



開発者ガイド

AWS Serverless Application Model



AWS Serverless Application Model: 開発者ガイド

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

Amazon の商標とトレードドレスは、Amazon 以外の製品またはサービスとの関連において、顧客に混乱を招いたり、Amazon の名誉または信用を毀損するような方法で使用することはできません。Amazon が所有しない他の商標はすべてそれぞれの所有者に帰属します。所有者は必ずしも Amazon との提携や関連があるわけではありません。また、Amazon の支援を受けているとはかぎりません。

Table of Contents

とは AWS SAM	1
主な特徴	1
関連情報	2
仕組み	2
AWS SAM テンプレートの仕様は何ですか？	3
AWS SAM プロジェクトと AWS SAM テンプレートとは	3
とは AWS SAMCLI	10
詳細はこちら	17
次のステップ	18
サーバーレスの概念	18
サーバーレスの概念	18
開始	20
前提条件	20
ステップ 1: AWS アカウントにサインアップする	21
ステップ 2: IAM ユーザーを作成します	21
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	22
ステップ 4: をインストールする AWS CLI	24
ステップ 5: AWS CLI を使用して AWS 認証情報を設定する	24
次のステップ	25
AWS SAM CLI のインストール	25
AWS SAM CLIのインストール	26
インストールエラーのトラブルシューティング	37
次のステップ	38
オプション: AWS SAMCLIインストーラを検証する	39
Hello World のチュートリアル	51
前提条件	53
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	53
ステップ 2: アプリケーションを構築する	57
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	58
ステップ 4: アプリケーションを実行する	63
ステップ 5: で関数を操作する AWS クラウド	65
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	65
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	69
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	71

トラブルシューティング	72
詳細はこちら	72
の使用方法 AWS SAM	73
AWS SAMCLI	73
AWS SAM CLI コマンドを文書化する方法	74
AWS SAM CLI の設定	75
Core コマンド	81
AWS SAM プロジェクト	83
テンプレートの構造分析	84
リソースとプロパティ	93
生成された リソース	413
サポートされているリソース属性	431
API Gateway 拡張機能	433
組み込み関数	434
アプリケーションの開発	436
アプリケーションを作成する	436
新しいサーバーレスアプリケーションを初期化する	437
sam init のオプション	443
トラブルシューティング	443
例	444
詳細はこちら	444
次のステップ	445
インフラストラクチャを定義する	445
アプリケーションリソースの定義	445
アクセスの設定	447
API アクセスの制御	529
レイヤーによる効率の向上	542
コードの再利用	545
時間ベースのイベントを管理する	549
アプリケーションのオーケストレーション	552
コード署名の設定	553
AWS SAM テンプレートファイルを検証する	557
アプリケーションを構築する	557
の概要 sam build	558
を使用したデフォルトビルド AWS SAM	572
ビルドをカスタマイズする	580

アプリケーションをテストする	606
の概要 sam local	606
sam local コマンドの使用	607
の概要 sam local generate-event	607
の概要 sam local invoke	614
の概要 sam local start-api	620
の概要 sam local start-lambda	625
関数をローカルに呼び出す	628
環境変数ファイル	629
レイヤー	631
詳細はこちら	631
API Gateway をローカルで実行する	631
環境変数ファイル	632
レイヤー	634
でテストする sam remote test-event	634
AWS SAM CLI で sam remote test-event を使用するようにセットアップする	635
sam remote test-event コマンドの使用	635
共有可能なテストイベントを使用する	638
共有可能なテストイベントを管理する	638
でテストする sam remote invoke	640
sam remote invoke コマンドの使用	641
sam リモート呼び出しコマンド オプションの使用	645
プロジェクト設定ファイルを設定する	651
例	651
関連リンク	667
統合テストの自動化	667
サンプルペイロードの生成	669
アプリケーションのデバッグ	670
ローカルデバッグ関数	670
AWS ツールキットの使用	671
デバッグモードで AWS SAM のローカル実行	673
複数のランタイム引数を渡す	674
cfn-lint による検証	674
例	675
詳細はこちら	675
アプリケーションとリソースをデプロイする	676

の概要 sam deploy	676
前提条件	677
sam deploy を使用したアプリケーションのデプロイ	677
ベストプラクティス	687
sam deploy のオプション	687
トラブルシューティング	688
例	688
詳細はこちら	696
デプロイオプション	696
を使用して AWS SAM CLI 手動でデプロイする方法	697
CI/CD システムとパイプラインを使用してデプロイする	697
段階的なデプロイ	698
AWS SAM CLI を使用したデプロイのトラブルシューティング	698
詳細はこちら	631
CI/CD システムとパイプラインを使用してデプロイする	699
パイプラインとは	700
スターターパイプラインを生成する	700
スターターパイプラインをカスタマイズする	706
デプロイを自動化する	708
OIDC 認証を使用する	712
デプロイ時にローカルファイルをアップロード	715
の概要 sam sync	723
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	724
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	725
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	726
sam sync コマンドのオプション	726
トラブルシューティング	728
例	729
詳細はこちら	735
アプリケーションをモニタリングする	736
Application Insights	736
を使用した CloudWatch Application Insights の設定 AWS SAM	736
次のステップ	740
ログの使用	740
AWS CloudFormation スタックによるログの取得	741
Lambda 関数名によるログの取得	741

ログ終端の表示	741
特定時間範囲のログの表示	741
ログのフィルタリング	741
エラーの強調表示	742
JSON の整形出力	742
AWS SAM リファレンス	743
AWS SAM 仕様と AWS SAM テンプレート	743
AWS SAM CLI コマンドリファレンス	743
AWS SAM ポリシーテンプレート	744
トピック	744
AWS SAMCLI コマンド	744
sam build	745
sam delete	751
sam deploy	752
sam init	758
sam list	761
sam local generate-event	769
sam local invoke	771
sam local start-api	776
sam local start-lambda	781
sam logs	785
sam package	788
sam pipeline bootstrap	792
sam pipeline init	796
sam publish	798
sam remote invoke	800
sam remote test-event	805
sam sync	811
sam traces	818
sam validate	819
AWS SAMCLI 管理	821
AWS SAM CLI 設定ファイル	821
AWS SAM CLI バージョンの管理	828
AWS 認証情報のセットアップ	838
AWS SAM CLI テレメトリ	839
トラブルシューティング	842

コネクタリファレンス	848
サポートされているコネクタリソースタイプ	848
コネクタによって作成された IAM ポリシー	858
Docker のインストール	881
Docker のインストール	882
次のステップ	885
イメージリポジトリ	885
イメージリポジトリ URI	886
例	888
段階的なデプロイ	888
初めて Lambda 関数を段階的にデプロイする	891
詳細はこちら	892
重要な注意事項	892
2023	893
2020	893
サンプルアプリケーション	895
DynamoDB イベントを処理する	895
開始する前に	895
ステップ 1: アプリケーションを初期化する	895
ステップ 2: アプリケーションのローカルでテストする	896
ステップ 3: アプリケーションをパッケージ化する	896
ステップ 4: アプリケーションをデプロイする	897
次のステップ	898
Amazon S3 イベントを処理する	898
開始する前に	898
ステップ 1: アプリケーションを初期化する	899
ステップ 2: アプリケーションをパッケージ化する	899
ステップ 3: アプリケーションをデプロイする	900
ステップ 4: アプリケーションをローカルでテストする	901
次のステップ	901
Terraform サポート	902
AWS SAM CLI Terraform のサポート	902
とは AWS SAMCLI	903
Terraform と連携させた AWS SAM CLI をどのように使用しますか？	903
次のステップ	904
開始	904

前提条件	904
Terraform と連携させた AWS SAM CLI	905
Terraform プロジェクト用のセットアップ	905
Terraform Cloud のセットアップ	911
Terraform と連携させた AWS SAM CLI を使用する	912
sam local invoke によるローカルテスト	913
sam local start-api によるローカルテスト	913
sam local start-lambda によるローカルテスト	915
Terraform の制限事項	915
AWS SAM CLI と Serverless.tf の使い方	916
Terraform リファレンス	916
AWS SAM サポートされている機能のリファレンス	917
Terraform 特有のリファレンス	917
sam metadata	917
AWS CDK サポート	921
開始	921
前提条件	921
AWS CDK アプリケーションの作成とローカルでのテスト	922
ローカルでのテスト	924
例	925
構築	926
例	926
デプロイ	927
他のユーザーが使用するための公開	928
前提条件	928
新しいアプリケーションの公開	930
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	930
ステップ 2: アプリケーションをパッケージ化する	930
ステップ 3: アプリケーションを公開する	931
ステップ 4: アプリケーションを共有する (オプション)	931
既存アプリケーションの新しいバージョンの公開	932
その他のトピック	932
メタデータセクションのプロパティ	932
プロパティ	932
ユースケース	935
例	936

ドキュメント履歴	938
.....	cmlxiv

AWS Serverless Application Model (AWS SAM) とは

AWS Serverless Application Model (AWS SAM) は、Infrastructure as Code (IaC) を使用してサーバーレスアプリケーションを構築するためのオープンソースフレームワークです。AWS SAMの短縮構文を使用すると、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](#)リソースと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 と AWS SAM プロジェクトの 2 つの主要コンポーネントが含まれています。AWS SAM プロジェクトは、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 sam init。AWS SAM プロジェクトには、AWS SAM テンプレート仕様 (リソースの宣言に使用する短縮構文) を含む テンプレートのようなファイルが含まれています。

主な特徴

AWS SAM には、開発者エクスペリエンスを向上させるさまざまな利点があります。

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 に直接デプロイ AWS CloudFormation 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 SAM CLI](#)」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アクセス許可をすばやく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 リソース間のアクセス許可を定義します。は、インテントを容易にするために必要な IAM アクセス許可にコードを AWS SAM 変換します。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

sam sync コマンドを使用してローカルの変更を AWS SAM CLI クラウドに自動的に同期し、開発ワークフローとクラウドテストワークフローを高速化します。詳細については、「[を使用して sam sync に同期する方法の概要 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 CLITerraform のサポート](#)」を参照してください。

関連情報

- の AWS SAM 仕組みについては、「」を参照してください[AWS SAM の仕組み](#)。
- の使用を開始するには AWS SAM、「」を参照してください[の開始方法 AWS SAM](#)。
- 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、「」を参照してください[の使用方法 AWS SAM](#)。

AWS SAM の仕組み

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 つの主要コンポーネントで構成されています。

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。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。これは、サーバーレスアプリケーションの関数、イベント、APIs、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を備えたオープンソースフレームワークです。
2. [AWS SAM CLI](#) – サーバーレスアプリケーションを構築して実行するために、AWS SAM プロジェクトおよび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。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現するには、リソースを定義し、AWS SAM プロジェクト内の AWS SAM テンプレートやその他のファイルでアプリケーションを開発します。を使用して、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ある AWS SAM プロジェクトでコマンド AWS SAM CLI を実行します。

サーバーレスは初めてですか？

を確認することをお勧めします[サーバーレスの概念](#)。

AWS SAM テンプレートの仕様は何ですか？

AWS SAM テンプレート仕様は、サーバーレスアプリケーションインフラストラクチャコードの定義と管理に使用できるオープンソースフレームワークです。AWS SAM テンプレートの仕様は次のとおりです。

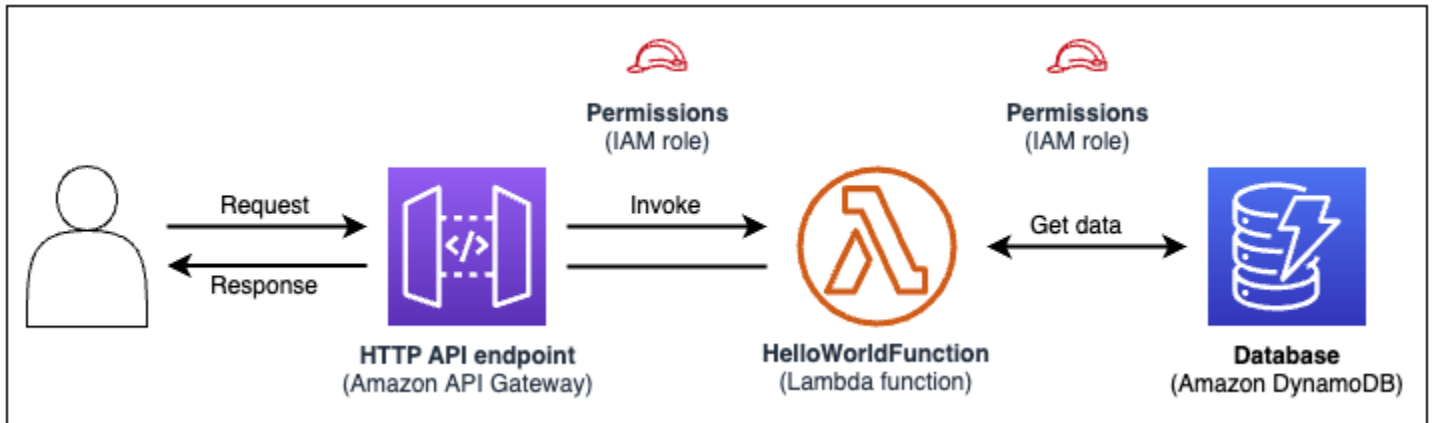
- **で構築** AWS CloudFormation — テンプレートで AWS CloudFormation 構文を直接使用し AWS SAM、リソースとプロパティの設定の広範なサポートを活用します。に既に慣れている場合は AWS CloudFormation、アプリケーションインフラストラクチャコードを管理するために新しいサービスを学ぶ必要はありません。
- **の拡張** AWS CloudFormation- サーバーレス開発の高速化に特に重点を置いた独自の構文 AWS SAM を提供します。同じテンプレート内で AWS CloudFormation 構文と AWS SAM 構文の両方を使用できます。
- **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** — テンプレートを を介し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変換するという複雑な作業を行います AWS CloudFormation。 AWS SAM

AWS SAM プロジェクトと 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ています。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 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あり AWS、操作を容易にする追加コンポーネントがいくつかあります。この意味では、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: nodejs12.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 サービスを使用する関数。
- 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やり取りするために必要な AWS Identity and Access Management (IAM) アクセス許可。

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、は 23 行のコードを、でこれらのリソースを生成するために必要な AWS CloudFormation 構文 AWS SAM に変換します AWS。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"
    }
  ]
}
}
```

を使用すると AWS SAM、23 行のインフラストラクチャコードを定義します。は、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 コードにコードを AWS SAM 変換します。

とは AWS SAMCLI

AWS SAMCLI 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統合でサーバーレスアプリケーションを構築して実行するために使用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 SAMCLI は、AWS SAM および AWS CloudFormation テンプレートと併用する場合に最適です。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、「[sam init コマンドを使用して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。

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

Build Succeeded

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

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

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。

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

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

詳細については、「[アプリケーションをテストする](#)」および「[アプリケーションのデバッグ](#)」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設定し、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。は、アプリケーションのデプロイ設定を構成する手順、テンプレートを変換する AWS SAMCLI手順 AWS CloudFormation、に AWS CloudFormation 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。は、 オプションを AWS SAMCLIガイドし、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 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]: 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てデプロイする](#)」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。

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

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。

```
-----  
Key           HelloWorldFunctionIamRole  
Description   Implicit IAM Role created for Hello World function  
Value        arn:aws:iam::513423067560:role/sam-app-HelloWorldFunctionRole-15GLOUR9LMT1W  
  
Key           HelloWorldApi  
Description   API Gateway endpoint URL for Prod stage for Hello World function  
Value        https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Prod/hello/  
  
Key           HelloWorldFunction  
Description   Hello World Lambda Function ARN  
Value        arn:aws:lambda:us-west-2:513423067560:function:sam-app-HelloWorldFunction-  
yQDNe17r9maD  
-----
```

```
Stack update succeeded. Sync infra completed.
```

```
Infra sync completed.
```

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

```
Syncing Lambda Function HelloWorldFunction...
```

```
Manifest is not changed for (HelloWorldFunction), running incremental build
```

```
Building codeuri: /Users/evzz/Demo/what-is/sam-app/hello_world runtime: python3.9 metadata: {} architecture: x86_64 functions: HelloWorldFunction
```

```
Running PythonPipBuilder:CopySource
```

```
Finished syncing Lambda Function HelloWorldFunction.
```

```
□
```

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

について学習を続けるには AWS SAM、次のリソースを参照してください。

- [完全な AWS SAM ワークショップ](#) — AWS SAM が提供する主要な機能の多くを学習するために設計されたワークショップ。
- [SAM を使用したセッション](#) — を使用して で Serverless Developer Advocate チームによって AWS 作成された動画シリーズ AWS SAM。

- [Serverless Land](#) – AWS サーバーレスの最新情報、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

を初めて使用する場合は AWS SAM、 「」を参照してくださいの[開始方法 AWS SAM](#)。

サーバーレスの概念

AWS Serverless Application Model () を使用する前に、サーバーレスの基本概念について説明します AWS SAM。

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 AWS Lambda 用の やデータベース管理用の Amazon DynamoDB などの個々の AWS サービスで構成され、それぞれが特殊なロールを実行します。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、 にデプロイして AWS、 リソース AWS を作成します。IaC では、プロビジョニングする内容をコードで定義 AWS します。詳細については、 ホワイトペーパーの「入門」の DevOps AWS AWS [「コードとしてのインフラストラクチャ」](#)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 によって行われるため AWS、 自動スケーリングや組み込みの高可用性など多くの利点があり、アイデアを迅速に本番環境に持ち込むことができます。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。

- [でのサーバーレス AWS](#)
- [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。

コア 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「Serverless 101: Understanding the serverless services」](#)を参照してください。

の開始方法 AWS SAM

このセクションのトピックを確認して完了 AWS SAM することで、 の使用を開始します。 [AWS SAM 前提条件](#)では、 AWS アカウントのセットアップ、 IAM ユーザーの作成、 キーアクセスの作成、 AWS SAM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しますCLI。前提条件を満たしたら、 の準備が整います。これは [AWS SAM CLI のインストール](#)Linux、Windows、macOS オペレーティングシステムで実行できます。インストールが完了したら、 オプションで Hello World AWS SAM チュートリアルを順を追って確認できます。このチュートリアルでは、 を使用して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 AWS SAM。チュートリアルを完了すると、「」で説明されている概念を確認する準備が整います[AWS Serverless Application Model \(AWS SAM\) の使用方法](#)。

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: Hello World アプリケーションをデプロイする](#)

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 () をインストールして使用する前に、次の前提条件を満たしてくださいAWS SAMCLI。

を使用するには AWS SAMCLI、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 ユーザーを作成します](#)
- [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](#)
- [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、次の手順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/signup> 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 ユーザーを作成します

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	AWS IAM Identity Center ユーザーガイドの「 開始方法 」の手順に従います。	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。AWS Command Line Interface

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
	<p>詳細については、IAM ユーザーガイドの「IAM でのセキュリティのベストプラクティス」を参照してください。</p>		
IAM 内 (非推奨)	<p>長期認証情報を使用して AWS にアクセスする。</p>	<p>IAM ユーザーガイドの「最初の IAM 管理者のユーザーおよびグループの作成」の手順に従います。</p>	<p>IAM ユーザーガイドの「IAM ユーザーのアクセスキーの管理」に従って、プログラムによるアクセスを設定します。</p>

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、「IAM [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。

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、 に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
<p>ワークフォースアイデンティティ</p> <p>(IAM Identity Center で管理されているユーザー)</p>	<p>一時的な認証情報を使用して、AWS SDKs AWS CLI、または AWS APIs。</p>	<p>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。</p> <ul style="list-style-type: none"> • については AWS CLI、「ユーザーガイド」の AWS CLI「を使用するための設定 AWS IAM Identity Center AWS Command Line Interface」を参照してください。 • AWS SDKs、ツール、AWS APIs「SDK とツールのリファレンスガイド」の 「IAM Identity Center 認証」を参照してください。 AWS SDKs
IAM	<p>一時的な認証情報を使用して、AWS SDKs AWS CLI、または AWS APIs。</p>	<p>「IAM 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。</p>
IAM	<p>(非推奨)</p> <p>長期認証情報を使用して、AWS SDKs AWS CLI、または AWS APIs。</p>	<p>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。</p> <ul style="list-style-type: none"> • 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。 • AWS SDKs「SDK とツールのリファレンスガイド」の 「長期的な認証情報を使

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
		<p>用した認証」を参照してください。AWS SDKs</p> <ul style="list-style-type: none">• AWS APIs 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。では、認証情報の設定などのアクティビティ AWS CLI に AWS SAMCLI 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、「[AWS Command Line Interface 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。

ステップ 5: AWS CLI を使用して AWS 認証情報を設定する

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 AWS SAM CLI しての使用を開始する準備ができました AWS SAM。をインストールするには、AWS SAM CLI 「」を参照してください [AWS SAM CLI のインストール](#)。

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。

夜間ビルドのアップグレード、アンインストール CLI、管理の方法など、現在インストールされている AWS SAM のバージョンの管理については、「」を参照してください [AWS SAM CLI バージョンの管理](#)。

① 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 アカウントにサインアップする。
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. のインストール AWS CLI。
5. AWS 認証情報の設定。

トピック

- [AWS SAM CLIのインストール](#)
- [インストールエラーのトラブルシューティング](#)
- [次のステップ](#)
- [オプション: AWS SAMCLIインストーラの整合性を検証する](#)

AWS SAM CLIのインストール

Note

2023年9月以降、AWSはAWS SAMCLI () のAWSマネージドHomebrewインストーラーを維持しなくなりますaws/tap/aws-sam-cli。Homebrewを使用してをインストールおよび管理する場合はAWS SAMCLI、次のオプション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。

をインストールするにはAWS SAMCLI、オペレーティングシステムの指示に従ってください。

Linux

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 SAMCLI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
```

正常にインストールされたことを確認するには、次のような出力が表示されますが、角括弧で囲まれたテキストは最新の SAM CLI バージョンに置き換えられます。

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

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 SAMCLI インストーラー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
```

- 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
```

- インストールを確認します。

```
$ sam --version
```

正常にインストールされたことを確認するには、次の角括弧で囲まれたテキストを利用可能な最新バージョンに置き換える出力が表示されます。

```
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 をインストールします。さらに、パッケージインストーラーには、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。インストールできるのは、すべてのユーザー または現在のユーザー のみです。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ーをダウンロードして をインストールするには AWS SAM CLI

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 を任意のディレクトリpkgにダウンロードします。
 - Intel プロセッサを実行している Mac の場合は、x86_64 --x86_64.pkg を選択します。[aws-sam-cli-macos](#)
 - Apple シリコンを実行している Mac の場合は、arm64 --arm64.pkg を選択します。[aws-sam-cli-macos](#)

Note

インストール前にインストーラの整合性を検証するオプションがあります。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

をダウンロードしてインストールするには AWS SAMCLI

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 を任意のディレクトリpkgにダウンロードします。
 - Intel プロセッサを実行している Mac の場合は、x86_64 --x86_64.pkg を選択します。[aws-sam-cli-macos](#)
 - Apple シリコンを実行している Mac の場合は、arm64 --arm64.pkg を選択します。[aws-sam-cli-macos](#)

Note

インストール前にインストーラの整合性を検証するオプションがあります。手順については、「[オプション: AWS SAMCLI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。
- 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。
 - [Install] (インストール) を押します。
 - 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

- `sudo` 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 `$PATH` への書き込み権限を持っている場合、`sudo` は必須ではありません。それ以外の場合、`sudo` が必要です。
- **`path-to`** — AWS SAM CLI プログラムをインストールした場所へのパス。例えば `/Users/myUser/Desktop` です。
- **`path-to-symlink-directory`** — `$PATH` 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

をダウンロードしてインストールするには AWS SAMCLI

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 を任意のディレクトリpkgにダウンロードします。
 - Intel プロセッサを実行している Mac の場合は、x86_64 --x86_64.pkg を選択します。[aws-sam-cli-macos](#)
 - Apple シリコンを実行している Mac の場合は、arm64 --arm64.pkg を選択します。[aws-sam-cli-macos](#)

Note

インストール前にインストーラの整合性を検証するオプションがあります。手順については、「[オプション: AWS SAMCLI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

をダウンロードしてインストールするには AWS SAMCLI

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 を任意のディレクトリ pkg にダウンロードします。
 - Intel プロセッサを実行している Mac の場合は、x86_64 --x86_64.pkg を選択します。[aws-sam-cli-macos](#)
 - Apple シリコンを実行している Mac の場合は、arm64 --arm64.pkg を選択します。[aws-sam-cli-macos](#)

Note

インストール前にインストーラの整合性を検証するオプションがあります。手順については、「[オプション: AWS SAMCLI 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、を *path-to-my-directory* に置き換えると /Users/myUser/Desktop、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が。

4. をインストールするには、以下を実行します AWS SAMCLI。

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

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

シンボリックリンクを作成するには

- `sudo` 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 `$PATH` への書き込み権限を持っている場合、`sudo` は必須ではありません。それ以外の場合、`sudo` が必要です。
- `path-to` — AWS SAM CLI プログラムをインストールした場所へのパス。例えば `/Users/myUser/Desktop` です。
- `path-to-symlink-directory` — `$PATH` 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  
$ sudo 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。

Note

32 ビットバージョンの Windows を使用している場合は、「[32 ビットの Windows で AWS SAMCLI をインストールする](#)」を参照してください。

- (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 SAMCLI インストーラーの整合性を検証する](#)」を参照してください。
- 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
```

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

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

- ロングパスを有効にします (Windows 10 以降のみ)。

Important

は、Windows の最大パス制限を超えるファイルパスとやり取り AWS SAMCLI する場合があります。これにより、Windows 10 MAX_PATH の制限 `sam init` により、実行時にエラーが発生する可能性があります。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。

- 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。

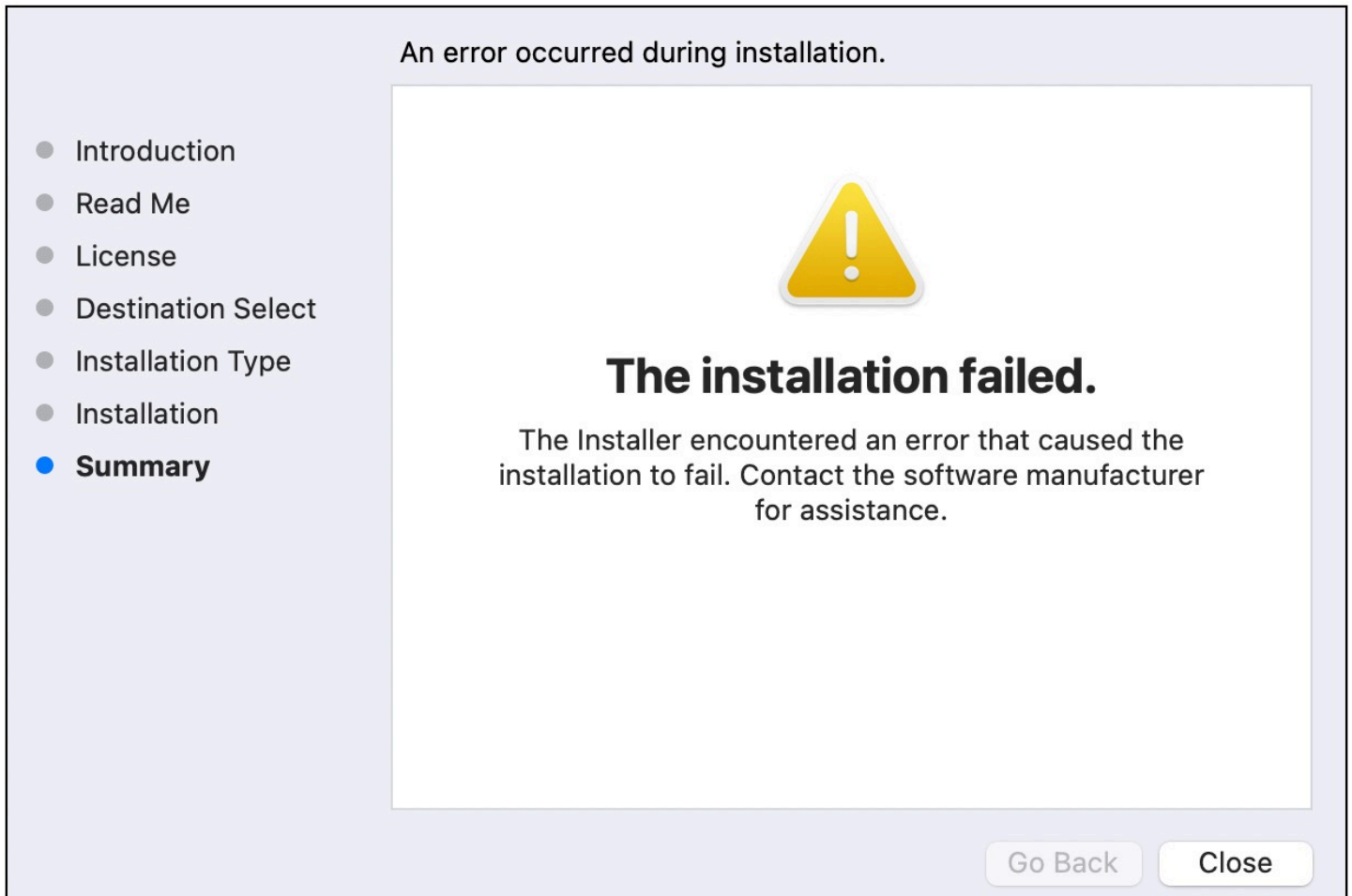
AWS SAMCLI エラー: "/lib64/libc.so.6: version `GLIBC_2.14' not found (/usr/local/aws-sam-cli/dist/libz.so.1 で必要)"

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: Hello World アプリケーションをデプロイする](#) – 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、デプロイするための Step-by-step 手順。
- [完全な AWS SAM ワークショップ](#) – AWS SAM が提供する主要な機能の多くを学習するために設計されたワークショップです。
- [AWS SAM サンプルアプリケーションとパターン](#) – 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 SAMCLI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 SAMCLI) をインストールする場合、インストール前に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の署名ファイルを確認します。
- パッケージインストーラ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 .zip AWS SAMCLI 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` です。

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

mQINBGRuSzMBEADsqiw0y78w7F4+sshaMFRiWRGNRM94p5Qey2KMZBxekFtoryVD
D9jE0nvupx4tvhfBHz5EcUHCE0d14MTqdBy6vVAshozgxVb9RE8JpECn5lw7XC69
4Y7Gy1TKKQMEwtDXElkGxIFdUwvWjSnPlzfnoXwQYGeE93CUS3h5dImP22Yk1Ct6
eGGh1cbg1X4L8EpFMj7GvcsU8f7ziVI/PyC1Xwy39Q8/I67ip5eU5ddx0/xHqrbL
YC7+8pJPbRMej2twT2LrcpWYAbprMtRoa6WfE0/thoo3xhHpIMhdPFAA86ZNGIN
kRLjGUg7jnPTRW40in3pCc8nT4Tfc1QERkHm641gTC/jUvpmQsM6h/FUVP2i5iE/
JHpJcMuL2Mg6zDo3x+3gTCf+Wqz3rZzxB+wQT3yryZs6efcQy7nR0iRxYBxCSXX0
2cNYzsYlb/bYaW8yqWIHD5IqKhW269gp2E5Khs60zgS3CorMb5/xHgXjUCVgcu8a
a8ncdf9fj13WS5p0ohetPb0Z2jWv+MaqrZ0mUIgKbA4RpWZ/fU97P5BW9y1wmIDB
sWy0cMxg8M1vSdLytPieogaM0qMg3u5qXRGBr6Wmevkty0qgnmpGGc5zPiUbt0E8
CnFFqyxBpj5I0nG0KZGVihvn+iRrxv6G07WW092+Dc6m94U0EEiBR7Qi0wARAQAB
tDRBV1MgU0FNIENMSSBQcm1tYXJ5IDxhd3MtY2FtLWNsaS1wcm1tYXJ5QGZfYXpv
bi5jb20+iQI/BBMBCQApBQJkbksZAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYC
AwECHgECF4AACgkQQv1fen0tiFqTuhAAzi5+ju5UV0WqHkev0JS008T4QB8HcqAE
SV03mY6/j29knkcL8ubZP/DbpV7QpHPI2PB5qSXsiDTP3IYPbeY78zHSDjljaIK3
njJLMScFeGPyfPpwMsuY4nziRiGAtXShPA8N/k4ZJcafnpNqKj7QnPxIC1KaIQWm
p0tvb8msUF3/s0UTa5Ys/1NRhVC0eGg32ogXGdojZA2kHZWdm9udLo4CDrDcrQT7
NtDcJASapXSQL63XfAS3snEc4e1941YxcjFYZ33rel8K9juyDZfi1s1WR/L3AviI
QFIaqSHzy0tP1oinUkoVwL8ThevKD3Ag9CZf1ZLzNCV7yq1F8R1hEZ4zce/3s9E1
WzCFsozb5HfE1AZonmrDh3Sy0EIBMCS6vG5dWnvJrAuSYv2rX38++K5Pr/MIAfOX
D0I1rtA+XDshNv91SwSy0lt+iClawZAN09IXCiN1r0YcVQlwzDFwCNWDgkwd0qS0
g0A2f8NF91E5nBbeEuYquo011Vy8+ICbg0F9LoWZlnVh7/RyY6ssowiU9vGUNHI
L8f9jqRspIz/Fm3JD86ntZxLVGkeZUz62FqErdohYfkFIVcv7GONTEyrz5HL1npv
FJ0MR0HjrMrZrn0VZnwBKhpLocTsH+3t5It4ReYEX0f1DIOL/KRwPvjMvBVkXY5
hb1RVDQo0Wc=
=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)

```

- 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* です。

```

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

mQINBGRtS20BEAC7GjaAwverrB1zNEu2q3EGI6HC37WzwL5dy30f4LirZ0WS3piK
oKfTqPjXPRLcF1GL2mMqUSgSnpEbPNXuvWTW1CfSnnjwuH8ZqbvvUQyHJwQyYpKm
KMwb+8V0bzzQkMzDVqoLYQCi5XyGpAuo3wroxXSzG6r/mIhbiq3aRnL+2lo4X0Yk
r7q9bhBqbJhzjkm7N62PhPWmi/+EGdEBakA1pReE+cKjP2UAp5L6CPSHQ12fRKL
9BumitNfFHHs1JJZgZSCCruiWny3XkUaXUEMfyoE9nNbfqNvuqV2KjWguZCXASgz2
ZSPF4DTVIBMfP+xrZGQSWdGU/67QdysDQW81TbF0jK9ZsRwwGC4kbg/K98IsCNHT
ril5RZbyr8pw3fw7jYyjI2E1AacRwP53iRzvutm5AruPpLfoKDQ/tKzBUYItBwlu
Z/diKgcqtW7xDlyqNyTN8xPFpM02I8IsZ2Pd1131htdFiZMiin1RQG9pV9p2vHS
eQVY2uKcNvnA6vFCQYKXP7p0IwReuPNzDvECUsidw8VTakTqZsANT/bU17e4KuKn
+JgbNrK0asJX37sDb/9ruysozLvy78ozYKJDLmC3yoRQ8DhEjviT4cnjORgNmvnZ
0a5AA/DJPPQW4buRrXdxu+fITzBxQn2+G0/iDNCxtJaq5SYVBKjTmTWPUJwARAQAB
tDBBV1MgU0FNIENSSBUZWFtIDxhd3MtZ2FtLWNsaS1zaWduZXJAYW1hem9uLmNv
bT6JAj8EEwEJACKFAMRtS20CGy8FCQPCZwAHCwkIBwMCAQYVCAIJCgsEFgIDAQIe
AQIXgAAKCRDHoF9D/grd+1E4D/4kJW65He2LNsBLTta7lcGfsEXCf4zgIvkytS7U
3R36zMD8IEyWjJlZ+aPkIP8/jFJrF14pVHbU7vX85Iut1vV7m+8BgWt25mJhnoJ9
KPjXGra9mYP+Cj8zFACjvt13NBAPodyfcfCTWsU3umF9ArOFICcrGCzHX2SS7wX5
h9n0vYRZxk5Qj5FsgskKAQLq33CKFAMlaqZnL5gWRvTeycSIxsysus+stX+8YBPC0
J64f7+y+MPIP1+m2nj1VXg1xLEMMVa08oWcc0MiakgzDev3LCrPy+wdwdn7Ut7oA
pna3DNy9aYnd2lh6vUCJeJ+Yi1B12jYpzLcCLKrHUmIn9/rRSz70rbg8P181kfPu
G/M7CD5FwhxP3p4+0XoGwxQefrV2jqpSnbLae7xbYJiJAhpjWDQhuNGUbPcDmqk
aH0Q3XU8AonJ8YqaQ/q3VZ3JBiH3TbBr0Xsvd59cwxYyf83aJ/WLCb2P8y75zDad
ln0P713ThF5J/Afj9Hj09waFV0Z2W2ZZe4rU20JTAiXEtM8xsFMrc7TCUacJtJGs
u4kdBmXREcVpSz65h9ImSy2ner9qktnVVCW4mZPj63IhB37YtoLAMyz3a3R2RFNk
viEX8fo0TUg1FmW HofTxZ9P91QwLoTajkDrh26ueIe45sG6Uxua2AP4Vo37cFfCj
ryV80okCHAQQAQkABgUCZG5MWAACKRBC/V96c62IWmg1D/9idU43kW8Zy8Af1j81

```

```
Am3lI4d9ks0leeKRZqxo/SZ5rovF32D02nw7XRXq1+EbhgJaI3Qww0i0U0pfAMVT
4b9TdxH+n+tzqCHh3jZqmo9sw+c9WFXyJN1hU9bLzcHXS8h0TbyoE2EuXx56ds9
L/BWCcd+LIvawp0lggFfavVx/QF4C7nBKjnJ66+xxwfgVIKR7oG1qDiHMfp9ZWh5
HhEqZo/nrNhdY0h3sczEdqC2N6eIa8mgHffHZdKudDMXIXHbgdhW9pcZXDIktVf7
j9wehsW0yYXiRgR0dz7DI26AUG4JLh5FTtx9XuSBdEsI69Jd4dJuibmgtImzbZjn
7un8DJWIyqi7Ckk96Tr4oXB9mYAXaWLR4C9j5XJhMNZgk0ycuY2DADnbGmSb+1kA
ju77H4ff84+vMDwUzUt2Wwb+GjzXu2g6Wh+bWhGSirYle1+6xYrI6beu1BDCFLq+
VZFE8WggjJHpwcl7CiqadfVIQaw4HY0jQFTSdwzPWhJvYjXF0hMkyCcjsbBtmB+z
/otfgySyQqThrD48RWS5GuyqCA+pK3UNmEJ11c1AXMdTn2VWInR1N0JNALQ2du3y
q8t1vMsErV0J7pkZ50F4ef17PE6DKrXX8ilwGFyVuX5ddydt/t9J5pC3sRwHWXVZx
GXwoX75FwIEHA3n5Q7rZ69Ea6Q==
=ZI07
-----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** です。

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 リポジトリで [問題を作成して](#)、AWS SAM チームにエスカレーションします。

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 リポジトリに問題[を作成して](#) AWS SAM チームにエスカレーションし、ダウンロードしたファイルを使用しないようにしてください。

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 .zip AWS SAMCLI 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* です。

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

mQINBGRuSzMBEADsqiw0y78w7F4+sshaMFRiWRGNRm94p5Qey2KMZBxekFtoryVD
D9jE0nvupx4tvhfBHz5EcUHCE0d14MTqdBy6vVAshozgxVb9RE8JpECn51w7XC69
4Y7Gy1TKKQMEwtDXE1kGxIFdUwvWjSnPlzfnoXwQYGeE93CUS3h5dImP22Yk1Ct6
eGGh1cbg1X4L8EpFMj7GvcsU8f7ziVI/PyC1Xwy39Q8/I67ip5eU5ddx0/xHqrbL
YC7+8pJPbRMej2twT2LrcpWYAbprMtRoa6WfE0/thoo3xhHpIMHdPFAA86ZNGIN
kRLjGUg7jnPTRw40in3pCc8nT4Tfc1QERkHm641gTC/jUvpmQsM6h/FUVP2i5iE/
JHpJcMuL2Mg6zDo3x+3gTCf+Wqz3rZzxB+wQT3yryZs6efcQy7nR0iRxYBxCSXX0
2cNYzsYlb/bYaW8yqWIHD5IqKhW269gp2E5Khs60zgS3CoRmb5/xHgXjUCVgcu8a
a8ncdf9fj13WS5p0ohetPb02ZjWv+MaqrZ0mUIgKbA4RpWZ/fU97P5BW9y1wmIDB
sWy0cMxg8M1vSdLytPieogaM0qMg3u5qXRGBr6Wmevkty0qgnmpGGc5zPiUbtOE8
```

```
CnFFqyxBpj5I0nG0KZGVihvn+iRrxrv6G07WW092+Dc6m94U0EEiBR7Qi0wARAQAB
tDRBV1MgU0FNIENMSSBQcmLtYXJ5IDxhd3Mtc2FtLWNsaS1wcm1tYXJ5QGFtYXpv
bi5jb20+iQI/BBMBCQApBQJkbkszAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYC
AwEChgECF4AACGkQQv1fen0tiFqTuhAAzi5+ju5UV0WqHkev0JS008T4QB8HcqAE
SV03mY6/j29knkcL8ubZP/DbpV7QpHPi2PB5qSXsiDTP3IYPbeY78zHSDjljaIK3
njJLMScFeGPyfPpwMsuY4nziRiGAtXShPA8N/k4ZJcafnpNqKj7QnPxIC1KaIQWm
p0tVb8msUF3/s0UTa5Ys/1NRhVC0eGg32ogXGdojZA2kHZWdm9udLo4CDrDcrQT7
NtDcJASapXSQL63XfAS3snEc4e1941YxcjfyZ33rel8K9juyDZfi1s1WR/L3AviI
QFIaqSHzy0tP1oinUkoVwL8ThevKD3Ag9CZf1ZLzNCV7yq1F8R1hEZ4zce/3s9E1
WzCFsozb5HfE1AZonmrDh3Sy0EIBMCS6vG5dWnvJrAuSYv2rX38++K5Pr/MIAf0X
DOI1rtA+XDshNv91SwSy01t+iClawZAN09IXCiN1r0YcVQ1wzDFwCNWDgkwd0qS0
g0A2f8NF91E5nBbeEuYquo0l1Vy8+ICbg0Fs9LoWZlnVh7/RyY6ssowiU9vGUnHI
L8f9jqRspIz/Fm3JD86ntZxLVGkeZuz62FqErdohYfkFIVcv7GONTEyrz5HL1npv
FJ0MR0HjrMrZrn0VZnwBKhpLocTsH+3t5It4ReYEX0f1DIOL/KRwPvjMvBVkXY5
hb1RVDQo0Wc=
=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* です。

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

mQINBGRtS20BEAC7GjaAwverrB1zNEu2q3EGI6HC37WzwL5dy30f4LirZOWS3piK
oKfTqPjXPrLCf1GL2mMqUSgSnpEbPNXuvWTW1CfSnnjwuH8ZqbvvUQyHJwQyYpKm
KMwb+8V0bzzQkMzDVqolYQCi5XyGpAuo3wroxXSzG6r/mIhbiq3aRnL+21o4X0Yk
```

```

r7q9bhBqbJhzjkm7N62PhPWmi/+/EGdEBakA1pReE+cKjP2UAp5L6CPSHQ12fRKL
9BumitNfFHHs1JZgZSCCruiWny3XkUaXUEMfyoE9nNbfqNvuqV2KjWguZCXASgz2
ZSPF4DTVIBMfP+xrZGQSWdGU/67QdysDQW81TbF0jK9ZsRwwGC4kbg/K98IsCNHT
ril5RZbyr8pw3fw7jYjyI2E1AacRwP53iRzvutm5AruPpLfokDQ/tKzBUYItBwlu
Z/diKgcqtW7xDlyqNyTN8xFPFqM02I8IsZ2Pd1131htdFiZMiin1RQG9pV9p2vHS
eQVY2uKcNvnA6vFCQYKXP7p0IwReuPNzDvECUsidw8VTakTqZsANT/bU17e4KuKn
+JgbNrK0asJX37sDb/9ruysozLvy78ozYKJDLmC3yoRQ8DhEjviT4cnjORgNmvnZ
0a5AA/DJPPQW4buRrXdxu+fITzBxQn2+G0/iDNCxtJaq5SYVBKjTmTWPUJwARAQAB
tDBBV1MgU0FNIENMSSBUZWFtIDxhd3MtZ2FtLWNsaS1zaWduZXJAYW1hem9uLmNv
bT6JAj8EEwEJACKFAMRtS20CGy8FCQPCZwAHCwkIBwMCAQYVCAIJCgsEFgIDAQIe
AQIXgAAKCRDHoF9D/grd+1E4D/4kJW65He2LNsblTta71cGfsEXCf4zgiVkytS7U
3R36zMD8IEyWJjLZ+aPKIP8/jFjrF14pVHbU7vX85Iut1vV7m+8BgWt25mJhnoJ9
KPjXGra9mYP+Cj8zFACjvt13NBAPodyfcfCTWsu3umF9ArOFICcrGCzHX2SS7wX5
h9n0vYRZxk5Qj5FsgskKAQLq33CKFAMlaqZnL5gWRvTeycSIxsysus+stX+8YBPC0
J64f7+y+MPIP1+m2nj1VXg1xLEMMVa08oWcc0MiakgzDev3LCrPy+wdwn7Ut7oA
pna3DNy9aYnd21h6vUCJeJ+Yi1B12jYpzLcCLKrHUmLn9/rRSz70rbg8P181kfPu
G/M7CD5FwhxP3p4+0XoGwxQefrV2jqpSnbLae7xbYJiJAhbpbjWDQhuNGUbPcDmqk
aH0Q3XU8AonJ8YqaQ/q3VZ3JBih3TbBr0Xsvd59cwxYyf83aJ/WLCb2P8y75zDad
ln0P713ThF5J/Afj9Hj09waFV0Z2WZZe4rU20JTAiXEtM8xsFMrc7TCUacJtJGs
u4kdBmXREcVpSz65h9ImSy2ner9qktnVVCW4mZPj63IhB37YtoLAMyz3a3R2RFNk
viEX8fo0TUg1FmwHoftxZ9P91QwLoTajkDrh26ueIe45sG6Uxua2AP4Vo37cFfCj
ryV80okCHAQQAQkABgUCZG5MWAACKRBC/V96c62IWmg1D/9idU43kW8Zy8Af1j81
Am31I4d9ks0leeKRZqxo/SZ5rovF32D02nw7XRXq1+EbhgJaI3Qww0i0U0pfAMVT
4b9TdxH+n+tzqCHh3jZqmo9sw+c9WFXyJN1hU9bLzchXS8h0TbyoE2EuXx56ds9
L/BWCcd+LIvapw0lggFfavVx/QF4C7nBKjnJ66+xxwfgVIKR7oG1qDiHMfp9ZWh5
HhEqZo/nrNhdY0h3sczEdqC2N6eIa8mgHffHZdKudDMXIXHbgdhW9pcZXDiktVf7
j9wehsW0yYXiRgR0dz7DI26AUG4JLh5FTtx9XuSBdEsI69Jd4dJuibmgtImzbZjn
7un8DJWIyqi7Ckk96Tr4oXB9mYAXaWLR4C9j5XJhMNZgk0ycuY2DADnbGmSb+1kA
ju77H4ff84+vMDwUzUt2Wwb+GjzXu2g6Wh+bWhGSirYle1+6xYrI6beu1BDCFLq+
VZFE8WggjJHpwcl7CiqadfVIQaw4HY0jQFTSdwzPWhJvYjXF0hMkyCcjsbBtmB+z
/otfgySyQqThrD48RWS5GuyqCA+pK3UNmEJ11c1AXMdTn2VWInR1N0JNALQ2du3y
q8t1vMsErV0J7pkZ50F4ef17PE6DKrXX8ilwGFyVuX5ddyT/t9J5pC3sRwHWXVZx
GXwoX75FwIEHA3n5Q7rZ69Ea6Q==
=ZI07
-----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** です。

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 リポジトリで[問題を作成して](#)、AWS SAM チームにエスカレーションします。

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 リポジトリに問題[を作成して](#) AWS SAM チームにエスカレーションし、ダウンロードしたファイルを使用しないようにしてください。

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 リポジトリで[問題を作成して](#)、AWS SAM チームにエスカレーションします。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 SAMCLI インストーラは、Windowsオペレーティングシステムの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ます。

```
c011d416e99a1142c0e0235118ef64c2681f3db9
```

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: Hello World アプリケーションをデプロイする

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 SAMCLI) を使用して以下を完了します。

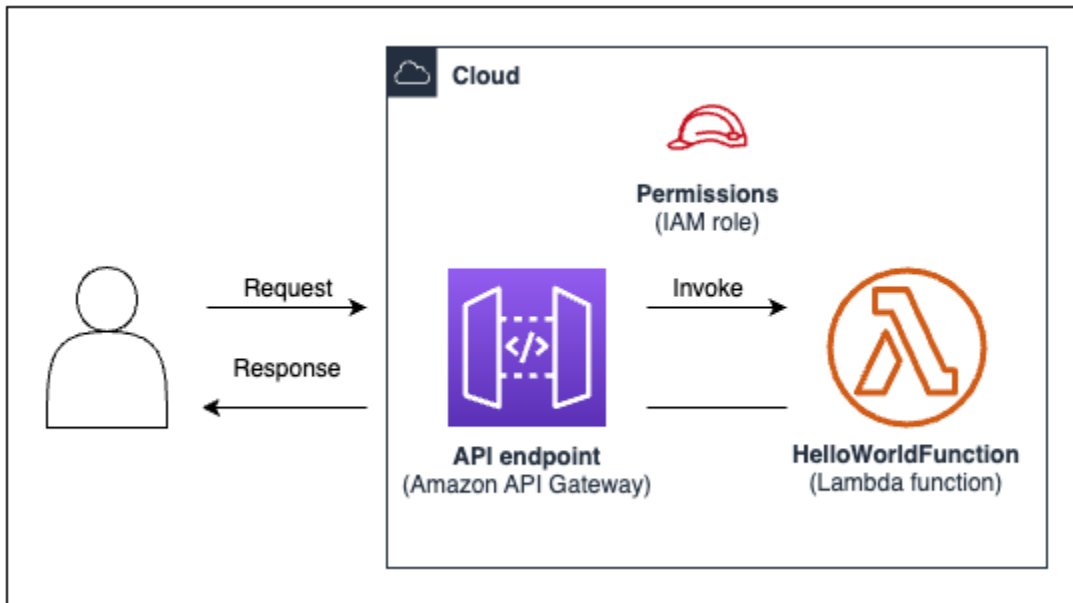
- サンプルの Hello World アプリケーションを初期化、構築、デプロイします。
- ローカルで変更を行い、 に同期します AWS CloudFormation。

- でアプリケーションをテストします AWS クラウド。
- 必要に応じて、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。

サンプルの 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ています。これは次のリソースで構成されます。

- Amazon API Gateway – 関数を呼び出すために使用する API エンドポイント。
- AWS Lambda – HTTP API GET リクエストを処理し、hello world メッセージを返す関数。
- 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

- [前提条件](#)
- [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](#)
- [ステップ 2: アプリケーションを構築する](#)
- [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](#)
- [ステップ 4: アプリケーションを実行する](#)
- [ステップ 5: で関数进行操作する AWS クラウド](#)

- [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](#)
- [詳細はこちら](#)

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 を使用して、ローカルマシン上にサンプルの 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、複数のファイルとフォルダが含まれます。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。Python のバージョンは、`sam init` コマンドが作成した `template.yaml` ファイルに記載されている Python のバージョン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通じてユーザーをガイドします。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. Python ランタイムと zip パッケージタイプを使用します。
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「[とは AWS X-Ray](#)」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 関数での JSON 形式の構造化ログ記録の設定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。

AWS SAM CLI インタラクティブフローを使用するには:

- 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. は開始テンプレート AWS SAMCLI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。次は AWS SAM CLI の出力例です。

```
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。AWS SAM CLI が作成したものの例を次に示します。

```
$ 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 – Lambda 関数に必要な Python 依存関係が含まれます。
- samconfig.toml – AWS SAMで使用す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 を使用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。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、および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

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

次は AWS SAM CLI の出力例です。

```
$ 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
```

AWS SAM CLI によって作成される `.aws-sam` ディレクトリの短い例を次に示します。

```
.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関数ごとに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` – アプリケーションの構築およびデプロイ時に AWS SAM CLI によって参照されるデフォルトのパラメータ値を保存する設定ファイル。

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 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。AWS SAMCLI 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、テンプレートをサービスにアップロード AWS CloudFormation して AWS リソースをプロビジョニングします。

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. AWS SAM CLI のインタラクティブフローに従って、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、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 承認が定義されていない状態で許可する – このメッセージは、API Gateway 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に表示されます。これは Hello World アプリケーション用に意図された設定であるため、AWS SAM CLI の続行を許可します。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. は、以下を実行してアプリケーションを AWS SAMCLI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。
 - は AWS SAM テンプレートを AWS SAMCLI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 は進行状況を表示します。以下は、その出力例です。

Looking for resources needed for deployment:

Managed S3 bucket: aws-sam-cli-managed-default-samclisourcebucket-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-samclisourcebucket-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-15GL0UR9LMT1W  
  
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 エンドポイントの値を取得するには

1. 前のステップで AWS SAM CLI によって表示された情報から、Outputs セクションを見つけます。このセクションで、HelloWorldApi リソースを見つけて HTTP エンドポイントの値を確認します。出力例を次に示します。

```
-----  
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: で関数进行操作する AWS クラウド

このステップでは、AWS SAM CLI を使用して AWS クラウド内の Lambda 関数を呼び出します。

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. はクラウドで関数を AWS SAMCLI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、sam sync --watch コマンドを使用してローカルの変更を AWS SAMCLI に同期します 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ようとしていることを確認するよう促すプロンプトを表示します。sam sync --watch コマンドはローカルの変更を AWS クラウド にリアルタイムで自動的にデプロイするため、開発環境のみにすることをお勧めします。

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_pt15g6_80000gr/T/tmpq3x9vh63.
Execute the following command to deploy the packaged template
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_pt15g6_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-
samclisourcebucket-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 SAMCLI 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. 選択した IDE 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. は変更 AWS SAMCLI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 エンドポイントに GET 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

このステップは、ローカルマシン上に Docker を必要とするため、オプションです。

Important

ローカルテストに AWS SAM CLI を使用するには、Docker をインストールして設定しておく必要があります。詳細については、「[Docker のインストール](#)」を参照してください。

このステップでは、`sam local` コマンドを使用して AWS SAM CLI アプリケーションをローカルでテストします。これを実現するために、AWS SAM CLI は Docker を使用してローカル環境を作成します。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 Docker コンテナを作成し、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\"}"}
```

API を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、Lambda 関数用にローカル Docker コンテナを作成し、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、sam delete コマンドを使用して AWS SAMCLI からアプリケーションを削除します 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 から確認するよう求められます。その後、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については、AWS SAMCLI「」を参照してください[AWS SAMCLI トラブルシューティング](#)。

詳細はこちら

について学習を続けるには 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。
- [SAM を使用したセッション](#) — を使用して で AWS Serverless Developer Advocate チームによって作成された動画シリーズ 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) の使用方法

アプリケーションの開発に使用する主なツールは、AWS SAMCLIと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、次の目的で使用します。

1. [アプリケーションの開発](#) (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. [アプリケーションのデバッグ](#)。
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](#)。

AWS SAM `aws 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、によって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、その中の最も重要なファイルはAWS SAM テンプレート (`template.yaml` という名前) です。このテンプレートでは、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コードを記述します。

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AWS SAMCLIが含まれています。具体的には、AWS SAMCLI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える際に使用します。

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 SAMCLI) は、AWS SAM アプリケーション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 SAMCLIを使用すると、AWS SAM アプリケーション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、同期できます。

AWS SAMCLI および 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して実行するためにサポートされているサードパーティー統合が付属しています。

トピック

- [AWS SAM CLI コマンドを文書化する方法](#)
- [AWS SAM CLI の設定](#)
- [AWS SAMCLI コアコマンド](#)

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

の利点の 1 AWS SAM つは、反復的なタスクを削除して開発者の時間を最適化することです。には、`samconfig`この目的のために という名前の設定ファイル AWS SAMCLIが含まれています。デフォルトでは、への設定 AWS SAMCLIは必要ありませんが、設定ファイルでカスタマイズしたパラメータ AWS SAM を参照できるようにすることで、より少ないパラメータでコマンドを実行できるように設定ファイルを更新できます。次の表の例は、コマンドを最適化する方法を示しています。

元	で最適化 <code>samconfig</code>
<code>sam build --cached --parallel --use-containers</code>	<code>sam build</code>

元	で最適化 samconfig
<pre>sam local invoke --env-vars locals.json</pre>	<pre>sam local invoke</pre>
<pre>sam local start-api --env-vars locals.json -- warm-containers EAGER</pre>	<pre>sam local start-api</pre>

AWS SAMCLI 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は、アプリケーションとデベロッパーの設定に基づいてオプションのフラグで設定できます。詳細については、「」の[AWS SAMCLI コンテンツを参照してください](#)。 [GitHub](#)

このセクションのトピックでは、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ために [AWS SAM CLI 設定ファイル](#) を作成し、デフォルト設定をカスタマイズする方法を示します。

トピック

- [設定ファイル \(samconfig ファイル\) を作成する方法](#)
- [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 (samconfig ファイル) を作成する方法

AWS SAMCLI 設定ファイル (ファイル名 samconfig) は、通常 TOML 構造を使用するテキストファイルですが、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、このファイルは sam init コマンドの実行時に作成されます。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ときに、このファイルを更新できます。

デプロイが完了すると、デフォルト値を使用したdefault場合、samconfigファイルには という名前のプロファイルが含まれます。deploy コマンドを再実行すると、はこのプロファイルに保存されている構成設定 AWS SAM を適用します。

samconfig ファイルの利点は、が deploy コマンドに加えて使用可能な他のコマンドの設定 AWS SAM を保存することです。新しいデプロイで作成されたこれらの値以外にも、samconfig ファイルで設定できる属性が多数あり、AWS SAM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CLI。

プロジェクト設定を構成する

で使用する設定ファイルで、AWS SAMCLIコマンドパラメータ値などのプロジェクト固有の設定を指定できます AWS SAMCLI。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行います。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](#)
- [の名前付きプロファイル AWS CLI](#)
- [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 SAMCLI コアコマンド

AWS SAMCLI 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あります。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を含むリンクを示します。

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

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、デベロッパーワークフローの後続のステップのためにアプリケーションを準備します。	<ul style="list-style-type: none"> • sam build コマンドを使用した構築の概要 • sam build
sam deploy	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 AWS CloudFormation。	<ul style="list-style-type: none"> • sam deploy コマンドを使用したデプロイの概要 • sam deploy
sam init	新しいサーバーレスアプリケーションを初期化して作成するためのオプションを提供します。	<ul style="list-style-type: none"> • sam init コマンドを使用して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。	<ul style="list-style-type: none"> • 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	AWS Lambda 関数の共有可能なテストイベントにアクセスして管理する方法を提供します。	<ul style="list-style-type: none"> • 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 クラウドでサポートされている AWS リソースを操作する方法を提供します。	<ul style="list-style-type: none"> • を使用したクラウドテストの概要 sam remote test-event • sam remote test-event

Command	その内容	関連トピック
sam sync	ローカルアプリケーションの変更を AWS クラウドにすばやく同期するオプションを提供します。	<ul style="list-style-type: none">• を使用して sam syncに同期する方法の概要 AWS クラウド• 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、AWS SAM はアプリケーションプロジェクトディレクトリを作成します。これは AWS SAM プロジェクトです。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、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、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コードを記述します。

Note

AWS SAM テンプレートの主要な要素は AWS SAM、テンプレート仕様です。この仕様は、と比較して、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 AWS CloudFormation、アクセス許可、APIs、およびその他のプロパティを定義するために使用できるコード行の数を減らすことができる簡潔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、データ型、リソース属性、組み込み関数、API Gateway 拡張機能を定義する方法について詳しく説明します。

AWS SAM テンプレートは AWS CloudFormation テンプレートの拡張であり、よりもコード行数が少ない短縮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の開発が高速化されます。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](#) AWS CloudFormation 」を参照してください。

トピック

- [AWS SAM テンプレートの構造](#)
- [AWS SAM リソースとプロパティ](#)
- [生成された AWS CloudFormation リソース](#)
- [でサポートされているリソース属性 AWS SAM](#)
- [API Gateway 拡張機能](#)
- [組み込み関数](#)

AWS SAM テンプレートの構造

AWS SAM テンプレートファイルは、「AWS CloudFormation ユーザーガイド」の AWS CloudFormation 「テンプレート構造」で[説明されているテンプレート](#)ファイルの形式に厳密に従います。AWS CloudFormation テンプレートファイルと AWS SAM テンプレートファイルの主な違いは次のとおりです。

- 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変換してテンプレートファイルとして識別します AWS SAM。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 セクションは固有です AWS SAM。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 テンプレートでは、Resourcesセクションに 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、「ユーザーガイド」の「[AWS リソースタイプとプロパティタイプのリファレンス](#)AWS CloudFormation」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の他の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、いくつかの主要なセクションを含めることができます。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、言語拡張を使用する場合は、次の例に示すように、サーバーレス変換 `AWS::LanguageExtensions` の前 (つまりの前 `AWS::Serverless-2016-10-31`) に追加する必要があります。

Transform:

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

テンプレートを構築するときは、次のリストに示す論理順序を使用することをお勧めします。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、このセクションに の値を含める必要がありますAWS::Serverless-2016-10-31。

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 に固有です AWS SAM。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ています。

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ています。

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ています。

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ています。

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 テンプレートでは、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、`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を含む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、「」を参照の[開始方法 AWS SAM](#)し、「」の手順に従います[チュートリアル: Hello World アプリケーションをデプロイする](#)。

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は、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 セクションは、次の AWS SAM リソースタイプをサポートします:

- `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 ThumbnailFunction の両方が Runtime に「nodejs12.x」、Timeout に「180」秒、および Handler に「index.handler」を使用しています。HelloWorldFunction は、継承された TABLE_NAME に加えて、MESSAGE 環境変数を追加します。ThumbnailFunction はすべての Globals プロパティを継承し、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 は、以下のリソースとプロパティをサポートします。

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

  Function:
    Architectures:
    AssumeRolePolicyDocument:
    AutoPublishAlias:
    CodeUri:
    DeadLetterQueue:
    DeploymentPreference:
```

```
Description:
Environment:
EphemeralStorage:
EventInvokeConfig:
Handler:
KmsKeyArn:
Layers:
MemorySize:
PermissionsBoundary:
PropagateTags:
ProvisionedConcurrencyConfig:
ReservedConcurrentExecutions:
Runtime:
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。

暗黙的な API

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

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、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 AWS SAM。これらのリソースとプロパティは AWS SAM、短縮構文を使用して定義します。は、AWS CloudFormation リソースタイプとプロパティタイプ AWS SAM もサポートしています。すべての AWS リソースおよびプロパティタイプ AWS CloudFormation と AWS SAM サポートのリファレンス情報については、「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](#) リソースを使用して、[AWS::Serverless::Api](#) を使用して API を定義および文書化する必要があります。これにより OpenApi、基盤となる Amazon API Gateway リソースをより詳細に設定できます。

AWS CloudFormation フックまたは IAM ポリシーを使用して、API Gateway リソースに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「[フックの登録](#)」および「[リポジトリ](#)」を参照してください。 [apigw-enforce-authorizer](#) GitHub

IAM ポリシーの使用の詳細については、「API Gateway デベロッパーガイド」の「[API ルートに認証を要求する](#)」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、「[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::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 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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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 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証を設定します。

を使用したアクセスの設定の詳細については、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](#)。

タイプ: [ApiAuth](#)

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 同等のプロパティはありません。

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 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。のリスト BinaryMediaTypes が 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 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 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 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 AWS SAM、OpenAPI 定義を変更する必要があります。でインライン OpenAPI 定義を作成して `DefinitionBody`、CORS を有効にします。

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

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、「[デプロイ時にローカルファイルをアップロード](#)」を参照してください。

Type: JSON

必須: いいえ

AWS CloudFormation 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指定されている場合、コンテンツは に渡される DefinitionBody 前に に挿入されたり変更されたりすることがあります CloudFormation。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、 によって参照される外部 OpenApi ファイルではサポートされていません DefinitionUri。代わりに、インクルード [変換で](#) DefinitionBody プロパティを使用して、テンプレートに OpenApi 定義をインポートします。

タイプ: 文字列 | [ApiDefinition](#)

必須: いいえ

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

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 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 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、[Gateway Responses の Api Gateway OpenApi 拡張機能の](#)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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

MergeDefinitions

AWS SAM は API イベントソースから OpenAPI 仕様を生成します。を指定 true して、AWS::Serverless::Api リソースで定義されているインライン OpenAPI 仕様にこれを AWS SAM マージします。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はありません。

デフォルト値: false

タイプ: ブール

必須: いいえ

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

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

Name

API Gateway RestApi リソースの名前

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::ApiGateway::RestApi リソースの [Name](#) プロパティに直接渡されます。

OpenApiVersion

OpenApi 使用する のバージョン。これは、2.0Swagger 仕様用でも、 のような OpenApi 3.0 バージョン用でもかまいません3.0.1。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 デフォルトで というステージを作成します。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。

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 同等のプロパティはありません。

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

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 CloudFormation が指定されているときに生成される AWS::Serverless::Api リソース](#)」を参照してください。生成された AWS CloudFormation リソースの一般的な情報については、「」を参照してください[生成された AWS CloudFormation リソース](#)。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 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、キーと値のペアで構成され CloudFormation 、その中の Tag オブジェクトのリストで構成されます。

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 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 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 関数を含む Hello World AWS SAM テンプレートファイル。これは、動作中の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

```
AWS::Serverless::Function:
  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 統合と CORS 設定とともに外部 Swagger ファイルで定義された API を含む AWS SAM テンプレートスニペット。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にすぎません。

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 を含む AWS SAM テンプレートスニペット。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にすぎません。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Cors: "'*'"
```

```
Auth:
  DefaultAuthorizer: MyCognitoAuthorizer
  Authorizers:
    MyCognitoAuthorizer:
      UserPoolArn:
        Fn::GetAtt: [MyCognitoUserPool, Arn]
```

ApiModelsExample

Models スキーマを含む API を含む AWS SAM テンプレートスニペット。これは AWS SAM テンプレートファイルの一部にすぎず、2 つのモデルスキーマを持つ [AWS::Serverless::Api](#) 定義を示しています。

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 関数を含む Hello World AWS SAM テンプレートファイル。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

```
AWS::Serverless::Api
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

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

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

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に AddApiKeyRequiredToCorsPreflight を設定すると、API キーが Options プロパティに追加される場合があります。

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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

AddDefaultAuthorizerToCorsPreflight

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

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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

ApiKeyRequired

true に設定すると、すべての API イベントに API キーが必要になります。API キーの詳細については、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。

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 同等のプロパティはありません。

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。

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](#)

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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

その他の注意点： SAM は Api OpenApi の定義にオーソライザーを追加します。

DefaultAuthorizer

API Gateway API のデフォルトオーソライザーを指定します。これは、デフォルトで API コールの認証に使用されます。

Note

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

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

ResourcePolicy

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。

タイプ: [ResourcePolicyStatement](#)

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 同等のプロパティはありません。

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、`EndpointConfiguration: PRIVATE` を使用した API に必要です。

UsagePlan

この API に関連付けられた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。

この AWS SAM プロパティは、このプロパティが設定されている場合、[AWS::ApiGateway::UsagePlan](#)、[AWS::ApiGateway::UsagePlanKey](#) および [AWS::ApiGateway::ApiKey](#) の 3 つの追加 AWS CloudFormation リソースを生成します [AWS::ApiGateway::ApiKey](#)。このシナリオの詳細については、「[UsagePlan プロパティが指定されています](#)」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、「[生成された AWS CloudFormation リソース](#)」を参照してください。

タイプ: [ApiUsagePlan](#)

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 同等のプロパティはありません。

例

CognitoAuth

Cognito 認証の例です。

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"
      }
    }
  }]
  IpRangeBlacklist:
    - "10.20.30.40"
```

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、次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 [AWS::ApiGateway::UsagePlan](#)、同じ AWS SAM テンプレート `CreateUsagePlan`: SHAREDにも存在する API 間で共有される [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

タイプ: 文字列

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 同等のプロパティはありません。

Description

使用量プランの説明です。

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::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 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 タグタイプ](#) を使用します。

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::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 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。

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::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、次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証スコープのリストです。

タイプ: リスト

必須: いいえ

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

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

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

タイプ: 文字列

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 同等のプロパティはありません。

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

このプロパティを使用して、オーソライザーの受信リクエスト IdentitySource で を指定できます。の詳細については、[ApiGateway 「Authorizer extension OpenApi IdentitySource」](#) 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 定義で認証のヘッダー名を指定します。

タイプ: 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 がオーソライザーの結果をキャッシュする期間を秒単位で指定する time-to-live (TTL) 期間。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。

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 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、次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::Lambda::Permissions リソースとオーソライザー Lambda 関数間のアクセス許可をプロビジョニングする AWS::Serverless::Api リソースが AWS SAM によって自動的に作成されない `true` ように指定します。

デフォルト値: `false`

タイプ: ブール

必須: いいえ

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

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 SAM が に指定されていると、FunctionArnは自動的に AWS::Lambda::Permissions リソースを作成します。AWS::Serverless::Api。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。

タイプ: 文字列

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 同等のプロパティはありません。

FunctionInvokeRole

Lambda オーソライザー OpenApi の定義にオーソライザー認証情報を追加します。

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 同等のプロパティはありません。

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

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

例

LambdaRequestAuth

YAML

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

LambdaRequestAuthorizationIdentity

このプロパティを使用して、オーソライザーの受信リクエスト `IdentitySource` でを指定できます。の詳細については、[ApiGateway 「Authorizer extension OpenApi IdentitySource」](#) 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
QueryString: 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 がオーソライザーの結果をキャッシュする期間を秒単位で指定する time-to-live (TTL) 期間。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、次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::Lambda::Permissions` リソースとオーソライザー Lambda 関数間のアクセス許可をプロビジョニングする `AWS::Serverless::Api` リソースが、によって自動的に作成 AWS SAM されない `true` ように 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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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 SAM が に指定されていると、FunctionArnは自動的に AWS::Lambda::Permissions リソースを作成します AWS::Serverless::Api。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。

タイプ: 文字列

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 同等のプロパティはありません。

FunctionInvokeRole

Lambda オーソライザー OpenApi の定義にオーソライザー認証情報を追加します。

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 同等のプロパティはありません。

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

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

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

このプロパティを使用して、オーソライザーの受信リクエスト IdentitySource で を指定できます。詳細については、「[オーソライザー OpenApi 拡張機能 IdentitySource](#)」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

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

プロパティ

Header

OpenApi 定義で認証のヘッダー名を指定します。

タイプ: 文字列

必須: いいえ

デフォルト: Authorization

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

ReauthorizeEvery

API Gateway がオーソライザーの結果をキャッシュする時間を指定する time-to-live (TTL) 期間を秒単位で指定します。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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。

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 同等のプロパティはありません。

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、次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。

型: 文字列のリスト

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 同等のプロパティはありません。

AwsAccountWhitelist

許可する AWS アカウント。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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

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

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

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

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

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

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

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

例

リソースポリシーの例

次の例では、2 つの IP アドレスとソース 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

API を定義する OpenAPI ドキュメントです。

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  
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 バケットの名前です。

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::RestApi S3Location 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 キーです。

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::RestApi S3Location データ型の [Key](#) 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は、OpenAPI ファイルのバージョンです。

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 S3Location 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: mybucket-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 AWS SAM、OpenAPI 定義を変更する必要があります。CORS `DefinitionBody` を有効にするには、`DefinitionBody` でインライン OpenAPI 定義を作成します。CORS Configuration が OpenAPI 定義およびプロパティレベルで設定されている場合、はそれらを AWS SAM 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、次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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

AllowHeaders

許可するヘッダーの文字列です。

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 同等のプロパティはありません。

AllowMethods

許可する HTTP メソッドが含まれる文字列です。

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 同等のプロパティはありません。

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することができます。

タイプ: 文字列

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 同等のプロパティはありません。

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。

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 同等のプロパティはありません。

例

CorsConfiguration

CORS 設定の例。これは、CORS が設定された [AWS::Serverless::Api](#) 定義とを示す AWS SAM テンプレートファイルの一部にすぎません [AWS::Serverless::Function](#)。Lambda プロキシ統合または HTTP プロキシ統合を使用する場合、バックエンドは 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: "'www.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': 'www.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。

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::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、は [RegionalCertificateArn](#) のにCertificateArnマッピングされま
すAWS::ApiGateway::DomainName。EndpointConfiguration がに設定されている場合EDGE、は [CertificateArn](#) のにCertificateArnマッピングされま
すAWS::ApiGateway::DomainName。

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ます](#)」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、「[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::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。

と入力します。 [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。

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::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。

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::DomainName リソースの [SecurityPolicy](#) プロパティに直接渡されます。

例

DomainName

DomainName 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。

タイプ: 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::::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。

EvaluateTargetHealth

EvaluateTargetHealth が true の場合、エイリアスレコードは、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、参照されるAWSリソースの正常性を継承します。

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::::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、 を true EvaluateTargetHealth に設定することはできません。

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。

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::::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。

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::`Route53::RecordSetGroup` `RecordSet` リソースの `HostedZoneName` プロパティに直接渡されます。

IPv6

このプロパティが設定されると、はAWS::`Route53::RecordSet`リソースAWS SAMを作成し、指定された <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-route53-recordset.html#cfn-route53-recordset-typeAAAA>のタイプを に設定します HostedZone。

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。

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::::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。

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::::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

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 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 [AWS::CloudFormation::Stack](#) リソースとしてデプロイされます。このリソースには、他の [AWS::Serverless::Application](#) リソースを含めたその他のリソースを複数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、「[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です。

テンプレート URL を指定する場合は、[CloudFormation TemplateUrl ドキュメント](#)で指定されている形式に従い、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、からアプリケーションを取得 [ApplicationLocationObject](#) するために を使用しません 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::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。SAM の Tags プロパティは、キーと値のペアで構成され CloudFormation、その中の Tag 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 が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 `serverlessrepo:semanticVersion: SemanticVersion` も自動的に追加します。

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の長さ。デフォルトではタイムアウトが設定されていません。は、ネストされたスタックが CREATE_COMPLETE 状態になったことを AWS CloudFormation 検出し、ネストされたスタックリソースを親スタック CREATE_COMPLETE のとしてマークし、親スタックの作成を再開します。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終了すると 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/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、次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。

タイプ: 文字列

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 同等のプロパティはありません。

SemanticVersion

アプリケーションのセマンティックバージョンです。

タイプ: 文字列

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 同等のプロパティはありません。

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、「」を参照してください。[AWS::Serverless::Connector を指定したときに生成された AWS CloudFormation リソース](#)。

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](#)。

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、「[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](#) | のリスト [ResourceReference](#)

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 同等のプロパティはありません。

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、リソースからのデータの読み取りを許可する AWS Identity and Access Management (IAM) アクションが含まれています。

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

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

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

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、Amazon DynamoDB テーブルに対して読み書きする AWS Lambda 関数を使用します。

```
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、上記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で使用します。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、次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。

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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。

タイプ: 文字列

必須: 条件に応じて異なります

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

Name

リソースの名前です。

タイプ: 文字列

必須: 条件に応じて異なります

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

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。

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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。

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

ResourceId

リソースの ID です。例: API Gateway API ID。

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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。

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

Type

リソース AWS CloudFormation のタイプ。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。

タイプ: 文字列

必須: 条件に応じて異なります

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

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、次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。

タイプ: 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 に固有 AWS SAM であり、同等の AWS CloudFormation プロパティはありません。

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、「[カスタムランタイムを使用した Lambda 関数の構築](#)」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、「[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  
ReservedConcurrentExecutions: Integer  
Role: String  
RolePath: String  
Runtime: String  
RuntimeManagementConfig: RuntimeManagementConfig  
SnapStart: SnapStart  
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Roleに 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。

Type: JSON

必須: いいえ

AWS CloudFormation 互換性: このプロパティは、AWS::IAM::Role resource. AWS SAM adds this [AssumeRolePolicyDocument](#) property to the generated IAM role for this function のプロパティに似ています。この関数にロールの Amazon リソースネーム (ARN) が提供されている場合、このプロパティは何も実行しません。

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、「[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 このプロパティが設定されている場合、は [AWS::Lambda::Version](#) および [AWS::Lambda::Alias](#) リソースを生成します。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ます](#)」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 [生成された AWS CloudFormation リソース](#)。

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 同等のプロパティはありません。

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 同等のプロパティはありません。

AutoPublishCodeSha256

使用すると、この文字列は CodeUri 値と連携して、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、次のデプロイの問題を解決するためによく使用されます。デプロイパッケージは Amazon S3 の場所に保存され、更新された Lambda 関数コードを持つ新しいデプロイパッケージに置き換えられますが、CodeUri プロパティは変更されません (新しいデプロイパッケージが新しい Amazon S3 の場所にアップロードされ、 が新しい場所に変更 CodeUri されるのとは対照的です)。

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 オブジェクトは、段階的なデプロイ用に設定されています (「」を参照 [サーバーレスアプリケーションの段階的なデプロイ](#))
- AutoPublishAlias プロパティが設定され、デプロイ間で変更されない
- CodeUri プロパティは設定されており、デプロイ間で変更されません。

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ン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 同等のプロパティはありません。

CodeSigningConfigArn

[AWS::Lambda::CodeSigningConfig](#) リソースの ARN で、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` です。
- 関数へのローカルパス。例えば `hello_world/` です。
- [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 を使用して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、「[を使用してデプロイ時にローカルファイルをアップロードする方法 AWS SAMCLI](#)」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::Language拡張機能変換](#) 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

処理できないイベントを Lambda が送信する 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 キューである場合は、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てください。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::CodeDeploy::Application](#) という名前の ServerlessDeploymentApplication (スタックごとに1つ)、という[AWS::CodeDeploy::DeploymentGroup](#)名前の *<function-logical-id>*DeploymentGroup、という[AWS::IAM::Role](#)名前の AWS SAM を作成します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 同等のプロパティはありません。

その他の参照資料: このプロパティの詳細については、「[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 同等のプロパティはありません。

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 同等のプロパティはありません。

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 [AWS::EFS::MountTarget](#) リソースが含まれている場合は、`DependsOn` リソース属性を指定して、関数の前にマウントターゲットが作成または更新されることを確実に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 URL を記述するオブジェクトです。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。

詳細については、「AWS Lambda デベロッパーガイド」の「[関数スケーリング](#)」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 同等のプロパティはありません。

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 関数のコンテナイメージ用の Amazon Elastic Container Registry (Amazon ECR) リポジトリの URI です。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、「[を使用したデフォルトビルド AWS SAM](#)」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。

タイプ: 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 ARN。

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::Functionリソースの [KmsKeyArn](#) プロパティに直接渡されます。

Layers

この関数が必要とする LayerVersion ARN のリストです。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、AWS SAMによって自動的に作成できますCLI。詳細については、「[sam deploy](#)」を参照してください。

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 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、関数のデフォルト 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](#)。
- [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](#)の 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 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 同等のプロパティはありません。

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。

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

この関数の実行ロールとして使用する IAM ロールの ARN 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 [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。

タイプ: 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識別子を指定すると、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、「[カスタムランタイムを使用した Lambda 関数の構築](#)」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。

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るデフォルトのロールに `lambda:createdBy:AWS SAM` タグ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。この `Tags` プロパティは、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るデフォルトのロールに `lambda:createdBy:AWS SAM` タグ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 AWS X-Ray を使用する](#)」を参照してください。

有効な値: [Active|Disabled|PassThrough]

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::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 Virtual Private Cloud (VPC) 内のプライベートリソースにアクセスすることを可能にする設定です。

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 関数の ARN です。

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:::my-bucket/*'
Events:
  ApiEvent:
    Type: Api
    Properties:
      Path: /path
      Method: get
```

ImageConfig 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 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:4c459dd0104ee29ec65dcad056c0b3ddb20d6db76b265ade7eda9a066859b1e
      UpdateRuntimeOn: Manual
```

SnapStart 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イベントを送信する 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 関数の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。sqs: SendMessage は SQS キュー用に追加され、sns: Publish は SNS トピック用に追加されます。

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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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

タイプ: 文字列

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 同等のプロパティはありません。

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、「[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、次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。

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

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

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

PassthroughCondition

True の場合、このデプロイ設定が有効になっている場合、関数の条件は生成された 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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

Role

トラフィックシフト CodeDeploy に使用する IAM ロール ARN。これが提供されている場合、IAM ロールは作成されません。

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 同等のプロパティはありません。

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 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、「[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。

タイプ: 文字列

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 同等のプロパティはありません。

例

DeploymentPreference トラフィック前およびトラフィック後のフックを持つ。

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
```

DeploymentPreference Fn::If 組み込み関数を使用する

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、「ユーザーガイド」の [CodeDeployLambdaAliasUpdate](#) 「ポリシー-AWS CloudFormation」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

```
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。

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 同等のプロパティはありません。

PreTraffic

トラフィックの移行前に実行される Lambda 関数です。

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 同等のプロパティはありません。

例

Hooks

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、次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 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、に存在しない追加のパラメータ「Type」が必要です CloudFormation。

MaximumEventAgeInSeconds

Lambda が処理のために関数に送信するリクエストの最大持続時間です。

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::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 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 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。SAM 限定の追加プロパティである Type 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。SAM 限定の追加プロパティである `Type` 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) です。

タイプ: 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ている自動生成された IAM ロールに必要な許可を追加します。

その他の注意事項: 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、Lambda、EventBridge です。

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 同等のプロパティはありません。

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になっている場合、SQS/SNS リソースは SAM によって自動生成されます。リソースを参照するには、SQS の場合は `<function-logical-id>.DestinationQueue`、SNS の場合は `<function-logical-id>.DestinationTopic` を使用します。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 設定に宛先が指定されていないため、SAM は暗黙的に 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、SAM はこの Lambda 関数に必要なアクセス許可を追加して、送信先の Lambda 関数を呼び出します。

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) です。

タイプ: 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、この関数に関連付けられている自動生成された IAM ロールに必要な許可を追加します。

その他の注意事項: 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、Lambda、EventBridge です。

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 同等のプロパティはありません。

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になっている場合、SQS/SNS リソースは SAM によって自動生成されます。リソースを参照するには、SQS の場合は `<function-logical-id>.DestinationQueue`、SNS の場合は `<function-logical-id>.DestinationTopic` を使用します。タイプが Lambda/ の場合 EventBridge、Destination は必須です。

例

EventInvoke SQS と Lambda の送信先を使用した設定例

この例では、SQS OnSuccess 設定に宛先が指定されていないため、SAM は暗黙的に 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、SAM はこの Lambda 関数に必要なアクセス許可を追加して、送信先の Lambda 関数を呼び出します。

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
```

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

Type

イベントタイプです。

有効な値:

AlexaSkill、Api、CloudWatchEvent、CloudWatchLogs、Cognito、DocumentDB、DynamoDB、

タイプ: 文字列

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 同等のプロパティはありません。

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、次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。

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 同等のプロパティはありません。

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 と body というキーstatusCode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、次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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

Method

この関数が呼び出される HTTP メソッドです。

タイプ: 文字列

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 同等のプロパティはありません。

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。

タイプ: 文字列

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 同等のプロパティはありません。

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

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

RestApiId

RestApi リソースの識別子。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、AWS SAM は生成された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。

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 同等のプロパティはありません。

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

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、次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 互換性: このプロパティは に AWS SAM 固有であり、AWS CloudFormation 同等のプロパティはありません。

AuthorizationScopes

この API、パス、およびメソッドに適用する認証スコープです。

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

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 SAM テンプレートで定義されているオーソライザーのAWS_IAMNONE論理 ID。

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 AWS SAM 固有であり、AWS CloudFormation 同等のプロパティはありません。

InvokeRole

InvokeRole 認証に使用する AWS_IAM を指定します。

タイプ: 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 AWS SAM 固有であり、AWS CloudFormation 同等のプロパティはありません。

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` として `true` を指定すると、`ApiKeyRequired`、`Authorizer` または `ResourcePolicy` に指定された値でグローバルオーソライザーが上書きされます。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つ指定する必要があります。例については、[for が指定されているときにグローバルオーソライザー AWS::Serverless::Api を上書き DefinitionBody を参照してください。](#)

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 同等のプロパティはありません。

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

例

Function-Auth

以下の例は、関数レベルで認証を指定します。

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上書きするには、 を Authorizerとして指定しますNONE。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
```

for が指定されているときにグローバルオーソライザー AWS::Serverless::Api を上書き DefinitionBody 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。

```
AWS::Serverless::Api
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、次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。

型: 文字列のリスト

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 同等のプロパティはありません。

AwsAccountWhitelist

許可する AWS アカウント。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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

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

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

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

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

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

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

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

例

リソースポリシーの例

次の例では、2 つの IP アドレスとソース 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、次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。

タイプ: 文字列

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 同等のプロパティはありません。

Required

特定の API エンドポイント OpenApi の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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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

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、次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

タイプ: ブール

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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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。

AWS Serverless Application Model (AWS SAM) このイベントタイプが設定されると、は [AWS::Events::Rule](#) リソースを生成します。

重要な注意点: [EventBridgeRule](#) は、ではなく、優先するイベントソースタイプです CloudWatchEvent。EventBridgeRule および は、同じ基盤となるサービス、API、AWS CloudFormation リソース CloudWatchEvent を使用します。ただし、AWS SAM は にのみ新機能のサポートを追加します EventBridgeRule。

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

```
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 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、AWS SAM を渡 `true` します。それ以外のENABLED場合は `false` を渡しますDISABLED。

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。

タイプ: 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。

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::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。

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::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「[のイベントとイベントパターン EventBridge](#)」を参照してください。

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。

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。

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::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、次の構文を使用します。

YAML

```
FilterPattern: String  
LogGroupName: String
```

プロパティ

FilterPattern

送信先 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。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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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と一致する場合、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は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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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、次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 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 UserPool 定義されている への参照

タイプ: 文字列

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 同等のプロパティはありません。

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](#) [AWS Lambda](#)」の [Amazon DocumentDB](#) での の使用」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、次の構文を使用します。

YAML

```
BatchSize: Integer
Cluster: String
CollectionName: String
DatabaseName: String
Enabled: Boolean
FilterCriteria: FilterCriteria
FullDocument: 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 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)。

タイプ: 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。

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。

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 DocumentDBEventSourceConfig データ型の [CollectionName](#) プロパティに直接渡されません。

DatabaseName

Amazon DocumentDB クラスター内で使用するデータベースの名前。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 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 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。

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 DocumentDBEventSourceConfig データ型の [FullDocument](#) プロパティに直接渡されません。

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。

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。

タイプ: 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 に固有 AWS SAM であり、同等の AWS CloudFormation プロパティはありません。

SourceAccessConfigurations

認証プロトコルまたは仮想ホストの配列です。 [SourceAccessConfigurations](#) データ型を使用してこれを指定します。

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 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。

タイプ: 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。

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、「[デベロッパーガイドAWS Lambda](#)」の「[Amazon DynamoDBでの使用](#)」を参照してください。AWS Lambda

AWS SAM このイベントタイプが設定されると、は [AWS::Lambda::EventSourceMapping](#) 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、次の構文を使用します。

YAML

```
BatchSize: Integer
BisectBatchOnFunctionError: Boolean
DestinationConfig: DestinationConfig
Enabled: Boolean
FilterCriteria: FilterCriteria
FunctionResponseTypes: List
MaximumBatchingWindowInSeconds: Integer
MaximumRecordAgeInSeconds: Integer
MaximumRetryAttempts: Integer
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ための Amazon Simple Queue Service (Amazon SQS) キューまたは Amazon Simple Notification Service (Amazon SNS) トピック送信先です。

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。

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。

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持続時間です。

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。

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

タイプ: 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 Streams の Amazon リソースネーム (ARN) です。

タイプ: 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

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。サーバーレス関数を 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 ユーザーガイド」の「Amazon とは EventBridge」](#)を参照してください。EventBridge

AWS SAM このイベントタイプが設定されると、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、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

Amazon Simple Queue Service (Amazon SQS) キューを設定します。は、ターゲット呼び出しが失敗した後にイベント EventBridge を送信します。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](#)」および「[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::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 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せん。

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。

タイプ: 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。

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::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。

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::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::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「[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](#)」および「[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::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。

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::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。

使用できる値: DISABLED | ENABLED

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::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 EventBridge がトリガーされたときに が呼び出す 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。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

がターゲット呼び出しの失敗後にイベント EventBridge 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。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「[イベント再試行ポリシー](#)」および「[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、次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。

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::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、 が AWS SAM 作成するデッドレターキューのカスタム名。

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。

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 同等のプロパティはありません。

Type

キューのタイプです。このプロパティが設定されると、 はデッドレターキュー AWS SAM を自動的に作成し、必要な [リソースベースのポリシー](#) をアタッチして、キューにイベントを送信するルールリソースにアクセス許可を付与します。

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

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 同等のプロパティはありません。

例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

ルールがトリガーされたときに 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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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

タイプのイベントソースを記述するオブジェクト HttpApi。

指定されたパスとメソッド OpenApi の定義が 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。

API に指定されたパスとメソッド OpenApi の定義が存在しない場合、SAM は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、次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、を指定しないこのテンプレートで定義された Api イベントで定義されたすべてのパスとメソッドの和集合を含む生成された OpenApi ドキュメント `ServerlessHttpApi` を使用して、というデフォルトの [AWS::Serverless::HttpApi](#) リソースが作成されます `ApiId`。

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。

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 同等のプロパティはありません。

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

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。

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 同等のプロパティはありません。

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。

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 同等のプロパティはありません。

PayloadFormatVersion

統合に送信されるペイロードの形式を指定します。

注: OpenAPI 定義を変更するには SAM PayloadFormatVersion が必要であり、DefinitionBody プロパティで OpenApi 定義されているインラインでのみ機能します。

タイプ: 文字列

必須: いいえ

デフォルト: 2.0

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

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 [AWS::ApiGatewayV2::Stage RouteSettings](#)」の「」を参照してください。

注: HttpApi リソースソースとイベントソースの両方が指定され AWS SAM ている場合、RouteSettings はイベントソースプロパティが優先されます。

タイプ: [RouteSettings](#)

必須: いいえ

AWS CloudFormation 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: OpenAPI 定義を変更するには SAM TimeoutInMillis が必要であり、DefinitionBodyプロパティで OpenApi 定義されているインラインでのみ機能します。

タイプ: 整数

必須: いいえ

デフォルト: 5000

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

例

デフォルト HttpApi イベント

HttpApi デフォルトパスを使用するイベント。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

HttpApi 特定のパスとメソッドを使用するイベント。

YAML

```
Events:
  HttpApiEvent:
    Type: HttpApi
    Properties:
      Path: /
      Method: GET
```

HttpApi 認可

HttpApi オーソライザーを使用するイベント。

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、次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証スコープです。

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せん (存在する場合)。

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 同等のプロパティはありません。

Authorizer

特定の関数用の Authorizer です。IAM 認証を使用するには、AWS_IAM を指定し、テンプレートの Globals セクションで EnableIamAuthorizer に対して 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。

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 同等のプロパティはありません。

例

Function-Auth

関数レベルでの認可の指定

YAML

```
Auth:
  Authorizer: OpenIdAuth
  AuthorizationScopes:
    - scope1
    - scope2
```

IAM 認証

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 認証を使用するには、テンプレートの Globals セクションで EnableIamAuthorizer の true を指定する必要があります。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 ルールを宣言する [AWS::IoT::TopicRule](#) リソースを作成します。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、次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使用する SQL ルールエンジンのバージョンです。

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::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 デベロップャーガイド](#)の AWS IoT SQL リファレンスを参照してください。

タイプ: 文字列

必須: はい

AWS CloudFormation 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](#)[AWS Lambda](#)」の[Amazon Kinesis](#)での使用を参照してください。

AWS SAM このイベントタイプが設定されると、は [AWS::Lambda::EventSourceMapping](#) 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、次の構文を使用します。

YAML

```
BatchSize: Integer
BisectBatchOnFunctionError: Boolean
DestinationConfig: DestinationConfig
Enabled: Boolean
FilterCriteria: FilterCriteria
FunctionResponseTypes: List
MaximumBatchingWindowInSeconds: Integer
MaximumRecordAgeInSeconds: Integer
MaximumRetryAttempts: Integer
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ための Amazon Simple Queue Service (Amazon SQS) キューまたは Amazon Simple Notification Service (Amazon SNS) トピック送信先です。

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。

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。

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持続時間です。

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。

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

タイプ: 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) です。

タイプ: 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

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。

AWS Serverless Application Model (AWS SAM) このイベントタイプが設定されている場合、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、次の構文を使用します。

YAML

```
BatchSize: Integer
Broker: String
DynamicPolicyName: Boolean
Enabled: Boolean
FilterCriteria: FilterCriteria
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 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。


タイプ: 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。

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 `SamAutoGeneratedAMQPolicy` のためにです。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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的に停止するには、false に設定します。

タイプ: ブール

必須: いいえ

AWS CloudFormation 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 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。

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。

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 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

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 AWS Secrets Manager。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。

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

SourceAccessConfigurations

認証プロトコルまたは仮想ホストの配列です。 [SourceAccessConfigurations](#) データ型を使用してこれを指定します。

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 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-3i45j6k17819
      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、「[デベロッパーガイド](#)」の「[Amazon MSK AWS Lambda での の使用](#)」を参照してください。AWS Lambda

AWS Serverless Application Model (AWS SAM) このイベントタイプが設定されると、は [AWS::Lambda::EventSourceMapping](#) リソースを生成します。

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
MaximumBatchingWindowInSeconds: Integer
```

[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。

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。

SourceAccessConfigurations

認証プロトコルの配列、VPC コンポーネント、イベントソースを保護して定義する仮想ホスト。

有効値: 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

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 [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) です。

タイプ: 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

既存のクラスターの Amazon MSK の例

以下は、AWS アカウントに既に存在する Amazon MSK クラスター用の MSK イベントソースタイプの例です。

YAML

```
Events:
  MSKEvent:
    Type: MSK
    Properties:
      StartingPosition: LATEST
```

```
Stream: 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ている Amazon MSK クラスター用の 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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 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 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。これにより、サーバーレス関数が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されます。詳細については、[「Amazon ユーザーガイド」の「Amazon EventBridgeとは」](#)を参照してください。

EventBridge

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 は、新しいスケジューリング機能 [Amazon EventBridge Scheduler](#) を提供す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、一元管理型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、より広範なターゲット API オペレーションとのセットにより、スケジュールされたルールよりも EventBridge スケーラビリティが向上します AWS のサービス。を使用して EventBridge Scheduler、スケジュールに従ってターゲットを呼び出す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

Amazon Simple Queue Service (Amazon SQS) キューを設定します。は、ターゲット呼び出しが失敗するとイベント EventBridge を送信します。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や、Lambda 関数を呼び出すための十分なアクセス許可 EventBridge がいない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「[イベントの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。

と入力します。 [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。

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::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。

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::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。

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::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「[イベントの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。

と入力します。 [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。

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::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。

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::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

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。ターゲット呼び出しが失敗すると、[ガイベント EventBridge](#) を送信します。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「[イベントの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。

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::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。

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 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 [リソーススペースのポリシー](#) 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

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 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。これにより、サーバーレス関数がスケジュールに従ってトリガーされる Amazon EventBridge Scheduler イベントのターゲットとして設定されます。詳細については、[ス EventBridge ケジューラユーザーガイドの「Amazon Scheduler とは EventBridge」](#) 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

Amazon Simple Queue Service (Amazon SQS) キューを設定します。は、ターゲット呼び出しが失敗するとイベント EventBridge を送信します。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や、Lambda 関数を呼び出すための十分なアクセス許可 EventBridge がない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EventBridge](#)」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。

と入力します。 [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。

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::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。

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::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。

と入力します。 [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。

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::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。

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::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。

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::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。

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::::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 `<Function-logical-IDTAkevent-source-name>` 形式のスケジュール名AWS SAMを生成して使用します。このプロパティを `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。

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::::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。

と入力します。 [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

スケジューラが呼び出されたときに EventBridge ケジューラがターゲットに使用する IAM ロールの ARN。

と入力します。 [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。

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::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。

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::Scheduler::Schedule リソースの [ScheduleExpressionTimezone](#) プロパティに直接渡されます。

StartDate

スケジューラ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。

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::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

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::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

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](#) [AWS Lambda](#)」の「[セルフマネージド Apache Kafka](#)」を使用する」を参照してください。

AWS Serverless Application Model (AWS SAM) このイベントタイプが設定されると、は [AWS::Lambda::EventSourceMapping](#) リソースを生成します。

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
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 同等のプロパティはありません。

SourceAccessConfigurations

認証プロトコルの配列、VPC コンポーネント、イベントソースを保護して定義する仮想ホスト。

有効値: 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

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 [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

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、次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 [GetSubscriptionAttributes](#) の「」を参照してください。

タイプ: [SnsFilterPolicy](#)

必須: いいえ

AWS CloudFormation 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。

タイプ: 文字列

必須: いいえ

デフォルト: MessageAttributes

AWS CloudFormation 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 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、は発信者の CloudFormation リージョンをデフォルトとして使用します。

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::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: Boolean | [SqsSubscriptionObject](#)

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 同等のプロパティはありません。

Topic

サブスクライブ先のトピックの ARN です。

タイプ: 文字列

必須: はい

AWS CloudFormation 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、次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。

タイプ: 文字列

必須: いいえ

デフォルト: 10

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

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

QueueArn

既存の SQS キュー ARN を指定します。

タイプ: 文字列

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 同等のプロパティはありません。

QueuePolicyLogicalId

[AWS::SQS::QueuePolicy](#) リソースのカスタム logicalId 名を指定します。

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 同等のプロパティはありません。

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。

タイプ: 文字列

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 同等のプロパティはありません。

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、「[デベロッパーガイド AWS Lambda](#)」の[Amazon SQS](#)での使用」を参照してください。AWS Lambda

このイベントタイプが設定されていると、SAM は [AWS::Lambda::EventSourceMapping](#) 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、次の構文を使用します。

YAML

```
BatchSize: Integer
Enabled: Boolean
FilterCriteria: FilterCriteria
FunctionResponseTypes: List
MaximumBatchingWindowInSeconds: Integer
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。

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。

Queue

キューの ARN です。

タイプ: 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

SQS ポーラーのスケールリング設定により、呼び出し速度を制御し、同時呼び出しの最大数を設定します。

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

基本的な 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た 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。

Syntax

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。

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::Lambda::Function Code 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。

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::Lambda::Function Code データ型の [S3Key](#) プロパティに直接渡されます。

Version

バージョンニングオブジェクトの場合、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。

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::Lambda::Function Code 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: mybucket-name
  Key: mykey-name
  Version: 121212
```

FunctionUrlConfig

指定され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、次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 が使用する認証のタイプ。AWS Identity and Access Management (IAM) を使用してリクエストを承認するには、`AuthType` を `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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 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

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::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 認証を使用します。

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、サーバーレスアプリケーションの API AWS AppSync GraphQL を作成および設定します。

の詳細については AWS AppSync、「[AWS AppSync デベロッパーガイド](#)」の「[とは AWS AppSync](#)」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

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。

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 同等のプロパティはありません。

Auth

GraphQL API の認証を設定します。

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 同等のプロパティはありません。

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 に接続 AWS AppSync するためのデータソースを作成します。 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 同等のプロパティはありません。

DomainName

GraphQL API のカスタムドメイン名です。

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 に直接渡されます。は [AWS::AppSync::DomainNameApiAssociation](#) リソース AWS SAM を自動的に生成します。

Functions

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。

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 同等のプロパティはありません。

Logging

GraphQL API の Amazon CloudWatch ログ記録を設定します。

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

GraphQL API の一意の名前です。

タイプ: 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

GraphQL API 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。

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::AppSync::GraphQLApi リソースの [Name](#) プロパティに直接渡されます。

Resolvers

GraphQL API のフィールドのリゾルバーを設定します。AWS SAM は、[JavaScript パイプライン リゾルバー](#)をサポートします。

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 同等のプロパティはありません。

SchemaInline

SDL 形式の GraphQL スキーマをテキストで表記したものです。

タイプ: 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です。

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。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、「[を使用してデプロイ時にローカルファイルをアップロードする方法 AWS SAMCLI](#)」を参照してください。

タイプ: 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 GraphQL API のタグ (キーと値のペア)。タグを使用して、リソースを識別し、カテゴリ分類します。

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。

例

DynamoDB データソースを使用する GraphQL API

この例では、DynamoDB テーブルをデータソースとして使用する GraphQL API を作成します。

```
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;
}
```

データソースとして Lambda 関数を使用する GraphQL API

この例では、Lambda 関数をデータソースとして使用する GraphQL API を作成します。

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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

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::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 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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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、次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: List of [AuthProvider](#)

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、同等の AWS CloudFormation プロパティはありません。

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 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 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 間のデフォルトの認証タイプ。

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「[認可と認証](#)」を参照してください。

Lambda オーソライザー () を指定すると AWS_LAMBDA、は GraphQL API と Lambda 関数間のアクセス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 AWS SAM を作成します。

タイプ: 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 [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 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
```

これにより、次の AWS SAM テンプレートが作成されます。

```
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、次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承認設定を指定します。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 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 互換性 : このプロパティは、AWS::AppSync::GraphQLApi [AdditionalAuthenticationProvider](#) オブジェクトの [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 間のデフォルトの認証タイプ。

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「[認可と認証](#)」を参照してください。

Lambda オーソライザー () を指定するとAWS_LAMBDA、は GraphQL API と Lambda 関数間のアクセス許可をプロビジョニングする AWS Identity and Access Management (IAM) ポリシー AWS SAM を作成します。

タイプ: 文字列

必須: はい

AWS CloudFormation 互換性 : このプロパティは、AWS::AppSync::GraphQLApi [AdditionalAuthenticationProvider](#) オブジェクトの [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 互換性 : このプロパティは、AWS::AppSync::GraphQLApi [AdditionalAuthenticationProvider](#) オブジェクトの [UserPoolConfig](#) 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、次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、次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 互換性: このプロパティはに固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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、次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。

タイプ: [DeltaSyncConfig](#)

必須: いいえ

AWS CloudFormation 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。

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::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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

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::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る AWS::Serverless::Connector リソースを使用してアクセス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、はデフォルト値の Read と AWS SAM を生成し

まずWrite。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。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 AWS リージョン の。指定しない場合、AWS SAM を使用します [AWS::Region](#)。

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::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。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。

タイプ: 文字列

必須: いいえ。指定しない場合、 のデフォルト値 AWS SAM が適用されますPermissions。

AWS CloudFormation 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。

タイプ: 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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

TableName

テーブルの名前。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 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 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、次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。

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::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。

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::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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

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::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。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。

タイプ: 文字列

必須: いいえ。指定しない場合、を使用して Write アクセス許可をプロビジョニング AWS SAM します [AWS SAM コネクタ](#)。

AWS CloudFormation 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、次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 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。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、「[を使用してデプロイ時にローカルファイルをアップロードする方法 AWS SAMCLI](#)」を参照してください。

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::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 です。

- 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。

のバリエーションが指定されている場合、AWS SAM [NONE | None | none] は `AWS::AppSync::DataSource` [Type](#) オブジェクト `None` の値を生成します。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

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::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。例えば「`Id: !GetAtt createPostItemFunc.FunctionId`」のようになります。
- 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。

を使用する場合 `Id`、他のすべてのプロパティは許可されません。AWS SAM は参照される関数の関数 ID を自動的に渡します。

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 同等のプロパティはありません。

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。

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::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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 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
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。

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::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 関数で使用されるランタイムを記述します。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。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 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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 のフィールドに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

```
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 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 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。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、「[を使用してデプロイ時にローカルファイルをアップロードする方法 AWS SAMCLI](#)」を参照してください。

CodeUri または が指定されていない場合、InlineCode はリクエストを最初のパイプライン関数に InlineCode 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 AWS SAM を生成します。

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::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。

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::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 が指定されていない場合、InlineCodeはリクエストを最初のパイプライン関数にInlineCode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 AWS SAM を生成します。

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::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。

タイプ: 文字列

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 同等のプロパティはありません。

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 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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。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 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 生成されたリゾルバー関数コードを使用し、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
}
```

テンプレートのスニペットを次に示します AWS SAM。

```
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 を指定しませんInlineCode。デプロイ時に、はリゾルバーの次のインラインコード AWS SAM を自動的に生成します。

```
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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

AWS CloudFormation フックまたは IAM ポリシーを使用して、API Gateway リソースに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「[フックの登録](#)」および「[リポジトリ](#)」を参照してください。 [apigw-enforce-authorizer](#) GitHub

IAM ポリシーの使用の詳細については、「API Gateway デベロッパーガイド」の「[API ルートに認証を要求する](#)」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、「[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::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 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証を設定します。

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

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。CORS では OpenAPI 定義を変更 AWS SAM する必要があるため、CORS は `DefinitionBody` プロパティが指定されている場合にのみ機能す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

CorsConfiguration が OpenAPI 定義とプロパティレベルの両方で設定されている場合、は両方の設定ソースをプロパティが優先される状態で AWS SAM マージします。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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。

タイプ: [RouteSettings](#)

必須: いいえ

AWS CloudFormation 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 を指定しない場合 DefinitionBody、 はテンプレート設定 DefinitionBody に基づいて AWS SAM を生成します。

Type: JSON

必須: いいえ

AWS CloudFormation 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指定されている場合、 に渡される DefinitionBody 前に にコンテンツを挿入したり、 を変更 AWS SAM したりできます

AWS CloudFormation。プロパティには、対応するAWS::Serverless::FunctionリソースEventSourceの Authタイプと タイプ HttpApi 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。を指定しない場合、DefinitionUriまたは が指定されている場合、DefinitionBodyはテンプレート設定に基づいて DefinitionBody AWS SAM 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、で参照する外部 OpenApi 定義ファイルではサポートされていません。DefinitionUri。OpenApi 定義をテンプレートにインポートするには、インクルード [変換](#) で 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 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。

を指定するとDescription、AWS SAM は descriptionフィールドを設定して HTTP API リソースOpenApi の定義を変更します。次のシナリオではエラーが発生します。

- DefinitionBody プロパティは、Open API 定義で設定された description フィールドで指定 AWS SAM されます。これにより、解決されない descriptionフィールドの競合が発生します。
- DefinitionUri プロパティが指定され AWS SAM ています。Amazon S3 から取得される Open API 定義は変更されません。

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 同等のプロパティはありません。

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`プロパティの代わりに `DefinitionUri`プロパティを指定するか、OpenAPI 定義 `disableExecuteApiEndpoint` で `x-amazon-apigateway-endpoint-configuration` を定義する必要があります。OpenAPI

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

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 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。

を指定するとName、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 プロパティは、Open API 定義で設定された title フィールドで指定されます。これにより、解決されない title フィールドの競合 AWS SAM が発生します。
- DefinitionUri プロパティが指定され AWS SAM でいます。Amazon S3 から取得される Open API 定義は変更されません。

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 同等のプロパティはありません。

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

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。

タイプ: [RouteSettings](#)

必須: いいえ

AWS CloudFormation 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、は API Gateway の \$default ステージ AWS SAM を使用します。

タイプ: 文字列

必須: いいえ

デフォルト: \$default

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

その他の注意点: Tags プロパティは OpenAPI 定義を変更する必要がある AWS SAM ため、タグは DefinitionBody プロパティが指定されている場合のみ追加されます。DefinitionUri

プロパティが指定されている場合、タグは追加されません。AWS SAM は、自動的に `httpapi:createdBy:SAM` タグを追加します。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 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。この例では、`HttpApi` が AWS SAM 作成するデフォルトの HTTP API を使用します。

YAML

```
AWS::SAM::TemplateFormatVersion: '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
```

HttpApi 認証付き

以下の例は、API エンドポイントで認証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"
```

HttpApi OpenAPI 定義を使用する

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。

は、この HTTP API を参照する HttpApi イベントの欠落している Lambda 統合 AWS SAM をすべて埋めることに注意してください。は、HttpApi イベントが参照する欠落しているパス AWS SAM も追加します。

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
```

HttpApi 設定付き

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証を設定します。

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、次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 互換性: このプロパティはに固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

その他の注意事項: OpenAPI 定義にオーソライザー AWS SAM 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。

タイプ: 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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

EnableIamAuthorizer

API ルートに IAM 認証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 に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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。

詳細と例については、[「API Gateway デベロッパーガイド」の APIs 「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、次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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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。デフォルトでは、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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

FunctionArn

API の認証を提供する 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。

タイプ: 文字列

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 同等のプロパティはありません。

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。

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 同等のプロパティはありません。

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

例

LambdaAuthorizer

LambdaAuthorizer 例

YAML

```
Auth:
```

```
Authorizers:
  MyLambdaAuthorizer:
    AuthorizerPayloadFormatVersion: 2.0
    FunctionArn:
      Fn::GetAtt:
        - MyAuthFunction
        - Arn
    FunctionInvokeRole:
      Fn::GetAtt:
        - LambdaAuthInvokeRole
        - Arn
    Identity:
      Headers:
        - Authorization
```

LambdaAuthorizationIdentity

プロパティを使用して、Lambda オーソライザーの受信リクエスト IdentitySource で を指定できます。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、次の構文を使用します。

YAML

```
Context: List
Headers: List
QueryString: 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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

QueryString

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

ReauthorizeEvery

API Gateway がオーソライザーの結果をキャッシュする時間を指定する time-to-live (TTL) 期間を秒単位で指定します。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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、次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証スコープのリストです。

タイプ: リスト

必須: いいえ

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

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。

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 同等のプロパティはありません。

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 の場合は小文字、 の場合は大文字 `Issuer`、 [AWS::ApiGatewayV2::AuthorizerAudience](#) のように使用できます。

タイプ: マップ

必須: いいえ

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

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: Cors では OpenAPI 定義を変更するために SAM が必要であるため、DefinitionBodyプロパティで OpenApi 定義されているインラインでのみ機能します。

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: HttpApiCorsConfiguration が OpenAPI と プロパティレベルの両方で設定されている場合、はプロパティ AWS SAM 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、次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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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

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

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

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

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

API を定義する OpenAPI ドキュメントです。

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

```
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 バケットの名称です。

タイプ: 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 キーです。

タイプ: 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

バージョン管理されたオブジェクトの場合は、OpenAPI ファイルのバージョンです。

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::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: mybucket-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。

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::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ます](#)」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、「[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::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。

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。

と入力します。 [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。

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::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。

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::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 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。

タイプ: 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::::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。

EvaluateTargetHealth

EvaluateTargetHealth が true の場合、エイリアスレコードは、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、参照されるAWSリソースの正常性を継承します。

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::::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 デイストリビューションの場合、`EvaluateTargetHealth` を `true` に設定することはできません。

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。

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::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称です。`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。

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::Route53::RecordSetGroup` `RecordSet` リソースの [HostedZoneName](#) プロパティに直接渡されます。

IpV6

このプロパティが設定されると、`AWS::Route53::RecordSet` リソース AWS SAM を作成し、指定された <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-route53-recordset.html#cfn-route53-recordset-typeAAAA> のタイプを に設定します `HostedZone`。

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。

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::`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。

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::::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::::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::::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、「[Lambda レイヤーの構築](#)」を参照してください。

重要な注意: の [UpdateReplacePolicy](#) リソース属性のリリース以降 AWS CloudFormation、[AWS::::LayerVersion](#) (推奨) は 同じ利点を提供します [AWS::::LayerVersion](#)。

サーバーレス LayerVersion が変換されると、SAM はリソースの論理 ID も変換するため、リソースが更新され CloudFormation でも古い LayerVersions は によって自動的に削除されません。

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、「[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
  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

この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基づいて解決されます。

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。

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::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) です。

タイプ: 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

こののライセンスに関する情報 LayerVersion。

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::LayerVersionリソースの [LicenseInfo](#) プロパティに直接渡されます。

RetentionPolicy

このプロパティ LayerVersion は、リソースを削除するときに、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 LayerVersion ときに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 同等のプロパティはありません。

その他の注意事項: を指定すると Retain、 は変換された AWS::Lambda::LayerVersion リソース DeletionPolicy: Retain に [でサポートされているリソース属性 AWS SAM](#) 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 のリソース ARN 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://my-bucket/my-layer.zip'
  CompatibleRuntimes:
    - nodejs10.x
    - nodejs12.x
  LicenseInfo: 'Available under the MIT-0 license.'
  RetentionPolicy: Retain
```

LayerContent

[Lambda レイヤー](#)のコンテンツが含まれる ZIP アーカイブです。

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

レイヤーアーカイブの Amazon S3 バケットです。

タイプ: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、AWS::Lambda::LayerVersion Content 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。

タイプ: 文字列

必須: はい

AWS CloudFormation 互換性: このプロパティは、AWS::Lambda::LayerVersion Content 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。

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::LayerVersion Content 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: mybucket-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。

DynamoDB のより高度な機能を使用するには、代わりに [AWS::DynamoDB::Table](#) リソースを使用してください。

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、「[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固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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。SAM の Tags プロパティは Key:Value ペアで構成され CloudFormation 、その中の Tag 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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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。

その他の注意事項: 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

タイプ: 文字列

必須: はい

AWS CloudFormation 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。ステートマシンを使用して、AWS Lambda 関数やその他の AWS リソースをオーケストレーションし、複雑で堅牢なワークフローを形成できます。

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

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、「[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::StateMachine
Properties:
  AutoPublishAlias: String
  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。を指定しない場合DeploymentPreference、AWS SAM は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トラフィックを設定します。

AWS SAM は、Retainデフォルトでバージョンの DeletionPolicyと UpdateReplacePolicyを 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。

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::StepFunctions::StateMachineAliasリソースの [Name](#)プロパティに直接渡されません。

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 JSON や YAML などの AWS SAM テンプレートファイルの形式と一致します。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

DefinitionSubstitutions

ステートマシン定義のプレースホルダー変数のマッピングを指定する string-to-string マップ。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 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。組み込み関数がインラインステートマシン定義で指定されている場合、はこのプロパティにエントリ AWS SAM を追加してステートマシン定義に挿入します。

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 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。

を指定するとDeploymentPreference、はStateMachineVersionArnサブプロパティ値を自動的に AWS SAM 生成します。

タイプ: [DeploymentPreference](#)

必須: いいえ

AWS CloudFormation 互換性: StateMachineVersionArnプロパティ値 AWS SAM を生成してにアタッチDeploymentPreferenceし、AWS::StepFunctions::StateMachineAliasリソースの [DeploymentPreference](#) プロパティDeploymentPreferenceに渡します。

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

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 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。

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::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。

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::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](#)。
- [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](#)の 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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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。

タイプ: 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::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。

タイプ: 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。

Tags

ステートマシンに追加されたタグと、対応する実行ロールを指定する string-to-string マップ。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 との互換性: このプロパティは、AWS::StepFunctions::StateMachine resource. AWS SAM automatically adds a stateMachine:createdBy:SAM tag to this resource and to the default role that is generated for it. の [Tags](#) プロパティに似ています。

Tracing

ステートマシンで AWS X-Ray が有効になってい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 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 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 関数がステートマシンを呼び出すこと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、Roleプロパティが呼び出しを許可する適切なポリシーを設定することを想定していること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エントリにより、ステートマシンは AWS SAM テンプレートファイルに宣言されたリソースを含めることができます。

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は 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。JSON でインラインステートマシン定義を宣言するには、JSON で AWS SAM テンプレートファイルを記述します。

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

Type

イベントタイプです。

有効な値: `Api`、`Schedule`、`ScheduleV2`、`CloudWatchEvent`、`EventBridgeRule`

タイプ: 文字列

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 同等のプロパティはありません。

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、次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証設定です。

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

Method

この関数が呼び出される HTTP メソッドです。

タイプ: 文字列

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 同等のプロパティはありません。

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。

タイプ: 文字列

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 同等のプロパティはありません。

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、AWS SAM は生成された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。

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 同等のプロパティはありません。

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

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

AuthorizationScopes

この API、パス、およびメソッドに適用する認証スコープです。

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

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。

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 同等のプロパティはありません。

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

例

StateMachine-認証

以下の例は、ステートマシンレベルで認証を指定します。

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、次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。

型: 文字列のリスト

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 同等のプロパティはありません。

AwsAccountWhitelist

許可する AWS アカウント。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 互換性: このプロパティは に固有 AWS SAM であり、AWS CloudFormation 同等のプロパティはありません。

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

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

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

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

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

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

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

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

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

例

リソースポリシーの例

次の例では、2 つの IP アドレスとソース 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。

AWS Serverless Application Model (AWS SAM) このイベントタイプが設定されている場合、は [AWS::Events::Rule](#) リソースを生成します。

重要な注意点: [EventBridgeRule](#) は、 の代わりに使用するイベントソースタイプで CloudWatchEvent。EventBridgeRule および は、同じ基盤となるサービス、API、AWS CloudFormation リソース CloudWatchEvent を使用します。ただし、AWS SAM は にのみ新機能のサポートを追加します EventBridgeRule。

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

```
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と、はデフォルトのイベントバス AWS SAM を使用します。

タイプ: 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。

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::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。

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::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「[のイベントとイベントパターン EventBridge](#)」を参照してください。

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。

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記述するオブジェクト。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 ユーザーガイド」](#)の[「Amazon とは EventBridge」](#)を参照してください。 EventBridge

AWS SAM このイベントタイプが設定されると、 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、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

Amazon Simple Queue Service (Amazon SQS) キューを設定します。 は、ターゲット呼び出しが失敗した後にイベント EventBridge を送信します。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、 EventBridge に Lambda 関数を呼び出すための十分なアクセス許可がない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」](#)および[「デッドレターキューの使用」](#)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::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 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せん。

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。

タイプ: 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。

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::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。

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::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 ユーザーガイド」の「Amazon EventBridge 入力変換」](#)を参照してください。 EventBridge

タイプ: [InputTransformer](#)

必須: いいえ

AWS CloudFormation 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 [「のイベントとイベントパターン EventBridge」](#)を参照してください。

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」](#) および [「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::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。

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::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]

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::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 EventBridge がトリガーされたときに が呼び出す 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。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

がターゲット呼び出しの失敗後にイベント EventBridge 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。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「[イベント再試行ポリシー](#)」および「[デッドレターキューの使用](#)」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、次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。

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::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、 が AWS SAM 作成するデッドレターキューのカスタム名。

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。

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 同等のプロパティはありません。

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場合、 はデッドレターキュー AWS SAM を自動的に作成し、必要な [リソースベースのポリシー](#) をアタッチして、キューにイベントを送信するルールリソースにアクセス許可を付与します。

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

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 同等のプロパティはありません。

例

```
DeadLetterConfig
```

```
DeadLetterConfig
```


YAML

```
DeadLetterConfig:  
  Type: SQS  
  QueueLogicalId: MyDLQ
```

Target

ルールがトリガーされたときに 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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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記述するオブジェクト。ステートマシンをスケジュールに基づいてトリガーする 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 ユーザーガイド」の「Amazon とは EventBridge」](#)を参照してください。 EventBridge

AWS Serverless Application Model (AWS SAM) このイベントタイプが設定されると、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、次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

Amazon Simple Queue Service (Amazon SQS) キューを設定します。は、ターゲット呼び出しが失敗した後にイベント EventBridge を送信します。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」](#) および [「デッドレターキューの使用」](#) 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::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 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せん。

Description

ルールの説明です。

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::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 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、AWS SAM を渡 `true` します。それ以外の ENABLED 場合は `false` を渡します DISABLED。

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。

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::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。名前を指定しない場合、一意の物理 ID AWS CloudFormation を生成し、その ID をルール名として使用します。

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::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「[イベント再試行ポリシー](#)」および「[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::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 ケジューラがターゲットに使用する 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 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。

タイプ: 文字列

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 [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。

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::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 EventBridge がトリガーされたときに呼び出す 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。このプロパティ AWS SAM のバージョンでは、単一のターゲットの論理 ID のみを指定できます。

例

CloudWatch イベントをスケジュールする

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

がターゲット呼び出しの失敗後にイベント EventBridge 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。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 [「イベント再試行ポリシー」](#) および [「デッドレターキューの使用」](#) 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、次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。

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::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、 が AWS SAM 作成するデッドレターキューのカスタム名。

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。

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 同等のプロパティはありません。

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場合、 はデッドレターキュー AWS SAM を自動的に作成し、必要な [リソースベースのポリシー](#) をアタッチして、キューにイベントを送信するルールリソースにアクセス許可を付与します。

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

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 同等のプロパティはありません。

例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

ルールがトリガーされたときに 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、次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。

タイプ: 文字列

必須: はい

AWS CloudFormation 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、スケジュールに従ってトリガーする Amazon EventBridge Scheduler 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 Scheduler とは」](#)を参照してください。EventBridge

AWS Serverless Application Model (AWS SAM) このイベントタイプが設定されると、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、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

Amazon Simple Queue Service (Amazon SQS) キューを設定します。は、ターゲット呼び出しが失敗した後にイベント EventBridge を送信します。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EventBridge」](#) 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::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 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。

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::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。

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::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。

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::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。

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::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。

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::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、その ID をスケジュール名として使用します。

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::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 *<State-machine-logical-ID><event-source-name>* 形式のスケジュール名 AWS SAM を生成して使用します。このプロパティを true に設定すると、は一意的物理 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 に固有 AWS SAM であり、同等の AWS CloudFormation プロパティはありません。

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。

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::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::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` ケジューラがターゲットに使用する IAM ロールの ARN。

タイプ: [RoleArn](#)

必須: いいえ

AWS CloudFormation 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。

タイプ: 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。

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::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。

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::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

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::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-demo-bucket
  Key: my-state-machine.asl.json
  Version: 3
Policies:
  - LambdaInvokePolicy:
      FunctionName: !Ref MyFunction
```

生成された AWS CloudFormation リソース

このセクションでは、が AWS テンプレート AWS SAM を処理するときに作成される AWS CloudFormation リソースの詳細について説明します。が AWS SAM 生成する AWS CloudFormation リソースのセットは、指定したシナリオによって異なります。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` プロパティも指定した場合、は AWS SAM さらに `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 リソースを一覧表示し、AWS SAM テンプレートファイルで生成された AWS CloudFormation リソースを参照する方法を示します。

生成された AWS CloudFormation リソースの参照

テンプレート AWS SAM ファイル内で生成された AWS CloudFormation リソースを参照するには、LogicalIdまたは参照可能なプロパティの2つのオプションがあります。

で生成された AWS CloudFormation リソースを参照する LogicalId

AWS SAM が生成する各 AWS CloudFormation リソースには [LogicalId](#)、テンプレートファイル内で一意の英数字 (A~Z、a~z、0~9) 識別子である [LogicalId](#) があります。は、テンプレートファイル内の LogicalIds AWS SAM リソースの AWS SAM を使用して、生成される AWS CloudFormation リソース LogicalIds の を構築します。生成された AWS CloudFormation リソース LogicalId の を使用して、明示的に宣言した リソースの場合と同様に AWS CloudFormation、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できます。AWS CloudFormation および AWS SAM テンプレートの詳細については、ユーザーガイド LogicalIds の [「リソースAWS CloudFormation」](#) 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。スタックが作成された後にのみ AWS Management Console AWS CLI、または AWS SDKs のいずれかを使用して取得できます。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。

参照可能なプロパティによる生成された AWS CloudFormation リソースの参照

生成された一部のリソースについて、はリソースの AWS SAM 参照可能なプロパティ AWS SAM を提供します。このプロパティを使用して、生成された AWS CloudFormation リソースとその AWS SAM テンプレートファイル内のプロパティを参照できます。

Note

生成されたすべての AWS CloudFormation リソース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 リソースシナリオ

次の表は、AWS SAM リソースを生成するシナリオを構成する AWS CloudFormation リソースとプロパティをまとめたものです。シナリオ列のトピックには、そのシナリオに対して生成する AWS SAM 追加の AWS CloudFormation リソースに関する詳細が記載されています。

AWS SAM リソース	ベース AWS CloudFormation リソース	シナリオ
<u>AWS::Serverless::Api</u>	<u>AWS::ApiGateway::RestApi</u>	<ul style="list-style-type: none"> • <u>DomainName</u> プロパティが指定されています • <u>UsagePlan</u> プロパティが指定されています
<u>AWS::Serverless::Application</u>	<u>AWS::CloudFormation::Stack</u>	<ul style="list-style-type: none"> • ベース AWS CloudFormation リソースを生成する以外に、このサーバーレスリソースには追加のシナリオはありません。
<u>AWS::Serverless::Function</u>	<u>AWS::Lambda::Function</u>	<ul style="list-style-type: none"> • <u>AutoPublishAlias</u> プロパティが指定されています • <u>Role</u> プロパティが指定されていない • <u>DeploymentPreference</u> プロパティが指定されています • <u>Api</u> イベントソースが指定されている • <u>HttpApi</u> イベントソースが指定されている • <u>ストリーミングイベントソース</u>が指定されている • <u>イベントブリッジ (またはイベントバス)</u> イベントソースが指定されている • <u>IoTRule</u> 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 CloudForm ation リソース	シナリオ
		<ul style="list-style-type: none"> OnSuccess (または OnFailure) プロパティが Amazon SNS イベントに指定されている OnSuccess (または OnFailure) プロパティが Amazon SQS イベントに指定されている
AWS::Serverless::HttpApi	AWS::ApiGatewayV2::Api	<ul style="list-style-type: none"> StageName プロパティが指定されています 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ます
AWS::Serverless::LayerVersion	AWS::Lambda::LayerVersion	<ul style="list-style-type: none"> ベース AWS CloudFormation リソースを生成する以外に、このサーバーレスリソースには追加のシナリオはありません。
AWS::Serverless::SimpleTable	AWS::DynamoDB::Table	<ul style="list-style-type: none"> ベース AWS CloudFormation リソースを生成する以外に、このサーバーレスリソースには追加のシナリオはありません。
AWS::Serverless::StateMachine	AWS::StepFunctions::StateMachine	<ul style="list-style-type: none"> 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

トピック

- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::Api リソース](#)
- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::Application リソース](#)

- [AWS::Serverless::Connector](#) を指定したときに生成された AWS CloudFormation リソース
- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::Function](#) リソース
- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::GraphQLApi](#) リソース
- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::HttpApi](#) リソース
- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::LayerVersion](#) リソース
- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::SimpleTable](#) リソース
- [AWS CloudFormation が指定されているときに生成される AWS::Serverless::StateMachine](#) リソース

AWS CloudFormation が指定されているときに生成される AWS::Serverless::Api リソース

AWS::Serverless::Api を指定すると、AWS Serverless Application Model (AWS SAM) は常に AWS::ApiGateway::RestApi 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: 該当なし (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 *MyRestApiGammaStage* 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、`MyRestApiDeployment926eeb5ff1` などです。

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、`APIGateway` を指定すると、`AWS::Serverless::Api` は次のシナリオで追加の AWS CloudFormation リソース AWS SAM を生成します。

シナリオ

- [DomainName プロパティが指定されています](#)
- [UsagePlan プロパティが指定されています](#)

DomainName プロパティが指定されています

の DomainName プロパティの Domain プロパティを指定すると、`AWS::Serverless::Api` は `AWS::ApiGateway::DomainName` AWS CloudFormation リソース AWS SAM 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。例えば、`ApiGatewayDomainName926eeb5ff1` などです。

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ます

の UsagePlan プロパティの Auth プロパティを指定すると、`AWS::Serverless::Api` は次のリソース AWS SAM を生成します:

`AWS::ApiGateway::UsagePlan`、`AWS::ApiGateway::UsagePlanKey`、および `AWS::ApiGateway::ApiKey`。

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 CloudFormation が指定されているときに生成される
AWS::Serverless::Application リソース**

を指定すると、AWS Serverless Application Model (AWS SAM)

AWS::Serverless::Applicationは AWS::CloudFormation::Stack 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: 該当なし (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」と入力します。

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 CloudFormation が指定されているときに生成される AWS::Serverless::Function リソース

AWS::Serverless::Function を指定すると、AWS Serverless Application Model (AWS SAM) は常に AWS::Lambda::Function ベース AWS CloudFormation リソースを作成します。

AWS::Lambda::Function

LogicalId: *<function-LogicalId>*

参照可能なプロパティ: 該当なし (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)

この AWS CloudFormation リソースに加えて、 を指定すると、AWS::Serverless::Function は次のシナリオの AWS CloudFormation リソース AWS SAM も生成します。

シナリオ

- [AutoPublishAlias](#) プロパティが指定されています
- [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ます
- [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
- [lotRule](#) イベントソースが指定されている
- [OnSuccess \(または OnFailure\)](#) プロパティが Amazon SNS イベントに指定されている
- [OnSuccess \(または OnFailure\)](#) プロパティが Amazon SQS イベントに指定されている

AutoPublishAlias プロパティが指定されています

の AutoPublishAlias プロパティを指定すると、AWS::Serverless::Function は次の AWS CloudFormation リソース AWS SAM を生成します: AWS::Lambda::Alias および AWS::Lambda::Version。

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 `live`、LogicalId は `Aliaslive MyFunction` 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、`MyFunctionVersion926eeb5ff1` などです。

参照可能なプロパティ: *<function-LogicalId>.Version*

Role プロパティが指定されていない

の Role プロパティが指定されていない場合、AWS::Serverless::Function は AWS::IAM::Role AWS CloudFormation リソース AWS SAM を生成します。

AWS::IAM::Role

LogicalId: *<function-LogicalId>*Role

参照可能なプロパティ: 該当なし (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ます

の DeploymentPreference プロパティを指定すると、AWS::Serverless::Function は次の AWS CloudFormation リソース AWS SAM を生成します: AWS::CodeDeploy::Application および AWS::CodeDeploy::DeploymentGroup。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、は AWS::IAM::Role AWS CloudFormation リソース AWS SAM も生成します。

AWS::CodeDeploy::Application

LogicalId: ServerlessDeploymentApplication

参照可能なプロパティ: 該当なし (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: 該当なし (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: 該当なし (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

の Eventプロパティ `AWS::Serverless::Function` が に設定されていても `Api`、 `RestApiId` プロパティが指定されていない場合、 は `AWS::ApiGateway::RestApi` AWS CloudFormation リソース `AWS SAM` を生成します。

AWS::ApiGateway::RestApi

LogicalId: `ServerlessRestApi`

参照可能なプロパティ: 該当なし (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

の Eventプロパティ `AWS::Serverless::Function` が に設定されていても `HttpApi`、 `ApiId` プロパティが指定されていない場合、 は `AWS::ApiGatewayV2::Api` AWS CloudFormation リソース `AWS SAM` を生成します。

AWS::ApiGatewayV2::Api

LogicalId: `ServerlessHttpApi`

参照可能なプロパティ: 該当なし (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

の Eventプロパティ `AWS::Serverless::Function` がストリーミングタイプの 1 つに設定されている場合、 は `AWS::Lambda::EventSourceMapping` AWS CloudFormation リソース `AWS SAM` 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: 該当なし (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

の Eventプロパティ `AWS::Serverless::Function` が イベントブリッジ (またはイベントバス) タイプの 1 つに設定されている場合、 は `AWS::Events::Rule` AWS CloudFormation リソース AWS SAM 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: 該当なし (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

の Eventプロパティ `AWS::Serverless::Function` が `IoTRule` に設定されている場合、 は `AWS::IoT::TopicRule` AWS CloudFormation リソース AWS SAM を生成します。

AWS::IoT::TopicRule

LogicalId: <function-LogicalId><event-LogicalId>

参照可能なプロパティ: 該当なし (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)

`OnSuccess` (または `OnFailure`) プロパティが Amazon SNS イベントに指定されている

のプロパティの `OnSuccess` (または `OnFailure`)

`DestinationConfigEventInvokeConfig` プロパティ `AWS::Serverless::Function` が指定され、送信先タイプが SNS であるが、送信先 ARN が指定されていない場合、 は `AWS::Lambda::EventInvokeConfig` および の AWS CloudFormation リソース AWS SAM を生成します `AWS::SNS::Topic`。

AWS::Lambda::EventInvokeConfig

LogicalId: <function-LogicalId>EventInvokeConfig

参照可能なプロパティ: 該当なし (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。

OnSuccess (または OnFailure) プロパティが Amazon SQS イベントに指定されている

のプロパティの OnSuccess (または OnFailure)

DestinationConfigEventInvokeConfigプロパティAWS::Serverless::Functionが

指定され、送信先タイプが SQSであるが、送信先 ARN が指定されていない場合、は

AWS::Lambda::EventInvokeConfigおよび の AWS CloudFormation リソース AWS SAM を生成しますAWS::SQS::Queue。

AWS::Lambda::EventInvokeConfig

LogicalId: *<function-LogicalId>*EventInvokeConfig

参照可能なプロパティ: 該当なし (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 CloudFormation が指定されているときに生成される

AWS::Serverless::GraphQLApi リソース

(AWS SAM) テンプレートで AWS::Serverless::GraphQLApiリソースを指定すると AWS Serverless Application Model 、 AWS SAM は常に次の基本 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: 該当なし (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: 該当なし (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: 該当なし (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: 該当なし (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: 該当なし (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、 を指定するAWS::Serverless::GraphQLApiと、次の AWS CloudFormation リソースも生成される AWS SAM 可能性があります。

AWS::AppSync::ApiCache

LogicalId: <graphqlapi-LogicalId>ApiCache

参照可能なプロパティ: 該当なし (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: 該当なし (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: 該当なし (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: 該当なし (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アクセス許可をプロビジョニングすることもできます。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 CloudFormation が指定されているときに生成される AWS::Serverless::HttpApi リソース

を指定すると、AWS Serverless Application Model (AWS SAM) AWS::Serverless::HttpApi は AWS::ApiGatewayV2::Api 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: 該当なし (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)

この AWS CloudFormation リソースに加えて、を指定すると、AWS::Serverless::HttpApi は 次のシナリオの AWS CloudFormation リソース AWS SAM も生成します。

シナリオ

- [StageName プロパティが指定されています](#)
- [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ます](#)

StageName プロパティが指定されています

の StageName プロパティを指定すると、

AWS::Serverless::HttpApi は AWS::ApiGatewayV2::Stage AWS CloudFormation リソース AWS SAM 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 *Gamma*、LogicalId は *MyHttpApiGamma* ステージになります。

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

の StageName プロパティが指定されていない場合、

AWS::Serverless::HttpApi は AWS::ApiGatewayV2::Stage AWS CloudFormation リソース AWS SAM 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ます

の DomainName プロパティの Domain プロパティを指定すると、AWS::Serverless::HttpApi は AWS::ApiGatewayV2::DomainName AWS CloudFormation リソース AWS SAM 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、`ApiGatewayDomainNameV2926eeb5ff1` などです。

参照可能なプロパティ: `<httpapi-LogicalId>.DomainName`

AWS CloudFormation が指定されているときに生成される AWS::Serverless::LayerVersion リソース

を指定すると、AWS Serverless Application Model (AWS SAM)

`AWS::Serverless::LayerVersion` は `AWS::Lambda::LayerVersion` 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: 該当なし (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 CloudFormation が指定されているときに生成される AWS::Serverless::SimpleTable リソース

を指定すると、AWS Serverless Application Model (AWS SAM)

`AWS::Serverless::SimpleTable` は `AWS::DynamoDB::Table` 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: 該当なし (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 CloudFormation が指定されているときに生成される AWS::Serverless::StateMachine 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: 該当なし (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)

この AWS CloudFormation リソースに加えて、 を指定すると、AWS::Serverless::StateMachine は次のシナリオの AWS CloudFormation リソース AWS SAM 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](#)

Role プロパティが指定されていない

の Role プロパティが指定されていない場合、AWS::Serverless::StateMachine は AWS::IAM::Role AWS CloudFormation リソース AWS SAM を生成します。

AWS::IAM::Role

LogicalId: *<statemachine-LogicalId>Role*

参照可能なプロパティ: 該当なし (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ます

の Event プロパティ AWS::Serverless::StateMachine が に設定されていても Api、RestApiId プロパティが指定されていない場合、 は AWS::ApiGateway::RestApi AWS CloudFormation リソース AWS SAM を生成します。

AWS::ApiGateway::RestApi

LogicalId: ServerlessRestApi

参照可能なプロパティ: 該当なし (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

の Event プロパティ `AWS::Serverless::StateMachine` が イベントブリッジ (またはイベントバス) タイプの 1 つに設定されている場合、は `AWS::Events::Rule` AWS CloudFormation リソース AWS SAM 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: 該当なし (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)

でサポートされているリソース属性 AWS SAM

リソース属性は、AWS SAM および AWS CloudFormation リソースに追加して、追加の動作や関係を制御できる属性です。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、で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 AWS CloudFormation。サポートされているリソース属性のうち、一部は対応するリソースの基本生成 AWS CloudFormation リソースにのみコピーされ AWS SAM、一部は対応する AWS CloudFormation リソースから生成されたすべての AWS SAM リソースにコピーされます。対応する AWS CloudFormation リソースから生成された AWS SAM リソースの詳細については、「」を参照してください [生成された AWS CloudFormation リソース](#)。

次の表は、によるリソース属性のサポートをまとめたものです。ただし AWS SAM、以下を前提と [例外](#) しています。

リソース属性	送信先で生成されるリソース
DependsOn	ベース AWS CloudFormation 生成リソースのみ。AWS SAM リソースとベースリソース間のマッピングについては、AWS CloudFormation 「」を参照してください 生成された AWS CloudFormation リソースシナリオ 。
Metadata ^{1, 2}	

リソース属性	送信先で生成されるリソース
条件	対応する AWS CloudFormation AWS SAM リソースから生成されたすべてのリソース。生成された AWS CloudFormation リソースのシナリオについては、「」を参照してください 生成された AWS CloudFormation リソースシナリオ 。
DeletionPolicy	
UpdateReplacementPolicy	

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、「[カスタムランタイムを使用した Lambda 関数の構築](#)」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、「[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。

- の場合 `AWS::Lambda::LayerVersion`、AWS SAM のみのカスタムフィールドは、生成された AWS CloudFormation リソース `DeletionPolicy` の `RetentionPolicy` 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、リソース属性は次の生成された AWS CloudFormation リソースにコピーされません。
 - `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`
- テンプレートに暗黙的に作成された API イベントソースを持つ複数の関数 AWS SAM が含まれている場合、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、関数のリソース属性が異なる場合、生成された `AWS::ApiGateway::RestApi` リソースについては、は、次の優先順位リストに従ってリソース属性 AWS SAM をコピーします。

- UpdateReplacePolicy:
 1. Retain
 2. Snapshot
 3. Delete
- DeletionPolicy:
 1. Retain
 2. Delete

API Gateway 拡張機能

向けに特別に設計された API Gateway 拡張機能は AWS、APIs を設計および管理するための追加のカスタマイズと機能を提供します。これらは、API Gateway に固有の AWS 固有の認証と API 統合をサポートする OpenAPI 仕様の拡張機能です。

API Gateway 拡張機能は、AWS 固有の認証と API Gateway 固有の API 統合をサポートする OpenAPI 仕様の拡張機能です。API Gateway 拡張機能の詳細については、「[OpenAPI への API Gateway 拡張機能の使用](#)」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。でサポートされている API Gateway 拡張機能を確認するには AWS SAM、次の表を参照してください。

API Gateway 拡張機能	でサポート AWS SAM
x-amazon-apigateway-any-メソッドオブジェクト	あり
x-amazon-apigateway-api-キーソースプロパティ	なし
x-amazon-apigateway-auth オブジェクト	あり
x-amazon-apigateway-authorizer オブジェクト	あり
x-amazon-apigateway-authtype プロパティ	あり
x-amazon-apigateway-binary-media-types プロパティ	あり
x-amazon-apigateway-documentation オブジェクト	なし

x-amazon-apigateway-endpoint- 設定オブジェクト	なし
x-amazon-apigateway-gateway-responses オブジェクト	あり
x-amazon-apigateway-gateway-responses.gatewayResponse オブジェクト	あり
x-amazon-apigateway-gateway-response s.responseParameters オブジェクト	あり
x-amazon-apigateway-gateway-response s.responseTemplates オブジェクト	あり
x-amazon-apigateway-integration オブジェクト	あり
x-amazon-apigateway-integration.requestTemplates オブジェクト	あり
x-amazon-apigateway-integration.requestParameters オブジェクト	なし
x-amazon-apigateway-integration.responses オブジェクト	あり
x-amazon-apigateway-integration.response オブジェクト	あり
x-amazon-apigateway-integration.responseTemplates オブジェクト	あり
x-amazon-apigateway-integration.responseParameters オブジェクト	あり
x-amazon-apigateway-request- 検証プロパティ	なし
x-amazon-apigateway-request- 検証オブジェクト	なし
x-amazon-apigateway-request-validators.requestValidator オブジェクト	なし

組み込み関数

組み込み関数は、ランタイムでのみ使用可能なプロパティに値を割り当てることができる組み込み関数です。AWS SAM では、特定の組み込み関数プロパティの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できません。そのため、これを解決するために `AWS::LanguageExtensions` 変換を追加することをお勧めします。`AWS::LanguageExtensions` は、によってホストされるマクロ AWS CloudFormation で、組み込み関数と、に含まれていないデフォルトであるその他の機能を使用できます AWS CloudFormation。

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場合、は値を正しく解析 AWS SAM できません。代わりにAWS::LanguageExtensions変換の使用を検討してください。
詳細については、[「」の「プロパティ」セクション AWS::Serverless::Function](#)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。

でサーバーレスアプリケーションを開発する AWS SAM

このセクションでは、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を提供します。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 APIs、アプリケーションのコード署名など、特定のユースケース AWS SAM での使用に関するトピックも含まれています。AWS 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

- [sam init コマンドを使用してアプリケーションを作成する](#)
- [でインフラストラクチャを定義する AWS SAM](#)
- [でアプリケーションを構築する AWS SAM](#)

sam init コマンドを使用してアプリケーションを作成する

[の開始方法](#)と読み取りが完了したら[AWS Serverless Application Model \(AWS SAM\) の使用方法](#)、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として機能します。sam init コマンドオプションのリスト AWS SAMCLI については、「」を参照してください[sam init](#)。

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) sam init コマンドには、以下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オプションが用意されています。

- インフラストラクチャコードを定義する 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

で説明されているように[チュートリアル: Hello World アプリケーションをデプロイする](#)、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、複数のファイルとフォルダが含まれます。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。Python のバージョンは、`sam init` コマンドが作成した `template.yaml` ファイルに記載されている Python のバージョン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. 開始する 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。
- にデプロイされると AWS クラウド、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。はイメージ AWS SAMCLIをダウンロードし、関数の ディレクトリ Dockerfileに 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ことを選択できます。詳細については、「[AWS X-Ray デベロッパーガイド](#)」の「[とは AWS X-Ray](#)」を参照してください。

アクティブ化すると、AWS SAMCLI が 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 によるモニタリングの設定

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 Application Insights CloudWatch](#)」を参照してください。

アクティブ化すると、AWS SAMCLI が 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

のトラブルシューティングについては、AWS SAMCLI「」を参照してください。[AWS SAMCLI トラブルシューティング](#)。

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。

- [Learning AWS SAM: sam init](#) – の Serverless Land 「Learning AWS SAM」シリーズ YouTube。
- [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する準備が整います。これを行うために必要なタスクの詳細については、[でインフラストラクチャを定義する AWS SAM](#)「」を参照してください。

でインフラストラクチャを定義する AWS SAM

プロジェクトを作成したので、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する準備が整いました AWS SAM。これを行うには、AWS SAM テンプレートを設定して、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。これは AWS SAM プロジェクト内の `template.yaml` ファイルです。

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 APIs を使用したリソースのオーケストレーション、アプリケーションへのコード署名、AWS SAM テンプレートの検証など、特定のユースケースの AWS リソースの定義に関するトピックも含まれています。

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 テンプレートでのリソースアクセスの設定と管理](#)
- [AWS SAM テンプレートを使用して API アクセスを制御する](#)
- [で Lambda レイヤーを使用して効率を向上させる AWS SAM](#)
- [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利用する AWS SAM](#)
- [でスケジューラを使用して EventBridge 時間ベースのイベントを管理する AWS SAM](#)
- [による AWS リソースのオーケストレーション AWS Step Functions](#)
- [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が使用する 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ときは、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 AWS SAM の短縮構文を使用できます。

AWS SAM 短縮構文	関連 AWS リソースでの動作
AWS::Serverless::Api	HTTPS エンドポイントを介して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	は、AWS AppSync GraphQL サーバーレスアプリケーション用の API を作成して設定します。
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 LayerVersion を含む Lambda を作成します。

AWS SAM 短縮構文	関連 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。ステートマシンを使用すると、AWS Lambda 関数やその他の AWS リソースをオーケストレーションして、複雑で堅牢なワークフローを作成できます。

上記のリソースもに記載されています[AWS SAM リソースとプロパティ](#)。

すべての AWS リソースおよびプロパティタイプ AWS CloudFormation と AWS SAM サポートのリファレンス情報については、「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 テンプレートでのリソースアクセスの設定と管理

AWS リソースが相互にやり取りするには、リソース間で適切なアクセス許可とアクセス許可を設定する必要があります。そのためには、AWS Identity and Access Management (IAM) ユーザー、ロール、ポリシーを設定して、安全な方法でやり取りを行う必要があります。

このセクションのトピックはすべて、テンプレートで定義されているリソースへのアクセスの設定に関連しています。このセクションでは、一般的なベストプラクティスから始めます。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間のアクセスとアクセス許可を設定するための 2 つのオプションである AWS SAM コネクタと AWS SAM ポリシーテンプレートについて説明します。最後のトピックでは、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 AWS CloudFormation 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。これを行うには、AWS SAM テンプレート内で相互にやり取りする方法を説明します。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間のデータとイベントのプロビジョニングとReadWriteアクセスをサポートします。AWS SAM コネクタの詳細については、「」を参照してください[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](#)。

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、AWS Lambda 関数、AWS Step Functions ステートマシン、およびそれらが操作するリソース間のアクセスとアクセス許可を管理するためにテンプレートに追加できる、事前定義されたアクセス許可のセットです。AWS SAM ポリシーテンプレートの詳細については、「」を参照してください[AWS SAMポリシーテンプレート](#)。

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 メカニズムによる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取りを容易にするために必要な IAM の専門知識を減らします。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 コネクタは、AWS SAM テンプレートのアクセス許可を定義するシンプルで直感的な短縮構文を提供し、IAM の専門知識を最低限に抑えます。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 コネクタを使用します。
- AWS SAM コネクタは、サポートされている送信 AWS SAM 元Readリソースと送信先リソース間のデータとイベントをプロビジョニングしてWriteアクセスできます。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 コネクタを使用します。

- 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制限されていますが、ポリシーテンプレートはすべての CRUD オペレーションをサポートします。サポートされ、シナリオの 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、「[AWS SAMポリシーテンプレート](#)」を参照してください。
- 他のすべてのシナリオ、または詳細度が必要な場合は、AWS CloudFormation メカニズムを使用します。

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

トピック

- [AWS SAM コネクタとは](#)
- [コネクタの例](#)
- [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](#)
- [コネクタの使用](#)
- [コネクタの仕組み](#)
- [AWS SAM コネクタの利点](#)
- [詳細はこちら](#)
- [フィードバックを送信する](#)

AWS SAM コネクタとは

コネクタは、として識別される AWS Serverless Application Model (AWS SAM) 抽象リソースタイプで `AWS::Serverless::Connector`、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します。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、そ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記述します。AWS SAM その後、は、必要なやり取り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。

```
AWS::TemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
...  
Resources:  
  <source-resource-logical-id>:
```

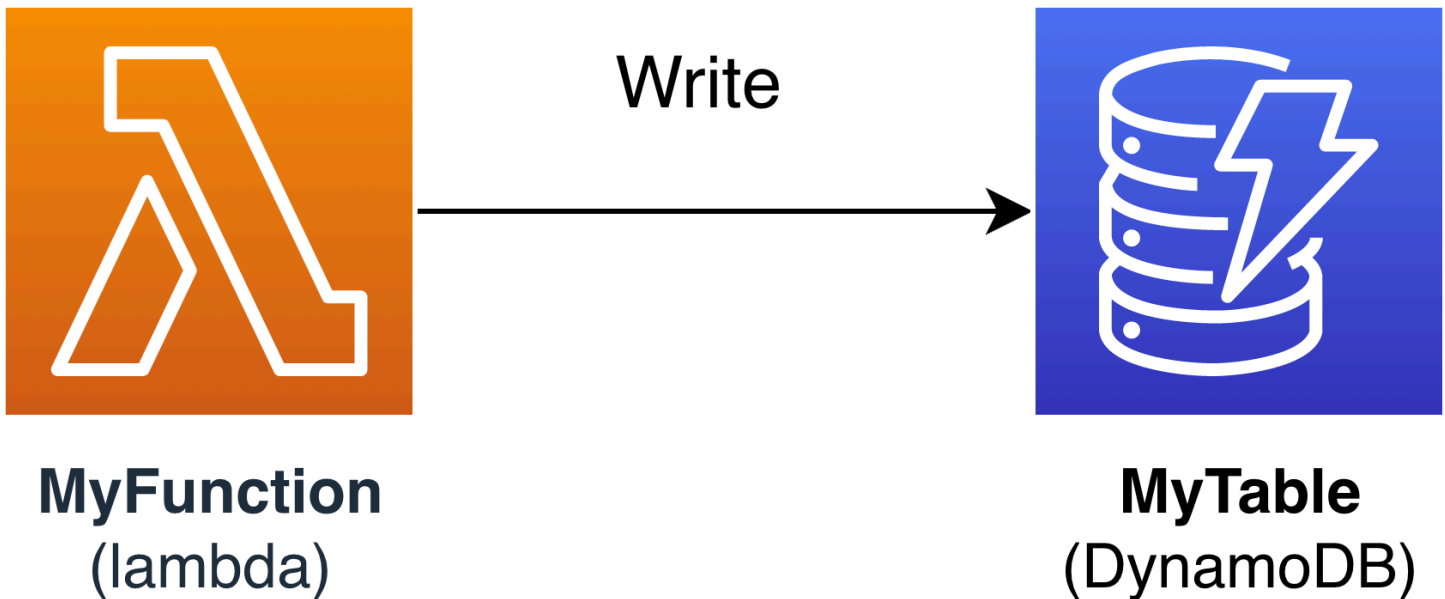
```

Type: <resource-type>
...
Connectors:
  <connector-name>:
    Properties:
      Destination:
        <properties-that-identify-destination-resource>
      Permissions:
        <permission-types-to-provision>
...

```

コネクタの例

この例では、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みます。



```

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で定義できます。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。

コネクタの使用

読み取りおよび書き込み許可を定義する

Read および Write 許可は、単一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
```

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

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>

```

Qualifier を使用して Amazon API Gateway リソースの範囲を絞り込む例を次に示します。

```

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
    ...
```

単一のソースから複数のコネクタを作成する

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: MyBucket
          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
  ...
```

複数の宛先コネクタを作成する

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
```

コネクタを使用してリソース属性を定義する

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。デプロイ時に AWS SAM テンプレートが変換されると、属性は生成されたリソースに渡されません。

```
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 `#source-resource-logical-id><embedded-connector-logical-id#` として自動的に作成されます。

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 テンプレートで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せん。

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。

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、認証機能、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作成し、アプリケーションアーキテクチャに集中することができます。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。

詳細はこちら

AWS SAM コネクタの使用の詳細については、「」を参照してください [AWS::Serverless::Connector](#)。

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スコープできます。

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を含む問題を AWS SAM GitHub プロジェクトに送信します。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ー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 MyQueueAmazon SQS キューの名前を取得します。このキューは、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

プレースホルダー値がない場合でも、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	から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 (AMI) を記述する許可を付与します。		
AthenaQueryPolicy	Athena クエリを実行する許可を付与します。		
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。		

ポリシーテンプレート	説明		
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	にメトリクスを送信するアクセス許可を付与しません CloudWatch。		
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。		
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。		
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	請求履歴の読み取り専用 Cost Explorer API に読み取り専用許可を付与します。		

ポリシーテンプレート	説明		
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。		
DynamoDBReadOnlyPolicy	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。		
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。		

ポリシーテンプレート	説明		
EFSWriteAccessPolicy	書き込みアクセス権と共に Amazon EFS ファイルシステムをマウントする許可を付与します。		
EKSDescribePolicy	Amazon EKS クラスターを記述またはリスト化する許可を付与します。		
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。		
ElasticMapReduceModifyInstancesFleetPolicy	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。		
ElasticMapReduceModifyInstancesGroupsPolicy	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。		

ポリシーテンプレート	説明		
ElasticMapReduceJobFlowsPolicy	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 POST アクセス許可を付与します。		
EventBridgePutEventsPolicy	にイベントを送信するアクセス許可を付与します EventBridge。		
FilterLogEventsPolicy	指定されたロググループから CloudWatch 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。		
FirehoseCreatePolicy	Firehose 配信ストリームを作成、書き込み、更新、削除するアクセス許可を付与します。		
FirehoseWritePolicy	Firehose 配信ストリームに書き込むアクセス許可を付与します。		
KinesisCreatePolicy	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。		
KMSTDecryptPolicy	AWS Key Management Service (AWS KMS) キーを使用して復号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		
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	子アカウントの名前と ID を一覧表示する読み取り専用許可を付与します。		
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。		

ポリシーテンプレート	説明		
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。		
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。		

ポリシーテンプレート	説明		
SageMakerCreateEndpointConfigPolicy	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 SageMaker。		
SageMakerCreateEndpointPolicy	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 SageMaker。		
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。		
SESBulkTemplatedCreatePolicy	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。		
SESBulkTemplatedCreatePolicy_v2	Amazon SES E 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。		
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。		
SESEmailTemplateCreatePolicy	Amazon SES E メールテンプレートを作成、取得、リスト化、更新、および削除する許可を付与します。		
SESSendBouncePolicy	Amazon Simple Email Service (Amazon SES) ID にアクセス SendBounce 許可を付与します。		

ポリシーテンプレート	説明		
SNSCrudPolicy	Amazon SNS トピックを作成、公開、およびサブスクライブする許可を付与します。		
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許可を付与します。		
SSMParameterReadPolicy	このアカウントのシークレットをロードするために、Amazon EC2 Systems Manager (SSM) パラメータストアからのパラメータにアクセスするための許可を付与します。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	このアカウントのシークレットをロードするために、Amazon EC2 Systems Manager (SSM) パラメータストアからのパラメータにアクセスするための許可を付与します。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。		

ポリシーテンプレート	説明		
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: {}
```

ポリシーテンプレートのリスト

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](#)
- [AWSecretsManagerGetSecretValuePolicy](#)
- [AWSecretsManagerRotationPolicy](#)
- [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](#)
- [KMSTDecryptPolicy](#)
- [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

から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 (AMI) を記述する許可を付与します。

```
"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"
        }
      }
    ]
  }
}
]

```

AWSecretsManagerGetSecretValuePolicy

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"
          }
        }
      ]
    }
  }
}
]

```

AWSecretsManagerRotationPolicy

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。

```
"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

に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",  
    ],  
    "Resource": "*"    
  }  
]
```

```
    "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

請求履歴の読み取り専用 AWS Cost Explorer (Cost Explorer) APIs 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

書き込みアクセス権と共に Amazon EFS ファイルシステムをマウントする許可を付与します。

```
"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リスト化する許可を付与します。

```
"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 PUT アクセス許可を付与します。

```
"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

指定されたロググループから CloudWatch 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 配信ストリームを作成、書き込み、更新、削除する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 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) キーを使用して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。keyId は AWS KMS キーエイリアスではなくキー ID である必要があることに注意してください。

```
"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

子アカウントの名前と ID を一覧表示する読み取り専用許可を付与します。

```
"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

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 SageMaker。

```
"Statement": [
  {
    "Action": [
      "sagemaker:CreateEndpointConfig"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint-
config/${endpointConfigName}",
        {
          "endpointConfigName": {
            "Ref": "EndpointConfigName"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

SageMakerCreateEndpointPolicy

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 SageMaker。

```
"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 SES E 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。

Note

ses:SendTemplatedEmail アクションにはテンプレート ARN が必要です。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 SES E 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。

```

"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 SES E メールテンプレートを作成、取得、リスト化、更新、および削除する許可を付与します。

```
"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、およびサブスクライブする許可を付与します。

```
"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"  
          }  
        }  
      ]  
    }  
  }  
]
```



```
        "queueName": {
          "Ref": "QueueName"
        }
      }
    ]
  }
}
```

SQSSendMessagePolicy

Amazon SQS キューにメッセージを送信する許可を付与します。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sqs:SendMessage*"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:${queueName}",
        {
          "queueName": {
            "Ref": "QueueName"
          }
        }
      ]
    }
  }
]
```

SSMParameterReadPolicy

このアカウントのシークレットをロードするために、Amazon EC2 Systems Manager (SSM) パラメータストアからのパラメータにアクセスするための許可を付与します。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

このアカウントのシークレットをロードするために、Amazon EC2 Systems Manager (SSM) パラメータストアからのパラメータにアクセスするための許可を付与します。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"
          }
        }
      ]
    }
  }
]
```

```
    {
      "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 メカニズムによるアクセス許可の管理

AWS リソースへのアクセスを制御するために、AWS Serverless Application Model (AWS SAM) は同じメカニズムを使用できます AWS CloudFormation。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つの主要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。

- 必要な管理ポリシー AWS をアタッチします。
- 特定の AWS Identity and Access Management (IAM) アクセス許可を付与します。

選択したオプションに応じて、ユーザーはアクセス許可を持つ AWS リソースを含む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

管理者権限をユーザーに付与すると、ユーザーは任意の組み合わせの AWS リソースを含む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 管理ポリシーのセットに含まれていることを確認してください。

例えば、[サンプルの Hello World アプリケーションをデプロイするには](#)、次の AWS マネージドポリシーで十分です。

- 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、[ポリシーステートメント](#)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](#)の場合、アプリケーションを実行するには十分AWSLambdaBasicExecutionRoleです。Lambda 実行ロールを作成したら、サンプルの 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、次のプロパティを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 APIs と、サーバーレスアプリケーションが安全であり、有効にした認可を通じてのみアクセスできるようになります。AWS SAM テンプレートで認証を有効にして、API Gateway APIs にアクセスできるユーザーを制御できます。

AWS SAM は、API Gateway APIs へのアクセスを制御するためのいくつか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	<code>AWS::Serverless::HttpApi</code>	<code>AWS::Serverless::Api</code>
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ンツまたは認証トークンによって、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、「[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なりません。

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、[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、[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、「[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、「[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、[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、認証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、を使用してそのメカニズムを使用するようにアプリケーション AWS SAM を設定する方法については、[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

を使用して AWS SAM、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、「[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

- [Lambda オーソライザーの例](#)
- [IAM 許可の例](#)
- [Amazon Cognito ユーザープールの例](#)
- [API キーの例](#)
- [リソースポリシーの例](#)
- [OAuth 2.0/JWT オーソライザーの例](#)
- [カスタマイズされたレスポンスの例](#)

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)

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で、APIs へのアクセスを制御できます。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は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)

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で、APIs へのアクセスを制御できます。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)

テンプレート内で Lambda オーソライザーを定義することで、HTTP APIs へのアクセスを制御できます AWS SAM 。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
```

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。

IAM アクセス許可に を使用する 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:
  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。

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。

ユーザープールの AWS SAM テンプレートセクションの例を次に示します。

```
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 ユーザープールの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

API キーの例

AWS SAM テンプレート内で APIs することで、API へのアクセスを制御できます。これを実行するには、[ApiAuth](#) データ型を使用します。

API キーの AWS SAM テンプレートセクションの例を次に示します。

```
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。

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。

プライベート API の AWS SAM テンプレートの例を次に示します。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。

```
AWS::Serverless::Api::Api
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 APIs「[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。

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。

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を次に示します。

```
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。

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エラーの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。

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 AWS SAM、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 レイヤーを使用すると、Lambda 関数から Lambda レイヤーにコードを抽出し、複数の Lambda 関数で使用できるようになります。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「[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、「[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。

レイヤーを含む Lambda 関数を使用した AWS SAM テンプレートの例を次に示します。

```
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</user>
Windows 8	C:\Users\ <user>\AppData\Roaming\AWS SAM</user>
Windows 10	C:\Users\ <user>\AppData\Roaming\AWS SAM</user>
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。は、ビルドするイメージの名前と、キャッシュに保持 LayerVersions されている AWS SAMCLI 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` を指定すると、`LayerVersionArn` は ARN から `LayerName` およびバージョンを AWS SAMCLI 抽出します。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_64-2dd7ac5ffb30d515926aeffffd
```

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利用する AWS SAM

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大規模なアプリケーションの一部であり、スタンドアロンアーティファクトまたは大規模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してデプロイできます。ネストされたアプリケーションを使用すると、頻繁に使用するコードとを独自のアプリケーションに変換し、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できます。

サーバーレスアーキテクチャが大きくなると、通常、複数のアプリケーションテンプレートで同じコンポーネントが定義されている一般的なパターンが現れます。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、および設定を別々の AWS SAM テンプレートで再利用できるため、1 つのソースからのコードのみを維持できます。これにより、コードと設定の重複が軽減されます。さらに、このモジュラーアプローチにより、開発が合理化され、コードの整理が強化され、サーバーレスアプリケーション間の一貫性が促進されます。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。

を使用して、サーバーレスアプリケーションでこれらのタイプのネストされたアプリケーションの両方 AWS SAM を定義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。

からネストされたアプリケーションを定義する AWS Serverless Application Repository

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。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 Serverless Application Repository、AWS SDK、AWS CLIまたは Lambda コンソールを使用できます。

AWS Serverless Application Repository サーバーレスアプリケーション AWS SAM テンプレートでホストされているアプリケーションを定義するには、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 SAM リソースとしてコピー 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。

- AWS Management Consoleにサインインしていることを確認します。
- デAWS Serverless Application Repository ペロツパーガイドの「アプリケーションの閲覧、検索、デプロイ」セクションのステップ AWS Serverless Application Repository を使用して、でネストするアプリケーションを見つけます。 <https://docs.aws.amazon.com/serverlessrepo/latest/devguide/serverless-app-consuming-applications.html#browse-and-search-applications>
- [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。
- 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

でホストされている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は、ネストされたアプリケーションの共有制限を AWS Serverless Application Repository 継承します。例えば、アプリケーションがパブリックに共有されているが、親アプリケーションを作成した AWS アカウントとのみプライベートに共有され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、AWS アカウントにネストされたアプリケーションをデプロイするアクセス許可がない場合、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。これを行うには、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、「[でアプリケーションとリソースをデプロイする AWS SAM](#)」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ときは、それを承認する必要があります。これを行うには、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。

でスケジューラを使用して EventBridge 時間ベースのイベントを管理する AWS SAM

Amazon EventBridge Scheduler とは

Amazon EventBridge Scheduler は、すべてのサービスで何千万もの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 AWS 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。これを使用して、イベントと定期的な時間ベースの呼び出しをスケジュールできます。また、1 回限りのイベントと、開始時刻と終了時刻のレート式と同期式もサポートしています。

Amazon Scheduler の詳細については、EventBridge スケジューラユーザーガイドの「[Amazon EventBridge Scheduler とは](#)」を参照してください。EventBridge

トピック

- [EventBridge でのスケジューラのサポート AWS SAM](#)
- [でのス EventBridge ケジューライベントの作成 AWS SAM](#)
- [例](#)
- [詳細はこちら](#)

EventBridge でのスケジューラのサポート AWS SAM

AWS Serverless Application Model (AWS SAM) テンプレート仕様は、AWS Lambda および のス EventBridge ケジューラでイベントをスケジュールするために使用できるシンプルで短い構文を提供します AWS Step Functions。

でのス EventBridge ケジューライベントの作成 AWS SAM

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、ス EventBridge 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 スケジューライベントスケジューリングは、未処理のイベントのデッドレターキュー (DLQ) もサポートしています。デッドレターキューの詳細については、スケジューラEventBridge ユーザーガイドの「[ス EventBridge 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、はスケジューラスケジュールのアクセス許可 AWS SAM 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、は DLQ AWS SAM リソースを作成します。

例

でス EventBridge ケジューライベントを定義する基本的な例 AWS SAM

```

Transform: AWS::Serverless-2016-10-31
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: python3.8
      InlineCode: |
        def handler(event, context):
            print(event)
            return {'body': 'Hello World!', 'statusCode': 200}
      MemorySize: 128
    Events:
      Schedule:
        Type: ScheduleV2
        Properties:

```



```
ScheduleExpression: rate(1 minute)
Input: '{"hello": "simple"}'
```

MySFNFunction:

```
Type: AWS::Serverless::Function
```

Properties:

```
Handler: index.handler
```

```
Runtime: python3.8
```

```
InlineCode: |
```

```
def handler(event, context):
    print(event)
    return {'body': 'Hello World!', 'statusCode': 200}
```

```
MemorySize: 128
```

StateMachine:

```
Type: AWS::Serverless::StateMachine
```

Properties:

```
Type: STANDARD
```

Definition:

```
StartAt: MyLambdaState
```

States:

```
MyLambdaState:
```

```
Type: Task
```

```
Resource: !GetAtt MySFNFunction.Arn
```

```
End: true
```

Policies:

```
- LambdaInvokePolicy:
```

```
FunctionName: !Ref MySFNFunction
```

Events:

```
Events:
```

Schedule:

```
Type: ScheduleV2
```

Properties:

```
ScheduleExpression: rate(1 minute)
```

```
Input: '{"hello": "simple"}'
```

詳細はこちら

スCHEDULEV2 EventBridge 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](#)。

による AWS リソースのオーケストレーション AWS Step Functions

[AWS Step Functions](#) を使用して、AWS Lambda 関数やその他の AWS リソースをオーケストレーションし、複雑で堅牢なワークフローを形成できます。Step Functions は、AWS Lambda 関数などの AWS リソースがいつ、どのような条件下で使用されるかをアプリケーションに指示します。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。を使用して [AWS::Serverless::StateMachine](#)、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、移行と条件が必要な場所に追加します。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 ステートマシンを含む AWS SAM テンプレートを管理するには、のバージョン 0.52.0 AWS SAM以降を使用する必要がありますCLI。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) で Step Functions をサポートすると、次のことを実行できます。

- AWS SAM テンプレート内で直接、または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 管理ポリシーを使用してステートマシン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 を直接呼び出しAPIs
- 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

次の AWS SAM テンプレートファイルのスニペット例では、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 ステートマシンを含むサンプル 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「[を使用して Step Functions ステートマシンを作成する AWS SAM](#)」を参照してください。

詳細情報

Step Functions とそのでの使用の詳細については AWS SAM、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: を使用して Step Functions ステートマシンを作成する AWS SAM](#)
- [AWS SAM 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 AWS SAM するには、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します。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の署名付き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、デプロイパイプラインでゲートキーパーコンポーネントを構築する負担から組織を解放できます。

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、Signer AWS 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。その後、[AWS::Lambda::CodeSigningConfig](#) リソースの Amazon リソースネーム (ARN) を使用して関数の [CodeSigningConfigArn](#) プロパティを指定することによって、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。が AWS SAM 作成する 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
```

上記のコマンドが正常に実行されると、署名プロファイルの ARN が返されたことを確認できます。例えば、このようになります。

```
{
  "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、コード署名設定を作成するときに使用する ARN が含まれています。

コード署名設定の作成と関数のコード署名の有効化

次のサンプル AWS SAM テンプレートは、[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 test-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 test-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 test-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。が AWS SAM 作成する 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

コード署名が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 促します。`sam deploy --guided` プロンプトの詳細については、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。

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` コマンドでは、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。

でアプリケーションを構築する AWS SAM

Infrastructure as Code (IaC) を AWS SAM テンプレートに追加したら、`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する準備が整います。このコマンドは、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は、ローカルテストや AWS クラウドへのデプロイなど、アプリケーションの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の準備をします。テストとデプロイの両方で、ビルドアーティファクトを入力として使用します。

を使用して `sam build` 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を使用して、カスタマイズされたビルドを作成できます。の使用法と理由の詳細については `sam build`、このセクションのトピックを参照してください。`sam build` コマンドの使用法の概要については、「[sam build コマンドを使用した構築の概要](#)」を参照してください。

トピック

- [sam build コマンドを使用した構築の概要](#)
- [を使用したデフォルトビルド AWS SAM](#)

- [でカスタマイズされたビルド AWS SAM](#)

sam build コマンド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 SAMCLI) `sam build` コマンドを使用して、ローカルテストやへのデプロイなど、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 AWS クラウド。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。

- の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。
- `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、「[sam init コマンドを使用して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](#)
- [ローカルでのテストとデプロイ](#)
- [ベストプラクティス](#)
- [sam build のオプション](#)
- [トラブルシューティング](#)
- [例](#)
- [詳細はこちら](#)

sam build を使用したアプリケーションの構築

を使用する前に `sam build`、以下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せん。Lambda レイヤーの詳細については、「[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、「[カスタムランタイムを使用した Lambda 関数の構築](#)」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を設定できます。

- `sam init` の使用の詳細については、「[sam init コマンドを使用して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、「[を使用したデフォルトビルド AWS SAM](#)」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 – ビルド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した。
- `build.toml` ファイル – で使用されるビルド設定を含む設定ファイル AWS SAM CLI。

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 AWS SAM テンプレートがそのディレクトリ内にあ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、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 AWS SAMCLI と見なします。

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。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 `.aws-sam` して、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 AWS SAMCLI 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": {
  "GITUB_TOKEN": "TOKEN1"
},
"MyFunction2": {
  "GITUB_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、一時ディレクトリに AWS SAM CLI ビルドされます。これには、ソースコードとプロジェクトファイルのコピーが含まれます。を使用すると `--build-in-source`、は AWS SAM ソースフォルダに直接 CLI ビルドされるため、一時ディレクトリにファイルを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。

トラブルシューティング

のトラブルシューティングについては、AWS SAM CLI 「」を参照してください [AWS SAM CLI トラブルシューティング](#)。

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: Hello World アプリケーションをデプロイする](#)」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
```

```
....
```

はアプリケーションを AWS SAMCLI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
```

```
....
```

はアプリケーションを AWS SAMCLI 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。はコンテナイメージ AWS SAMCLIをダウンロード

ドし、ネイティブ を使用して関数を構築し GoModulesBuilder、結果のバイナリを `.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` コマンドの使用法の詳細については、次を参照してください。

- [Learning AWS SAM: sam build](#) – Serverless Land の「Learning AWS SAM」シリーズYouTube。
- [Learning AWS SAM | sam build | E3](#) – でのサーバーレスランドの「Learning AWS SAM」シリーズYouTube。
- [AWS SAM build: how it provides artifacts for deployment \(Sessions with SAM S2E8\)](#) – での AWS SAM シリーズを含むセッションYouTube。
- [AWS SAM カスタムビルド: Makefiles を使用して SAM \(S2E9\) でビルドをカスタマイズする方法](#) – での AWS SAM シリーズを含むセッションYouTube。

を使用したデフォルトビルド AWS SAM

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](#)
- [の外部での関数の構築 AWS SAM](#)

.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](#) 用に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ない場合、`x86_64` はデフォルトで AWS SAM 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、関数を AWS クラウドにデプロイするときに適切な形式になります。

`--use-container` オプションを使用すると、はデフォルトで [Amazon ECR Public](#) からコンテナイメージを AWS SAM プルします。などの別のリポジトリからコンテナイメージをプルする場合は DockerHub、`--build-image` オプションを使用して代替コンテナイメージの URI を指定できま

す。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するための2つのコマンドの例を次に示します。

```
# 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
```

で使用できる URIs のリストについては `--build-image`、サポートされている多数のランタイム DockerHub URIs を含む [イメージリポジトリ](#) 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。

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: Hello World アプリケーションのデプロイの「[チュートリアル: Hello World アプリケーションをデプロイする](#)」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すると、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、一時ディレクトリに AWS SAM CLI ビルドされます。これには、ソースコードとプロジェクトファイルのコピーが含まれます。を使用すると `--build-in-source`、は AWS SAM ソースフォルダに直接 CLI ビルドされるため、一時ディレクトリにファイルを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。

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

次の AWS SAM テンプレートはコンテナイメージとして構築されます。

```
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
```

の外部での関数の構築 AWS SAM

デフォルトでは、`sam build` を実行すると `sam build` はすべての関数リソース AWS SAM 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。

- の外部ですべての関数リソース AWS SAMを構築する – すべての関数リソースを手動で構築する場合、または別のツールを使用して構築する場合、`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。
- の外部で一部の関数リソース AWS SAMを構築する – の外部で他の関数リソースを構築しながら、関数リソースの一部を構築する場合は AWS SAM、AWS SAM テンプレートでこれを指定できます。AWS SAM

の外部で一部の関数リソースを構築する AWS SAM

の使用時に 関数を AWS SAM スキップするには `sam build`、AWS SAM テンプレートで以下を設定します。

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 `sam build` は以下 AWS SAM を実行します。

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
```

でカスタマイズされたビルド AWS SAM

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したい場合は、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、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 AWS SAM。これには、カスタムランタイムを使用した Lambda 関数の構築と Lambda レイヤーの構築が含まれます。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」に記載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、サーバーレス関数と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代わりに) Lambda レイヤーのみを構築すると、いくつかの点でメリットが得られます。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。

関数の詳細については、「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

- [esbuild による Node.js Lambda関数の構築](#)
- [ネイティブ AOT コンパイルを使用した .NET Lambda 関数の構築](#)
- [Cargo Lambda を使用した Rust Lambda 関数の構築](#)
- [カスタムランタイムを使用した Lambda 関数の構築](#)
- [Lambda レイヤーの構築](#)

esbuild による 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には、esbuild JavaScript バンドルヤ AWS SAMCLIでを使用できます。esbuild バンドラーは、で記述する Lambda 関数をサポートします TypeScript。

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、は esbuild AWS SAM 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、はエントリポイントに Lambda 関数ハンドラー AWS SAM を使用します。

EntryPoints

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]

アプリケーション内で生成された 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 of 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 AWS SAM テンプレートスニペット例では、esbuild を使用して、の TypeScript コードから Node.js Lambda 関数を作成しますhello-world/app.ts。

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello-world/
      Handler: app.handler
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /hello
            Method: get
      Environment:
        Variables:
          NODE_OPTIONS: --enable-source-maps
    Metadata:
      BuildMethod: esbuild
      BuildProperties:
        Format: esm
        Minify: false
        OutExtension:
          - .js=.mjs
        Target: "es2020"
        Sourcemap: true
        EntryPoints:
          - app.ts
      External:
        - "<package-to-exclude>"
```

ネイティブ AOT コンパイルを使用した .NET Lambda 関数の構築

コール AWS Lambda ドスタート時間を短縮するために、ネイティブ先行 (AOT AWS SAM) コンパイルを利用して、AWS Serverless Application Model () を使用して .NET 8 AWS Lambda 関数を構築してパッケージ化します。

トピック

- [.NET 8 ネイティブ AOT の概要](#)
- [.NET 8 Lambda 関数 AWS SAM での の使用](#)
- [インストール条件](#)
- [テンプレートで AWS SAM .NET 8 Lambda 関数を定義する](#)
- [AWS SAM CLI を使用してアプリケーションを構築する](#)
- [詳細はこちら](#)

.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、Dotnet GitHub リポジトリの「[ネイティブ AOT の使用](#)」を参照してください。

.NET 8 Lambda 関数 AWS SAM での の使用

(AWS SAM) を使用して .NET 8 Lambda 関数を設定するには、AWS Serverless Application Model 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。
- AWS SAM テンプレートで .NET 8 Lambda 関数を定義します。
- を使用してアプリケーションを構築します AWS SAMCLI。

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。

- の 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. をインストールするには、AWS SAMCLI「」を参照してください[AWS SAM CLI のインストール](#)。
3. インストールされている のバージョンをアップグレードするにはCLI、AWS SAM「」を参照してください[AWS SAM CLI のアップグレード](#)。

.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 .NET Core Global Tool の詳細については、「」の「[.AWS NET CLI リポジトリの拡張機能](#)」を参照してください GitHub。

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、「[AWS SAM CLI で Docker を使用するためのインストール方法](#)」を参照してください。

テンプレートで AWS SAM .NET 8 Lambda 関数を定義する

AWS SAM テンプレートで .NET8 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. を選択して、開始テンプレートAWS Quick Start Templatesを選択します。
3. Hello World Example テンプレートを選択します。
4. 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n。
5. ランタイムには、 を選択しますdotnet8。
6. パッケージタイプで、 を選択しますZip。
7. スターターテンプレートで、 を選択しますHello World Example using native AOT。

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、「[AWS SAM CLI で Docker を使用するためのインストール方法](#)」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。PublishAot プロパティが .NET 8 プロジェクトファイルで定義されている場合、AWS SAMCLI はネイティブ AOT コンパイルを使用して構築されます。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、「[の .NET 8 ランタイムの紹介 AWS Lambda](#)」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。

Cargo Lambda を使用した Rust Lambda 関数の構築

この機能は のプレビューリリースであり、変更 AWS SAM される可能性があります。

Rust AWS Lambda 関数で AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) を使用します。

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 言語

Rust をインストールするには、Rust 言語ウェブサイトの「[Rust をインストールする](#)」を参照してください。

Cargo Lambda

AWS SAM CLI では、Cargo のサブコマンドである [Cargo Lambda](#) のインストールが必要です。インストール手順については、Cargo Lambda ドキュメントの「[インストール](#)」を参照してください。

Docker

Rust Lambda 関数の構築とテストには Docker が必要です。インストール手順については、「[Docker のインストール](#)」を参照してください。

AWS SAM CLI ベータ機能にオプトインする

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 AWS SAM CLI のコマンドを使用する場合は、--beta-features オプションを使用します。例:

```
$ sam build --beta-features
```

4. AWS SAM CLI でオプトインするよう促すプロンプトが表示されたら、オプション y を選択します。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 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: Rust Lambda 関数で AWS SAM CLI を使用する

AWS SAM テンプレートで任意の AWS SAMCLI コマンドを使用します。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、ランタイムとして Rust を使用してサンプルの Hello World 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ています。

```
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。AWS SAM CLI は `.aws-sam` ディレクトリを作成し、そこにビルドアーティファクトを整理します。関数は Cargo Lambda を使用して構築され、実行可能バイナリとして `.aws-sam/build/HelloWorldFunction/bootstrap` に保存されます。

Note

MacOS で `sam local invoke` コマンドを実行する場合は、呼び出す前に異なる関数を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にお勧めしません。

```
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-
samclisourcebucket-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 関数を呼び出します。

```
$ 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"
```

カスタムランタイムを使用した Lambda 関数の構築

[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な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

Cargo Lambda を使用して Lambda 関数を構築することをお勧めします。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。

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 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: Makefile Builder for Python3.12 (バンドルされたビルダーを使用する代わりに)

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
```

Lambda レイヤーの構築

を使用して AWS SAM カスタム Lambda レイヤーを構築できます。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代わりに) Lambda レイヤーのみを構築すると、いくつかの点でメリットが得られます。これにより、デプロイパッケージのサイズを縮小し、コア関数ロジックと依存関係を分離し、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。

で Lambda レイヤーを構築する方法 AWS SAM

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。これを行う方法の詳細と例については、「」を参照してくださいで [Lambda レイヤーを使用して効率を向上させる AWS SAM](#)。

カスタムレイヤーを構築するには、AWS Serverless Application Model (AWS SAM) テンプレートファイルに宣言し、BuildMethod エントリに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な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、AWS Lambda は環境変数に応じてレイヤーコードを検索します。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は、AWS SAM 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、独立したオブジェクトとして、または AWS Lambda 関数の依存関係としてレイヤーを構築できます。

- 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

次のサンプル AWS SAM テンプレートは、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

次のサンプル AWS SAM テンプレートでは、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、ビルドアーティファクトは python/サブディレクトリにコピーされるため、AWS Lambda はレイヤーコードを見つけることができます。

```
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
```

でサーバーレスアプリケーションをテストする AWS SAM

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確認するためにアプリケーションをテスト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) を使用すると、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、パフォーマンスを確認しながら、対処が必要な問題 (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従うことができる一般的なプラクティス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では、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。デプロイ前のテストは、問題をプロアクティブに特定し、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、その使用の利点を説明し、テストの実行方法を示す例を示します。アプリケーションをテストしたら、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](#)
- [を使用したクラウドテストの概要 sam remote test-event](#)
- [を使用したクラウドでのテストの概要 sam remote invoke](#)
- [を使用してローカル統合テストを自動化する AWS SAM](#)
- [サンプル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 SAMCLI) 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。

の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンドを使用した構築の概要](#).
- [sam deploy コマンドを使用したデプロイの概要](#).

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。

- [の概要 sam local generate-event](#) – ローカルテスト用の AWS のサービス イベントを生成します。
- [の概要 sam local invoke](#) – 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。
- [の概要 sam local start-api](#) – 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 SDKs 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 SAMCLI) sam local generate-event サブコマンドを使用して、サポートされている 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。

- の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。
- 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、`ガアクション`または`タスク` AWS のサービス を実行したときに生成される 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

`sam local generate-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](#).

`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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンドを使用した構築の概要](#).
- [sam deploy コマンドを使用したデプロイの概要](#).

サンプルイベントを生成する

`sam local generate-event` サブコマンドを使用して AWS SAMCLI、サポートされている のイベントを生成します 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
```

- 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

- `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
```

- 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": "EXAMPLE123/5678abcdefghijklmbdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "example-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::example-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:

<code>--region TEXT</code>	Specify the region name you'd like, otherwise the default = us-east-1
<code>--partition TEXT</code>	Specify the partition name you'd like, otherwise the default = aws
<code>--bucket TEXT</code>	Specify the bucket name you'd like, otherwise the default = example-bucket
<code>--key TEXT</code>	Specify the key name you'd like, otherwise the default = test/key
<code>--debug</code>	Turn on debug logging to print debug message generated by AWS SAM CLI and display timestamps.
<code>--config-file TEXT</code>	Configuration file containing default parameter values. [default: samconfig.toml]
<code>--config-env TEXT</code>	Environment name specifying default parameter values in the configuration file. [default: default]
<code>-h, --help</code>	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 MyBucket
```

```
{
  "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": "MyBucket",
    "ownerIdentity": {
      "principalId": "EXAMPLE"
    },
    "arn": "arn:aws:s3:::MyBucket"
  },
  "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 MyBucket > 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
```

詳細はこちら

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。での [SAM シリーズを使用した Serverless Land Sessions YouTube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](#)。

を使用したテストの概要 `sam local invoke`

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。

- の概要については、AWS SAM CLI「」を参照してください [とは AWS SAM CLI](#)。
- `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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンドを使用した構築の概要](#).
- [sam deploy コマンドを使用したデプロイの概要](#).

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、既にクラウド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指す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、「[Terraform と連携させた AWS SAM CLI をローカルでのデバッグおよびテストに使用する方法](#)」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。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 AWS SAM。

```

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。での [SAM シリーズを使用した Serverless Land Sessions YouTube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](#)。

を使用したテストの概要 `sam local start-api`

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) `sam local start-api` サブコマンドを使用して、関数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 AWS Lambda。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。

- の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。
- `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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンドを使用した構築の概要](#).
- [sam deploy コマンドを使用したデプロイの概要](#).

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指す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`

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) `sam local start-lambda` サブコマンドを使用して、AWS Command Line Interface (AWS CLI) または SDK を介して AWS Lambda 関数を呼び出します。SDKs このコマンドは、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。

- の概要については、AWS SAM CLI 「」を参照してくださいとは [AWS SAM CLI](#)。
- `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

`sam local start-lambda` 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-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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンドを使用した構築の概要](#).
- [sam deploy コマンドを使用したデプロイの概要](#).

sam local start-lambda の使用

`sam local start-lambda` 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。

`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. 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、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、に AWS SDK を使用する例です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

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指す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。

詳細はこちら

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

クラウドでテストまたはデプロイする前に Lambda 関数をローカルに呼び出すと、さ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速くテストできます。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を特定する可能性が低くなり、不要なコストを回避できます。さらに、ローカルテストによりデバッグが容易になります。

[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で、Lambda 関数をローカルに呼び出すことができます。はイベント `stdin` として `sam local invoke` も受け入れます。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 [AWS Lambda](#)」の「[を他ののサービスで使用する](#)」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. オーバーライド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": "testBucket",
    "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": "testBucket",  
  },  
  "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 JSON ファイル (`env.json` など) に保存します。

環境変数の値のオーバーライド

JSON ファイルで定義された環境変数で環境変数をオーバーライドするには、`--env-vars` 引数を `invoke` または `start-api` コマンドで使用します。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、「[で Lambda レイヤーを使用して効率を向上させる AWS SAM](#)」を参照してください。

詳細はこちら

関数をローカルで呼び出す実践的な例については、「[完全な AWS SAM ワークショップ](#)」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

ローカルで実行される Amazon API Gateway には、さまざまな利点があります。例えば、API Gateway 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る場合、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、コストを削減できます。さらに、ローカルで実行すると、デバッグが容易になります。

HTTP リクエスト/レスポンス機能のテストに使用できる 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 SAMCLI コマンドを使用します。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、「[を使用したテストの概要 sam local start-api](#)」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、および `HttpApi` 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 Gateway [デベロッパーガイド](#)」の「[HTTP API の AWS Lambda プロキシ統合の使用 APIs](#)」を参照してください。`Api` リソースタイプでのプロキシ統合の詳細については、API Gateway [デベロッパーガイド](#)の「[API Gateway Lambda プロキシの統合について理解する](#)」を参照してください。

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。関数は、定義された HTTP パスにマウントされます。

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. オーバーライド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": "testBucket",
    "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": "testBucket",
  },
  "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 JSON ファイル (env.json など) に保存します。

環境変数の値のオーバーライド

JSON ファイルで定義された環境変数で環境変数をオーバーライドするには、`--env-vars` 引数を `invoke` または `start-api` コマンドで使用します。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、「[で Lambda レイヤーを使用して効率を向上させる AWS SAM](#)」を参照してください。

を使用したクラウドテストの概要 `sam remote test-event`

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` 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て管理します。

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](#).

が既にインストールされている場合は AWS SAM CLI、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、「[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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンドを使用した構築の概要](#).
- [sam deploy コマンドを使用したデプロイの概要](#).
- [を使用して sam sync に同期する方法の概要 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

sam remote test-event コマンドを使用するには、以下のセットアップステップを実行します AWS SAM CLI。

1. 使用するようにを設定する AWS SAM CLI AWS アカウント — Lambda の共有可能なテストイベントは、同じ内のユーザーがアクセスおよび管理できます AWS アカウント。使用するように AWS SAM CLI を設定するには、AWS アカウント「」を参照してください [AWS SAM CLI の設定](#)。
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にアクセスして管理するために使用できる以下のサブコマンドが用意されています。

- 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から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。
- `put` – ローカルファイルから EventBridge スキーマレジストリにイベントを保存します。

を使用してこれらのサブコマンドを一覧表示するには AWS SAM CLI、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。から以下を実行することもできます AWS SAM CLI。

```
$ sam remote test-event delete --help
```

共有可能なテストイベントを取得する

`get` サブコマンドと以下を使用して、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。から以下を実行することもできます AWS SAM CLI。

```
$ 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。から以下を実行することもできます AWS SAM CLI。

```
$ 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
```

同じ名前の共有可能なテストイベントが EventBridge スキーマレジストリに存在する場合、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。から以下を実行することもできます AWS SAM CLI。

```
$ 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、sam remote invoke コマンド AWS クラウド を使用して 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、ローカルで変更して、の Lambda 関数でローカルテストイベントを使用できます AWS クラウド。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アップロードし、EventBridge スキーマレジストリに保存します。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。

の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

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

- [sam remote invoke コマンドの使用](#)
- [sam リモート呼び出しコマンド オプションの使用](#)
- [プロジェクト設定ファイルを設定する](#)
- [例](#)
- [関連リンク](#)

前提条件

`sam remote 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](#).

また、最新バージョンの AWS SAM にアップグレードすることをお勧めします CLI。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 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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンドを使用した構築の概要](#).
- [sam deploy コマンドを使用したデプロイの概要](#).
- [を使用して sam sync に同期する方法の概要 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。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 ID については、「[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。AWS SAM CLI は Lambda 関数を呼び出し、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 ID については、「[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。`sam remote invoke` を使用する場合、AWS SAM CLI は Lambda 設定を自動的に検出して、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

`sam remote test-event` コマンドを使用して AWS SAM CLI、共有可能なテストイベントにアクセスして管理できます。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に変更し、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、「[を使用したクラウドテストの概要 sam remote test-event](#)」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内の 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 SQS で を使用する

Amazon SQSキューにメッセージを送信できます。は AWS SAM CLI以下を返します。

- メッセージ ID
- メッセージ本文の MD5
- 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 ID については、「[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 を使って実行する

Step Functions ステート マシンを呼び出して実行を開始できます。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 ID については、「[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-c9dc5c6fceb Version: $LATEST
END RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceb
REPORT RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceb 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\"}"%
```

AWS SAMCLI レスポンス出力を設定します。

`sam remote invoke` でサポートされているリソースを呼び出すと、AWS SAMCLI は次の内容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、出力レスポンスを設定できます AWS SAM CLI。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レスポンスが JSON 構造で返されます。レスポンスには完全な SDK 出力が含まれます。
- `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":
"U1RBULQgUmVxdWVzdElk0iBkMzBkMjgwZi04MTg4LTQzNzItYmM5NC1jZTBmMTYwM2I2YmIgVmVyc2l1b2JogJExBVEVTV
"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\"}}%
```

Boto3 パラメータをカスタマイズする

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、Boto3 パラメータをカスタマイズすることができます。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 コンピューティングブログの [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 の ARN を指定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"
}%
```

コマンドラインに JSON 文字列をイベントとして指定します。

```
$ 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"
}%
```

レスポンスを AWS SAM CLI JSON 形式で返します。

```
$ 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": "Q3yBcgTwtPaQTV26IKclbECmZikUY0zKY+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 出力を標準出力に返します。

```
$ 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": "4964625141191480677598090639777867595039988349006774274",
  "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

ARN をリソース ID 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。

物理 ID は、 を使用してデプロイするときに作成されます 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 を使用して、レスポンスをストリーミングするように設定された Lambda 関数が含まれる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 関数を操作します。

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 レスポンスストリーミングの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 という名前を付けます `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

```

は、次の構造でプロジェクト AWS SAMCLI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、「[を使用して sam sync に同期する方法の概要 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。設定ファイルで `--watch` オプションが設定されていることから、AWS SAM CLI が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て、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_pt15g6_80000gr/T/tmpavrzdhgp.

Execute the following command to deploy the packaged template

```
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_pt15g6_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-  
samclisourcebucket-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::Stack	lambda-streaming-	
Initiated	ack	nodejs-app	
CREATE_IN_PROGRESS	AWS::IAM::Role	StreamingFunctionRole	-
CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	AwsSamAutoDependencyLayerNest	-
Initiated	ack	erNestStack	
CREATE_IN_PROGRESS	AWS::IAM::Role	StreamingFunctionRole	Resource
Initiated			
CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	AwsSamAutoDependencyLayerNest	Resource
Initiated	ack	erNestStack	
CREATE_COMPLETE	AWS::IAM::Role	StreamingFunctionRole	-
CREATE_COMPLETE	AWS::CloudFormation::Stack	AwsSamAutoDependencyLayerNest	-
Initiated	ack	erNestStack	
CREATE_IN_PROGRESS	AWS::Lambda::Function	StreamingFunction	-
CREATE_IN_PROGRESS	AWS::Lambda::Function	StreamingFunction	Resource
Initiated			
CREATE_COMPLETE	AWS::Lambda::Function	StreamingFunction	-
CREATE_IN_PROGRESS	AWS::Lambda::Url	StreamingFunctionUrl	-

Event	Source	Resource	Target
CREATE_IN_PROGRESS creation	AWS::Lambda::Url	StreamingFunctionUrl	Resource
Initiated CREATE_COMPLETE	AWS::Lambda::Url	StreamingFunctionUrl	-
CREATE_COMPLETE	AWS::CloudFormation::Stack	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-gUmh0833A0vZ
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。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されたレスポンスのリアルタイムでの出力を直ちに開始します。

```
$ 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 が変更を即座に検出して、それらを直ちにデプロイします。以下は、関数コードに変更を行った後の AWS SAM CLI 出力の例です。

```
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 の例

基本的な例

ARN をリソース ID 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については AWS SAMCLI、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](#)

を使用してローカル統合テストを自動化する AWS SAM

[を使用したテストの概要 sam local invoke](#) を使用してコードを手動でテストすることもできますが、では自動統合テストを使用してコードをテスト AWS SAM 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。

で自動統合テストを作成するには AWS SAM、まずローカル Lambda 関数に対してテストを実行してから AWS クラウドにデプロイします。[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ため、テストを 1 回記述し、ローカル Lambda 関数またはデプロイされた Lambda 関数に対して (変更なしで) 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 テンプレートを含む ディレクトリで次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、AWS SDK を使用してテストデータで Lambda 関数を呼び出し、応答を待機し、応答が期待どおりであることを確認します。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。

Python の例を次に示します (他の言語の AWS SDKs の設定は似ています)。

```
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 AWS Lambda でに接続するように AWS SDK がセットアップされます。

サンプル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 のサービスによってトリガーされたときに Lambda 関数が受け取るデータを模倣するサンプルイベントペイロードを生成してカスタマイズできます。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。このアプローチにより、関数をテストするための AWS サービスイベントサンプルを手動で作成するよりも、時間を節約できます。

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
```

でサーバーレスアプリケーションをデバッグする AWS SAM

アプリケーションをテストしたら、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) を使用すると、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は、アプリケーション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。ステップスルーデバッグは、コードを一度に 1 行または命令実行する方法です。内のデバッグモードで Lambda AWS SAM 関数をローカルで呼び出すと CLI、その関数にデバッガーをアタッチできます。デバッガーを使用すると、コード行を行単位でステップスルーし、さまざまな変数の値を表示し、他のアプリケーションと同じように問題を修正できます。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

- [で関数をローカルにデバッグする AWS SAM](#)
- [でデバッグするときに複数のランタイム引数を渡す AWS SAM](#)
- [AWS CloudFormation Linter で AWS SAM アプリケーションを検証する](#)

で関数をローカルにデバッグする AWS SAM

をさまざまな AWS ツールキットやデバッガー AWS SAM とともに使用して、サーバーレスアプリケーションをローカルでテストおよびデバッグできます。Lambda 関数のステップスルーデバッグを使用すると、ローカル環境で一度に 1 行または命令でアプリケーションの問題を特定して修正できます。

ローカルステップスルーデバッグを実行する方法には、ブレークポイントの設定、変数の検査、関数コードの 1 行ずつの実行などがあります。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 を使用してデバッグしたり、AWS SAM デバッグモードで を実行したりできます。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。

AWS ツールキットの使用

AWS ツールキットは、ブレークポイントの設定、変数の検査、関数コードの実行など、多くの一般的なデバッグタスクを一度に 1 行ずつ実行できる統合開発環境 (IDE) プラグインです。AWS ツールキットを使用すると、 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、デプロイが容易になります AWS SAM。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。

で使用できる AWS ツールキットの詳細については AWS SAM、以下を参照してください。

- [AWS Toolkit for Visual Studio Code](#)
- [AWS Cloud9](#)
- [AWS Toolkit for JetBrains](#)

IDEs とランタイムのさまざまな組み合わせで動作するさまざまな AWS ツールキットがあります。次の表に、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を示します。

IDE	ランタイム	AWS ツールキット	ステップスルーデバッグの手順
Visual Studio Code	<ul style="list-style-type: none"> • 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 ツールキット	ステップスルーデバッグの手順
AWS Cloud9	<ul style="list-style-type: none"> Node.js Python 	AWS Cloud9ツールキットが有効になっている AWS ¹	AWS Cloud9 ユーザーガイドの Toolkit を使用した AWSAWS サーバーレスアプリケーションの操作 。
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」
GoLand	Go	AWS Toolkit for JetBrains ²	AWS Toolkit for JetBrains の 「Running (invoking) or debugging a local function」

注意:

1. AWS Cloud9 を使用して AWS SAM アプリケーションをステップスルーデバッグするには、AWS ツールキットを有効にする必要があります。詳細については、「[ユーザーガイド](#)」の [AWS 「ツールキットの有効化AWS Cloud9」](#) を参照してください。
2. を使用して AWS SAM アプリケーションを AWS Toolkit for JetBrains ステップスルーデバッグするには、まずの「[のインストール AWS Toolkit for JetBrains](#)」にある手順に従ってインストールして設定する必要がありますAWS Toolkit for JetBrains。

デバッグモードで AWS SAM のローカル実行

Toolkits との統合に加えて、AWS 「デバッグモード AWS SAM」で を実行して、[ptvsd](#) や [の探索](#)などのサードパーティーのデバッガーにアタッチすることもできます。

AWS SAM デバッグモードで を実行するには、コマンドまたは を [sam local invoke](#) または `--debug-port -d` オプション [sam local start-api](#) とともに使用します。

例:

```
# 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。

でデバッグするときに複数のランタイム引数を渡す AWS SAM

追加のランタイム引数をに渡す AWS SAM ことで、問題を検査したり、変数をより効果的にトラブルシューティングしたりできます。これにより、デバッグプロセスに対する制御と柔軟性が向上し、カスタマイズされたランタイム設定と環境に役立ちます。

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 で AWS SAM アプリケーションを検証する

AWS CloudFormation Linter (cfn-lint) は、テンプレートの AWS CloudFormation 詳細な検証を実行するために使用できるオープンソースツールです。Cfn-lint には、AWS CloudFormation リソース仕様によって導かれる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 `sam validate` で を実行することで、AWS SAM コマンドライン インターフェイス AWS Serverless Application Model (AWS SAM) を介して (AWS SAMCLI) 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、cfn-lint AWS CloudFormation GitHub リポジトリの「[Config 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。

でアプリケーションとリソースをデプロイする AWS SAM

アプリケーションをデプロイすると、AWS リソースが AWS クラウドでプロビジョニングおよび設定され、アプリケーションがクラウドで実行されます。は、基盤となるデプロイメカニズム [AWS CloudFormation](#) として AWS SAM を使用します。AWS SAM 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を、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します。

では AWS SAM、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) システムで 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自動化された一連のステップです。

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を提供します。アプリケーションを手動でデプロイするには、AWS SAMCLI 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。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を提供します。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

- [sam deploy コマンドを使用したデプロイの概要](#)
- [で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](#)
- [CI/CD システムとパイプラインを使用して でデプロイする AWS SAM](#)
- [を使用して sam sync に同期する方法の概要 AWS クラウド](#)

sam deploy コマンドを使用したデプロイの概要

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) `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。

- の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。
- `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](#)
- [sam deploy を使用したアプリケーションのデプロイ](#)
- [ベストプラクティス](#)
- [sam deploy のオプション](#)
- [トラブルシューティング](#)
- [例](#)
- [詳細はこちら](#)

前提条件

sam deploy 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 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](#).
- [sam init コマンドを使用してアプリケーションを作成する](#).
- [sam build コマンド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。AWS SAM CLI は、アプリケーションのデプロイ設定を構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。

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中に、AWS SAM CLI はアプリケーションのデプロイ設定を構成するためのオプションをプロンプトで表示します。

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。

AWS SAM CLI プロンプトに回答して値を指定します。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。

AWS SAM CLI は、レスポンスをプロジェクトの `samconfig.toml` ファイルに書き込みます。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、はアプリケーションを に AWS SAMCLI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、はパッケージを AWS SAMCLI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含む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 ECR) にアップロードします。必要に応じて、AWS SAM CLI は新しいリポジトリを作成します。
- AWS SAMCLI 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。
- は AWS SAMCLI、デプロイされた AWS SAM テンプレートを Lambda 関数の新しいCodeUri値で変更します。

AWS SAM CLI デプロイ出力の例を次に示します。

```

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 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-
samclisourcebucket-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 creation	AWS::Lambda::Function	HelloWorldFunction	Resource
Initiated			
CREATE_COMPLETE	AWS::Lambda::Function	HelloWorldFunction	-
CREATE_IN_PROGRESS	AWS::ApiGateway::RestA pi	ServerlessRestApi	-
CREATE_IN_PROGRESS creation	AWS::ApiGateway::RestA pi	ServerlessRestApi	Resource
Initiated			
CREATE_COMPLETE	AWS::ApiGateway::RestA pi	ServerlessRestApi	-
CREATE_IN_PROGRESS	AWS::Lambda::Permissio n	HelloWorldFunctionHell oWorldPermissionProd	-
CREATE_IN_PROGRESS	AWS::ApiGateway::Deplo yment	ServerlessRestApiDeplo yment47fc2d5f9d	-
CREATE_IN_PROGRESS creation	AWS::Lambda::Permissio n	HelloWorldFunctionHell oWorldPermissionProd	Resource
Initiated			
CREATE_IN_PROGRESS creation	AWS::ApiGateway::Deplo yment	ServerlessRestApiDeplo yment47fc2d5f9d	Resource
Initiated			
CREATE_COMPLETE	AWS::ApiGateway::Deplo yment	ServerlessRestApiDeplo yment47fc2d5f9d	-
CREATE_IN_PROGRESS	AWS::ApiGateway::Stage	ServerlessRestApiProdS tage	-
CREATE_IN_PROGRESS creation	AWS::ApiGateway::Stage	ServerlessRestApiProdS	Resource

```

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-11Z0GSCG28H0M

Key                HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World
                    function
Value              https://njzfhdm1s0.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

AWS CloudFormation 変更セットを表示し、デプロイ前に確認を求める AWS SAM CLI のように設定できます。

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、AWS SAM CLI はデプロイ前に変更を確認するように求めます。以下に例を示します。

```
Waiting for changeset to be created..
```

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

Operation	LogicalResourceId	ResourceType	
Replacement			
+ Add	HelloWorldFunctionHelloWorldPermissionProd	AWS::Lambda::Permission	N/A
+ Add	HelloWorldFunctionRole	AWS::IAM::Role	N/A
+ Add	HelloWorldFunction	AWS::Lambda::Function	N/A
+ Add	ServerlessRestApiDeployment47fc2d5f9d	AWS::ApiGateway::Deployment	N/A
+ Add	ServerlessRestApiProdStage	AWS::ApiGateway::Stage	N/A
+ Add	ServerlessRestApi	AWS::ApiGateway::RestApi	N/A

```

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 テンプレートの Parameters 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。そのためには、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。AWS SAM CLI は、コード署名を設定するように促すプロンプトを表示します。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

- `sam deploy` を使用すると、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

のトラブルシューティングについては、AWS SAMCLI「」を参照してください[AWS SAMCLI トラブルシューティング](#)。

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「[ステップ 3: アプリケーションをデプロイする 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。AWS SAM CLI は、`Docker` を使用して `Lambda` 関数をローカルで構築します。

```

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。AWS SAM CLI は、デプロイ設定の構成をガイドします。次に、はアプリケーションをに AWS SAM CLI 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-samclisourcebucket-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-samclisourcebucket-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 oWorldPermissionProd	AWS::Lambda::Permissio n	N/A
+ Add	HelloWorldFunctionRole	AWS::IAM::Role	N/A
+ Add	HelloWorldFunction	AWS::Lambda::Function	N/A
+ Add	ServerlessRestApiDeplo yment47fc2d5f9d	AWS::ApiGateway::Deplo yment	N/A
+ Add	ServerlessRestApiProdS tage	AWS::ApiGateway::Stage	N/A
+ Add	ServerlessRestApi	AWS::ApiGateway::RestA pi	N/A

```
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 ResourceStatusReason	ResourceType	LogicalResourceId	
CREATE_IN_PROGRESS Initiated	AWS::CloudFormation::Stack	sam-app	User
CREATE_IN_PROGRESS	AWS::IAM::Role	HelloWorldFunctionRole	-
CREATE_IN_PROGRESS creation	AWS::IAM::Role	HelloWorldFunctionRole	Resource Initiated
CREATE_COMPLETE	AWS::IAM::Role	HelloWorldFunctionRole	-
CREATE_IN_PROGRESS	AWS::Lambda::Function	HelloWorldFunction	-
CREATE_IN_PROGRESS creation	AWS::Lambda::Function	HelloWorldFunction	Resource Initiated
CREATE_COMPLETE	AWS::Lambda::Function	HelloWorldFunction	-
CREATE_IN_PROGRESS	AWS::ApiGateway::RestApi	ServerlessRestApi	-
CREATE_IN_PROGRESS creation	AWS::ApiGateway::RestApi	ServerlessRestApi	Resource Initiated
CREATE_COMPLETE	AWS::ApiGateway::RestApi	ServerlessRestApi	-
CREATE_IN_PROGRESS	AWS::Lambda::Permission	HelloWorldFunctionHell	-

	n	oWorldPermissionProd	
CREATE_IN_PROGRESS	AWS::ApiGateway::Deplo	ServerlessRestApiDeplo	-
	yment	yment47fc2d5f9d	
CREATE_IN_PROGRESS creation	AWS::Lambda::Permissio	HelloWorldFunctionHell	Resource
	n	oWorldPermissionProd	Initiated
CREATE_IN_PROGRESS creation	AWS::ApiGateway::Deplo	ServerlessRestApiDeplo	Resource
	yment	yment47fc2d5f9d	Initiated
CREATE_COMPLETE	AWS::ApiGateway::Deplo	ServerlessRestApiDeplo	-
	yment	yment47fc2d5f9d	
CREATE_IN_PROGRESS	AWS::ApiGateway::Stage	ServerlessRestApiProdS	-
		tage	
CREATE_IN_PROGRESS creation	AWS::ApiGateway::Stage	ServerlessRestApiProdS	Resource
		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-
JFML1J0KHJ71
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
```

詳細はこちら

sam deploy コマンドの使用 AWS SAMCLIの詳細については、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 SAMCLI。

で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

を使用すると AWS SAM、アプリケーションを手動でデプロイしたり、デプロイを自動化したりできます。を使用して AWS SAMCLI、アプリケーションを手動でデプロイします。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説明します。

トピック

- [を使用して AWS SAMCLI 手動でデプロイする方法](#)
- [CI/CD システムとパイプラインを使用してデプロイする](#)
- [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](#)

を使用して AWS SAMCLI 手動でデプロイする方法

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。

プロンプトを使用してデプロイを AWS SAM ガイドするには、`--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、コマンドは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てデプロイする

AWS SAM は、パイプラインと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デプロイを自動化するのに役立ちます。を使用して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 AWS SAM できます。複数の CI/CD システムは AWS SAM ビルドコンテナイメージをサポートし、のデプロイのベストプラクティスをカプセル化する複数の CI/CD AWS システム用のデフォルトのパイプラインテンプレートのセット AWS SAM も提供します。

詳細については、「[CI/CD システムとパイプラインを使用してデプロイする AWS SAM](#)」を参照してください。

段階的なデプロイ

アプリケーションを一度にすべてではなく AWS SAM 徐々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、[「ユーザーガイド」の「でのデプロイ設定 CodeDeploy」](#)の使用AWS CodeDeploy」を参照してください。

徐々にデプロイするように AWS SAM アプリケーションを設定する方法については、「」を参照してください[サーバーレスアプリケーションの段階的なデプロイ](#)。

AWS SAM CLI を使用したデプロイのトラブルシューティング

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、認証なしで設定された 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。

- 認証を使用してアプリケーションを設定する。認証の設定については、「[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、認証なしで設定された API Gateway API が存在するアプリケーションをデプロイしてもよいことを示す。

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、「[完全な AWS SAM ワークショップ](#)」の以下を参照してください。

- [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 SAMCLI。

- [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てでデプロイする AWS SAM

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、デプロイ頻度の加速、変更のリードタイムの短縮、デプロイエラーの軽減など、最小限の労力で CI/CD の利点を実現できるようにします。

AWS SAM は、ビルドコンテナイメージを使用して、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、サポートされている多数の AWS Lambda ランタイム用の AWS SAMCLI およびビルドツールが含まれています。これにより、AWS SAM 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。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、「」を参照してください [イメージリポジトリ](#)。

複数の CI/CD システムが AWS SAM ビルドコンテナイメージをサポートしています。どの CI/CD システムを使用するかは、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、のデプロイ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のパイプラインテンプレート AWS のセットも提供します。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。

を使用してサーバーレスアプリケーションをデプロイ AWS SAM するには、主に 2 つのオプションがあります。1) AWS SAMCLI コマンドを使用するように既存のパイプライン設定を変更するか、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 パイプライン設定の例を生成します。

トピック

- [パイプラインとは](#)
- [スターター CI/CD パイプラインを生成する](#)

- [スターターパイプラインをカスタマイズする方法](#)
- [AWS SAM アプリケーションのデプロイを自動化する](#)
- [AWS SAM パイプラインで OIDC 認証を使用する方法](#)
- [を使用してデプロイ時にローカルファイルをアップロードする方法 AWS SAMCLI](#)

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自動化された一連のステップです。を使用すると AWS SAM、[Jenkins](#)、CI/CD [AWS CodePipeline](#)、[GitHub アクション](#) など、多くの一般的な [GitLab CI/CD](#) システムを使用してアプリケーションをデプロイできます。

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 AWS デプロイに役立つ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。

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なら、の AWS SAM 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パイプラインを数分で生成できます。

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。現在、AWS SAM CLI は、[Jenkins](#) [AWS CodePipeline](#)、CI/CD、[GitHub アクション](#)、および [Bitbucket Pipelines](#) のスターター [GitLab CI/CD](#) パイプライン設定の生成をサポートしています。<https://www.jenkins.io/>

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の作成 - パイプラインには、必要なアクセス許可を持つ IAM ユーザーとロール、Amazon S3 バケット、オプションで Amazon ECR リポジトリなど、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。

- を使用している場合は AWS CodePipeline、「」を参照してください [AWS CodePipeline 向けのスターターパイプラインの生成](#)。
- Jenkins、GitLab CI/CD、GitHub Actions、または Bitbucket Pipelines を使用している場合は、「」を参照してください [Jenkins、GitLab CI/CD、GitHub Actions、または Bitbucket Pipelines 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で必要で、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、以前に `sam deploy` コマンドを実行した後、「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」の手順に従って接続を完了します。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。

Jenkins、GitLab CI/CD、GitHub Actions、または Bitbucket Pipelines 用のスターターパイプラインの生成

Jenkins、GitLab CI/CD、GitHub Actions、または Bitbucket Pipelines のスターターパイプライン設定を生成するには、次のタスクをこの順序で実行します。

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。コマンドが 2 つあるのは、管理者 (IAM ユーザーやロールなどのインフラストラクチャ AWS リソースをセットアップするためのアクセス許可を必要とするユーザー) が、デベロッパー (つまり、個々のパイプラインをセットアップするためのアクセス許可だけを必要とし、必要なインフラストラクチャ AWS リソースをセットアップするアクセス許可を必要としないユーザー) よりも多くのアクセス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

を使用するパイプラインには、必要なアクセス許可を持つ IAM ユーザーやロール、Amazon S3 バケット、オプションで Amazon ECR リポジトリなど、特定の AWS リソース AWS SAM 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。

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。これは、以降のステップで必要になるためです。

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 GitHub リポジトリでのアクション
2. GitLab GitLab リポジトリを使用した CI/CD
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 以外の Git リポジトリを使用している場合は GitLab、「外部リポジトリの接続」に関する [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、キー ID とシークレットキーごとに 1 つずつ、2 つの「保護された変数」を作成します。 [GitLab ドキュメント](#) の指示に従ってください。次のステップには 2 つの「可変キー」が必要です。
- Actions を使用している場合は GitHub、キーとシークレットキーごとに 1 つずつ、2 つの「暗号化されたシークレット」を作成します。 [GitHub ドキュメント](#) の指示に従ってください。次のステップには 2 つの「シークレット名」が必要です。
- 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。

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。このキーマッピングは、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。このキーマッピ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 例: [2 ステージ 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

では AWS SAM、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、このセクションの例では、AWS SAM ビルドコンテナイメージでのサーバーレスアプリケーションの構築を自動化するようにさまざまな CI/CD システムを設定する方法を示しています。これらのビルドコンテナイメージにより、AWS SAM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CLI。

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、ビルド AWS SAM 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示します。

トピック

- [を使用したデプロイ AWS CodePipeline](#)
- [Bitbucket Pipelines を使用したデプロイ](#)
- [Jenkins を使用したデプロイ](#)
- [GitLab CI/CD を使用したデプロイ](#)
- [GitHub アクションを使用したデプロイ](#)

を使用したデプロイ AWS CodePipeline

AWS SAM アプリケーションのビルド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ファイルに次の操作を行う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します。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、AWS SAM テンプレートファイル内のすべての関数とレイヤーを `runtime: nodejs20.x` で宣言し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、「[イメージリポジトリ](#)」を参照してください。

Bitbucket Pipelines を使用したデプロイ

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` ファイルには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。

- 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します。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、AWS SAM テンプレートファイル内のすべての関数とレイヤーを `ImageUri` で宣言していることを前提としています `runtime: nodejs20.x`。

```
image: public.ecr.aws/sam/build-nodejs20.x

pipelines:
  branches:
    main: # branch name
      - step:
          name: Build and Package
          script:
            - sam build
            - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

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、「[イメージリポジトリ](#)」を参照してください。

Jenkins を使用したデプロイ

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 [Jenkins](#) パイプラインを設定するには、以下を実行する行が含まれている Jenkinsfile 必要があります。

- 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。
- 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します。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、AWS SAM テンプレートファイル内のすべての関数とレイヤーを `ImageUri` で宣言していることを前提としています `runtime: nodejs20.x`。

```
pipeline {
  agent { docker { image 'public.ecr.aws/sam/build-nodejs20.x' } }
  stages {
    stage('build') {
```



```
    steps {
      sh 'sam build'
      sh 'sam deploy --no-confirm-changeset --no-fail-on-empty-changeset'
    }
  }
}
```

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、「[イメージリポジトリ](#)」を参照してください。

GitLab CI/CD を使用したデプロイ

[GitLab](#) パイプラインを設定して AWS SAM アプリケーションの構築とデプロイを自動化するには、`gitlab-ci.yml` ファイルには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します。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、AWS SAM テンプレートファイル内のすべての関数とレイヤーを `runtime: nodejs20.x` で宣言し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、「[イメージリポジトリ](#)」を参照してください。

GitHub アクションを使用したデプロイ

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 [GitHub](#) パイプラインを設定するには、まずホストに AWS SAM コマンドラインインターフェイス (CLI) をインストールする必要があります。GitHub ワークフローの [GitHub アクション](#) を使用して、この設定に役立てることができます。

次の GitHub ワークフロー例では、一連の GitHub アクションを使用して Ubuntu ホストを設定し、AWS SAM CLI コマンドを実行して AWS SAM アプリケーションを構築してデプロイします。

```
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、「[イメージリポジトリ](#)」を参照してください。

AWS SAM パイプラインで OIDC 認証を使用する方法

AWS Serverless Application Model (AWS SAM) は、Bitbucket、アクション、GitHub 継続的インテグレーションおよび GitLab 継続的デリバリー (CI/CD) プラットフォームの 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。それ以外の場合は、AWS SAM パイプラインへのアクセスを制御するために複数の AWS Identity and Access Management (IAM) ユーザーを作成および管理する必要があります。

AWS SAM パイプラインで OIDC をセットアップする

Sam pipeline bootstrap 設定プロセス中に、次の操作を実行して 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。

- 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。
 - GitHub コードリポジトリが属する組織。
 - GitHub リポジトリ名。
 - デプロイ元のブランチ。
- AWS SAM は、入力された 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。
- 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) リソースがすでに存在する場合 AWS アカウント、AWS SAM は新しいリソースを作成する代わりにそれを参照します。

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 パイプラインでの OIDC の使用の詳細については、「」を参照してください[sam pipeline bootstrap](#)。

を使用してデプロイ時にローカルファイルをアップロードする方法 AWS SAMCLI

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。基本的な例として、AWS Lambda 関数コードをインフラストラクチャコードから分離します。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。

アプリケーションを にデプロイする場合 AWS クラウド、 は、まずローカルファイルを Amazon Simple Storage Service (Amazon S3) AWS などのアクセス可能なサービスにアップロード AWS CloudFormation する必要があります。AWS SAM CLI を使用すると、この処理を自動化できます。sam deploy または sam package コマンドを使用して、次を実行します。

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を自動的に更新します。

トピック

- [デモ: AWS SAM CLI を使用して Lambda 関数コードをアップロードする](#)
- [対応するユースケース](#)
- [詳細はこちら](#)

デモ: AWS SAM CLI を使用して Lambda 関数コードをアップロードする

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。AWS SAM CLI を使用して Lambda 関数コードを Amazon S3 に自動的にアップロードし、その新しいパスをアプリケーションテンプレートで参照します。

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/aws-labs/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-
samclisourcebucket-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

```

デプロイ時、AWS SAM CLI により 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には、Amazon S3 バケットパスが反映されます。

```

AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...

```

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://aws-sam-cli-managed-default-samclisourcebucket-1a4x26zbcdkqr/demo/
da3c598813f1c2151579b73ad788cac8
      Handler: app.lambda_handler
      ...
```

対応するユースケース

AWS SAMCLI 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
AWS::AppSync::FunctionConfiguration	CodeS3Location RequestMappingTemplateS3Location ResponseMappingTemplateS3Location
AWS::AppSync::GraphQLSchema	DefinitionS3Location
AWS::AppSync::Resolver	CodeS3Location RequestMappingTemplateS3Location

リソース	プロパティ
	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
AWS::Serverless::Function	CodeUri ImageUri
AWS::Serverless::GraphQLApi	SchemaUri Function.CodeUri Resolver.CodeUri
AWS::Serverless::HttpApi	DefinitionUri
AWS::Serverless::LayerVersion	ContentUri

リソース	プロパティ
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。

詳細はこちら

AWS::Include 変換の詳細については、「ユーザーガイド」の [AWS::Include 「変換AWS CloudFormation」](#) を参照してください。

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 の [API Gateway HTTP API から SQS へのパターン](#) を参照してください。

を使用して sam sync に同期する方法の概要 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を提供します 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、「[でアプリケーションとリソースをデプロイする AWS SAM](#)」を参照してください。

aws sam sync コマンドはの一部です AWS SAM Accelerate。は、でサーバーレスアプリケーションの開発とテストの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AWS SAM Accelerateを提供します 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](#)
- [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](#)
- [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 コマンドのオプション](#)
- [トラブルシューティング](#)
- [例](#)
- [詳細はこちら](#)

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 aws 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、aws 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、デフォルト設定を使用してアプリケーションを AWS CloudFormation 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構成設定。
 - 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、AWS SAM CLI はその旨を通知し、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 は実行を継続し、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. ローカルの変更を に同期 AWS クラウドする – ローカルの変更を行うと、 はそれらの変更 AWS SAMCLIを検出し、利用可能な最も迅速な方法で に同期 AWS クラウド 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 APIsをサポートしている場合、AWS SAMCLIはそれを使用して変更をデプロイします。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 APIs、AWS SAMCLI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 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、「完全な AWS SAM ワークショップ」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` コマンドの変更可以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
```

AWS SAMCLI 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` 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できます。

- を実行するたびに AWS CloudFormation デプロイを要求する `--no-skip-deploy-sync` に `sam 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と同期され AWS CloudFormation、ドリフトが防止されます。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。AWS SAMCLI 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節約できます AWS クラウド。

変更が検出されない場合でも、AWS SAMCLI は次のシナリオで 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、「[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利用する AWS SAM](#)」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、は一時ディレクト

りに AWS SAM CLIビルドされます。これには、ソースコードとプロジェクトファイルのコピーが含まれます。を使用すると `--build-in-source`、は AWS SAM ソースフォルダに直接CLIビルドされるため、一時ディレクトリにファイルを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` ファイルを変更しても、は AWS SAM CLI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`同期を開始し、ファイルが含まれます。

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

のトラブルシューティングについては、AWS SAMCLI「」を参照してください[AWS SAMCLI トラブルシューティング](#)。

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: Hello World アプリケーションをデプロイする](#)」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_pt15g6_80000gr/T/tmpx_5t4u3f.
```

```
Execute the following command to deploy the packaged template
```

```
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_pt15g6_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-
samclisourcebucket-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-
            BUFVM02PJIYF

```

```

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-2P1N6TPTQoco
-----
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。はこの変更 AWS SAMCLIを検出し、アプリケーションを に同期します AWS クラウド。AWS Lambda は AWS サービス APIs をサポートしているため、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 API をサポートしていないため、AWS SAMCLIは自動的に AWS CloudFormation デプロイを実行します。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_pt15g6_80000gr/T/tmpuabo0jb9.
Execute the following command to deploy the packaged template
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_pt15g6_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-
samclisourcebucket-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 AwsSamAutoDependencyLayerNestedSt	AWS::CloudFormation::Stack -	ack
UPDATE_COMPLETE AwsSamAutoDependencyLayerNestedSt	AWS::CloudFormation::Stack -	ack
UPDATE_IN_PROGRESS ServerlessRestApi	AWS::ApiGateway::RestApi -	
UPDATE_COMPLETE ServerlessRestApi	AWS::ApiGateway::RestApi -	
CREATE_IN_PROGRESS ServerlessRestApiDeployment8cf30e	AWS::ApiGateway::Deployment -	d3cd
UPDATE_IN_PROGRESS HelloWorldFunctionHelloWorldPermi	AWS::Lambda::Permission Requested update requires the	ssionProd
	creation of a new physical	
	resource; hence creating one.	
UPDATE_IN_PROGRESS HelloWorldFunctionHelloWorldPermi	AWS::Lambda::Permission Resource creation Initiated	ssionProd
CREATE_IN_PROGRESS ServerlessRestApiDeployment8cf30e	AWS::ApiGateway::Deployment Resource creation Initiated	d3cd
CREATE_COMPLETE ServerlessRestApiDeployment8cf30e	AWS::ApiGateway::Deployment -	d3cd
UPDATE_IN_PROGRESS ServerlessRestApiProdStage	AWS::ApiGateway::Stage -	
UPDATE_COMPLETE ServerlessRestApiProdStage	AWS::ApiGateway::Stage -	
UPDATE_COMPLETE HelloWorldFunctionHelloWorldPermi	AWS::Lambda::Permission -	ssionProd
UPDATE_COMPLETE_CLEANUP_IN_PROGRE SS	AWS::CloudFormation::Stack -	sam-app
DELETE_IN_PROGRESS HelloWorldFunctionHelloWorldPermi	AWS::Lambda::Permission -	ssionProd
DELETE_IN_PROGRESS ServerlessRestApiDeployment47fc2d	AWS::ApiGateway::Deployment -	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-
BUFVM02PJIYF

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.

詳細はこちら

すべての `aws-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。

でサーバーレスアプリケーションをモニタリングする AWS SAM

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のモニタリングについて詳しく説明します。これには、異常を検出したときに通知 CloudWatch するように Amazon を設定する方法が含まれます。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、末尾付けのヒントなど、ログの操作に関する情報も提供します。

トピック

- [CloudWatch Application Insights 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](#)
- [ログの使用](#)

CloudWatch Application Insights 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

Amazon CloudWatch Application Insights は、アプリケーション内の AWS リソースをモニタリング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がないか確認し、自動ダッシュボードを構築して可視化できます。AWS Serverless Application Model (AWS SAM) アプリケーションで使用するように CloudWatch Application Insights を設定できます。CloudWatch Application Insights の詳細については、[「Amazon ユーザーガイド」の「Amazon CloudWatch Application Insights」](#)を参照してください。 CloudWatch

トピック

- [を使用した CloudWatch Application Insights の設定 AWS SAM](#)
- [次のステップ](#)

を使用した CloudWatch Application Insights の設定 AWS SAM

AWS SAM コマンドラインインターフェイス (AWS SAMCLI) 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

でアプリケーションを初期化するときは `sam init`、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 をアクティブ化します。

AWS SAM CLI インタラクティブフローを通じて CloudWatch Application Insights をアクティブ化するには、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` オプションを使用して CloudWatch Application Insights をアクティブ化するには、次の手順を実行します。

```
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 テンプレートを使用して を設定する

AWS SAM テンプレートで `AWS::ResourceGroups::Group` および `CloudWatch AWS::ApplicationInsights::Application` リソースを定義して Application Insights をアクティブ化します。

```
AWS::TemplateFormatVersion: '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` – 多数の AWS 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、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。

両方のリソースは、アプリケーションのデプロイ AWS CloudFormation 時に自動的に渡されます。AWS SAM テンプレート AWS CloudFormation の構文を使用して、Application Insights CloudWatch をさらに設定できます。詳細については、「Amazon [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。 CloudWatch

`aws sam init --application-insights` コマンドを使用すると、これらのリソースの両方が AWS SAM テンプレートで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/aws-labs/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/aws-labs/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/aws-labs/
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`を使用してアプリケーションをデプロイします。CloudWatch Application Insights がサポートするすべてのリソースは、モニタリング用に設定されます。

- 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 にアクセスする方法については、「[Amazon ユーザーガイド](#)」の「[Access CloudWatch Application Insights](#)」を参照してください。 CloudWatch

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、を使用してデプロイする AWS Lambda 関数だけでなく、すべての関数で機能します AWS SAM。

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 参考資料が含まれています。これには、AWS SAMCLIコマンドのAWS SAMCLIRifalens情報などのリファレンス資料や、設定AWS SAMCLI、バージョン管理、トラブルシューティング情報などの追加情報が含まれます。さらに、このセクションには、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 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 仕様と AWS SAM テンプレート

このAWS SAM 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 仕様の最新バージョンは [入手できます](#) [AWS SAM プロジェクトと AWS SAM テンプレート](#)。AWS SAM 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、APIs、設定、およびアクセス許可を定義するために使用する簡略化された短縮構文が付属しています。

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 仕様を操作しますsam init。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 テンプレートはテンプレートの拡張機能ですAWS CloudFormation。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 SAMCLI) 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でサーバーレスアプリケーションを構築して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。コマンドは、事前設定された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選択した言語で生成します。
- `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。AWS リソースを作成し、AWS SAM テンプレートで定義されているアクセス許可やその他の設定を設定します。

のインストール手順については、AWS SAMCLI「」を参照してください[AWS SAM CLI のインストール](#)。

AWS SAM ポリシーテンプレート

では AWS SAM、ポリシーテンプレートのリストから選択して、アプリケーションが使用するリソースに対する関数のアクセス許可の範囲を AWS Lambda 絞り込むことができます。

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](#)
- [イメージリポジトリ](#)
- [AWS SAM CLI でのテレメトリ](#)
- [AWS SAM テンプレートでのリソースアクセスの設定と管理](#)

AWS SAM CLI コマンドリファレンス

このセクションには、AWS SAMCLI コマンドに関するリファレンス情報が含まれています。これには、使用状況の詳細、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

追加情報が含まれます。該当する場合、追加情報には引数、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。のインストール手順については、AWS SAMCLI「」を参照してください[AWS SAM CLI のインストール](#)。

トピック

- [sam build](#)
- [sam delete](#)
- [sam deploy](#)
- [sam init](#)
- [sam list](#)
- [sam local generate-event](#)
- [sam local invoke](#)
- [sam local start-api](#)
- [sam local start-lambda](#)
- [sam logs](#)
- [sam package](#)
- [sam pipeline bootstrap](#)
- [sam pipeline init](#)
- [sam publish](#)
- [sam remote invoke](#)
- [sam remote test-event](#)
- [sam sync](#)
- [sam traces](#)
- [sam validate](#)

sam build

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 SAMCLI) sam build コマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してください[とは AWS SAMCLI](#)。

- `sam build` コマンドの使用に関するドキュメントについては、AWS SAMCLI「」を参照してください[sam build コマンド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単一のリソースを構築する AWS SAM ように指示します。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の URI です。デフォルトで、AWS SAM は Amazon ECR Public からコンテナイメージをプルします。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スペアを使用できます。文字列を指定する場合は、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の URI になります。例えば、`sam build --use-container --build-image amazon/aws-sam-cli-build-image-python3.8` などです。キーバリュースペアを指定する場合、キーはリソース名で、値はそのリソースに使用するコンテナイメージの URI です。例えば、`sam build --use-container --build-image Function1=amazon/aws-sam-cli-build-image-python3.8` などです。キーバリュースペアを使用すると、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 – [sam init --runtime](#) オプションでサポートされている Node.js ランタイム。
- 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、`samconfig.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 `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 `1.1` から `1.2` に変更された場合 `1.2`、AWS SAM はキャッシュされていないビルドを実行するまで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が含まれる JSON ファイルのパスとファイル名です。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。

`--docker-network TEXT`

デフォルトのブリッジネットワークと共に、Lambda Docker コンテナが接続する必要がある既存の Docker ネットワークの名前または ID を指定します。指定されていない場合、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。

`--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`

AWS SAM CLI 機能の拡張に使用されるフックの名前。

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。

`--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

Terraform設定ファイルまたは関数ソースコード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これらのファイルがルートモジュールを含むディレクトリの外部にある場合は、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。

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 SAMCLI) `sam delete` コマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI「」を参照してくださいとは [AWS SAMCLI](#)。

`sam delete` コマンドは、AWS CloudFormation スタック、Amazon S3 と Amazon ECR 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、および AWS SAM テンプレートファイルを削除して、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をチェックし、デプロイされている場合は、そのスタックと Amazon ECR リポジトリの削除についてユーザーに表示します。--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。

`--help`

このメッセージを表示して終了します。

--no-prompts

を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。

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 SAMCLI) sam deploy コマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。
- sam deploy コマンドの使用に関するドキュメントについては、AWS SAMCLI 「」を参照してください [sam deploy コマンドを使用したデプロイの概要](#)。

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

SAM_CLI_POLL_DELAY 環境変数を 秒の値で設定して、AWS SAM CLI が AWS CloudFormation スタックの状態をチェックする頻度を設定します。これは、からのスロットリングを確認するときに便利です AWS CloudFormation。この env 変数は、の実行中に行われる describe_stack API 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*

が特定のスタックを作成 AWS CloudFormation できるよう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、どちらの機能でも指定できます。カスタム名を持つ IAM リソースがある場合は、CAPABILITY_NAMED_IAM を指定する必要があります。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。

`--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`

AWS SAM CLI が計算された変更セットをデプロイするかどうかの確認を求めます。

`--debug`

デバッグロギングを有効にして、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。

`--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ロールバックされます。`--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して、デプロイ全体で AWS SAM CLI によるプロンプトを用いたガイドを有効にします。

`--help`

このメッセージを表示して終了します。

`--image-repositories` *TEXT*

関数の Amazon ECR リポジトリ URI へのマッピングです。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 リポジトリの名前です。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 バケットに保管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。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 AWS CloudFormation に関連付ける Amazon Simple Notification Service (Amazon SNS) トピック ARNs 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`

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=ParameterValue InstanceType=t1.micro` です。

`--profile` *TEXT*

認証情報を取得する AWS 認証情報ファイルの特定のプロファイル。

`--region` *TEXT*

AWS リージョン デプロイ先の 。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および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 `--resolve-image-repos` を無視します。

Note

が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、AWS SAM CLI は --resolve-s3 を無視します。--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。

--role-arn *TEXT*

変更セットを適用するときが 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 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 です。

--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。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 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-json`

AWS CloudFormation テンプレートの JSON を出力します。デフォルトの出力は YAML です。

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 SAMCLI) `sam init` コマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。
- `sam init` コマンドの使用に関するドキュメントについては、AWS SAMCLI「」を参照してください [sam init コマンドを使用して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、0.30.0 以降の AWS SAM CLI バージョンのみで利用できます。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、「[CloudWatch Application Insights 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](#)」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/dotnetcore3.1-base` | `amazon/go1.x-base` | `amazon/java21-base` | `amazon/java17-base` | `amazon/java11-base` | `amazon/java8.al2-base` | `amazon/java8-base` | `amazon/nodejs20.x-base` | `amazon/nodejs18.x-base` | `amazon/nodejs16.x-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。

--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) です。

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 | dotnetcore3.1 | go1.x | java21 | java17 | java11 | java8 | java8.al2 | nodejs20.x | nodejs18.x | nodejs16.x | 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を設定 AWS SAM 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。

sam 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 SAMCLI) `sam list` コマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

`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 AWS CloudFormation 時に作成された AWS Serverless Application Model (AWS SAM) テンプレートに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 SAMCLI) `sam list endpoints` サブコマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。

`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` 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。

`--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

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 `test-stack`。

```
$ sam list endpoints --stack-name test-stack --output json

[
  {
    "LogicalResourceId": "HelloWorldFunction",
    "PhysicalResourceId": "sam-app-test-list-HelloWorldFunction-H85Y7yIV7ZLq",
    "CloudEndpoint": "https://zt55oi7kbljxjmcoahsj3cknwu0rposq.lambda-url.us-east-1.on.aws/",
    "Methods": "-"
  },
  {
    "LogicalResourceId": "ServerlessRestApi",
    "PhysicalResourceId": "uj80uoe2o2",
    "CloudEndpoint": [
```

```
    "https://uj80uoe2o2.execute-api.us-east-1.amazonaws.com/Prod",
    "https://uj80uoe2o2.execute-api.us-east-1.amazonaws.com/Stage"
  ],
  "Methods": [
    "/hello['get']"
  ]
}
]
```

sam list resources

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 SAMCLI) `sam list resources` サブコマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

`sam list resources` サブコマンドは、デプロイ時に変換 AWS CloudFormation によって作成された AWS Serverless Application Model (AWS SAM) テンプレート内の AWS SAM リソースを表示します。

デプロイ前に AWS SAM テンプレート `sam list resources` を使用して、作成されるリソースを確認します。AWS CloudFormation スタック名を指定して、デプロイされたリソースを含む統合リストを表示します。

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` 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。

`--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

テンプレートからのローカルリソース AWS SAM と、 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テーブル形式で表示します `test-stack`。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 SAMCLI) `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。

の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

`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 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。

--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

という名前の AWS CloudFormation スタック内のリソースの出力をテーブル形式で表示します `test-stack`。

```
$ sam list stack-outputs --stack-name test-stack --output table
```

OutputKey Description	OutputValue	
HelloWorldFunctionIamRole Implicit IAM Role created for Hello function	arn:aws:iam:: <i>account-number</i> :role/sam- app-test-list-HelloWorldFunctionRole-	World
HelloWorldApi Gateway endpoint URL for Prod for Hello World function	SRJDMJ6F7F41 https://uj80uoe2o2.execute-api.us- east-1.amazonaws.com/Prod/hello/	API stage
HelloWorldFunction World Lambda Function ARN	arn:aws:lambda:us- east-1: <i>account-number</i> :function:sam-app- test-list- HelloWorldFunction-H85Y7yIV7ZLq	Hello

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 SAMCLI) `sam local generate-event` サブコマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してくださいとは [AWS SAMCLI](#)。
- `sam local generate-event` コマンドの使用に関するドキュメントについては、AWS SAMCLI「」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
```

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 SAMCLI) `sam local invoke` サブコマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。
- `sam local invoke` サブコマンドの使用に関するドキュメントについては、AWS SAMCLI「」を参照してください [を使用したテストの概要 sam local invoke](#)。

`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 1 つの Lambda 関数が含まれている場合、AWS SAM CLI はその関数を呼び出します。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にホスト名を IP アドレスマッピングに渡します。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。macOS 上の Docker コンテナで AWS SAM CLI を実行する場合は、host.docker.internal を指定できます。コンテナをとは異なるホストで実行する場合は AWS SAMCLI、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。

--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が含まれる JSON ファイルです。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が含まれる JSON ファイルです。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。stdin からの JSON を入力するには、値「-」を渡す必要があります。さまざまな AWS サービスからのイベントメッセージ形式の詳細については、「[デベロッパーガイド](#)」の「[他のサービスの使用](#)」を参照してください。

AWS Lambda

--force-image-build

レイヤーがある Lambda 関数を呼び出すために使用されるイメージを AWS SAM CLI が再構築すべきかどうかを指定します。

--help

このメッセージを表示して終了します。

--hook-name TEXT

AWS SAM CLI 機能の拡張に使用されるフックの名前。

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の URI。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 [イメージリポジトリ](#)。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` です。

--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

(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確認し、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。

このオプションを指定して、Lambda ランタイム環境の最新の 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。

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 ロードします。

`--terraform-plan-file`

Terraform Cloud で AWS SAMCLI を使用する場合はローカル Terraform プランファイルへの相対パスまたは絶対パス。このオプションは `--hook-name` を terraform に設定する必要があります。

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 SAMCLI) `sam local start-api` サブコマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してくださいとは [AWS SAMCLI](#)。
- `sam local start-api` サブコマンドの使用に関するドキュメントについては、AWS SAMCLI「」を参照してください [を使用したテストの概要 sam local start-api](#)。

`sam local start-api` サブコマンドは関数を AWS Lambda 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。

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にホスト名を IP アドレスマッピングに渡します。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。macOS 上の Docker コンテナで AWS SAM CLI を実行する場合は、host.docker.internal を指定できます。コンテナをとは異なるホストで実行する場合は AWS SAMCLI、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。

--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、リモートマシンと一致するようにこの値を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が含まれる JSON ファイルです。

`--force-image-build`

が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するかどうか AWS SAM CLI を指定します。

`--help`

このメッセージを表示して終了します。

`--hook-name TEXT`

AWS SAM CLI 機能の拡張に使用されるフックの名前。

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の URI。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。文字列を指定する場合は、アプリケーション内のすべての関数に使用するコンテナイメージの URI になります。例えば、`sam local start-api --invoke-image public.ecr.aws/sam/emu-python3.8` などです。キーバリューペアを指定する場合、キーはリソース名で、値はそのリソースに使用するコンテナイメージの URI です。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` などです。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`

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。と同じ形式を使用します AWS CLI。例えば、'`ParameterKey=, ==KeyValuePairName , ParameterValueMyKey ParameterKey=InstanceTypeParameterValuet1.micro`' です。

`--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`

Lambda ランタイムの最新 Docker イメージのプルダウンを CLI がスキップすべき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: なし)。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 (CSS/JavaScript/HTML など) ファイルは、`--static-dir` 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、`--template` 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 ロードします。

`--terraform-plan-file`

Terraform Cloud で AWS SAMCLI を使用する場合はローカル Terraform プランファイルへの相対パスまたは絶対パス。このオプションは `--hook-name` を terraform に設定する必要があります。

`--warm-containers` [*EAGER* | *LAZY*]

オプション。AWS SAM CLI が各関数のコンテナを管理する方法を指定します。

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。

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 SAMCLI) `sam local start-lambda` サブコマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してくださいとは [AWS SAMCLI](#)。
- `sam local start-lambda` サブコマンドの使用に関するドキュメントについては、AWS SAMCLI「」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にホスト名を IP アドレスマッピングに渡します。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。macOS 上の Docker コンテナで AWS SAM CLI を実行する場合は、host.docker.internal を指定できます。コンテナをとは異なるホストで実行する場合は AWS SAMCLI、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。

--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が含まれる JSON ファイルです。

`--force-image-build`

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 CLI が再構築すべきかどうかを指定します。

`--help`

このメッセージを表示して終了します。

`--hook-name TEXT`

AWS SAM CLI 機能の拡張に使用されるフックの名前。

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の URI。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` です。

`--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

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。と同じ形式を使用します AWS CLI。例えば、'ParameterKey=, ==KeyValuePairName , ParameterValueMyKey ParameterKey=InstanceTypeParameterValue1.micro' です。このオプションは `--hook-name` と互換性がありません。

--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*

認証情報を取得する AWS 認証情報ファイル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

Lambda ランタイムの最新 Docker イメージのプルダウンを CLI がスキップすべきかどうかを指定します。

--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 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

Terraform Cloud で AWS SAMCLI を使用する場合はローカル Terraform プランファイルへの相対パスまたは絶対パス。このオプションは --hook-name を terraform に設定する必要があります。

--warm-containers *[EAGER | LAZY]*

オプション。AWS SAM CLI が各関数のコンテナを管理する方法を指定します。

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。

sam logs

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 SAMCLI) sam logs コマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。

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 Logs CloudWatch ロググループのログが含まれます。このオプションを とともに指定すると name、 は名前付き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 AWS SAM を含めます。

--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、単純なキーワード（「エラー」など）でも、Amazon CloudWatch Logs でサポートされているパターンでもかまいません。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 template 内の関数リソースの論理 ID になります。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、名前の前にネストされたスタック名を付加して、そのリソースからログをプルできます (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、 は指定したスタック内のすべてのリソースのログ AWS SAM 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*

認証情報を取得する AWS 認証情報ファイル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。

`--stack-name` *TEXT*

リソースが属する AWS CloudFormation スタックの名前。

`--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。は、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。

このコマンドを使用する場合、AWS SAM CLI はデフォルトで、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、.aws-sam サブフォルダにあり、という名前の [sam build](#) コマンドを使用して構築されたテンプレートファイルを見つけようとし、template.yaml。AWS SAM CLI は次に、現在の作業ディレクトリで、template.yaml または template.yml と命名されているテンプレートファイルを見つけようとし、--template オプションを指定すると、AWS SAM CLI デフォルトの動作は上書きされ、はその 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 1 つの Lambda 関数が含まれている場合、AWS SAM CLI はその関数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。

`--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 URI です。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 バケットに保管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。Amazon S3 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。

--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。アーティファクトが 51,200 バイトを超える場合は、--s3-bucket または --resolve-s3 オプションが必要です。--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` です。

`--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 パッケージ化します。

`--use-json`

AWS CloudFormation テンプレートの JSON を出力します。デフォルトでは、YAML が使用されます。

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 SAMCLI) `sam local pipeline bootstrap` サブコマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。

`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、以下の 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の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、IAM 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ために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。

- OIDC を通じてサポートされている CI/CD プラットフォーム。AWS SAM パイプラインで OIDC を使用する方法の概要については、「[AWS SAM パイプラインで OIDC 認証を使用する方法](#)」を参照してください。
- 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、コンテナイメージの Lambda デプロイパッケージを保持するための Amazon ECR イメージリポジトリ (パッケージタイプが Image のリソースがある場合)。

使用方法

```
$ sam pipeline bootstrap <options>
```

オプション

`--bitbucket-repo-uuid` *TEXT*

Bitbucket リポジトリの UUID です。このオプションは、アクセス許可に Bitbucket OIDC を使用する場合に固有です。

Note

この値は `https://bitbucket.org/workspace/repository/admin/addon/admin/pipelines/openid-connect` にあります。

`--bucket` *TEXT*

AWS SAM アーティファクトを保持する Amazon S3 バケットの ARN。

`--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のデプロイ AWS CloudFormation 中に が引き受ける IAM ロールの ARN。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

Amazon ECR イメージリポジトリが提供されていない場合に、それを作成するかどうかを指定します。Amazon ECR リポジトリは、パッケージタイプが Image の Lambda 関数またはレイヤーのコンテナイメージを保持します。デフォルトは `--no-create-image-repository` です。

--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し、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 Actions OIDC を使用することに固有です。

--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 Actions OIDC を使用すること GitHub に固有です。

--github-repo *TEXT*

デプロイ元の 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 Actions OIDC を使用することに固有です。

--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 GitLab OIDC を使用することに固有です。

`--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 GitLab OIDC を使用することに固有です。

`--help`, `-h`


このメッセージを表示して終了します。

`--image-repository` *TEXT*

パッケージタイプが Image の Lambda 関数またはレイヤー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の ARN です。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 `--use-oidc-provider` と一緒に指定される場合は、OIDC プロバイダーの必須パラメータをすべて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設定されたクライアント ID です。

`--oidc-provider` [*github-actions* | *gitlab* | *bitbucket-pipelines*]

OIDC アクセス許可に使用される 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 プロバイダーの URL です。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*

このステージでの動作のためにパイプラインユーザーが引き受ける IAM ロールの ARN です。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*

アクセスキー ID とシークレットアクセスキーが CI/CD システムと共有されている IAM ユーザーの Amazon リソースネーム (ARN) です。これは、この IAM ユーザーに、対応する AWS アカウントへのアクセス許可を付与するために使用されます。提供されていない場合、コマンドによってアクセスキー ID とシークレットアクセスキーの認証情報と共に IAM ユーザーロールが作成されます。

`--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 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。

`--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せん。

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 SAMCLI) `sam local pipeline init` サブコマンドのリファレンス情報を提供します。

の概要については、AWS SAM CLI「」を参照してくださいとは [AWS SAM CLI](#)。

`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` コマンドを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`, `-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。

sam publish

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 SAMCLI) `sam publish` コマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

`sam publish` コマンドは、AWS SAM アプリケーションを に発行します AWS Serverless Application Repository。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、AWS SAM テンプレートに公開に必要なアプリケーションメタデータを含む Metadata セクションが含まれていることを想定しています。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 セクションの詳細については、「」を参照してください [を使用したアプリケーションの公開 AWS SAMCLI](#)。

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 AWS Serverless Application Repository する前に、それらを に個別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*

認証情報を取得する AWS 認証情報ファイル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。

`--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ン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のパス[default: template.[yaml|yml]]。

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 SAMCLI) `sam remote invoke` コマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。
- `sam remote invoke` コマンドの使用に関するドキュメントについては、AWS SAMCLI「」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) – リソースの ARN。

Tip

`sam list stack-outputs --stack-name <stack-name>` を使用してリソースの ARN 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 ID または物理 ID の指定で、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場合、または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 はリソース ID を AWS CloudFormation スタックの論理 ID 値として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すると、AWS SAM CLI は次のロジックを使用して 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. 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進みます。
 - 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のリソースが 1 つある場合、AWS SAM CLI はそのリソースを呼び出します。リソースタイプのリソースがスタックに複数存在する場合、AWS SAM CLI はエラーを返します。

以下は、 の動作 AWS SAM CLI の例です。

- 2 つの Lambda 関数と Amazon SQS キューを含むスタック – AWS SAM CLI スタックには複数の Lambda 関数が含まれているため、 は Lambda リソースタイプを検索し、 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 スタックには 1 つの Lambda リソースが含まれているため、 は 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スタックに 1つの Amazon SQS キューが含まれているため、は 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*

AWS SAM CLI 設定ファイルから、使用する環境を指定します。

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。そうすることで、AWS SAM CLI が生成したデバッグメッセージとタイムスタンプを出力します。

`--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のレスポンスが JSON 構造で返されます。レスポンスには完全な 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

呼び出されているリソースに渡すことができる追加の [Boto3](#) パラメータです。

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)

次の追加パラメータは Amazon SQS キューにメッセージを送信するために使用できます。

- 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*

認証情報を取得するための AWS 認証情報ファイルからの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 です。

--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。

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ーフェイス (AWS SAMCLI) `sam remote test-event` コマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。
- `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、AWS SAMCLI「」を参照してください [を使用したクラウドテストの概要 sam remote test-event](#)。

`sam remote test-event` コマンドは、Amazon EventBridge スキーマレジストリの共有可能なテストイベントとやり取りします。

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

ローカルファイルから EventBridge スキーマレジストリにイベントを保存します。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 SAMCLI) `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。

- の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。
- `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、AWS SAMCLI「」を参照してください [を使用したクラウドテストの概要 sam remote test-event](#)。

`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ている 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 SAMCLI) `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。

- の概要については、AWS SAMCLI 「」を参照してくださいとは [AWS SAMCLI](#)。
- `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、AWS SAMCLI 「」を参照してください [を使用したクラウドテストの概要 sam remote test-event](#)。

`sam remote test-event get` 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ている 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 SAMCLI) `sam remote test-event list` サブコマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してくださいとは [AWS SAMCLI](#)。
- `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、AWS SAMCLI「」を参照してください [を使用したクラウドテストの概要 sam remote test-event](#)。

`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ている 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 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 SAMCLI) sam remote test-event put サブコマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI 「」を参照してください [とは AWS SAMCLI](#)。
- 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、AWS SAMCLI 「」を参照してください [を使用したクラウドテストの概要 sam remote test-event](#)。

sam remote test-event put サブコマンドは、ローカルマシンから Amazon EventBridge スキーマレジストリに共有可能なテストイベントを保存します。

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。

同じ名前の共有可能なテストイベントが EventBridge スキーマレジストリに存在する場合、AWS SAM CLI はそれを上書きしません。上書きするには、`--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ている 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 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 SAMCLI) `sam sync` コマンドのリファレンス情報を提供します。

- の概要については、AWS SAMCLI「」を参照してくださいとは [AWS SAMCLI](#)。
- の使用に関するドキュメントについては、AWS SAMCLI「」を参照してください [AWS SAMCLI](#)。

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基づいて解決されます。

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](#) の URI です。デフォルトでは、は Amazon [Elastic Container Registry \(Amazon ECR\) Public](#) の [コンテナイメージリポジトリ](#) URI `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に使用するコンテナイメージ URI 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、そのリソースで使用するコンテナイメージ URI 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ごとに異なるコンテナイメージ URI 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 – [sam init --runtime](#) オプションでサポートされている Node.js ランタイム。
- 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*

が特定のスタックを作成 AWS CloudFormation 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。例えば、新しい 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、はを通じてデプロイするのではなく、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) リポジトリの名前です。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 URI へのマッピングです。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の暗号化に使用される 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*

スタック AWS CloudFormation に関連付ける Amazon Simple Notification Service (Amazon SNS) トピック ARNs のリスト。

`--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 (AWS CLI) と同じ形式を使用します。例えば `ParameterKey=ParameterValue InstanceType=t1.micro` です。

`--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` です。

`--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` です。

`--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 SAMCLI 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してい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 AWS Lambda のような Docker コンテナ内に関数を構築します。

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` を `--dependency-layer` と共に使用すると AWS SAMCLI にメッセージが表示され、`--no-dependency-layer` で処理が続行されます。

`--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、更新時にアプリケーション内のすべてのリソースが AWS SAM 同期されます。このオプションでは、は初期 AWS CloudFormation デプロイ AWS SAM を実行します。次に、AWS はサービス APIs AWS SAM を使用してコードリソースを更新します。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しても、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。

sam traces

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 SAMCLI) sam traces コマンドのリファレンス情報を提供します。

の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

sam traces コマンドは、の AWS アカウント で AWS X-Ray トレースを取得します AWS リージョン。

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設定 AWS SAM 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 SAMCLI) `sam validate` コマンドのリファレンス情報を提供します。

の概要については、AWS SAM CLI 「」を参照してくださいとは [AWS SAM CLI](#)。

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検証します。

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。

`--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、リポジトリの「[cfn-lint](#)AWS CloudFormation GitHub」を参照してください。

`--profile` *TEXT*

認証情報を取得する AWS 認証情報ファイル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。

--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。

AWS SAMCLI 管理

このセクションでは、AWS SAM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。これには、プロジェクトレベルの設定ファイルを使用してAWS SAMCLIコマンドパラメータ値を設定する方法に関する情報が含まれます。また、のさまざまなバージョンの管理CLI、がユーザーに代わってAWS SAM AWS サービスを呼び出すためのAWS SAM 出するためのAWS 認証情報の設定、および 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。AWS SAM。このセクションでは、一般的なAWS SAM トラブルシューティングのセクションで終わります。

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 SAMCLI) は、AWS SAMCLIコマンドパラメータ値の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ています。

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](#)

- [サポートされている設定ファイル形式](#)
- [設定ファイルの指定](#)
- [設定ファイルの基本](#)
- [パラメータ値のルール](#)
- [設定の優先順位](#)
- [設定ファイルの作成と変更](#)

デフォルトの設定ファイルの設定

AWS SAM は、次のデフォルト設定ファイル設定を使用します。

- [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 のドキュメントの「[TOML](#)」を参照してください。

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と、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 は次にこのファイルを検索します。
3. `samconfig.[yaml|yml]` ファイル — `samconfig.toml` がプロジェクトのルートに存在しない場合、AWS SAM CLI はこのファイルを探します。

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
```

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。

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

Command

このコマンドは、パラメータ値を指定する AWS SAM CLI コマンドです。

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。

AWS SAM CLI コマンドを参照するときは、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 AWS SAM CLI コマンドのオプション名です。
- 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` です。
- 文字列値の場合、引用符 (") を使用します。例えば `region = "us-west-2"` です。
- 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\""` です。
- 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` です。
- 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` です。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 のインタラクティブフローでは、設定値が括弧 ([]) 内に表示されます。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際に、AWS SAM CLI はグローバル値を次のように処理します。

- 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 はその値を特定のコマンドセクションに書き込みません。
- パラメータ値が `global` セクション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 はグローバル値を優先してその特定のエントリを削除します。

AWS SAM CLI バージョンの管理

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) のバージョンを管理するには、アップグレード、ダウングレード、アンインストールを行います。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

は、インストールに使用したのと同じ方法でアップグレード AWS SAM CLI する必要があります。パッケージインストーラーを使用して AWS SAM をインストールおよびアップグレードすることをお勧めします CLI。

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

をアップグレードするには CLI、Windows AWS SAM のインストール手順をもう一度繰り返し [AWS SAM CLI のインストール](#) 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 バイナリが配置されているパスが表示されます。例：

```
/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
```

- シンボリックリンクを削除します。

```
sudo rm /usr/local/bin/sam
```

- 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。パッケージインストーラーを使用して AWS SAM をインストールすることをお勧めします CLI。

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

- 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
```

- 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。
 - 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。
- 以下を変更して実行 \$PATH して、 を削除します AWS SAMCLI。

```
$ sudo rm -rf /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-symlink-directory** – \$PATH 環境変数。デフォルトの場所は `/usr/local/bin` です。
- 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. をアンインストールすることを確認します AWS SAMCLI。

Homebrew を使用する AWS SAM CLI の管理から移行する

Homebrew を使用して をインストールおよびアップグレードする場合は CLI、サポートされている AWS SAM メソッドを使用する AWS ことをお勧めします。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. リポジトリ [sam-cli-nightly](#) の `aws-sam-cli` GitHub 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. プラットフォームのパッケージインストーラーをリポジトリ [sam-cli-nightly](#) の `aws-sam-cli` GitHub から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

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 AWS SAM CLI。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 日現在、AWS SAM CLI は のサポートを中止しています Python 3.7。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はAWS SAMCLI () のAWS マネージドHomebrewインストーラーを維持しなくなります`aws/tap/aws-sam-cli`。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

これらの手順では、コミュニティマネージド AWS SAMCLI Homebrew インストーラを使用します。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
```

が正常にインストールされると AWS SAMCLI、次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 マネージド Homebrew インストーラ (aws/tap/aws-sam-cli) を使用していて、を引き続き使用する場合は Homebrew、コミュニティマネージド Homebrew インストーラ () に切り替えることをお勧めします aws-sam-cli。

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. の AWS マネージド Homebrew バージョンをアンインストールします AWS SAM CLI。

```
$ 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 マネージド AWS SAM CLI タップを削除します。

```
$ 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 マネージドインストーラーのキャッシュされた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. コミュニティ管理バージョンの AWS SAM をインストールします CLI。

```
$ 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

のインストール時または使用時にエラーが発生した場合はCLI、AWS SAM「」を参照してください[AWS SAM CLI トラブルシューティング](#)。

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、ユーザーに代わって AWS 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 を呼び出します AWS CloudFormation。

SDKs や の 1 つなどの AWS ツールを操作するための AWS 認証情報を AWS 既に設定している可能性があります AWS CLI。このトピックでは、AWS 認証情報を設定するための推奨アプローチを示します。

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 がインストールされているかどうかを確認します。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を使用する場合

AWS CLI がインストールされている場合は、`aws configure` コマンドを使用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か、環境変数を設定できます。

- 認証情報ファイル – ローカルシステムの認証情報ファイルで AWS 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

では AWS、お客様とのやりとりから学んだことに基づいてサービスを開発し、起動します。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、環境変数を一時的に設定できます。

コマンドプロンプトを使用する場合：

```
set SAM_CLI_TELEMETRY 0
```

を使用している場合 PowerShell：

```
$env:SAM_CLI_TELEMETRY=0
```

コマンドプロンプトまたは のいずれかで環境変数を永続的に設定するには PowerShell、次のコマンドを使用します。

```
setx SAM_CLI_TELEMETRY 0
```

Note

ターミナルが閉鎖され、再び開くまで、変更は有効ではありません。

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。
- システムおよび環境情報 - Python バージョン、オペレーティングシステム (Windows、Linux、macOS) AWS CodeBuild、 が AWS SAMCLI実行される環境 (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

が AWS SAMCLI収集す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ています。詳細については、次を参照してください。

- [AWS サービス条件](#)

- [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

AWS Serverless Application Model コマンドラインインターフェイス () の使用、インストール、管理時にエラーメッセージをトラブルシューティングしますAWS SAMCLI。

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

に関連するトラブルシューティングガイダンスについては、AWS SAMCLI 「」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](#)。

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を確認し AWS CLI、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
```

つまり、`g` AWS SAMCLI AWS サービスコールを実行できるように AWS 認証情報を設定していないということです。これを修正するには、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 認証情報のセットアップ](#)」を参照してください。

Windows `FileNotFoundException` でのエラー

Windows AWS SAMCLI でコマンドを実行すると、次のエラーが表示されることがあります。

```
Error: FileNotFoundException
```

考えられる原因: は、Windows の最大パス制限を超えるファイルパスとやり取り AWS SAMCLI する可能性があります。

解決策: この問題を解決するには、新しいロングパス動作を有効にする必要があります。これを行うには、Microsoft [Windows アプリケーション開発ドキュメント](#) 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。
- を使用すると AWS SAMCLI、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。

`sam remote invoke` コマンドは AWS SAM CLI バージョン AWS SAM CLI 1.88.0 でリリースされました。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となります。

この問題を解決するには、以下のオプションがあります。

- 認証を使用してアプリケーションを設定する。認証の設定については、「[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 のHomebrewインストーラーは維持されなくなります AWS SAM 。

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: Homebrew サポートを維持し AWS なくなった。

2023 年 9 月以降、AWS は AWS SAMのHomebrewインストーラーを維持しなくなりますCLI。

解決策: AWS サポートされているインストール方法を使用します。

- AWS サポートされているインストール方法については、「」を参照してください[AWS SAM CLI のインストール](#)。

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の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、ソースリソースタイプと宛先リソースタイプごとに定義する必要があるプロパティを参照します。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。

新しい接続をリクエストするには、`serverless-application-model` AWS GitHub [リポジトリ](#) で [新しい問題を送信します](#)。

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
<code>AWS::ApiGateway::RestApi</code>	<code>AWS::Lambda::Function</code>	Write	Id または Qualifier、ResourceId、および Type	Id または Arn および Type
<code>AWS::ApiGateway::RestApi</code>	<code>AWS::Serverless::Function</code>	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::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
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

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::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
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::Event Bus	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

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::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
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

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::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
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

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::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
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

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::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
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

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::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
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

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::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
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

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::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
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

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::Step Functions::StateMachine	AWS::Serverless::Function	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Step Functions::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::Step Functions::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::Step Functions::StateMachine	AWS::SNS::Topic	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Step Functions::StateMachine	AWS::SQS::Queue	Write	Id または RoleName および Type	Id または Arn および Type
AWS::Step Functions::StateMachine	AWS::Step Functions::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、コネクタを使用する AWS SAM とき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::::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}"
}
```

AWS SAM CLI で Docker を使用するためのインストール方法

Docker はマシン上でコンテナを実行するアプリケーションです。AWS SAM を使用すると Docker、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 が必要になるのは、アプリケーションをローカルでテストする場合と、`--use-container` オプションを使用してデプロイパッケージをビルドする場合だけです。

トピック

- [Docker のインストール](#)
- [次のステップ](#)

Docker のインストール

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

Linux

Docker は、CentOS、Debian、および Ubuntu など、最新の Linux ディストリビューションを含めたさまざまなオペレーティングシステムで使用できます。特定のオペレーティングシステムへの Docker のインストールについては、Docker Docs ウェブサイトで「[Get Docker](#)」を参照してください。

Amazon Linux 2 または Amazon Linux 2023 に Docker をインストールするには

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 Docker Community Edition パッケージをインストールします。

- 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. Docker サービスを開始します。

```
$ sudo service docker start
```

4. ec2-user を docker グループに追加して、sudo を使用せずに Docker コマンドを実行できるようにします。

```
$ 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。これを実行するには、現在の SSH ターミナルウィンドウを閉じ、新しいウィンドウで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、ホストマシンとは異なる命令セットアーキテクチャで Lambda 関数をビルドして実行するには、Docker を設定するための追加の手順が必要です。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` を設定します。

Docker のインストールで問題が発生した場合は、「[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。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 Desktop がサポートされていますが、AWS SAM CLI バージョン 1.47.0 以降では、Docker ランタイムを使用する代替手段を使用できます。

1. Docker をインストールする

AWS SAM CLI では、macOS Sierra 10.12 以降で動作する Docker がサポートされています。Docker のインストール方法については、Docker Docs ウェブサイトの「[Install Docker Desktop for Mac](#)」を参照してください。

2. 共有ドライブの構成

AWS SAMCLI 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一覧表示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、Docker ドキュメントの「[ボリュームのマウントにはファイル共有が必要です](#)」のトラブルシューティングのトピックを参照してください。

3. インストールの検証

Docker がインストールされたら、動作していることを検証します。また、コマンドラインから Docker コマンドを実行できることを確認します (docker ps など)。コンテナをインストール、フェッチ、またはプルする必要はありません。これらは、AWS SAM CLI が必要に応じて自動的に行います。

Docker のインストールで問題が発生した場合は、トラブルシューティングのヒントを紹介している Docker Docs ウェブサイトの「[Troubleshoot and diagnose](#)」セクションを参照してください。

Windows

Note

AWS SAM は Docker Desktop を正式にサポートしています。ただし、AWS SAM CLI バージョン 1.47.0 以降では、Docker ランタイムを使用する代替手段を使用できます。

1. Docker をインストールします。

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 の旧バージョンでは、Docker Toolbox を使用できます。正しい手順で Docker のインストールを行うため、Windows のバージョンを選択してください。

- Windows 10 用 Docker をインストールするには、Docker Docs ウェブサイトの「[Install Docker Desktop for Windows](#)」を参照してください。
- 以前のバージョンの Windows Docker をインストールするには、[DockerToolbox](#) GitHub リポジトリの Docker 「ツールボックス」を参照してください。

2. 共有ドライブを設定します。

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一覧表示されている必要があります。Docker を正しく機能させるには、ドライブを共有する必要がある場合があります。

3. インストールを確認します。

Docker がインストールされたら、動作していることを検証します。また、コマンドラインから Docker コマンドを実行できることを確認します (docker ps など)。コンテナをインストール、フェッチ、またはプルする必要はありません。これらは、AWS SAM CLI が必要に応じて自動的に行います。

Docker のインストールで問題が発生した場合は、トラブルシューティングのヒントを紹介している Docker Docs ウェブサイトの「[Troubleshoot and diagnose](#)」セクションを参照してください。

次のステップ

のインストール方法については、AWS SAM CLI 「」を参照してください。[AWS SAM CLI のインストール](#)。

イメージリポジトリ

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てデプロイする](#)」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、そのイメージ AWS SAMCLIに含まれる のバージョンがタグ付けされます。タグがない URI を指定すると、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、`public.ecr.aws/sam/build-nodejs20.x:1.24.1` は AWS SAM CLI バージョン 1.24.1 が含まれるイメージを使用します。

のバージョン 1.33.0 AWS SAM以降ではCLI、 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 AWS SAM以前はCLI、 はコンテナイメージを AWS SAMCLIプルしたデフォルトのリポジトリ DockerHub でした。バージョン 1.22.0 からは、デフォルトリポジトリが 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 URI

次の表に、でサーバーレスアプリケーションを構築およびパッケージ化するために使用できる [Amazon ECR Public](#) ビルドコンテナイメージの URIs を示します AWS SAM。

Note

Amazon ECR Public は、AWS SAM CLI バージョン1.22.0 以降で DockerHub を置き換えました。以前のバージョンの AWS SAMを使用している場合は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 Public
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 Public
カスタムランタイム	public.ecr.aws/sam/build-provided
Go 1.x	public.ecr.aws/sam/build-go1.x
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 (AL2)	public.ecr.aws/sam/build-java8.al2
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 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 3.12	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。

DockerHub からプルしたコンテナイメージを使用して Node.js 20 アプリケーションを構築します:

```
$ sam build --use-container --build-image public.ecr.aws/sam/build-nodejs20.x
```

DockerHub からプルした Python 3.12 コンテナイメージを使用して関数リソースを構築します:

```
$ sam build --use-container --build-image Function1=public.ecr.aws/sam/build-python3.12
```

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、段階的な AWS Lambda デプロイを提供するために [CodeDeploy](#) が組み込まれています。わずか数行の設定で、 は次の AWS SAM 操作を行います。

- 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ると、デプロイを自動的にロールバックします。

Note

AWS SAM テンプレートを使用し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、 から直接表示できます AWS Management Console。

例

次の例は、を使用して、顧客 CodeDeploy を新しくデプロイされたバージョンの Lambda 関数に徐々に移行する方法を示しています。

```
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.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リビジョンは、次の操作を行います。

- **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、次のようになります AWS SAM。
- 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 関数のバージョンングとエイリアスになじみがない場合は、[AWS Lambda 関数のバージョンングとエイリアス](#)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、はデプロイを CodeDeploy 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。もう 1 つの例は、指定

した [AWS Lambda](#) または 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: トラフィックシフトを開始する前に、 はトラフィック前フックの Lambda 関数を CodeDeploy 呼び出します。この Lambda 関数は を呼び出し、成功または失敗 CodeDeploy を示す必要があります。関数が失敗した場合、関数は中止し、失敗を に報告します AWS CloudFormation。関数が成功 CodeDeploy すると、 はトラフィックシフトに進みます。
- PostTraffic: トラフィックシフトが完了すると、 CodeDeploy はトラフィック後フックの Lambda 関数を呼び出します。これは、トラフィック前フックに似ています。このフックでは、関数が を呼び出し CodeDeploy で成功または失敗を報告する必要があります。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 関数を徐々にデプロイする場合、 はトラフィックを移行するために以前にデプロイされた関数バージョン CodeDeploy 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。

最初の段階的なデプロイを 2 つのステップで実行すると CodeDeploy、以前の Lambda 関数バージョンからトラフィックを移行できます。

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: nodejs12.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: nodejs12.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。

重要な注意事項

このセクションには、AWS Serverless Application Model () に関する重要な注意事項とお知らせが含まれていますAWS SAM。

トピック

- [2023 年の重要な注意事項](#)
- [2020 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 end-of-life ステータスを取得しました。は、2023 AWS SAM CLI 年 10 月 24 Python 3.7 日に 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 [お知らせ](#) 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。

- を使用して から Python 3.7 AWS SAM CLI インストールする場合 pip。
- 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。

別の方法で をインストールおよび管理 AWS SAM CLI しても、影響を受けません。

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。

2020 年の重要な注意事項

2020 年 6 月

32 ビットの Windows で AWS SAM CLI をインストールする


32 ビット Windows での AWS SAM CLI の使用に対するサポートは、間もなく廃止される予定です。32 ビットシステムで動作させている場合は、64 ビットシステムにアップグレードして、「[AWS SAM CLI のインストール](#)」の手順を実行することが推奨されます。

64 ビットシステムにアップグレードできない場合は、32 ビットシステムで AWS SAM CLI に [レガシー Docker Toolbox](#) を使用することが可能ですが、ただし、これにより、AWS SAM で特定の制限が発生する可能性があります CLI。例えば、32 ビットシステムでは 64 ビットの Docker コンテナを

実行できません。このため、Lambda 関数が 64 ビットのネイティブにコンパイルされたコンテナに依存している場合、32 ビットシステムでローカルにテストすることができません。

32 ビットシステムに AWS SAM CLI をインストールするには、以下のコマンドを実行します。

```
pip install aws-sam-cli
```

 Important

この `pip install aws-sam-cli` コマンドは 64 ビット Windows でも動作しますが、64 ビットシステムへの AWS SAM CLI のインストールには [64 ビットの MSI](#) の使用が推奨されます。

サンプルサーバーレスアプリケーション

以下の例では、追加のサーバーレスアプリケーションをダウンロード、テスト、およびデプロイする方法が説明されています。これには、イベントソースと AWS リソースの設定方法が含まれます。

トピック

- [DynamoDB イベントを処理する](#)
- [Amazon S3 イベントを処理する](#)

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 `gh:` では、GitHub URL に展開されることに注意してください `https://github.com/`。

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` — DynamoDB アプリケーションの読み取りに必要な 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などの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 SAMCLI を使用してデプロイパッケージを作成します。このパッケージは、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ので、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には

- `aws-sam-cli` の `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
```

コマンドでは、`--capabilities` パラメータは AWS CloudFormation が IAM ロールを作成できるようにします。

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には

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. テーブルのメトリクスタブに移動し、すべての CloudWatch メトリクスを表示を選択します。CloudWatch コンソールで、ログ出力を表示できるようにするログを選択します。

次のステップ

AWS SAM GitHub リポジトリには、ダウンロードして試す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。

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。この演習では、Lambda 関数を使用して AWS リソースにアクセスし、AWS サービス呼び出しを行う方法を示します。

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 を呼び出します。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。この例は、Lambda 関数をローカルでテストする前に、AWS リソースを作成し、IAM アクセス許可を設定する必要があるためです。AWS CloudFormation を使用してリソースを作成し、アクセス許可を設定します。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 アプリケーションに必要な 3 つの AWS リソースを定義します。Lambda 関数、Amazon S3 バケット、DynamoDB テーブルです。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 SAMCLI 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。このデプロイでは、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 リソースとアクセス許可が作成されます。

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ので、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。次に、AWS クラウドで呼び出し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には

- `aws-sam-cli` の `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
```

コマンドでは、`--capabilities` パラメータは AWS CloudFormation が IAM ロールを作成できるようにします。

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には

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、まず によって作成された 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 キー名とバケット名を取得します AWS CloudFormation。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、ダウンロードして試す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 CLITerraform のサポート

このセクションでは、AWS Serverless Application Model プロジェクト Terraform と Terraform クラウドでの コマンドラインインターフェイス (AWS SAM CLI) の使用について説明します。

フィードバックや機能に関する要望を送るには、[GitHub Issue](#) を作成してください。

トピック

- [Terraform の AWS SAM CLI サポートの概要](#)
- [AWS SAM CLI での Terraform のサポートの準備](#)
- [Terraform と連携させた AWS SAM CLI をローカルでのデバッグおよびテストに使用する方法](#)
- [AWS SAM CLI と Serverless.tf を使用したローカルデバッグとテスト](#)
- [Terraform リファレンス付きのAWS SAM CLI](#)

Terraform の AWS SAM CLI サポートの概要

コマンド AWS Serverless Application Model ラインインターフェイス (AWS SAM CLI) を Terraform プロジェクトで使用するか、Terraform Cloud を使用して、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 API。

Terraform の概要については、HashiCorp Terraform ウェブサイトで「[Terraform とは](#)」を参照してください。

フィードバックや機能に関する要望を送るには、[GitHub Issue](#) を作成してください。

Note

の統合の解析ステップの一環として、AWS SAM CLI はユーザーコマンド AWS SAM CLI 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しないでください。特定の環境では、環境またはランナーが出力に追加のログや情報を注入することがあります。

トピック

- [とは AWS SAMCLI](#)
- [Terraform と連携させた AWS SAM CLI をどのように使用しますか？](#)
- [次のステップ](#)

とは AWS SAMCLI

AWS SAMCLI 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サーバーレスアプリケーションTerraformを構築して実行できるコマンドラインツールです。の概要については、AWS SAMCLI「」を参照してください [とは AWS SAMCLI](#)。

は、 の次のコマンド AWS SAMCLI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、「[を使用したテストの概要 sam local invoke](#)」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 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。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-api](#)」を参照してください。
- `sam local start-lambda` – AWS Command Line Interface (AWS CLI) または SDKsローカルエンドポイントを起動します。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 と連携させた AWS SAM CLI をどのように使用しますか？

[コア Terraform ワークフロー](#)は、作成、計画、適用の 3 つの段階で構成されています。AWS SAMCLI のサポートによりTerraform、Terraformワークフローを使用して でアプリケーションを引き続き管理しながら、`sam local`一連のコマンドを利用できます AWS SAMCLI AWS。一般的には、以下の操作を実行する必要があります。

- 作成 — Terraform を使用してインフラストラクチャをコードとして作成します。
- テストとデバッグ — AWS SAM CLI を使用して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。

AWS SAMCLI を使用する例についてはTerraform、AWS コンピューティングブログの「[Better together: AWS SAMCLI andHashiCorp Terraform](#)」を参照してください。

次のステップ

前提条件をすべて満たして Terraform を設定するには、「[AWS SAM CLI での Terraform のサポートの準備](#)」を参照してください。

AWS SAM CLI での Terraform のサポートの準備

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 SAMCLI) の使用を開始する方法について説明しますTerraform。

フィードバックや機能に関する要望を送るには、[GitHub Issue](#) を作成してください。

トピック

- [AWS SAMCLITerraform の前提条件](#)
- [Terraform と連携させた AWS SAM CLI](#)
- [Terraform プロジェクト用のセットアップ](#)
- [Terraform Cloud のセットアップ](#)

AWS SAMCLITerraform の前提条件

Terraform プロジェクトで AWS SAM CLI を使い始めるには、すべての前提条件を満たしてください。

1. AWS SAM CLI をインストールまたはアップグレードする

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
```

AWS SAM CLI がインストールされている場合は、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。

AWS SAM CLI のインストール手順とインストールに必要な前提条件については、「[AWS SAM CLI のインストール](#)」を参照してください。

2. Terraform をインストールする

Terraform 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。

```
$ terraform -version
```

Terraform をインストールするには、Terraform レジストリの「[Install Terraform](#)」を参照してください。

3. ローカルテスト用の Docker のインストール

AWS SAM CLI でローカルテストを実行するには Docker が必要です。Docker をインストールするには、「[AWS SAM CLI で Docker を使用するためのインストール方法](#)」を参照してください。

Terraform と連携させた AWS SAM CLI

サポートされている AWS SAM CLI コマンドを実行するときは、`--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 AWS SAM CLI 設定ファイルで次のように設定できます。

```
hook_name = "terraform"
```

Terraform プロジェクト用のセットアップ

AWS SAM CLI と Terraform プロジェクトを使用するには、このトピックの手順を実行してください。

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。の使用を開始するには、AWS SAM [Terraform と連携させた AWS SAM CLI をローカルでのデバッグおよびテストに使用する方法](#)「」を参照してくださいCLI。

Terraform プロジェクト内の Lambda アーティファクトをビルドするには、以下の操作を実行する必要があります。

1. 3.8 Python 以降をインストールする

2. Make ツールをインストールします。
3. Terraform プロジェクト内で Lambda アーティファクトのビルドロジックを定義します。
4. ビルドロジックの AWS SAM CLI を通知する `awscli` リソースを定義します。
5. `sam build` コマンドを使用して AWS SAMCLI Lambda アーティファクトを構築します。

3.8 Python 3.8 以降をインストールする

Python で使用するには、3.8 AWS SAM 以降が必要です。CLI。 `sam build` を実行すると、AWS SAM CLI が Lambda アーティファクトを構築する Python コマンドを含む `Makefile` を作成します。

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。

以下を実行して、Python 3.8 以降がマシンパスに追加されていることを確認します。

```
$ python --version
```

出力には、3.8 以降のバージョンの Python が表示されるはずです。

Make ツールをインストールする

GNU [Make](#) は、プロジェクトの実行ファイルやその他の非ソースファイルの生成を制御するツールです。AWS SAM CLI は このツールに依存して Lambda アーティファクトを構築する `Makefile` を作成します。

Make をローカルマシンにまだインストールしていない場合は、先に進む前にインストールしてください。

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

`awscli` Terraform リソースタイプを使用して Lambda ビルドロジックを定義します。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 "aws_lambda_function" "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"
  }
}
```

sam metadata リソースを定義します。

sam metadata リソースは、Lambda アーティファクトを見つけるために必要な情報を AWS SAM CLI に提供する `null_resource` Terraform リソースタイプです。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、Terraform レジストリの「[null_resource](#)」を参照してください。

sam metadata リソースを定義します。

1. リソースには `sam_metadata_` で始まる名前を付けます。sam metadata リソースとして認識できるようにするためです。
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 リソースの詳細は、Lambda リソースタイプ (関数またはレイヤー) やパッケージタイプ (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 AWS SAM CLI コマンドを使用すると、AWS SAM CLI は AWS SAM CLI コマンドを実行する前にメタデータファイルを生成します。このファイルを生成したら、今後の AWS SAM CLI コマンドで `--skip-prepare-infra` オプションを使用することで、メタデータの生成プロセスをスキップして時間を節約できます。このオプションは、インフラストラクチャに新しい Lambda 関数や新しい API エンドポイントの作成などの変更を加えていない場合にのみ使用してください。

AWS SAM CLI を使用して Lambda アーティファクトを構築します。

Sam build コマンドを使用して AWS SAM CLI Lambda アーティファクトを構築します。Sam build を実行すると、AWS SAM CLI が次のことをします。

1. Terraform プロジェクト内の sam metadata リソースを探して、Lambda リソースについて調べたり、見つけたりします。
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. Sam local コマンドで使用する Terraform AWS SAM CLI プロジェクトを整理する `.aws-sam` ディレクトリを作成します。

サムビルドでビルドするには

1. Terraform ルートモジュールを含むディレクトリから、以下の操作を実行する。

```
$ sam build --hook-name terraform
```

- 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
```

許容される ID は、Lambda 関数名または完全な Terraform リソースアドレス (`aws_lambda_function.list_books` または `module.list_book_function.aws_lambda_function.this[0]` など) です。

関数のソースコードやその他の Terraform 設定ファイルが Terraform ルートモジュールを含むディレクトリの外部にある場合は、その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

`sam build` コマンドを実行する AWS SAM CLI ときに、ローカル Docker コンテナを使用してアプリケーションを構築する AWS SAM CLI ように設定できます。

Note

Docker がインストールされ、設定されている必要があります。手順については、「[AWS SAM CLI で Docker を使用するためのインストール方法](#)」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

- Terraform、Python、および Make ツールを含む Dockerfile を作成します。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 libgcc libgcc-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. [docker build](#) を使用して Docker イメージをビルドします。

以下に例を示します。

```
$ docker build --tag terraform-build:v1 <path-to-directory-containing-Dockerfile>
```

3. `--use-container` および `--build-image` オプションを使用して `sam build` コマンドを実行します AWS SAMCLI。

以下に例を示します。

```
$ sam build --use-container --build-image terraform-build:v1
```


次のステップ

Terraform プロジェクトで AWS SAM CLI を使い始めるには、「[Terraform と連携させた AWS SAM CLI をローカルでのデバッグおよびテストに使用する方法](#)」を参照してください。

Terraform Cloud のセットアップ

Terraform v1.6.0 以降を使用することをお勧めします。古いバージョンを使用している場合は、Terraform プランファイルをローカルで生成する必要があります。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

Terraform v1.6.0 以降のバージョンでは、これらの手順は不要です。CLI での使用を開始するには AWS SAM、Terraform Cloud「」を参照してください[Terraform と連携させた AWS SAM CLI を使用する](#)。

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、Terraform Cloud のドキュメントの「[API トークン](#)」を参照してください。
2. API トークンの環境変数を設定する — 以下はコマンドラインからの例です。

```
$ export TOKEN="<api-token-value>"
```

3. 実行 ID を取得する – Terraform Cloud コンソールから、で使用する Terraform 実行の実行 ID AWS SAM を見つけます CLI。

実行 ID は実行のブレットドクラムパスにあります。

my-sample-sam-org / Projects & workspaces / my-sample-sam-workspace / Runs / run-QUFkZxG9jafBd1gU

my-sample-sam-workspace

ID: ws-xwa1CyQRgFaF2H7b 

No workspace description available. [Add workspace description.](#)



4. プランファイルの取得 — API トークンを使用して、ローカルの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、Terraform Cloud と連携させた AWS SAM CLI の使用の準備ができました。サポートされている AWS SAM CLI コマンドを使用するときは、`--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 GitHub リポジトリの「[api_gateway_v2_tf_cloud](#)」を参照してください。

次のステップ

Terraform Cloud と連携させた AWS SAM CLI の使用を開始するには、「[Terraform と連携させた AWS SAM CLI をローカルでのデバッグおよびテストに使用する方法](#)」を参照してください。

Terraform と連携させた AWS SAM CLI をローカルでのデバッグおよびテストに使用する方法

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 SAMCLI) コマンドを Terraform プロジェクトとで使用する方法について説明します Terraform Cloud。

フィードバックや機能に関する要望を送るには、[GitHub Issue](#) を作成してください。

トピック

- [sam local invoke によるローカルテスト](#)
- [sam local start-api によるローカルテスト](#)
- [sam local start-lambda によるローカルテスト](#)
- [Terraform の制限事項](#)

sam local invoke によるローカルテスト

Note

AWS SAM CLI を使用してローカルでテストを実行するには、Docker をインストールし、構成する必要があります。手順については、「[AWS SAM CLI で Docker を使用するためのインストール方法](#)」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、「[を使用したテストの概要 sam local invoke](#)」を参照してください。

sam local start-api によるローカルテスト

Terraform で sam local start-api を使用するには、以下を実行してください。

```
$ 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、「[を使用したテストの概要 sam local start-api](#)」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、AWS SAM CLI 「」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](#)。
- Terraform での Lambda オーソライザーの使用に関する詳細は、Terraform レジストリの「[Resource: aws_api_gateway_authorizer](#)」を参照してください。

sam local start-lambda によるローカルテスト

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. AWS SAM CLI を使用してローカルテスト環境を作成します。

```
$ 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 の制限事項

Terraform と連携させた AWS SAM CLI を使用する場合は次のとおりです。

- 複数のレイヤーにリンクされている Lambda 関数。
- リソース間のリンクを定義する Terraform のローカル変数。
- まだ作成されていない Lambda 関数の参照。これには、REST API リソースの本文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。

AWS SAM CLI と Serverless.tf を使用したローカルデバッグとテスト

AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) 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。以下の 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 以降では、AWS SAM CLI 統合がサポートされています。

Serverless.tf モジュール AWS SAMCLIでの使用を開始するには、最新バージョンの Serverless.tf と AWS SAMに更新します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、`sam build` コマンドに必要なメタデータリソースが生成されます AWS SAMCLI。

の詳細についてはServerless.tf、「」を参照してください[terraform-aws-lambda-module](#)。

Terraform リファレンス付きのAWS SAM CLI

このセクションは、で AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) を使用してローカルのデバッグとテストTerraformを行うためのリファレンスです。

フィードバックや機能に関する要望を送るには、[GitHub Issue](#) を作成してください。

AWS SAM サポートされている機能のリファレンス

Terraform での使用がサポートされている AWS SAM CLI の機能に関するリファレンスは、次の場所にあります。

- [sam build](#)
- [sam local invoke](#)
- [sam local start-api](#)
- [sam local start-lambda](#)

Terraform 特有のリファレンス

Terraform で AWS SAM CLI を使用する際のリファレンスドキュメントは、次の場所にあります。

- [sam metadata resource](#)

sam metadata resource

このページには、Terraform プロジェクトで使用する sam metadata resource リソースタイプに関するリファレンスが含まれています。

- で AWS Serverless Application Model コマンドラインインターフェイス (AWS SAMCLI) を使用する方法の概要についてはTerraform、「」を参照してください[Terraform の AWS SAM CLI サポートの概要](#)。
- Terraform と連携させた AWS SAM CLI を使用するには、「[Terraform と連携させた AWS SAM CLI をローカルでのデバッグおよびテストに使用する方法](#)」を参照してください。

トピック

- [引数](#)
- [例](#)

引数

引数	説明
<code>built_output_path</code>	AWS Lambda 関数のビルドされたアーティファクトへのパス。
<code>docker_build_args</code>	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。
<code>docker_context</code>	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。 。
<code>docker_file</code>	Docker ファイルへのパス。このパスは、 <code>docker_context</code> 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 <code>Dockerfile</code> です。
<code>docker_tag</code>	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。
<code>depends_on</code>	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、Terraform レジストリの「 depends_on 引数 」を参照してください。
<code>original_source_code</code>	Lambda 関数が定義されている場所へのパス。この値には文字列、文字列の配列、または文字列としてデコードされた JSON オブジェクトを使用できます。 <ul style="list-style-type: none">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は、<code>source_code_property</code> も定義する必要があります。
<code>resource_name</code>	Lambda 関数の名前。
<code>resource_type</code>	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。 <ul style="list-style-type: none"><code>IMAGE_LAMBDA_FUNCTION</code><code>LAMBDA_LAYER</code><code>ZIP_LAMBDA_FUNCTION</code>

引数	説明
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

パッケージタイプが ZIP の Lambda 関数を参照する sam metadata resource

```
# 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]
}
```

AWS CDK を使用して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 SAMCLI

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。は AWS CDK プロジェクト構造内で AWS SAMCLI 機能するため、[AWS CDK ツールキット](#) を使用して AWS CDK アプリケーションを作成、変更、デプロイできます。

のインストールと設定の詳細については AWS CDK、「AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「[の開始 AWS CDK 方法](#)」を参照してください。

Note

は、バージョン 1.135.0 以降の AWS CDK v1 と、バージョン 2.0.0 以降の AWS CDK v2 AWS SAMCLI をサポートしています。

トピック

- [AWS SAM と の開始方法 AWS CDK](#)
- [アプリケーションをローカルでテスト AWS CDK する](#)
- [AWS CDK アプリケーションの構築](#)
- [AWS CDK アプリケーションのデプロイ](#)

AWS SAM と の開始方法 AWS CDK

このトピックでは、AWS SAMCLI を AWS CDK アプリケーションで使用するために必要な内容と、シンプルな AWS CDK アプリケーションを構築してローカルでテストする手順について説明します。

前提条件

AWS SAMCLI を使用するには AWS CDK、AWS CDK をインストールする必要があります AWS SAMCLI。

- のインストールの詳細については AWS CDK、「AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「[の開始 AWS CDK 方法](#)」を参照してください。

- のインストールについては、AWS SAM CLI 「」を参照してください [AWS SAM CLI のインストール](#)。

AWS CDK アプリケーションの作成とローカルでのテスト

を使用して AWS CDK アプリケーションをローカルでテストするには AWS SAM CLI、Lambda 関数を含むアプリケーションが必要です AWS CDK。Lambda 関数を使用して基本的な AWS CDK アプリケーションを作成するには、次のステップに従います。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

は、バージョン 1.135.0 以降の AWS CDK v1 と、バージョン 2.0.0 以降の AWS CDK v2 AWS SAM CLIをサポートしています。

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、を使用する AWS CDK アプリケーションを初期化します TypeScript。

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

を使用して AWS SAMCLI、AWS CDK アプリケーションで定義した Lambda 関数をローカルで呼び出すことができます。そのためには、関数コンストラクト識別子と合成された 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
```

CLI を使用して AWS CDK アプリケーションをテストするために使用できるオプションの詳細については、AWS SAM 「」を参照してください [アプリケーションをローカルでテスト AWS CDK する](#)。

アプリケーションをローカルでテスト AWS CDK する

を使用して AWS SAMCLI アプリケーションをローカルでテストするには、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次のコマンドを実行します AWS CDK 。

- [sam local invoke](#)
- [sam local start-api](#)
- [sam local start-lambda](#)

AWS CDK アプリケーションでsam localコマンドを実行する前に、`cdk synth`を実行する必要があります。

実行sam local invoke時には、呼び出す関数コンストラクト識別子と、合成された 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 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 アプリケーションが 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 `sam build` 前に `cdk synth` を実行してから を実行します。を実行する場合 `sam build`、など、ランタイム固有のコンストラクトを使用する Lambda 関数やレイヤーは構築 AWS SAM されません [NodejsFunction](#)。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次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 CDK アプリケーションのデプロイ

AWS SAMCLI はアプリケーションのデプロイをサポートしていません AWS CDK。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。

を使用したアプリケーションの公開 AWS SAMCLI

他のユーザーが AWS SAM アプリケーションを検索してデプロイできるようにするには、を使用してアプリケーションを AWS SAMCLI に公開します AWS Serverless Application Repository。を使用してアプリケーションを公開するには AWS SAMCLI、AWS SAM テンプレートを 사용하여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があります。

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。(実行する内容は、アプリケーションが にすでに存在するかどうか AWS Serverless Application Repository、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よって異なります)。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

AWS Serverless Application Repository を使用して にアプリケーションを発行する前に AWS SAMCLI、以下が必要です。

- 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 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 を使用するようしてください。Condition 要素の式により、AWS Serverless Application Repository は指定された AWS アカウントからアプリケーションにのみアクセスするアクセス許可を持ちます。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、AWS SAM テンプレートに Metadata 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 セクションの詳細については、「」を参照してください [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](#)。

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 に発行するには AWS Serverless Application Repository、次の AWS SAMCLI 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公開するか、特定の AWS アカウントリストにアクセス許可を付与する必要があります。

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 CLI、「AWS Serverless Application Repository デベロッパーガイド」の [AWS Serverless Application Repository 「リソースベースのポリシーの例」](#) 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

にアプリケーションを公開したら AWS Serverless Application Repository、新しいバージョンのアプリケーションを公開できます。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 CloudFormation [組み込み関数](#)はメタデータAWS::ServerlessRepo::Applicationキーではサポートされていません。

プロパティ

この表は、AWS SAM テンプレートの Metadataセクションのプロパティに関する情報を示しています。このセクションは、AWS Serverless Application Repository を使用してアプリケーションを発行するために必要です AWS SAMCLI。

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。 sam package コマンドを使用してパッケージ化されていない AWS SAM テンプレートファイルには、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ができます。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
			<p>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。</p>
ReadmeUrl	文字列	FALSE	<p>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。</p> <p>sam package コマンドを使用してパッケージ化されていない AWS SAM テンプレートファイルには、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ができます。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。</p> <p>最大サイズ: 5 MB。</p>
Labels	文字列	FALSE	<p>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。</p> <p>最小長: 1 最大長 = 127。ラベルの最大数:10。</p> <p>パターン: <code>"^[a-zA-Z0-9+\\-_:\\/@]+\$"</code>;</p>
HomePageUrl	文字列	FALSE	<p>アプリケーションの GitHub リポジトリの場所など、アプリケーションに関する詳細情報を含む URL。</p>

プロパティ	タイプ	必須	説明
SemanticVersion	文字列	FALSE	<p>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ンングの仕様については、「セマンティックバージョンング」のウェブサイトを参照してください。</p> <p>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。</p>
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 – アカウントに一致する名前 AWS Serverless Application Repository のアプリケーションがない場合、新しいアプリケーションが作成されます。
 - 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 – アカウントの名前が一致するアプリケーションが AWS Serverless Application Repository に既に存在しており、SemanticVersionが変更されている場合、アプリケーションバージョンが作成されます。
 - 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 – アカウントに一致する名前 AWS Serverless Application Repository のアプリケーションが にすでにあり、SemanticVersionが変更されていない場合、アプリケーションは更新されます。
 - 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
```

のドキュメント履歴 AWS SAM

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。

- ドキュメントの最終更新日：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成と更新	検出可能性とユーザビリティを向上させるために、ガイドを再編成し、再構成しました。タイトルを更新し、改善しました。トピックと概念を紹介するときに、追加の詳細を提供しました。	2024年6月20日
Ruby 3.3 の AWS SAMCLI のサポートを追加	Ruby 3.3 がランタイムおよびイメージリポジトリとして利用可能になりました。詳細については、「 イメージリポジトリと 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を追加	コマンド sam local start-api : <code>--ssl-cert-file PATH</code> 、の新しいオプションが利用可能です <code>--ssl-key-file PATH</code> 。さらに、新しいコマンドラインオプション <code>--add-host LIST</code> は、 sam local invoke 、 sam local start-api 、 sam local start-lambda で使用できます。	2024年3月20日

[.NET 8 AWS SAMCLIのサポートを追加](#)

.NET 8 がランタイムおよびイメージリポジトリとして利用可能になりました。.NET Core 3.1、Node.js 14、Node.js 12、Python 3.7、Ruby 2.7 の実行時間とイメージリポジトリはサポートされなくなりました。[「イメージリポジトリ」](#)と[「sam init」](#)を参照してください。

2024 年 2 月 22 日

[の AWS SAMCLI arm64 パッケージインストーラを追加 Linux](#)

手順については、[「のインストーラ」](#)を参照してください [AWS SAMCLI](#)。

2023 年 12 月 6 日

[sam sync コマンドに --watch-exclude AWS SAMCLI オプションを追加](#)

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

[sam sync コマンドに --build-in-source option AWS SAMCLI を追加](#)

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

[sam build コマンドに --build-in-source option AWS SAMCLI を追加](#)

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 リモート呼び出しコマンドの新しいリソースサポートを追加](#)

`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われません。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 SAMCLI リモート test-event コマンドを追加](#)

を使用して AWS SAM CLI、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、で Lambda 関数をテストします AWS クラウド。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

[AWS SAMCLI のサポートを追加 Terraform Cloud](#)

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 SAMCLI設定YAMLファイルのファイル形式のサポートを追加	が [.yaml .yml] ファイル形式をサポートする AWS SAMCLI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
のコマンドサポートを追加 AWS SAMCLIsam local start-apiTerraform	AWS SAMCLI のサポート内容セクションTerraform が更新され、のsam local start-api コマンドサポートが追加され AWS SAMCLIました Terraform。	2023 年 7 月 6 日
新しい AWS SAMCLIリモート呼び出しコマンドを追加	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
サーバーレスリソースタイプを追加 AWS AppSyncGraphQL API	でGraphQL APIリソースを定義する方法を説明する新しい AWS::Serverless::GraphQLApi セクションを作成します AWS SAM。	2023 年 6 月 22 日
3.2 RubyAWS SAMCLIのサポートを追加	「 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のオプション手順を追加しました。](#)

「[AWS SAM CLI のインストーラ](#)」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 オプションを追加](#)

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する sam sync します。詳細については、「[最初の 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を追加](#)

AWS SAM テンプレート仕様で、AWS::Serverless::Function リソースの DocumentDB イベントソースタイプがサポートされ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

[の外部で関数リソースを構築する AWS SAM](#)

sam build コマンド使用時の関数のスキップに関するガイドを追加しました。詳細については、「[の外部で関数を構築する AWS SAM](#)」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 のフォーマットプロパティと 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

[リソース EventSource の AWS::Serverless::StateMachine に追加されたターゲットプロパティ。](#)

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 ポーラーのスケールを設定する](#)

ScalingConfig プロパティを使用して AWS::Serverless::Function に SQS ポーラーのスケールを設定します。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 で AWS SAM アプリケーションを検証する](#)

cfn-lint を使用して、AWS SAMを通じて AWS SAM テンプレートを検証できますCLI。詳細については、「[Validate with cfn-lint](#)」(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 を使用してサーバーレスアプリケーションをモニタリングする](#)

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、「[Application Insights 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 CloudWatch](#)」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、「[のインストール](#)」を参照してください [AWS SAM CLI](#)。

2022 年 12 月 6 日

[Lambda のサポートを追加 SnapStart](#)

初期化された関数のキャッシュ状態であるスナップショットを作成するように Lambda 関数 SnapStart 用を設定します。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 applications.To 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。詳細については、[「リソース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](#)

ネイティブ先行 (AOT) コンパイルを利用して、.NET 7 Lambda 関数を で構築してパッケージ化し AWS SAM、Lambda 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のための 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

[ス EventBridge ジョーラ AWS SAM のサポートを追加](#)

AWS Serverless Application Model (AWS SAM) テンプレート仕様は、AWS Lambda およびのス EventBridge ジョーラでイベントをスケジュールするために使用できるシンプルで短い構文を提供します AWS Step Functions。詳細については、「[スケ EventBridge ジョーラ を使用した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については、「[のインストール](#)」を参照してください [AWS SAM CLI](#)。

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](#)

を使用して Lambda 関数を段階的にデプロイするには、2 つのステップ AWS CodeDeploy 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 FilterCriteria プロパティが追加されました。

2022 年 10 月 13 日

[AWS SAM パイプラインに OpenID Connect \(OIDC\) サポートを追加](#)

AWS SAM は、Bitbucket、アクション、GitHub 継続的インテグレーションおよび GitLab 継続的デリバリー (CI/CD) プラットフォームの 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 「パイプラインでの OIDC ユーザーアカウントの使用」](#)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

Function との新しいプロパティ StateMachine EventSource	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 コネクタが一般利用可能に	コネクタは、として識別される AWS SAM 抽象リソースタイプで <code>AWS::Serverless::Connector</code> 、サーバーレスアプリケーションリソース間でアクセス許可をプロビジョニングするシンプルで安全な方法を提供します。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	<code>sam sync</code> に <code>--dependency-layer</code> および <code>--use-container</code> 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	<code>sam deploy</code> に <code>--on-failure</code> 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には、[esbuild JavaScript バンドル](#)や AWS SAM CLI で使用できません。

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 関数をローカルで呼び出す](#)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[アーキテクチャ](#)」プロパティと「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 CLI/AWS 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、を使用して AWS CDK アプリケーションをパッケージ AWS SAMCLI 化およびデプロイできるようになりました。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 AWS SAMCLI。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 ビルドコンテナイメージを使用して、Jenkins AWS CodePipeline、CI/CD、GitHub Actions などの一般的な GitLab 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

[Toolkit を使用した AWS SAM アプリケーションのデバッグ AWS](#)

AWS ツールキットは、統合開発環境 (IDEs) とランタイムのより多くの組み合わせによる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 ツールキットの使用](#)」を参照してください。

2021 年 5 月 20 日

[AWS SAM CLI AWS CDK の統合 \(プレビュー\)](#)

を使用して、AWS CDK アプリケーションをローカル AWS SAM CLI でテストおよび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](#)

デフォルトのコンテナイメージリポジトリが [Amazon ECR Public](#) DockerHub 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](#)

毎晩構築 AWS SAM CLIされるのプレリリースバージョン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、「[のインストール](#)」の「[選択した OS サブトピックの夜間ビルド](#)」[セクションを参照してください AWS SAM CLI](#)。

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

のデッドレターキューのサポート EventBridge

サーバーレス関数 EventBridge とステートマシンの および Schedule イベントソース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 FunctionResponseType プロパティを参照してください。

2021 年 1 月 29 日

[タブラ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ブラ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 `TablingWindowInSeconds` 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、「アプリケーションの [コード署名の設定 AWS SAM](#)」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、「[による AWS リソースのオーケストレーション AWS Step Functions](#)」および「[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	が AWS SAM 生成する AWS CloudFormation リソースの詳細と、それらを参照する方法について説明しました。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 認証情報の設定	AWS SDKsやなどの他の AWS ツールで使用するよう AWS 認証情報をまだ設定していない場合に備えて、認証情報を設定する手順を追加しました AWS CLI。詳細については、「 認証情報のセットアップ AWS 」を参照してください。	2020 年 1 月 17 日
AWS SAM 仕様と AWS SAMCLI更新	から AWS SAM 仕様を移行しました GitHub。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。また、RekognitionFacesPolicy との 2 つのポリシーテンプレートも更新されました ElasticsearchHttpPostPolicy。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、「[の開始方法 AWS SAM](#)」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、[「を使用したサーバレスアプリケーションの発行」](#)を参照してください [AWS SAM CLI](#)。

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](#)

に Linuxbrew (Linux)、MSI (Windows)、および Homebrew (macOS) のインストールオプションを追加しました AWS SAM CLI。詳細については、[「のインストール」を参照してください AWS SAM CLI](#)。

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。