Terraform「Better together:」を参照してください。 AWS SAM CLI また、HashiCorp Terraform AWS コンピューティングブログの 。

次のステップ

すべての前提条件を完了して を設定するには Terraform「の開始方法 Terraform のサポート AWS SAM CLI」を参照してください。

AWS SAM CLI を使用して AWS CDK アプリケーションをローカルでテストおよび構築する

AWS SAM CLI を使用して、AWS Cloud Development Kit (AWS CDK) を使用して定義されたサーバーレスアプリケーションをローカルでテストおよび構築できます。AWS SAM CLI は AWS CDK プロジェクト構造内で機能するため、ユーザーは AWS CDK アプリケーションの作成、変更、およびデプロイに引き続き AWS CDK ツールキットを使用できます。

AWS CDK のインストールと設定については、AWS Cloud Development Kit (AWS CDK) デベロッパーガイドの「Getting started with the AWS CDK」を参照してください。

> **Note**
>
> AWS SAM CLI は、バージョン 1.135.0 以降の AWS CDK v1 およびバージョン 2.0.0 以降の AWS CDK v2 をサポートします。

トピック

AWS SAM と AWS CDK の開始方法

このトピックでは、AWS CDK アプリケーションで AWS SAM CLI を使用するために必要な事柄と、シンプルな AWS CDK アプリケーションを構築してローカルにテストするための手順を説明します。

前提条件

AWS CDK で AWS SAM CLI を使用するには、AWS CDK と AWS SAM CLI をインストールする必要があります。

- AWS CDK のインストールについては、AWS Cloud Development Kit (AWS CDK) デベロッパーガイドの「Getting started with the AWS CDK」を参照してください。

- AWS SAM CLI のインストールについては、「AWS SAM CLI のインストール」を参照してください。

AWS CDK アプリケーションの作成とローカルでのテスト

AWS SAM CLI を使用して AWS CDK アプリケーションをローカルにテストするには、Lambda 関数が含まれる AWS CDK アプリケーションが必須です。次のステップを使用して、Lambda 関数を使用する基本的な AWS CDK アプリケーションを作成します。詳細については、AWS Cloud Development Kit (AWS CDK) デベロッパーガイドの「AWS CDKを使用して、サーバーレスアプリケーションを作成する」を参照してください。

> **Note**
>
> AWS SAM CLI は、バージョン 1.135.0 以降の AWS CDK v1 およびバージョン 2.0.0 以降の AWS CDK v2 をサポートします。

ステップ 1: AWS CDK アプリケーションを作成する

このチュートリアルでは、TypeScript を使用するAWS CDK アプリケーションを初期化します。

実行するコマンド:

AWS CDK v2

```
mkdir cdk-sam-example
cd cdk-sam-example
cdk init app --language typescript
```

AWS CDK v1

```
mkdir cdk-sam-example
cd cdk-sam-example
cdk init app --language typescript
npm install @aws-cdk/aws-lambda
```

ステップ 2: アプリケーションに Lambda 関数を追加する

`lib/cdk-sam-example-stack.ts` のコードを、以下のコードに置き換えます。

AWS CDK v2

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as lambda from 'aws-cdk-lib/aws-lambda';

export class CdkSamExampleStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    new lambda.Function(this, 'MyFunction', {
      runtime: lambda.Runtime.PYTHON_3_9,
      handler: 'app.lambda_handler',
      code: lambda.Code.fromAsset('./my_function'),
    });
  }
}
```

AWS CDK v1

```
import * as cdk from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';

export class CdkSamExampleStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    new lambda.Function(this, 'MyFunction', {
      runtime: lambda.Runtime.PYTHON_3_9,
      handler: 'app.lambda_handler',
      code: lambda.Code.fromAsset('./my_function'),
    });
  }
}
```

ステップ 3: Lambda 関数コードを追加する

`my_function` という名前のディレクトリを作成します。そのディレクトリに `app.py` という名前のファイルを作成します。

実行するコマンド:

```
mkdir my_function
cd my_function
touch app.py
```

次のコードを app.py に追加します。

```
def lambda_handler(event, context):
    return "Hello from SAM and the CDK!"
```

ステップ 4: Lambda 関数をテストする

AWS CDK アプリケーションで定義されている Lambda 関数は、AWS SAM CLI を使用してローカルに呼び出すことができます。これを行うには、呼び出す関数のコンストラクト識別子と、合成した AWS CloudFormation テンプレートへのパスが必要です。

実行するコマンド:

```
cdk synth --no-staging
```

```
sam local invoke MyFunction --no-event -t ./cdk.out/CdkSamExampleStack.template.json
```

出力例:

```
Invoking app.lambda_handler (python3.9)

START RequestId: 5434c093-7182-4012-9b06-635011cac4f2 Version: $LATEST
"Hello from SAM and the CDK!"
END RequestId: 5434c093-7182-4012-9b06-635011cac4f2
REPORT RequestId: 5434c093-7182-4012-9b06-635011cac4f2 Init Duration: 0.32 ms Duration:
 177.47 ms Billed Duration: 178 ms Memory Size: 128 MB Max Memory Used: 128 MB
```

AWS SAM CLI を使用した AWS CDK アプリケーションのテストに使用できるオプションの詳細については、「AWS SAM を使用する AWS CDK アプリケーションのローカルでのテスト」を参照してください。

AWS SAM を使用する AWS CDK アプリケーションのローカルでのテスト

AWS SAM CLI を使用して、AWS CDK アプリケーションのプロジェクトルートディレクトリから以下のコマンドを実行することで、AWS CDK アプリケーションをローカルにテストできます。

- sam local invoke
- sam local start-api
- sam local start-lambda

AWS CDK アプリケーションで、sam local コマンドのいずれかを実行する前に、`cdk synth` を実行する必要があります。

sam local invoke の実行時、呼び出す関数のコンストラクト識別子と、合成した AWS CloudFormation テンプレートへのパスが必要です。アプリケーションでネストされたスタックを使用する場合、名前の競合を解決するには、関数が定義されているスタックの名前が必要です。

使用:

```
# Invoke the function FUNCTION_IDENTIFIER declared in the stack STACK_NAME
sam local invoke [OPTIONS] [STACK_NAME/FUNCTION_IDENTIFIER]

# Start all APIs declared in the AWS CDK application
sam local start-api -t ./cdk.out/CdkSamExampleStack.template.json [OPTIONS]

# Start a local endpoint that emulates AWS Lambda
sam local start-lambda -t ./cdk.out/CdkSamExampleStack.template.json [OPTIONS]
```

例

以下のサンプルで宣言されているスタックと関数について考えてみましょう。

```
app = new HelloCdkStack(app, "HelloCdkStack",
   ...
)
class HelloCdkStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    ...
    new lambda.Function(this, 'MyFunction', {
```

```
        ...
    });

    new HelloCdkNestedStack(this, 'HelloNestedStack' ,{
        ...
    });
}

class HelloCdkNestedStack extends cdk.NestedStack {
  constructor(scope: Construct, id: string, props?: cdk.NestedStackProps) {
    ...
    new lambda.Function(this, 'MyFunction', {
        ...
    });
    new lambda.Function(this, 'MyNestedFunction', {
        ...
    });
}
```

以下のコマンドは、上記の例で定義されている Lambda 関数をローカルに呼び出します。

```
# Invoke MyFunction from the HelloCdkStack
sam local invoke -t ./cdk.out/HelloCdkStack.template.json MyFunction
```

```
# Invoke MyNestedFunction from the HelloCdkNestedStack
sam local invoke -t ./cdk.out/HelloCdkStack.template.json MyNestedFunction
```

```
# Invoke MyFunction from the HelloCdkNestedStack
sam local invoke -t ./cdk.out/HelloCdkStack.template.json HelloNestedStack/MyFunction
```

AWS SAM を使用する AWS CDK アプリケーションの構築

AWS SAM CLI は、sam build により AWS CDK アプリケーションで定義された Lambda 関数とレイヤーを構築するためのサポートを提供します。

zip Artifact を使用する Lambda 関数の場合は、`sam local` コマンドを実行する前に `cdk synth` を実行してください。`sam build` は、必須ではありません。

AWS CDK アプリケーションでイメージタイプの関数を使用する場合は、`sam local` コマンドを実行する前に `cdk synth` を実行し、`sam build` を実行してください。`sam build` を実行する

と、AWS SAM は、NodejsFunction などのランタイム固有の構造を使用する Lambda 関数またはレイヤーを構築しません。`sam build` はバンドルされたアセットをサポートしていません。

例

AWS CDK プロジェクトルートディレクトリから以下のコマンドを実行すると、アプリケーションが構築されます。

```
sam build -t ./cdk.out/CdkSamExampleStack.template.json
```

AWS SAM での AWS CDK アプリケーションのデプロイ

AWS SAM CLI では、AWS CDK アプリケーションのデプロイはサポートされません。`cdk deploy` を使用してアプリケーションをデプロイします。詳細については、AWS Cloud Development Kit (AWS CDK) デベロッパーガイドの「AWS CDK ツールキット (cdk command)」を参照してください。

AWS SAM CLI を使用してアプリケーションを公開する

他のユーザーが AWS SAM アプリケーションを見つけてデプロイできるようにするには、AWS SAM CLI を使用してアプリケーションを AWS Serverless Application Repository に公開できます。AWS SAM CLI を使用してアプリケーションを公開するには、AWS SAM テンプレートを使用してアプリケーションを定義する必要があります。それをローカルで、または AWS クラウドでテストする必要もあります。

このトピックの手順に従って、新しいアプリケーションを作成、既存アプリケーションの新しいバージョンを作成、または既存アプリケーションのメタデータを更新します。(何を実行するかは、アプリケーションがすでに AWS Serverless Application Repository に存在しているかどうか、およびアプリケーションメタデータが変更されているかどうかに応じて異なります。) アプリケーションメタデータの詳細については、「AWS SAM テンプレートのメタデータセクションのプロパティ」を参照してください。

前提条件

以下は、AWS SAM CLI を使用してアプリケーションを AWS Serverless Application Repository に公開するための前提条件です。

- インストール済みの AWS SAM CLI。詳細については、「AWS SAM CLI のインストール」を参照してください。AWS SAM CLI がインストールされているかどうかを確認するには、以下のコマンドを実行します。

```
sam --version
```

- 有効な AWS SAM テンプレート。
- AWS SAM テンプレートが参照するアプリケーションのコードと依存関係。
- セマンティックバージョン (アプリケーションをパブリックに共有する場合にのみ必要)。この値は 1.0 といったシンプルなものを使用できます。
- アプリケーションのソースコードをポイントする URL。
- `README.md` ファイル。このファイルには、お客様がアプリケーションを使用する方法、およびお客様独自の AWS アカウントでアプリケーションをデプロイする前の設定方法が説明されている必要があります。
- `LICENSE.txt` ファイル (アプリケーションをパブリックに共有する場合にのみ必要)。

- アプリケーションにネストされたアプリケーションが含まれている場合は、それらがすでに AWS Serverless Application Repository で公開されている必要があります。
- アプリケーションをパッケージ化するときに Amazon Simple Storage Service (Amazon S3) にアップロードするアーティファクトに対する読み取り許可をサービスに付与する、有効な Amazon S3 バケットポリシー。このポリシーを設定するには、以下を実行します。

  1. https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
  2. アプリケーションのパッケージ化に使用した Amazon S3 バケットの名前を選択します。
  3. [Permissions] (アクセス許可) をクリックします。
  4. [Permissions] (アクセス許可) タブの [Bucket policy] (バケットポリシー) で [Edit] (編集) をクリックします。
  5. [Edit bucket policy] (バケットポリシーを編集) ページで、[Policy] (ポリシー) エディタに以下のポリシーステートメントを貼り付けます。ポリシーステートメントでは、`Resource` エレメントにお使いのバケット名、AWS エレメントにお使いの `Condition` アカウント ID を使用するようにしてください。`Condition` エレメントの式は、AWS Serverless Application Repository が、指定された AWS アカウントからのアプリケーションのみにアクセスする許可を持っていることを確実にします。ポリシーステートメントの詳細については、IAM ユーザーガイドの「IAM JSON ポリシーの要素のリファレンス」を参照してください。

     ```
     {
         "Version": "2012-10-17",
         "Statement": [
             {
                 "Effect": "Allow",
                 "Principal": {
                     "Service":  "serverlessrepo.amazonaws.com"
                 },
                 "Action": "s3:GetObject",
                 "Resource": "arn:aws:s3:::<your-bucket-name>/*",
                 "Condition" : {
                     "StringEquals": {
                         "aws:SourceAccount": "123456789012"
                     }
                 }
             }
         ]
     }
     ```

  6. [Save changes] (変更の保存) をクリックします。

新しいアプリケーションの公開

ステップ 1: AWS SAM テンプレートに **Metadata** セクションを追加する

まず、Metadata テンプレートに AWS SAM セクションを追加します。AWS Serverless Application Repository に公開するアプリケーション情報を入力します。

以下は、Metadata セクションの例です。

```
Metadata:
  AWS::ServerlessRepo::Application:
    Name: my-app
    Description: hello world
    Author: user1
    SpdxLicenseId: Apache-2.0
    LicenseUrl: LICENSE.txt
    ReadmeUrl: README.md
    Labels: ['tests']
    HomePageUrl: https://github.com/user1/my-app-project
    SemanticVersion: 0.0.1
    SourceCodeUrl: https://github.com/user1/my-app-project

Resources:
  HelloWorldFunction:
    Type: AWS::Lambda::Function
      Properties:
        ...
        CodeUri: source-code1
        ...
```

AWS SAM テンプレートの Metadata セクションに関する詳細については、「AWS SAM テンプレートのメタデータセクションのプロパティ」を参照してください。

ステップ 2: アプリケーションをパッケージ化する

以下の AWS SAM CLI コマンドを実行します。このコマンドは、アプリケーションアーティファクトを Amazon S3 にアップロードし、packaged.yaml と呼ばれる新しいテンプレートファイルを出力します。

```
sam package --output-template-file packaged.yaml --s3-bucket <your-bucket-name>
```

この packaged.yaml テンプレートファイルは、次のステップでアプリケーションを AWS Serverless Application Repository に公開するために使用します。このファイルは元のテンプレートファイル (template.yaml) と似ていますが、CodeUri、LicenseUrl、および ReadmeUrl プロパティが、それぞれのアーティファクトが含まれる Amazon S3 バケットとオブジェクトをポイントするという重要な相違点があります。

packaged.yaml テンプレートファイルの例から次のスニペットは、CodeUri プロパティを示しています。

```
MySampleFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://bucketname/fbd77a3647a4f47a352fc0bjectGUID

...
```

ステップ 3: アプリケーションを公開する

AWS SAM アプリケーションのプライベートバージョンを AWS Serverless Application Repository に公開するには、以下の AWS SAM CLI コマンドを実行します。

```
sam publish --template packaged.yaml --region us-east-1
```

sam publish コマンドの出力には、AWS Serverless Application Repository 上のアプリケーションへのリンクが含まれます。また、AWS Serverless Application Repository ランディングページに直接移動して、アプリケーションを検索することもできます。

ステップ 4: アプリケーションを共有する (オプション)

アプリケーションはデフォルトでプライベートに設定されているため、他の AWS アカウントには表示されません。アプリケーションを他のユーザーと共有するには、アプリケーションをパブリックにするか、特定の AWS アカウントのリストに許可を付与する必要があります。

AWS CLI を使用してアプリケーションを共有する方法については、AWS Serverless Application Repository デベロッパーガイドの「AWS Serverless Application Repository Resource-Based Policy Examples」を参照してください。AWS Management Console を使用したアプリケーションの共有については、AWS Serverless Application Repository デベロッパーガイドの「Sharing an Application」を参照してください。

既存アプリケーションの新しいバージョンの公開

アプリケーションを AWS Serverless Application Repository に公開した後、新しいバージョンの公開が必要になる場合があります。例えば、Lambda 関数コードを変更したり、アプリケーションアーキテクチャに新しいコンポーネントを追加した場合などです。

以前に公開したアプリケーションを更新するには、上記と同じプロセスを使用してアプリケーションを再度公開します。AWS SAM テンプレートファイルの `Metadata` セクションには、最初に公開したときのものと同じアプリケーション名を入力しますが、新しい `SemanticVersion` 値を含めてください。

例えば、アプリケーションが `SampleApp` という名前と、`1.0.0` の `SemanticVersion` で公開されたとします。そのアプリケーションを更新するには、AWS SAM テンプレートにアプリケーション名 `SampleApp` と、`1.0.1` の `SemanticVersion` (または `1.0.0` 以外の任意の番号) を使用する必要があります。

その他のトピック

- AWS SAM テンプレートのメタデータセクションのプロパティ

AWS SAM テンプレートのメタデータセクションのプロパティ

`AWS::ServerlessRepo::Application` は、AWS Serverless Application Repository に公開するアプリケーション情報を指定するために使用できるメタデータキーです。

> **Note**
>
> `AWS::ServerlessRepo::Application` メタデータキーは AWS CloudFormation の組み込み関数をサポートしません。

プロパティ

この表には、`Metadata` テンプレートの AWS SAM セクションのプロパティに関する情報が記載されています。このセクションは、AWS SAM CLI を使用して AWS Serverless Application Repository にアプリケーションを公開するために必要です。

プロパティ	タイプ	必須	説明
`Name`	文字列	TRUE	アプリケーションの名前。 最小長: 1 最大長 = 140。 パターン: `"[a-zA-Z0-9\\-]+";`
`Description`	文字列	TRUE	アプリケーションの説明。 最小長: 1 最大長 = 256。
`Author`	文字列	TRUE	アプリケーションを公開する作成者の名前。 最小長: 1 最大長 = 127。 パターン: `"^[a-z0-9](([a-z0-9]|-(?!-))*[a-z0-9])?$";`
`SpdxLicenseId`	文字列	FALSE	有効なライセンス識別子。有効なライセンス識別子のリストを確認するには、Software Package Data Exchange (SPDX) ウェブサイトの「SPDX License List」を参照してください。
`LicenseUrl`	文字列	FALSE	アプリケーションの spdxLicenseID 値に一致する、ローカルライセンスファイルへの参照、またはライセンスファイルへの Amazon S3 リンク。 AWS SAM コマンドを使用してパッケージ化されていない `sam package` テンプレートファイルにこのプロパティのローカルファイルへの参照を含めることはできますが、`sam publish` コマンドを使用してアプリケーションを公開するには、このプロパティが Amazon S3 バケットへの参照である必要があります。 最大サイズ: 5 MB。 アプリケーションをパブリックにするには、このプロパティに値を指定する必要があります。アプリ

プロパティ	タイプ	必須	説明
			ケーションが公開された後は、このプロパティを更新できないことに注意してください。このため、アプリケーションにライセンスを追加するには、まずライセンスを削除する、または別の名前で新しいアプリケーションを公開する必要があります。
`ReadmeUrl`	文字列	FALSE	アプリケーションとその仕組みに関するより詳細な説明が含まれたローカル readme ファイルへの参照、またはその readme ファイルへの Amazon S3 リンク。 AWS SAM コマンドを使用してパッケージ化されていない `sam package` テンプレートファイルにこのプロパティのローカルファイルへの参照を含めることはできますが、`sam publish` コマンドを使用して公開するには、このプロパティが Amazon S3 バケットへの参照である必要があります。 最大サイズ: 5 MB。
`Labels`	文字列	FALSE	検索結果でアプリケーションを発見しやすくするラベル。 最小長: 1 最大長 = 127。ラベルの最大数:10。 パターン: `"^[a-zA-Z0-9+\\-_:\\/@]+$";`
`HomePageUrl`	文字列	FALSE	アプリケーションに関する詳細情報が含まれた URL。例えば、アプリケーションの GitHub リポジトリの場所などです。

プロパティ	タイプ	必須	説明
SemanticVersion	文字列	FALSE	アプリケーションのセマンティックバージョンです。セマンティックバージョニングの仕様については、「セマンティックバージョニング」のウェブサイトを参照してください。 アプリケーションをパブリックにするには、このプロパティに値を指定する必要があります。
SourceCodeUrl	文字列	FALSE	アプリケーションのソースコードを含むパブリックリポジトリへのリンク。

ユースケース

このセクションでは、アプリケーション公開のユースケースと、そのユースケースで処理される Metadata プロパティを一覧表示します。特定のユースケースで一覧表示されていないプロパティは無視されます。

- 新しいアプリケーションの作成 - 新しいアプリケーションは、アカウントに一致する名前を持つアプリケーションが AWS Serverless Application Repository にない場合に作成されます。
  - Name
  - SpdxLicenseId
  - LicenseUrl
  - Description
  - Author
  - ReadmeUrl
  - Labels
  - HomePageUrl
  - SourceCodeUrl
  - SemanticVersion
  - AWS SAM テンプレートの内容 (イベントソース、リソース、Lambda 関数コードなど)

- アプリケーションバージョンの作成 - アプリケーションバージョンは、アカウントに一致する名前を持つアプリケーションがすでに AWS Serverless Application Repository に存在し、かつ SemanticVersion が変更されている場合に作成されます。
  - `Description`
  - `Author`
  - `ReadmeUrl`
  - `Labels`
  - `HomePageUrl`
  - `SourceCodeUrl`
  - `SemanticVersion`
  - AWS SAM テンプレートの内容 (イベントソース、リソース、Lambda 関数コードなど)

- アプリケーションの更新 - アプリケーションは、アカウントに一致する名前を持つアプリケーションがすでに AWS Serverless Application Repository に存在し、かつ SemanticVersion が変更されていない場合に更新されます。
  - `Description`
  - `Author`
  - `ReadmeUrl`
  - `Labels`
  - `HomePageUrl`

例

以下は、`Metadata` セクションの例です。

```
Metadata:
  AWS::ServerlessRepo::Application:
    Name: my-app
    Description: hello world
    Author: user1
    SpdxLicenseId: Apache-2.0
    LicenseUrl: LICENSE.txt
    ReadmeUrl: README.md
    Labels: ['tests']
```

```
    HomePageUrl: https://github.com/user1/my-app-project
    SemanticVersion: 0.0.1
    SourceCodeUrl: https://github.com/user1/my-app-project
```

AWS SAM のドキュメント履歴

以下の表には、AWS Serverless Application Model デベロッパーガイドの各リリースにおける重要な変更点が説明されています。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。

- ドキュメント最終更新日: 2024 年 6 月 20 日

変更	説明	日付
デベロッパーガイド全体のコンテンツの再構築と更新	見つけやすさと使いやすさを向上させるために、ガイドが再編成され、再構築されました。タイトルが更新され、改善されました。トピックと概念を紹介する際に、追加の詳細が提供されました。	2024 年 6 月 20 日
Ruby 3.3 の AWS SAM CLI サポートが追加されました	Ruby 3.3 がランタイムおよびイメージリポジトリとして提供されるようになりました。詳細については、「Image repositories」および「sam init」を参照してください。	2024 年 4 月 4 日
AWS SAMCLI コマンドオプションが追加されました	コマンド sam local start-api の新しいオプションである `--ssl-cert-file PATH` 、`--ssl-key-file PATH` が提供されるようになりました。さらに、新しいコマンドラインオプション `--add-host LIST` が、sam local invoke、sam local start-api、および sam local start-lambda で使用できます。	2024 年 3 月 20 日

NET 8 の AWS SAM CLI サポートが追加されました	.NET 8 がランタイムおよびイメージリポジトリとして提供されるようになりました。.NET Core 3.1、Node.js 14、Node.js 12、Python 3.7、Ruby 2.7 のランタイムおよびイメージリポジトリはサポートされなくなりました。「Image repositories」および「sam init」を参照してください。	2024 年 2 月 22 日
Linux 用の AWS SAM CLI arm64 パッケージインストーラが追加されました	手順については、「AWS SAMCLI のインストール」を参照してください。	2023 年 12 月 6 日
AWS SAMCLI sam sync コマンド用の --watch-exclude オプションが追加されました	ファイルとフォルダを同期の開始から除外します。詳細については、「同期を開始しないファイルとフォルダを指定する」を参照してください。	2023 年 12 月 6 日
AWS SAMCLI sam sync コマンドの --build-in-source オプションが追加されました	ビルドプロセスを高速化するには、ソースフォルダにプロジェクトを構築します。詳細については、「ソースフォルダにプロジェクトを構築することでビルド時間を短縮する」を参照してください。	2023 年 12 月 6 日
AWS SAMCLI sam build コマンドの --build-in-source オプションが追加されました	ビルドプロセスを高速化するには、ソースフォルダにプロジェクトを構築します。詳細については、「ソースフォルダにプロジェクトを構築することでビルド時間を短縮する」を参照してください。	2023 年 12 月 6 日

AWS SAM CLI リモート呼び出しコマンドの新しいリソースサポートが追加されました	`sam remote invoke` は、Kinesis Data Streams アプリケーション、Amazon SQS キュー、Step Functions ステートマシンで使用します。詳細については、「sam remote invoke を使用する」を参照してください。	2023 年 11 月 15 日
共有可能なテストイベント用の新しい AWS SAM CLI リモートテストイベントコマンドが追加されました	AWS SAM CLI を使用して、EventBridge スキーマレジストリから共有可能なテストイベントにアクセスして管理し、AWS クラウド で Lambda 関数をテストします。詳細については、「Using sam remote test-event」を参照してください。	2023 年 10 月 3 日
Terraform の AWS SAM CLI サポートの一般提供が開始されました	Terraform をサポートする AWS SAM CLI について詳しくは、「AWS SAM CLITerraform サポート」を参照してください。	2023 年 9 月 5 日
Terraform Cloud の AWS SAM CLI サポートが追加されました	AWS SAM CLI は、Terraform Cloud のローカルテストをサポートするようになりました。詳細については、「Terraform Cloudのセットアップ」を参照してください。	2023 年 9 月 5 日

AWS SAM CLI 設定ファイル用の YAML ファイル形式のサポートが追加されました	AWS SAM CLI は、[.yaml|.yml] ファイル形式をサポートするようになりました。「AWS SAM CLI の設定」と「AWS SAM CLI 設定ファイル」のページが更新されました。	2023 年 7 月 18 日
Terraform の AWS SAM CLIsam local start-api コマンドのサポートが追加されました	「Terraform の AWS SAM CLI サポートの概要」セクションが更新され、Terraform の AWS SAM CLI `sam local start-api` コマンドサポートが新たに記載されました。	2023 年 7 月 6 日
新しい AWS SAM CLI remote invoke コマンドが追加されました	`sam remote invoke` の使用を開始するには、「sam remote invoke の使用」を参照してください。	2023 年 6 月 22 日
AWS AppSyncGraphQL API サーバーレスリソースタイプが追加されました	AWS SAM を使用して GraphQL API リソースを定義する方法を説明する「`AWS::Serverless::GraphQLApi`」セクションを作成しました。	2023 年 6 月 22 日
Ruby 3.2 の AWS SAM CLI サポートが追加されました	「sam init」ページを更新して、新しいベースイメージとランタイム値を追加しました。Ruby 3.2 Amazon ECR URI で「イメージリポジトリ」ページを更新しました。	2023 年 6 月 6 日

AWS SAM CLI パッケージインストーラの整合性検証におけるオプションの手順が追加されました	「AWS SAM CLI のインストール」ページを更新して、オプションの手順を反映します。手順を文書化するために、「AWS SAM CLI インストーラの整合性を確認」ページを作成します。	2023 年 5 月 31 日
インフラストラクチャの同期をスキップする sam sync オプションが追加されました	`sam sync` を実行するたびに AWS CloudFormation デプロイが必要かどうかをカスタマイズします。詳細については、「初期 AWS CloudFormation デプロイをスキップする」を参照してください。	2023 年 3 月 23 日
DocumentDB イベントソースタイプのサポートが追加されました	AWS SAM テンプレート仕様は、`AWS::Serverless::Function` リソース向けに `DocumentDB` イベントソースタイプをサポートするようになりました。詳細については、「DocumentDB」を参照してください。	2023 年 3 月 10 日
Cargo Lambda を使用して Rust Lambda 関数を構築する	AWS SAM CLI を使用して Cargo Lambda で Rust Lambda 関数を構築します。詳細については、「Cargo Lambda を使用した Rust Lambda 関数の構築」を参照してください。	2023 年 2 月 23 日

関数リソースを AWS SAM の外部に構築する	sam build コマンド使用時の関数のスキップに関するガイダンスを追加しました。詳細については、「AWS SAM の外部で関数を構築する」を参照してください。	2023 年 2 月 14 日
新しい埋め込みコネクタ構文	`AWS::Serverless::Connector` リソースを定義するために、新しい埋め込みコネクタ構文を使用します。詳細については、「AWS SAM コネクタによるリソースに対するアクセス許可の管理」を参照してください。	2023 年 2 月 8 日
AWS SAM CLI 用に新しい sam list コマンドを追加した	サーバーレスアプリケーションのリソースに関する重要な情報を表示するために sam list を使用します。詳細については、「sam リスト」を参照してください。	2023 年 2 月 2 日
esbuild の Format および OutExtension ビルドプロパティを追加	esbuild を使用した Node.js Lambda 関数の構築で、`Format` および `OutExtension` ビルドプロパティがサポートされるようになりました。詳細については、「esbuild による Node.js Lambda 関数の構築」を参照してください。	2023 年 2 月 2 日

AWS SAM テンプレートの仕様にランタイム管理オプションを追加	Lambda 関数のランタイム管理オプションを設定します。詳細については、「RuntimeManagementConfig 」を参照してください。	2023 年 1 月 24 日
AWS::Serverless::StateMachine リソースの Target プロパティに EventSource を追加。	AWS::Serverless::StateMachine リソースタイプは、Target プロパティの EventBridgeRule および Schedule イベントソースをサポートします。	2023 年 1 月 13 日
Lambda 関数の SQS ポーラーのスケーリングを設定する	ScalingConfig プロパティを使用して AWS::Serverless::Function に SQS ポーラーのスケーリングを設定します。詳細については、「 ScalingConfig 」を参照してください。	2023 年 1 月 12 日
cfn-lint を使用して AWS SAM アプリケーションを検証する	AWS SAM CLI で cfn-lint を使用すると、AWS SAM テンプレートを検証できます。詳細については、「cfn-lint による検証」を参照してください。	2023 年 1 月 11 日
CloudWatch Application Insights を使用したサーバーレスアプリケーションのモニタリング	AWS SAM アプリケーションをモニタリングするように Amazon CloudWatch Application Insights を設定します。詳細については、「CloudWatch Application Insights を使用したサーバーレスアプリケーションのモニタリング」を参照してください。	2022 年 12 月 19 日

macOS 用の AWS SAM CLI パッケージインストーラを追加した	新しい macOS パッケージインストーラを使用して AWS SAM CLI をインストールします。詳細については、「AWS SAM CLI のインストール」を参照してください。	2022 年 12 月 6 日
Lambda SnapStart のサポートを追加	スナップショットを作成する SnapStart を Lambda 関数に設定します。このスナップショットには、初期化された関数がキャッシュされます。詳細については、「`AWS::Serverless::Function`」を参照してください。	2022 年 11 月 28 日
nodejs18.x の AWS SAM CLI サポートを追加した	AWS SAM CLI で nodejs18.x ランタイムがサポートされるようになりました。詳細については、「sam init」を参照してください。	2022 年 11 月 17 日
アクセスとアクセス許可の設定に関するガイダンスを追加	AWS SAM には、サーバーレスアプリケーションのアクセスとアクセス許可の管理をシンプルにする 2 つのオプションが用意されています。詳細については、「リソースへのアクセスとアクセス許可の管理」を参照してください。	2022 年 11 月 17 日

ネイティブ AOT コンパイルによる .NET 7 Lambda 関数の構築のサポートを追加	.NET 7 Lambda 関数を AWS SAM で構築してパッケージ化し、ネイティブの事前 (AOT) コンパイルを利用して Lambda のコールドスタート時間を短縮します。詳細については、「ネイティブ AOT コンパイルによる .NET 7 Lambda 関数の構築」を参照してください。	2022 年 11 月 15 日
ローカルでのデバッグとテストための AWS SAM CLITerraform サポートを追加した	Terraform プロジェクト内で AWS SAM CLI を使用して、ローカルで Lambda 関数とレイヤーのデバッグやテストを実行します。詳細については、「AWS SAM CLI Terraform のサポート」を参照してください。	2022 年 11 月 14 日
EventBridge スケジューラに AWS SAM のサポートを追加	AWS Serverless Application Model (AWS SAM) テンプレートの仕様には、EventBridge スケジューラを使用して AWS Lambda や AWS Step Functions のイベントをスケジュールできる、シンプルで簡潔な構文が用意されています。詳細については、「EventBridge スケジューラによるイベントのスケジューリング」を参照してください。	2022 年 11 月 10 日

AWS SAM CLI のインストール手順を簡略化する	AWS SAM CLI の前提条件とオプションの手順は、別のページに移動されました。サポートされているオペレーティングシステムのインストール手順は、「AWS SAM CLI のインストール」に記載されています。	2022 年 11 月 4 日
長いパスを使用できるようにする Windows 10 ユーザー向けの修正を追加	AWS SAM CLI アプリテンプレートリポジトリには、Windows 10 での MAX_PATH の制限により、`sam init` の実行時にエラーを引き起こす可能性のある長いファイルパスがいくつか含まれています。詳細については、「AWS SAM CLI のインストール」を参照してください。	2022 年 11 月 4 日
初回デプロイ時の段階的デプロイプロセスの更新	AWS CodeDeploy で Lambda 関数を段階的にデプロイするには、2 つのステップが必要です。詳細については、「初めて Lambda 関数を段階的にデプロイする」を参照してください。	2022 年 10 月 13 日
より多くの種類のイベントに対応する Lambda イベントフィルタリングの追加サポート	`MSK`、`MQ`、および `SelfManagedKafka` イベントソースタイプに `FilterCritera` プロパティが追加されました。	2022 年 10 月 13 日

AWS SAM パイプラインへの OpenID Connect (OIDC) のサポートの追加	AWS SAM は、Bitbucket、GitHub Actions、および GitLab の継続的インテグレーションおよび継続的デリバリー (CI/CD) のプラットフォームに対して OpenID Connect (OIDC) ユーザー認証をサポートしています。詳細については、「AWS SAM パイプラインで OIDC ユーザーアカウントを使用する」を参照してください。	2022 年 10 月 13 日
jwtConfiguration プロパティに関する注意	`OAuth2Authorizer` の `JwtConfiguration` で `issuer` プロパティおよび `audience` プロパティの定義に関する注意を追加しました。	2022 年 10 月 7 日
関数と StateMachine EventSource の新しいプロパティ	`AWS::Serverless::Function` の `CloudWatchEvent` イベントソースに `Enabled` プロパティおよび `State` プロパティが追加されました。`AWS::Serverless::Function` および `AWS::Serverless::StateMachine` の `Schedule` イベントソースに `State` プロパティが追加されました。	2022 年 10 月 6 日

AWS SAM コネクタの一般公開	コネクタは、`AWS::Serverless::Connector` として識別された AWS SAM 抽象リソースタイプで、サーバーレスアプリケーションリソース間のアクセス許可を簡単かつ安全にプロビジョニングする方法を提供します。詳細については、「AWS Serverless Application Model コネクタによるリソースに対するアクセス許可の管理」を参照してください。	2022 年 10 月 6 日
AWS SAM CLI への新しい sam 同期オプションを追加した	sam sync に `--dependency-layer` および `--use-container` オプションが追加されました。	2022 年 9 月 20 日
AWS SAM CLI への新しい sam デプロイオプションを追加した	sam deploy に `--on-failure` オプションが追加されました。	2022 年 9 月 9 日
esbuild の一般公開	Node.js Lambda 関数を構築してパッケージ化する場合、AWS SAM CLI と esbuild JavaScript バンドラーを使用できます。	2022 年 9 月 1 日
AWS SAM CLI テレメトリの更新	収集されたシステムおよび環境情報の説明が更新され、使用属性のハッシュ値を含むようになりました。	2022 年 9 月 1 日

AWS SAM CLI へのローカル環境変数のサポートを追加した	Lambda 関数を口ーカルで呼び出すときと API Gateway をローカルで実行するときは、AWS SAM CLI で環境変数を使用します。	2022 年 9 月 1 日
Lambda 命令セットアーキテクチャのサポート	AWS SAM CLI を使用して、x86_64 または arm64 命令セットアーキテクチャの Lambda 関数と Lambda レイヤーを構築します。詳細については、AWS::Serverless::Function リソースタイプの Architectures プロパティおよび AWS::Serverless::LayerVersion リソースタイプの CompatibleArchitectures プロパティを参照してください。	2021 年 10 月 1 日
サンプルパイプライン設定の生成	AWS SAM CLI を使用して、複数の CI/CD システム用サンプルパイプラインを生成します。これには、新しい sam pipeline bootstrap と sam pipeline init コマンドを使用します。詳細については、「サンプル CI/CD パイプラインの生成」を参照してください。	2021 年 7 月 21 日

AWS SAM CLIAWS CDK の統合 (プレビュー、フェーズ 2)	パブリックプレビューリリースのフェーズ 2 では、AWS CDK アプリケーションのパッケージ化とデプロイに AWS SAM CLI を使用できるようになりました。AWS SAM CLI を使用して、サンプル AWS CDK アプリケーションを直接ダウンロードすることもできます。詳細については、「AWS Cloud Development Kit (AWS CDK) (プレビュー)」を参照してください。	2021 年 7 月 13 日
関数のイベントソースとしての RabbitMQ のサポート	サーバーレス関数のイベントソースとしての RabbitMQ のサポートが追加されました。詳細については、「`AWS::Serverless::Function`」リソースタイプの `MQ` イベントソースの「`SourceAccessConfigurations`」プロパティを参照してください。	2021 年 7 月 7 日
Amazon ECR ビルドコンテナイメージを使用したサーバーレスアプリケーションのデプロイ	Amazon ECR ビルドコンテナーイメージを使用して、AWS CodePipeline、Jenkins、GitLab CI/CD、および GitHub Actions などの一般的な CI/CD システムでサーバーレスアプリケーションをデプロイします。詳細については、「サーバーレスアプリケーションのデプロイ」を参照してください。	2021 年 6 月 24 日

AWS ツールキットを使用した AWS SAM アプリケーションのデバッグ	AWS Toolkit が、統合開発環境 (IDE) とランタイムのより多くの組み合わせでのステップスルーデバッグをサポートするようになりました。詳細については、「AWS Toolkit の使用」を参照してください。	2021 年 5 月 20 日
AWS SAM CLIAWS CDK の統合 (プレビュー)	AWS SAM CLI を使用して AWS CDK アプリケーションをローカルにテストし、構築できるようになりました。これはパブリックプレビューリリースです。詳細については、「AWS Cloud Development Kit (AWS CDK) (プレビュー)」を参照してください。	2021 年 4 月 29 日
デフォルトコンテナイメージリポジトリの Amazon ECR Public への変更	デフォルトコンテナイメージリポジトリが DockerHub から Amazon ECR Public に変更されました。詳細については、「イメージリポジトリ」を参照してください。	2021 年 4 月 6 日
夜間 AWS SAM CLIビルド	夜間に構築される、プレリリースバージョンの AWS SAM CLI のインストールが可能になりました。詳細については、「AWS SAM CLI のインストール」で、選択する OS サブトピックの「ナイトリービルド」セクションを参照してください。	2021 年 3 月 25 日

ビルドコンテナの環境変数のサポート	ビルドコンテナに環境変数を渡すことが可能になりました。詳細については、「sam build」の --container-env-var と --container-env-var-file オプションを参照してください。	2021 年 3 月 4 日
新しい Linux インストールプロセス	ネイティブ Linux インストーラを使用して AWS SAM CLI をインストールできるようになりました。詳細については、「Linux への AWS SAM CLI のインストール」を参照してください。	2021 年 2 月 10 日
EventBridge 向けのデッドレターキューのサポート	サーバーレス関数とステートマシンの EventBridge および Schedule イベントソース向けのデッドレターキューのサポートが追加されました。詳細については、「AWS::Serverless::Function」と「AWS::Serverless::StateMachine」リソースタイプの両方について、EventBridgeRule および Schedule イベントソースの DeadLetterConfig プロパティを参照してください。	2021 年 1 月 29 日

カスタムチェックポイントのサポート	サーバーレス関数の DynamoDB および Kinesis イベントソースのためのカスタムチェックポイントのサポートが追加されました。詳細については、「AWS::Serverless::Function」リソースタイプの「Kinesis」と「DynamoDB」データ型の FunctionResponseTypes プロパティを参照してください。	2021 年 1 月 29 日
タンブリングウィンドウのサポート	サーバーレス関数の DynamoDB および Kinesis イベントソースのためのタンブリングウィンドウのサポートが追加されました。詳細については、「AWS::Serverless::Function」リソースタイプの「Kinesis」と「DynamoDB」データ型の TumblingWindowInSeconds プロパティを参照してください。	2020 年 12 月 17 日
ウォームコンテナのサポート	AWS SAM CLI コマンドの sam local start-api および sam local start-lambda を使用したローカルでのテスト時におけるウォームコンテナのサポートが追加されました。詳細については、これらのコマンドの --warm-containers オプションを参照してください。	2020 年 12 月 16 日

Lambda コンテナイメージのサポート	Lambda コンテナイメージのサポートが追加されました。詳細については、「アプリケーションの構築」を参照してください。	2020 年 12 月 1 日
コード署名のサポート	サーバーレスアプリケーションコードのコード署名と信頼できるデプロイのサポートが追加されました。詳細については、「AWS SAM アプリケーションのためのコード署名の設定」を参照してください。	2020 年 11 月 23 日
並列ビルドとキャッシュビルドのサポート	関数とレイヤーを順次的ではなく並列的に構築する `--parallel` オプション、および再構築を必要とする変更が行われなかった場合に以前のビルドからのビルドアーティファクトを使用する `--cached` オプションの 2 つのオプションを `sam build` コマンドに追加することによって、サーバーレスアプリケーションビルドのパフォーマンスを向上させました。	2020 年 11 月 10 日

Amazon MQ と相互 TLS 認証のサポート	サーバーレス関数のイベントソースとしての Amazon MQ のサポートが追加されました。詳細については、「AWS::Serverless::Function」リソースタイプの「EventSource」および「MQ」データ型を参照してください。API Gateway API および HTTP API のための相互 Transport Layer Security (TLS)認証のサポートも追加されました。詳細については、「AWS::Serverless::Api」リソースタイプの「DomainConfiguration」データ型、または「AWS::Serverless::HttpApi」リソースタイプの「HttpApiDomainConfiguration」データ型を参照してください。	2020 年 11 月 5 日
HTTP API 用の Lambda オーソライザーのサポート	AWS::Serverless::HttpApi リソースタイプ用の Lambda オーソライザーのサポートが追加されました。詳細については、「Lambda オーソライザーの例 (AWS::Serverless::HttpApi)」を参照してください。	2020 年 10 月 27 日

複数の設定ファイルと環境のサポート	AWS SAM CLI コマンドのデフォルトパラメータ値を保存するための複数の設定ファイルと環境のサポートが追加されました。詳細については、「AWS SAM CLI 設定ファイル」を参照してください。	2020 年 9 月 24 日
X-Ray の Step Functions との使用、および API へのアクセスを制御するときのリファレンスのサポート	サーバーレスステートマシンのイベントソースとしての X-Ray のサポートが追加されました。詳細については、「AWS::Serverless::StateMachine」リソースタイプの「Tracing」プロパティを参照してください。API へのアクセスの制御時におけるリファレンスのサポートも追加されました。詳細については、「ResourcePolicyStatement」データ型を参照してください。	2020 年 9 月 17 日
Amazon MSK のサポート	サーバーレス関数のイベントソースとしての Amazon MSK のサポートが追加されました。これは、Amazon MSK トピックのレコードが Lambda 関数をトリガーすることを可能にします。詳細については、「AWS::Serverless::Function」リソースタイプの「EventSource」および「MSK」データ型を参照してください。	2020 年 8 月 13 日

Amazon EFS のサポート	ローカルディレクトリへの Amazon EFS ファイルシステムのマウントのサポートが追加されました。これは、Lambda 関数コードが共有リソースにアクセスして変更を行うことを可能にします。詳細については、「AWS::Serverless::Function」リソースタイプの「FileSystemConfigs」プロパティを参照してください。	2020 年 6 月 16 日
サーバーレスアプリケーションのオーケストレーション	AWS SAM を使用した Step Functions ステートマシンの作成による、アプリケーションのオーケストレーションのサポートが追加されました。詳細については、「AWS Step Functions を使用した AWS リソースのオーケストレーション」と「AWS::Serverless::StateMachine」リソースタイプを参照してください。	2020 年 5 月 27 日
カスタムランタイムの構築	カスタムランタイムを構築する機能が追加されました。詳細については、「カスタムランタイムの構築」を参照してください。	2020 年 5 月 21 日

レイヤーの構築	個々の `LayerVersion` リソースを構築する機能が追加されました。詳細については、「レイヤーの構築」を参照してください。	2020 年 5 月 19 日
生成された AWS CloudFormation リソース	AWS SAM が生成する AWS CloudFormation リソースと、それらを参照する方法の詳細情報を掲載しました。詳細については、「生成された AWS CloudFormation リソース」を参照してください。	2020 年 4 月 8 日
AWS 認証情報のセットアップ	AWS SDK の 1 つ、または AWS CLI などの AWS のその他ツールでの使用向けの AWS がセットアップされていない場合のために、それらをセットアップする手順を追加しました。詳細については、「AWS 認証情報のセットアップ」を参照してください。	2020 年 1 月 17 日
AWS SAM の仕様と AWS SAM CLI の更新	AWS SAM の仕様が GitHub から移行されました。詳細については、「AWS SAM の仕様」を参照してください。`sam deploy` コマンドへの変更に伴い、デプロイワークフローも更新されました。	2019 年 11 月 25 日

API Gateway API へのアクセスの制御に対する新しいオプションとポリシーテンプレートの更新	API Gateway API へのアクセスを制御するための新しいオプションである IAM 許可、API キー、およびリソースポリシーが追加されました。詳細については、「API Gateway API へのアクセスの制御」を参照してください。RekognitionFacesPolicy と ElasticsearchHttpPostPolicy の 2 つのポリシーテンプレートも更新されました。詳細については、「AWS SAM ポリシーテンプレート」を参照してください。	2019 年 8 月 29 日
開始方法の更新	AWS SAM CLI および Hello World チュートリアルの改善されたインストール手順で「開始方法」章が更新されました。詳細については、「AWS SAM の使用開始」を参照してください。	2019年7月25日
API Gateway API へのアクセスの制御	API Gateway API へのアクセスの制御に対するサポートが追加されました。詳細については、「API Gateway API へのアクセスの制御」を参照してください。	2019 年 3 月 21 日

sam publish を AWS SAM CLI に追加した	AWS SAM CLI の新しい sam publish コマンドは、AWS Serverless Application Repository でサーバレスアプリケーションを公開するプロセスをシンプル化します。詳細については、「AWS SAM CLIを使用したサーバーレスアプリケーションの公開」を参照してください。	2018 年 12 月 21 日
ネストされたアプリケーションとレイヤーのサポート	ネストされたアプリケーションとレイヤーのサポートを追加しました。詳細については、「ネストされたアプリケーション」と「レイヤーの使用」を参照してください。	2018 年 11 月 29 日
sam build を AWS SAM CLI に追加した	AWS SAM CLI の新しい sam build コマンドは、依存関係を持つサーバーレスアプリケーションをコンパイルするプロセスをシンプル化して、これらのアプリケーションをローカルでテストおよびデプロイできるようにします。詳細については、「アプリケーションの構築」を参照してください。	2018 年 11 月 19 日

AWS SAM CLI の新しいインストールオプションを追加した	AWS SAM CLI に Linuxbrew (Linux)、MSI (Windows)、および Homebrew (macOS) のインストールオプションが追加されました。詳細については、「AWS SAM CLI のインストール」を参照してください。	2018 年 11 月 7 日
新しいガイド	これは AWS Serverless Application Model デベロッパーガイドの初回リリースです。	2018 年 10 月 17 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。