このコンテンツは公開から3年以上経過しており内容が古い可能性があります
最新情報についてはサービス別資料もしくはサービスのドキュメントをご確認ください

[AWS Black Belt Online Seminar] AWS Serverless Application Model (AWS SAM)

サービスカットシリーズ

Solutions Architect
今村 優太

2019/8/14

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

自己紹介

- 名前：

  今村 優太

- 所属：

  技術統括本部 Developer Advocacy チーム

  ソリューションアーキテクト

- 好きな AWS のサービス：

  AWS Lambda　AWS Step Functions　AWS AppSync

aws certified Solutions Architect Professional

aws certified DevOps Engineer Professional

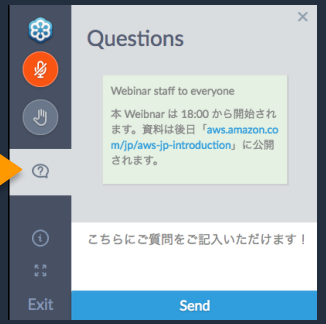
AWS Black Belt Online Seminar とは

「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。

質問を投げることができます！

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問はお答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック



Twitter ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では2019年8月14日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

アジェンダ

- AWS SAM とは
- AWS SAM の機能と文法
- AWS SAM Command Line Interface
- その他考慮事項

アジェンダ

- AWS SAM とは
- AWS SAM の機能と文法
- AWS SAM Command Line Interface
- その他考慮事項

Serverless な Application

何らかの**「イベント」**に応じて Lambda 関数を起動し処理を行う性質を持つ

イベント

データの追加・変更・削除

エンドポイントへのアクセス

リソースの状態変化 など

イベントが発生

Lambda 関数

処理ロジックを実行

必要に応じアクセス

その他のサービス

…etc

サーバーレスを取り巻く主なコンポーネント

外部I/F

REST API:
Amazon API Gateway

GraphQL:
AWS AppSync

処理ロジック

Java/Go/Node.js/Python/
.Net C#/Ruby and more:
AWS Lambda

データ保持/管理

NoSQL:
Amazon DynamoDB

Queue:
Amazon SQS

File:
Amazon S3

ストリーム

流入データ:
Amazon Kinesis Data Streams

AWS Serverless Application Model (SAM)

- イベント駆動の性質を持つサーバーレス アプリケーションのデプロイに特化した、AWS CloudFormation の拡張機能
  - CloudFormation で同一の内容を定義した場合と比較して、簡潔にテンプレートを書くことができる
- CloudFormation 同様に YAML もしくは JSON 形式で、SAM テンプレートを記述する
- サーバーレス アプリケーションの開発を支援するAWS SAM コマンドラインインターフェイス (CLI) が提供されている
- GitHub 上でオープンソースで開発が行われている

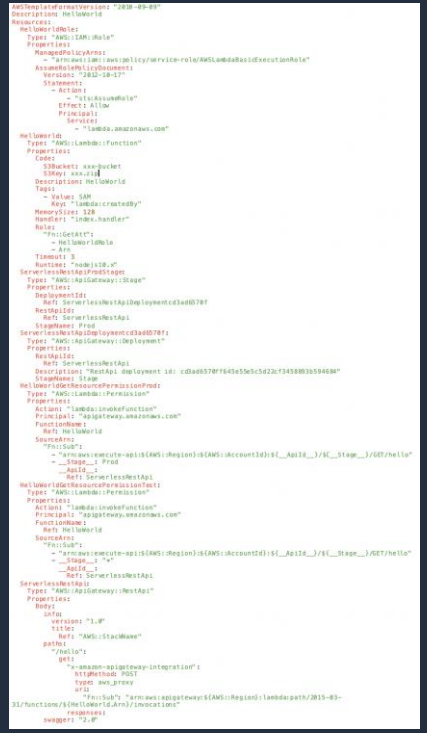
SAM テンプレートの特長 (1/3)

SAM を利用すると、**より簡潔に**サーバーレスなアプリケーションを定義できる

SAM を利用したテンプレート

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx-bucket/xxx.zip
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /hello
            Method: get
```

素の状態の CloudFormation テンプレート



```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: HelloWorld
Resources:
  HelloWorldRole:
    Type: "AWS::IAM::Role"
    Properties:
      ManagedPolicyArns:
        - "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Action:
              - "sts:AssumeRole"
            Effect: Allow
            Principal:
              Service:
                - "lambda.amazonaws.com"
  HelloWorld:
    Type: "AWS::Lambda::Function"
    Properties:
      Code:
        S3Bucket: xxx-bucket
        S3Key: xxx.zip
      Description: HelloWorld
      Tags:
        - Value: SAM
          Key: "lambda:createdBy"
      MemorySize: 128
      Handler: "index.handler"
      Role:
        "Fn::GetAtt":
          - HelloWorldRole
          - Arn
      Timeout: 3
      Runtime: "nodejs10.x"
  ServerlessRestApiProdStage:
    Type: "AWS::ApiGateway::Stage"
    Properties:
      DeploymentId:
        Ref: ServerlessRestApiDeployment cd3ad6570f
      RestApiId:
        Ref: ServerlessRestApi
      StageName: Prod
  ServerlessRestApiDeployment cd3ad6570f:
    Type: "AWS::ApiGateway::Deployment"
    Properties:
      RestApiId:
        Ref: ServerlessRestApi
      Description: "RestApi deployment id: cd3ad6570f645e55e5c5d23cf3458ff3b5946f"
      StageName: Stage
  HelloWorldGetResourcePermissionProd:
    Type: "AWS::Lambda::Permission"
    Properties:
      Action: "lambda:invokeFunction"
      Principal: "apigateway.amazonaws.com"
      FunctionName:
        Ref: HelloWorld
      SourceArn:
        "Fn::Sub":
          - "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${__ApiId__}/${__Stage__}/GET/hello"
          - __Stage__: Prod
            __ApiId__:
              Ref: ServerlessRestApi
  HelloWorldGetResourcePermissionTest:
    Type: "AWS::Lambda::Permission"
    Properties:
      Action: "lambda:invokeFunction"
      Principal: "apigateway.amazonaws.com"
      FunctionName:
        Ref: HelloWorld
      SourceArn:
        "Fn::Sub":
          - "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${__ApiId__}/${__Stage__}/GET/hello"
          - __Stage__: "*"
            __ApiId__:
              Ref: ServerlessRestApi
  ServerlessRestApi:
    Type: "AWS::ApiGateway::RestApi"
    Properties:
      Body:
        info:
          version: "1.0"
          title:
            Ref: "AWS::StackName"
        paths:
          "/hello":
            get:
              "x-amazon-apigateway-integration":
                httpMethod: POST
                type: aws_proxy
                uri:
                  "Fn::Sub": "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-
31/functions/${HelloWorld.Arn}/invocations"
              responses:
        swagger: "2.0"
```

SAM テンプレートの特長 (2/3)

CloudFormation でデプロイする際に、**本来の CloudFormation の文法に変換**される

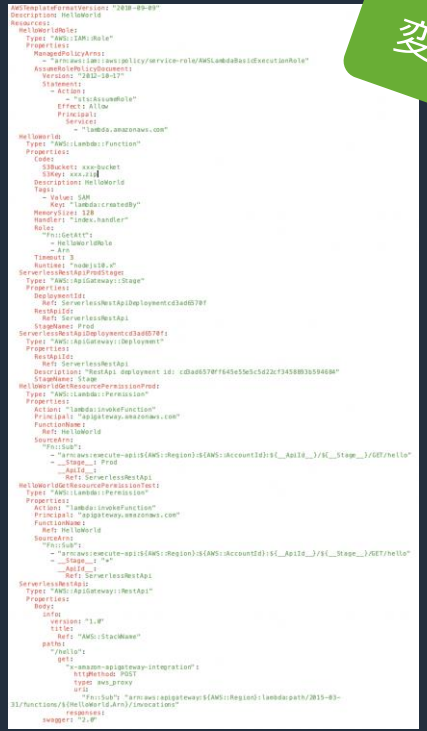
変換前

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx-bucket/xxx.zip
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /hello
            Method: get
```

CloudFormation のコンソールや CLI でデプロイ

AWS CloudFormation

変換後

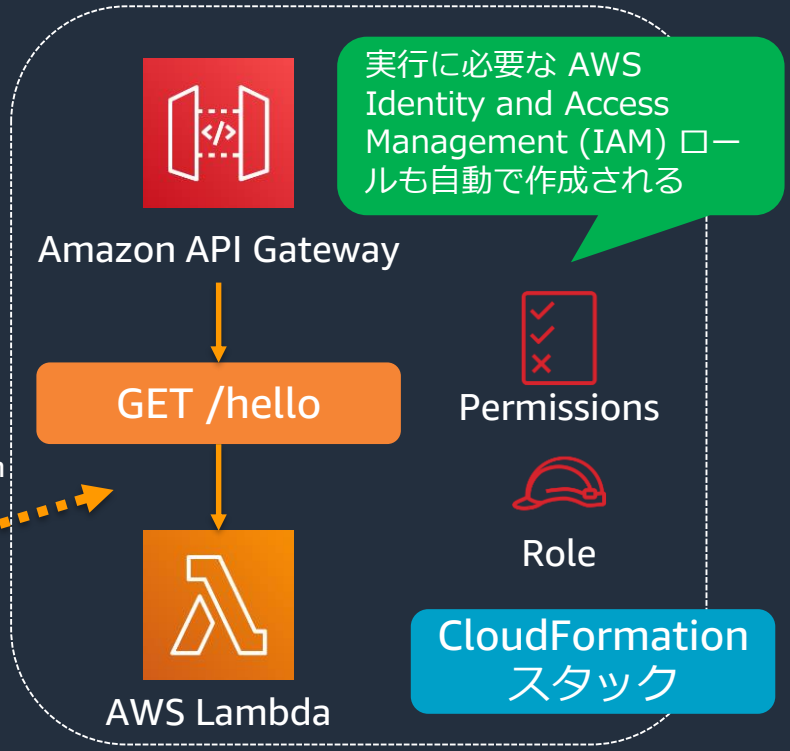


```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: HelloWorld
Resources:
  HelloWorldRole:
    Type: "AWS::IAM::Role"
    Properties:
      ManagedPolicyArns:
        - "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Action:
              - "sts:AssumeRole"
            Effect: Allow
            Principal:
              Service:
                - "lambda.amazonaws.com"
  HelloWorld:
    Type: "AWS::Lambda::Function"
    Properties:
      Code:
        S3Bucket: xxx-bucket
        S3Key: xxx.zip
      Description: HelloWorld
      Tags:
        - Value: SAM
          Key: "lambda:createdBy"
      MemorySize: 128
      Handler: "index.handler"
      Role:
        "Fn::GetAtt":
          - HelloWorldRole
          - Arn
      Timeout: 3
      Runtime: "nodejs10.x"
  ServerlessRestApiProdStage:
    Type: "AWS::ApiGateway::Stage"
    Properties:
      DeploymentId:
        Ref: ServerlessRestApiDeploymentcd3add570f
      RestApiId:
        Ref: ServerlessRestApi
      StageName: Prod
  ServerlessRestApiDeploymentcd3add570f:
    Type: "AWS::ApiGateway::Deployment"
    Properties:
      RestApiId:
        Ref: ServerlessRestApi
      Description: "RestApi deployment id: cd3add570ff645e55e5c5d22cf34588b3b594604"
      StageName: Stage
  HelloWorldGetResourcePermissionProd:
    Type: "AWS::Lambda::Permission"
    Properties:
      Action: "lambda:invokeFunction"
      Principal: "apigateway.amazonaws.com"
      FunctionName:
        Ref: HelloWorld
      SourceArn:
        "Fn::Sub":
          - "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${__ApiId__}/${__Stage__}/GET/hello"
          - __Stage__: Prod
            __ApiId__:
              Ref: ServerlessRestApi
  HelloWorldGetResourcePermissionTest:
    Type: "AWS::Lambda::Permission"
    Properties:
      Action: "lambda:invokeFunction"
      Principal: "apigateway.amazonaws.com"
      FunctionName:
        Ref: HelloWorld
      SourceArn:
        "Fn::Sub":
          - "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${__ApiId__}/${__Stage__}/GET/hello"
          - __Stage__: "*"
            __ApiId__:
              Ref: ServerlessRestApi
  ServerlessRestApi:
    Type: "AWS::ApiGateway::RestApi"
    Properties:
      Body:
        info:
          version: "1.0"
          title:
            Ref: "AWS::StackName"
        paths:
          "/hello":
            get:
              "x-amazon-apigateway-integration":
                httpMethod: POST
                type: aws_proxy
                uri:
                  "Fn::Sub": "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-
31/functions/${HelloWorld.Arn}/invocations"
              responses:
        swagger: "2.0"
```

SAM テンプレートの特長 (3/3)

より直感的に分かりやすくサーバーレスなアプリケーションを定義できる

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx-bucket/xxx.zip
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /hello
            Method: get
```



変換の仕組み

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx-bucket/xxx.zip
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /hello
            Method: get
```

Transform セクション

- もともと、CloudFormation には Transform というセクションを定義でき、CloudFormation テンプレートの変換方式 (マクロ) を用意できる
- 'AWS::Serverless-2016-10-31' というマクロが、あらかじめ CloudFormation 側で用意されている
- AWS::Serverless というプレフィックスの Type のリソースを、CloudFormation の本来の文法に変換する
- 自身で定義した CloudFormation マクロとの共存も可能

通常の CloudFormation テンプレートとの統合

SAM は CloudFormation の拡張なので、**通常の CloudFormation の文法と共存**できる

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs10.x
      CodeUri: s3://xxx-bucket/xxx.zip
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: my-bucket
```

Type が AWS::Serverless でない、通常の CloudFormation の文法で記述するリソースも、同一のテンプレートに含められる

アジェンダ

- AWS SAM とは
- AWS SAM の機能と文法
- AWS SAM Command Line Interface
- その他考慮事項

SAM テンプレート – ヘッダー

用意する SAM テンプレートのバージョンや、デプロイする内容について説明を書く

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: 'This is a sample template.'
```

プロパティ	型	内容	指定
AWSTemplateFormatVersion	String	CloudFormation テンプレートのバージョンを指定。2019/08/14 時点では '2010-09-09' のみが許容される	任意
Transform	String	SAM テンプレートのバージョンを指定。2019/08/14 時点では 'AWS::Serverless-2016-10-31' のみが許容される	**必須**
Description	String	このテンプレートに関する説明を記載	任意

SAM テンプレート – リソース

Resources の配下に、実際にデプロイするリソースの詳細を定義していく
Type に指定する値によって、どのようなリソースを配置するのかが決まる

例

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: 'This is a sample template.'
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx-bucket/xxx.zip
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
```

Type により、指定可能なプロパティの内容はそれぞれ異なる

Type に指定可能な値は次の 5 種類

- AWS::Serverless::Function
  - AWS Lambda をデプロイ。Lambda の呼び出し元となるリソースも同時にデプロイされる場合がある
- AWS::Serverless::Api
  - Amazon API Gateway をデプロイ
- AWS::Serverless::SimpleTable
  - Amazon DynamoDB のテーブルをデプロイ
- AWS::Serverless::LayerVersion
  - Lambda Layer をデプロイ
- AWS::Serverless::Application
  - Serverless Application Repository に存在するアプリケーションをデプロイ

SAM テンプレートのリソースタイプ – 5 種類

- AWS::Serverless::Function
- AWS::Serverless::Api
- AWS::Serverless::SimpleTable
- AWS::Serverless::LayerVersion
- AWS::Serverless::Application

AWS::Serverless::Function リソースタイプ

AWS Lambda 関数をデプロイすると同時に、Lambda を起動するイベントを設定する。
イベントの種類によっては、**イベント発生元となるリソースも同時にデプロイ**される。

例

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://my-bucket/app.zip
      Handler: index.handler
      Runtime: nodejs8.10
      Events:
        ApiEvent1:
          Type: Api
          Properties:
            Path: /hello
            Method: get
(...省略...)
```

AWS Lambda の実装や設定値

＋

Lambda を起動するイベント

Properties の中に
この 2 種類を記載することが基本

AWS::Serverless::Function で紹介するプロパティ

- AWS::Serverless::Function
  - Handler, Runtime プロパティ
  - CodeUri, InlineCode プロパティ
  - Events プロパティ
  - AutoPublishAlias プロパティ
  - DeploymentPreference プロパティ
  - Policies プロパティ

Handler, Runtime プロパティ

Lambda 関数で利用するランタイムと、実行する関数の位置 (ハンドラ) を指定する

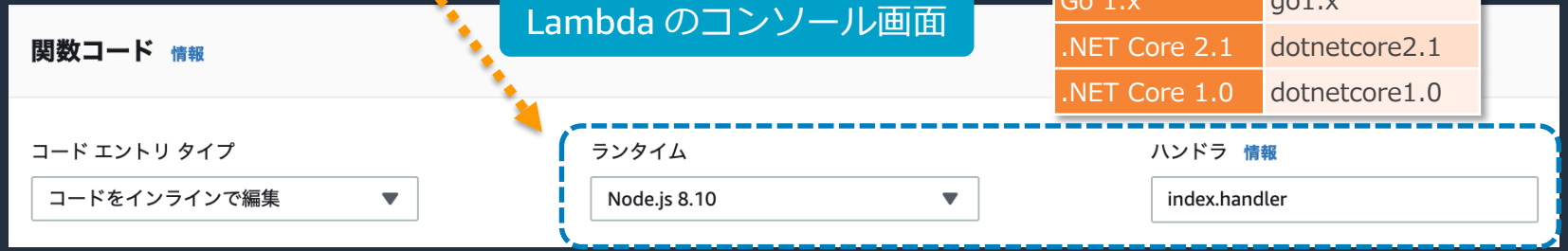
例

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
(...省略...)
```

以下の識別子が Runtime に指定可能

ランタイム	識別子
Node.js 10	nodejs10.x
Node.js 8.10	nodejs8.10
Python 3.7	python3.7
Python 3.6	python3.6
Python 2.7	python2.7
Ruby 2.5	ruby2.5
Java 8	java8
Go 1.x	go1.x
.NET Core 2.1	dotnetcore2.1
.NET Core 1.0	dotnetcore1.0

Lambda のコンソール画面



CodeUri, InlineCode プロパティ

Lambda 関数の実装内容は CodeUri もしくは InlineCode プロパティに指定する
そのため、CodeUri か InlineCode プロパティいずれかの指定が必須となる

CodeUri の場合：

例

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://my-bucket/app.zip

(...省略...)
```

S3 に事前にアップロードした ZIP ファイルの URI を指定する
(ZIP ファイルに関数の実装を固めておく)

InlineCode の場合：

例

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      InlineCode: |
        exports.handler = async (event, context, callback) => {
          return event
        }
      Handler: index.handler

(...省略...)
```

テンプレートの中に直接コードを書く

CodeUri – AWS Command Line Interface (CLI) によるデプロイ (1/2)

template.yml

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: ./src
      Events:
        ApiEvent1:
          Type: Api
          Properties:
            Path: /hello
            Method: GET
```

ローカル環境

my-bucket

output.yml

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://my-bucket/9a94737fdebabfcd740708aa35fffe43
      Events:
        ApiEvent1:
          Type: Api
          Properties:
            Path: /hello
            Method: GET
```

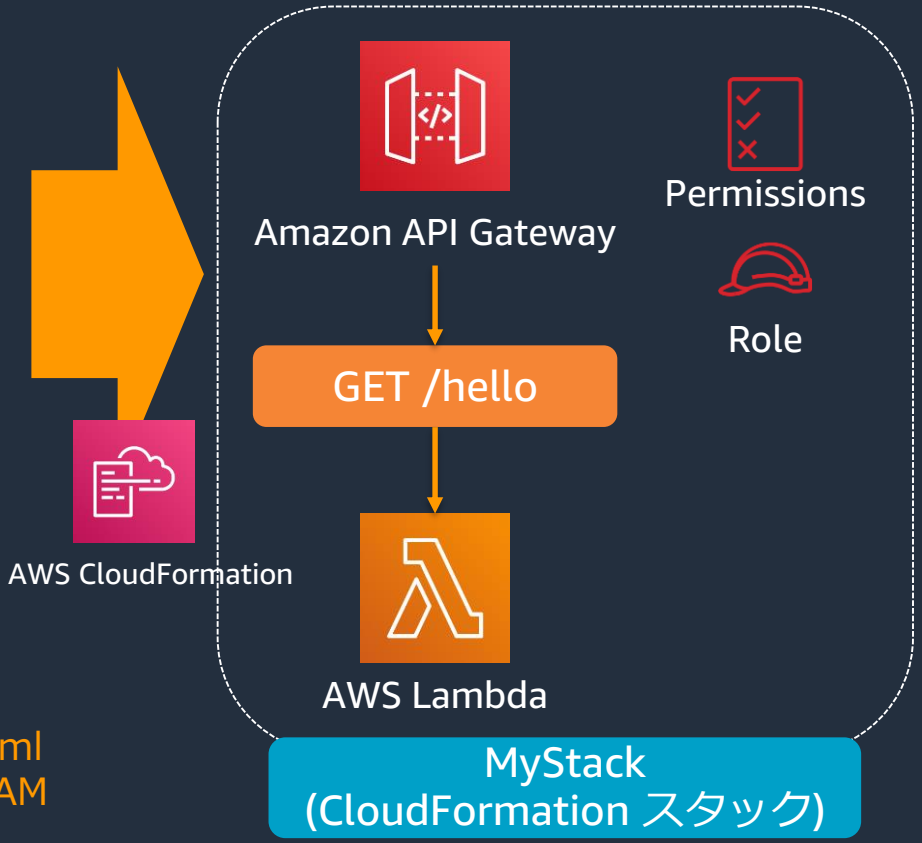
aws cloudformation package --template-file template.yml --s3-bucket my-bucket --output-template-file output.yml

コマンドを実行すると、CodeUri に指定したファイルやディレクトリを ZIP 形式に固め、指定した S3 のバケットにアップロードし、CodeUri の値が S3 の URI に変更される。

CodeUri – AWS Command Line Interface (CLI) によるデプロイ (2/2)

output.yml

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://my-bucket/9a94737fdebabfcd740708aa35fffe43
      Events:
        ApiEvent1:
          Type: Api
          Properties:
            Path: /hello
            Method: GET
```



aws cloudformation deploy --template-file output.yml
--stack-name MyStack --capabilities CAPABILITY_IAM

Events プロパティ

Lambda 関数が実行されるトリガーを Events プロパティ配下に指定する (複数可)
Type プロパティに AWS のどのサービスをイベントソースにするかを指定する

例

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Events:
        GetHello:
          Type: Api
          Properties:
            Path: /hello
            Method: get
(...省略...)
```

Type により、指定可能なプロパティの内容はそれぞれ異なる

(GetHello の箇所は論理 ID なので、
任意の値を定義して指定)

Type の指定	対応する AWS のサービス
S3	Amazon Simple Storage Service (S3)
Api	Amazon API Gateway
DynamoDB	Amazon DynamoDB
SNS	Amazon Simple Notification Service (SNS)
SQS	Amazon Simple Queue Service (SQS)
Kinesis	Amazon Kinesis Data Streams
Schedule	Amazon CloudWatch Events (スケジュール)
CloudWatchEvent	Amazon CloudWatch Events (イベント)
CloudWatchLogs	Amazon CloudWatch Logs
IoTRule	AWS IoT ルール
AlexaSkill	Amazon Alexa スキル

Events – Amazon S3 (Type: S3)

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        BucketEvent1:
          Type: S3
          Properties:
            Bucket: !Ref Bucket1
            Events:
              - 's3:ObjectCreated:*'
              - 's3:ObjectRemoved:*'
            Filter:
              S3Key:
                Rules:
                  - Name: prefix
                    Value: images/
                  - Name: suffix
                    Value: .jpg
  Bucket1:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: my-bucket
```

プロパティ	型	内容	指定
Bucket	String	イベントソースとする S3 のバケット	**必須**
Events	String もしくは String の配列	Lambda をトリガーするイベントの種類。サポートされるイベントについては下記 URL を参照 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/dev/NotificationHowTo.html#notification-how-to-event-types-and-destinations	**必須**
Filter	Amazon S3 Notification Filter	S3Key -> Rules の配下に、Lambda 関数をトリガーする条件を、Name と Value の配列形式で書く。 Name には prefix もしくは suffix を指定できる。prefix を指定した場合、S3 に置かれたファイル名の接頭辞が、suffix を指定した場合、S3 に置かれたファイルの接尾辞を条件にできる。条件の内容は Value に書く	任意

S3 をイベントソースとする場合、テンプレート内でバケットを作成し、参照する必要がある点に注意
(手動で作ったバケットを使うことはできない)

Events – Amazon API Gateway (Type: Api)

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        ApiEvent1:
          Type: Api
          Properties:
            Path: /hello
            Method: get
            RestApiId: !Ref: HelloWorldApi
            Auth:
              Authorizer: MyCognitoAuth
            RequestModel:
              Model: User
              Required: true
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
      Auth:
        DefaultAuthorizer: AWS_IAM
        Authorizers:
          MyCognitoAuth:
            UserPoolArn: arn:aws:cognito-
idp:ap-northeast-1:0123456789012:userpool/ap-
northeast-1_HtXewla1P
```

```yaml
      Models:
        User:
          type: object
          properties:
            username:
              type: string
```

プロパティ	型	内容	指定
Path	String	Lambda 関数が実行される API Gateway のパスを指定	**必須**
Method	String	Lambda 関数が実行される API Gateway のメソッドを指定	**必須**
RestApiId	String	AWS::Serverless:Api リソースタイプと紐付ける場合に利用し、同リソースタイプの論理 ID を指定する (後述)	任意
Auth	Function Auth Object	Authorizer に、このパスに対するリクエストの認可方式 (AWS_IAM 等) を指定する。AWS::Serverless:Api リソースタイプで定義したオーソライザーを利用できる	任意
RequestModel	Function Request Model Object	AWS::Serverless:Api リソースタイプ で定義したモデルを利用する。Model に、このパスに対するリクエストに紐付けるモデルを指定	任意

Path と Method プロパティのみを指定した場合、API Gateway のリソースも併せて作成される

Events – CloudWatch Events スケジュール (Type: Schedule)

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        Schedule1:
          Type: Schedule
          Properties:
            Schedule: "cron(0 1 * * ? *)"
            Input: |
              {
                  "region": "tokyo"
              }
```

プロパティ	型	内容	指定
Schedule	String (Cron / Rate Expression)	Lambda 関数を実行する頻度を指定。cron 式、もしくは rate 式で指定することができる 例： cron(0 12 * * ? *) rate(5 minutes) その他、詳細な書式は下記 URL を参照 https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/events/ScheduledEvents.html	**必須**
Input	String (JSON)	Lambda 関数に対して、定数入力として渡す値を JSON で記載する	任意

Events – CloudWatch Events (Type: CloudWatchEvent)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        CWEvent:
          Type: CloudWatchEvent
          Properties:
            Pattern:
              source:
                - aws.ec2
              detail-type:
                - 'EC2 Instance State-change
Notification'
              detail:
                state:
                  - running
            InputPath: $.detail.instance-id
```

例

プロパティ	型	内容	指定
Pattern	Event Pattern Object	Lambda 関数をトリガーするイベントの条件を指定。左記の例では、EC2 が running の状態になった場合を指定している。指定可能な値は下記を参照：https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/events/CloudWatchEventsandEventPatterns.html	**必須**
Input	String (JSON)	Lambda 関数に対して、定数入力として渡す値を JSON で記載する	任意、ただし InputPath と併用不可
InputPath	String (JSON Path 構文)	CloudWatch Events のイベントの内容から一部を切り出して Lambda 関数に対してデータを渡す場合に利用する。JSON Path と呼ばれる構文を使用する JSON Path についてはこちらを参照：https://github.com/json-path/JsonPath	任意、ただし Input と併用不可

Events – Amazon DynamoDB (Type: DynamoDB)

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        DynamoDbEvent1:
          Type: DynamoDB
          Properties:
            Stream: arn:aws:dynamodb:ap-northeast-1:123456789012:table/TestTable/stream/2019-08-14T00:00:00.000
            StartingPosition: TRIM_HORIZON
            BatchSize: 10
            Enabled: true
```

プロパティ	型	内容	指定
Stream	String	DynamoDB ストリームの ARN	**必須**
StartingPosition	String	TRIM_HORIZON もしくは LATEST を指定。LATEST の場合、DynamoDB ストリームの内部の最新レコードからデータの読み取りを開始する。 TRIM_HORIZON の場合、最も古いレコードからデータの読み取りを開始する	**必須**
BatchSize	Integer	一度の関数実行で処理するレコードの数を指定	任意
Enabled	Boolean	このイベントソースを有効化するかどうかを指定	任意

Events – Kinesis Data Streams (Type: Kinesis)

例

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        KinesisEvent1:
          Type: Kinesis
          Properties:
            Stream: arn:aws:kinesis:ap-northeast-
1:123456789012:stream/my-stream
            StartingPosition: TRIM_HORIZON
            BatchSize: 10
            Enabled: true
```

プロパティ	型	内容	指定
Stream	String	Kinesis Data Streams の ARN	**必須**
StartingPosition	String	TRIM_HORIZON もしくは LATEST を指定。LATEST の場合、シャードの内部の最新レコードからデータの読み取りを開始する。TRIM_HORIZON の場合、最も古いレコードからデータの読み取りを開始する	**必須**
BatchSize	Integer	一度の関数実行で処理するレコードの数を指定	任意
Enabled	Boolean	このイベントソースを有効化するかどうかを指定	任意

Events – Amazon SNS (Type: SNS)

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        Sns:
          Type: SNS
          Properties:
            Topic: arn:aws:sns:ap-northeast-
1:123456789012:mysns
            FilterPolicy:
              store:
                - example_corp
              price_usd:
                - numeric:
                    - ">="
                    - 100
```

プロパティ	型	内容	指定
Topic	String	SNS の ARN を指定する	**必須**
FilterPolicy	Amazon SNS Policy	SNS の サブスクリプション フィルタポリシーを指定する	任意

Events – Amazon SQS (Type: SQS)

例

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        Sqs:
          Type: SQS
          Properties:
            Queue: arn:aws:sqs:ap-northeast-
1:123456789012:my-queue
            BatchSize: 10
            Enabled: true
```

プロパティ	型	内容	指定
Queue	String	SQS の ARN を指定する	**必須**
BatchSize	Integer	一度の実行で Lambda 関数が処理するメッセージの数を指定	任意
Enabled	Boolean	このイベントソースを有効化するかどうかを指定	任意

Events – CloudWatch Logs (Type: CloudWatchLogs)

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        CWLogs:
          Type: CloudWatchLogs
          Properties:
            LogGroupName: /aws/batch/job
            FilterPattern: 'ERROR Exception'
```

プロパティ	型	内容	指定
LogGroupName	String	CloudWatch Logs のロググループ名	**必須**
FilterPattern	String	ログの内容がどのようなパターンに一致したときに Lambda 関数を実行するかを指定する。指定可能な内容については下記を参照：https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/logs/FilterAndPatternSyntax.html	**必須**

Events – AWS IoT ルール (Type: IoTRule)

例

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        IoT:
          Type: IoTRule
          Properties:
            Sql: "SELECT temperature FROM
'iot/topic' WHERE temperature > 50"
            AwsIotSqlVersion: 2016-03-23
```

プロパティ	型	内容	指定
Sql	String	AWS IoT トピックをクエリする SQL を指定する。SQL の文法については下記 URL を参照：https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/iot-sql-reference.html	**必須**
AwsIotSqlVersion	String	利用する AWS IoT ルールエンジンのバージョンを指定	任意

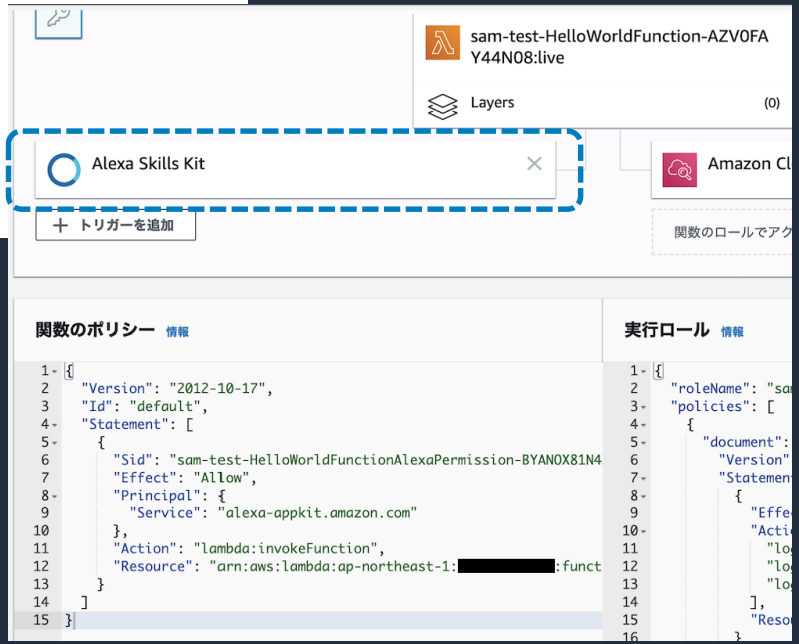
Events – Alexa スキル (Type: AlexaSkill)

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  SamFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: s3://xxx/xxx
      Events:
        Alexa:
          Type: AlexaSkill
```

プロパティは無し

このイベントを設定すると、Alexa から Lambda 関数を呼び出す権限 (関数のポリシー) が付与される



AutoPublishAlias プロパティ

AutoPublishAlias を指定すると、SAM が Lambda 関数の更新をバージョン管理する。Lambda 関数にデプロイする内容が変化すると、新しいバージョンを発行し、設定したエイリアスを最新版に変更する

例

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      AutoPublishAlias: live
(...省略...)
```

実装内容を更新

Version: 1

Version: 2

エイリアス : live へアクセス

新規バージョンを発行し、エイリアスを新しいバージョンに割り当ててくれる

DeploymentPreference プロパティ (1/2)

DeploymentPreference と AutoPublishAlias を指定すると、**AWS CodeDeploy と自動で連携**する最新のコードを一部だけデプロイし、**段階的にデプロイを完了させる**こと（カナリアリリース）が可能

例

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      AutoPublishAlias: live
      DeploymentPreference:
        Type: Linear10PercentEvery10Minutes
```

(…省略…)

Lambda の Alias 機能を利用できる

デプロイが完了すると、最新のバージョンにエイリアスが割り当てられる

CodeDeploy の画面

デプロイ中

デプロイ完了

DeploymentPreference プロパティ (2/2)

DeploymentPreference の Type に指定することができる値は以下のとおり。
指定した内容により、どのように最新バージョンがリリースされるか動作が変わる

デプロイ戦略	説明
Canary10Percent30Minutes	Lambda 関数の実行のうち 10 % は最新のバージョンを実行し、30 分後にすべてが最新バージョンで実行される
Canary10Percent5Minutes	Lambda 関数の実行のうち 10 % は最新のバージョンを実行し、5 分後にすべてが最新バージョンで実行される
Canary10Percent10Minutes	Lambda 関数の実行のうち 10 % は最新のバージョンを実行し、10 分後にすべてが最新バージョンで実行される
Canary10Percent15Minutes	Lambda 関数の実行のうち 10 % は最新のバージョンを実行し、15 分後にすべてが最新バージョンで実行される
AllAtOnce	すべての Lambda 関数の実行を最新バージョンで行う
Linear10PercentEvery10Minutes	Lambda 関数の実行のうち 10 % をまず最新のバージョンで実行し、10 分毎に 10 % ずつ最新のバージョンで実行される範囲を増加していく
Linear10PercentEvery1Minute	Lambda 関数の実行のうち 10 % をまず最新のバージョンで実行し、1 分毎に 10 % ずつ最新のバージョンで実行される範囲を増加していく
Linear10PercentEvery2Minutes	Lambda 関数の実行のうち 10 % をまず最新のバージョンで実行し、2 分毎に 10 % ずつ最新のバージョンで実行される範囲を増加していく
Linear10PervcentEvery3Minutes	Lambda 関数の実行のうち 10 % をまず最新のバージョンで実行し、3 分毎に 10 % ずつ最新のバージョンで実行される範囲を増加していく

Policies プロパティ

よく利用されるポリシーについて**簡単に設定**できるよう、あらかじめ用意されているポリシーがある

変換前

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Policies:
        - DynamoDBCrudPolicy:
            TableName: mytable
(...省略...)
```

DynamoDBCrudPolicy が変換される

変換後

```
- PolicyName: HelloWorldFunctionRolePolicy0
  PolicyDocument:
    Statement:
      - Action:
          - "dynamodb:GetItem"
          - "dynamodb:DeleteItem"
          - "dynamodb:PutItem"
          - "dynamodb:Scan"
          - "dynamodb:Query"
          - "dynamodb:UpdateItem"
          - "dynamodb:BatchWriteItem"
          - "dynamodb:BatchGetItem"
          - "dynamodb:DescribeTable"
        Resource:
          - "Fn::Sub":
              -
"arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:
table/${tableName}"
                - tableName: mytable
          - "Fn::Sub":
              -
"arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:
table/${tableName}/index/*"
                - tableName: mytable
        Effect: Allow
```

(SAM は Lambda 関数に必要な IAM ポリシーを自動でアサインするが、SAM 定義外の AWS サービスにアクセスしたい場合、自身でポリシーを定義する)

指定可能なポリシー名については下記を参照：

https://github.com/awslabs/serverless-application-model/blob/0f1a0823546acc7369a9059de302bcac4267d2eb/examples/2016-10-31/policy_templates/all_policy_templates.yaml

AWS::Serverless::Function プロパティ一覧 (1/4)

プロパティ	型	内容	指定	書き方の例
Handler	String	Lambda 関数のハンドラを指定	**必須**	`Handler: index.handler`
Runtime	String	Lambda 関数を動作させるランタイムを指定。指定可能な値については AWS の公式ドキュメントを参照：https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/lambda-runtimes.html	**必須**	`Runtime: nodejs8.10`

CodeUri	String もしくは S3 Location Object	Lambda 関数にデプロイするコードの場所を指定。Bucket (String)、Key (String)、Version (Integer) で指定する方法と、S3 の URI (s3://xxx-bucket/xxx.zip) を直に指定する方法がある	**CodeUri か InlineCode が必須**	`CodeUri:`
				`  Bucket: my-bucket`
				`  Key: assets/app.zip`

InlineCode	String	Lambda 関数にデプロイするコードを直接記載	**CodeUri か InlineCode が必須**	`InlineCode: |`
				`  exports.handler = async (event, context) => {`
				`  return event`
				`  }`

FunctionName	String	Lambda 関数の名前を指定。指定しない場合は CloudFormation が自動で生成	任意	`FunctionName: MyFunction`
Description	String	Lambda 関数の説明を指定	任意	`Description: 'Sample App'`
MemorySize	Integer	Lambda 関数で利用するメモリの量 (MB 単位) を指定。未指定の場合は 128 MB になる	任意	`MemorySize: 256`

AWS::Serverless::Function プロパティ一覧 (2/4)

プロパティ	型	内容	指定	書き方の例
Timeout	Integer	Lambda 関数が実行時にタイムアウトする秒数を指定。指定しない場合は 3 秒になる	任意	`Timeout: 180`
Role	String	Lambda 関数の実行ロールを ARN で指定。未指定の場合は、SAM が自動的にデフォルトのロールを紐付ける	任意	`Role: arn:aws:iam::0123456789012:role/LambdaRole`

Environment	Function environment object	Variables に環境変数のキーと値を指定	任意	`Environment:`
				`  Variables:`
				`  MyEnv: Prod`

VpcConfig	Vpc config object	SubnetIds に Lambda 関数がアクセスするサブネットの ID を指定し、SecurityGroupIds に Lambda 関数に適用するセキュリティグループを指定	任意	`VpcConfig:`
				`  SubnetIds:`
				`  - subnet-0757a65240a1b7755`
				`  SecurityGroupIds:`
				`  - sg-0309cb8853632edf2`

Tags	Map (String to String)	Lambda 関数に割り当てるタグを、キーと値の組み合わせで指定	任意	`Tags:` `  Env: Prod`
Tracing	String	Active もしくは PassThrough を指定。Active が指定された場合、X-Ray によるアクティブトレースが有効になる	任意	`Tracing: Active`

AWS::Serverless::Function プロパティ一覧 (3/4)

プロパティ	型	内容	指定	書き方の例
AutoPublishAlias	String	新しいソースコードで Lambda 関数を更新した際に、最新バージョンに付与するエイリアスを定義する	任意	`AutoPublishAlias: live`

DeploymentPreference	Deployment Preference Object	AutoPublishAlias と組み合わせて使う。指定すると CodeDeploy を利用した Lambda のカナリアリリースを行う。その他、指定可能なプロパティの詳細は下記 URL を参照：https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#deploymentpreference-object	任意	`DeploymentPreference:`
				`  Enabled: true`
				`  Type: Linear10PercentEvery10Minutes`

DeadLetterQueue	Dead Letter Queue Object	Lambda 関数実行失敗時の情報を格納するデッドレターキューを指定する。
		Type に SQS もしくは SNS を指定。TargetArn に SQS もしくは SNS の ARN を指定
			任意	`DeadLetterQueue:`
				`  Type: SNS`
				`  TargetArn: arn:aws:sns:ap-northeast-1:0123456789012:mytopic`

Layers	List (String)	Lambda 関数に適用する Lambda レイヤーを ARN で指定	任意	`Layers:` `  - arn:aws:lambda:ap-northeast-1:0123456789012:layer:MyLayer:2`
VersionDescription	String	AutoPublishAlias と組み合わせて使う。デプロイするバージョンの説明	任意	`VersionDescription: 'New Version'`

AWS::Serverless::Function プロパティ一覧 (4/4)

プロパティ	型	内容	指定	書き方の例

Policies	List (SAM Policy Templates) 等	Lambda 関数に追加で設定する IAM ポリシーを指定する	任意	`Policies:`
				`  - DynamoDBCrudPolicy:`
				`  TableName: mytable`
				`  - EC2DescribePolicy: {}`

PermissionsBoundary	String	Lambda 関数のロールに適用する Permissions boundary を ARN で指定する	任意	`PermissionsBoundary: arn:aws:iam::123456789012:policy/mypolicy`

Events	Map (String to Event Source Object)	Lambda 関数をトリガーするイベントを定義する	任意	`Events:`
				`  GetResource:`
				`  Type: Api`
				`  Properties:`
				`  Path: /hello`
				`  Method: get`

ReservedConcurrentExecutions	Integer	Lambda 関数の同時実行数を指定	任意	`ReservedConcurrentExecutions: 100`
KmsKeyArn	String	Lambda 関数で利用する環境変数について、暗号化を行う KMS のキーを ARN で指定	任意	`KmsKeyArn: arn:aws:kms:ap-northeast-1:0123456789012:key/28dbd6f7-53bc-45fe-8aff-8c8ffcc506f2`

SAM テンプレートのリソースタイプ – 5 種類

- AWS::Serverless::Function
- AWS::Serverless::Api
- AWS::Serverless::SimpleTable
- AWS::Serverless::LayerVersion
- AWS::Serverless::Application

aws

AWS::Serverless::Api リソースタイプ

API Gateway をデプロイする。AWS::Serverless::Function の **API イベントタイプと組み合わせて使う**場合が多く、**より詳細な API Gateway の設定を行いたい場合に利用**する

例

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://my-bucket/246fce1ec8c05c948655dace8bb1c9bc
      Handler: index.handler
      Runtime: nodejs8.10
      Events:
        ApiEvent1:
          Type: Api
          Properties:
            RestApiId: !Ref HelloWorldApi
            Path: /hello
            Method: get
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
(...省略...)
```

API イベントタイプの RestApiId プロパティで、AWS::Serverless::Api タイプのリソースの論理 ID (この場合は HelloWorldApi / 任意に指定可能) を指定

aws

AWS::Serverless::Api で紹介するプロパティ

- AWS::Serverless::Api
  - StageName プロパティ
  - CanarySetting プロパティ
  - DefinitionUri, DefinitionBody プロパティ
  - Auth プロパティ

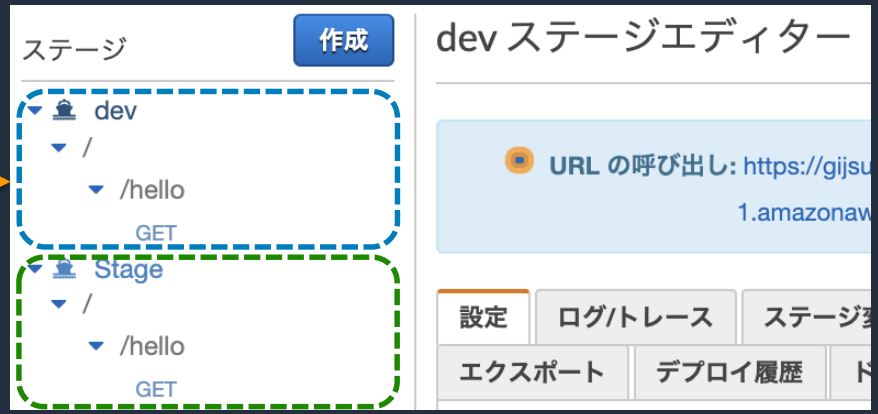
StageName プロパティ

API Gateway の API のデプロイ先となる**ステージ名は必須、かつ１つのみ指定**ができる
そのため、dev、test、prod といった**複数ステージをデプロイすることはできない**

例

```
Resources:
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
      DefinitionUri:
        Bucket: my-bucket
        Key: sam-swagger.yml

(...省略...)
```



別途、"Stage" という名称のステージも作られる。これは、次回のデプロイ時に、SAM テンプレートで API 定義が更新された際の互換性担保の役割で自動生成される

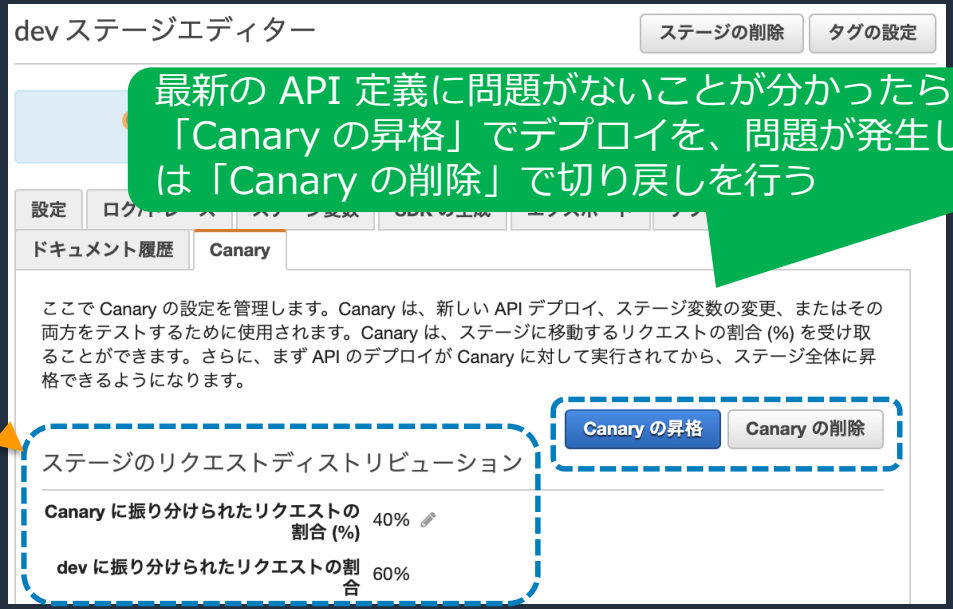
CanarySetting プロパティ

CanarySetting プロパティを利用すると、**API 定義をアップデートした際に、一部のトラフィックのみを最新版 (Canary) に流す**ことができる

例

```
Resources:
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
      CanarySetting:
        PercentTraffic: 40
      DefinitionUri:
        Bucket: my-bucket
        Key: sam-swagger.yml

(...省略...)
```



最新の API 定義に問題がないことが分かったら「Canary の昇格」でデプロイを、問題が発生した場合は「Canary の削除」で切り戻しを行う

PercentTraffic プロパティに、最新版の API に流すトラフィックの比率を指定 (この場合は 40 %)

DefinitionUri, DefinitionBody プロパティ

API Gateway の **API 定義を独自に設定する場合**、DefinitionUri もしくは DefinitionBody プロパティに指定する。DefinitionUri か DefinitionBody いずれかのみ指定可能

DefinitionUri の場合：

例

```
Resources:
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
      DefinitionUri: s3://my-bucket/sam-swagger.yml

(...省略...)
```

S3 に事前にアップロードした API 定義を指定する

Definition Body の場合：

例

```
Resources:
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
      DefinitionBody:
        swagger: "2.0"
        paths:
          /hello:
            get:
              x-amazon-apigateway-integration:
                uri: "arn:aws:apigateway:ap-northeast-1:lambda:path/2015-03-31/functions/arn:aws:lambda:ap-northeast-1:0123456789012:function:helloworld:live/invocations"
                httpMethod: "POST"
                type: "aws_proxy"

(...省略...)
```

テンプレートの中に直接 API 定義を書く

Auth プロパティ (1/2)

API Gateway のオーソライザーを作成し認可方式が設定できる
Auth プロパティを使う場合、DefinitionUri による API 定義は使えないことに注意

例

```
Resources:
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
      DefinitionBody:
        swagger: "2.0"
        paths:
          /hello:
            get:
              x-amazon-apigateway-integration:
                uri: "arn:aws:apigateway:ap-northeast-
1:lambda:path/2015-03-31/functions/arn:aws:lambda:ap-northeast-
1:0123456789012:function:HelloWorld/invocations"
                httpMethod: "POST"
                type: "aws_proxy"
      Auth:
        DefaultAuthorizer: AWS_IAM

(...省略...)
```

利用するオーソライザーの方式を指定する

IAM を利用する場合、AWS_IAM を指定する

Auth プロパティ (2/2)

Amazon Cognito のユーザープールを認可に利用する場合の設定は以下の通り

例

```
Resources:
  HelloWorldApi:
    Type: 'AWS::Serverless::Api'
    Properties:
      StageName: dev
      DefinitionBody:
        swagger: "2.0"
        paths:
          /hello:
            get:
              x-amazon-apigateway-integration:
                uri: "arn:aws:apigateway:ap-northeast-
1:lambda:path/2015-03-31/functions/arn:aws:lambda:ap-northeast-
1:0123456789012:function:HelloWorld/invocations"
                httpMethod: "POST"
                type: "aws_proxy"
      Auth:
        DefaultAuthorizer: MyCognitoAuth
        Authorizers:
          MyCognitoAuth:
            UserPoolArn: arn:aws:cognito-idp:ap-northeast-
1:0123456789012:userpool/ap-northeast-1_HtXewla1P
(...省略...)
```

UserPoolArn に Cognito ユーザープールの ARN を指定し、DefaultAuthorizer にオーソライザーの論理 ID (MyCognitoAuth / 自身で定義した任意の値) を指定する

AWS::Serverless::Api プロパティ一覧 (1/4)

プロパティ	型	内容	指定	書き方の例
Name	String	API Gateway の名称を指定	任意	`Name: MyApi`
StageName	String	API Gateway のステージ名を指定	**必須**	`StageName: prod`

DefinitionUri	String もしくは S3 Location Object	Swagger ドキュメントが配置されている場所を指定。Bucket (String)、Key (String)、Version (Number) で指定する方法と、S3 の URI (s3://xxx/xxx) を直に指定する方法がある	任意 Definition Body との併用不可	`DefinitionUri:`
				`  Bucket: my-bucket`
				`  Key: swagger.yml`

DefinitionBody	JSON もしくは YAML	Swagger ドキュメントを直接、JSON もしくは YAML 形式で指定	任意 Definition Uri との併用不可	`DefinitionBody:`
				`  swagger: "2.0"`
				`  paths:`
				`  /hello:`
				`  get:`
				`  x-amazon-apigateway-integration:`
				`  uri: "(省略)"`
				`  httpMethod: "POST"`
				`  type: "aws_proxy"`

EndpointConfiguration	String	API Gateway のエンドポイント種別を指定。REGION、EDGE、もしくは PRIVATE のうちいずれかの値を指定。デフォルトは EDGE (エッジ最適化) になる	任意	`EndpointConfiguration: REGION`
TracingEnabled	Boolean	X-Ray によるトレースを有効化するかどうかを指定	任意	`TracingEnabled: true`

AWS::Serverless::Api プロパティ一覧 (2/4)

プロパティ	型	内容	指定	書き方の例
CacheClusterEnabled	Boolean	API キャッシュを有効にするかどうかを指定	任意	`CacheClusterEnabled: true`
CacheClusterSize	String	API キャッシュで利用する、キャッシュのサイズを指定。0.5、1.6、6.1、13.5、28.4、58.2、118、237 のうちいずれかを指定	CacheClusterEnabled が指定されている場合は必須	`CacheClusterSize: '6.1'`
Variables	Map (String to String)	ステージ変数のキーと値を指定	任意	`Variables:` `  Env: Prod`

MethodSettings	CloudFormation Method Settings Property	API Gateway のメソッドごとに、ログレベル等の詳細設定を行う。CloudFormation で設定可能な内容と同値のため、下記を参照 https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/aws-properties-apigateway-deployment-stagedescription-methodsetting.html	任意	`MethodSettings:`
				`  - DataTraceEnabled: true`
				`  LoggingLevel: 'ERROR'`
				`  ResourcePath: '/*'`
				`  HttpMethod: '*'`

BinaryMediaTypes	List (String)	API Gateway が取り扱うことのできるバイナリ形式を MIME タイプで指定する。ただし、"/" は "~1" に変更する必要がある点に注意 (例：image~1gif)	任意	`BinaryMediaTypes:` `  - image~1gif`

AWS::Serverless::Api プロパティ一覧 (3/4)

プロパティ	型	内容	指定	書き方の例
MinimumCompressionSize	Integer	レスポンス ボディの圧縮をトリガーするしきい値を、バイト単位で指定	任意	`MinimumCompressionSize: true`
Cors	String もしくは Cors Configuration	String として指定する場合、クロスドメインアクセスを許可するドメインを指定。その場合、"'example.com'"のように、ダブルクォートとシングルクォートの両方で囲む必要がある。CorsConfiguration についてはこちらを参照：https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#cors-configuration	任意 DefinitionUri が指定されている場合、利用不可	`Cors: "'example.com'"`

Auth	API Auth Object	オーソライザーを定義する。また、デフォルトで利用するオーソライザーを設定する https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#api-auth-object	任意 DefinitionUri が指定されている場合使用不可	`Auth:`
				`  DefaultAuthorizer: AWS_IAM`
				`  Authorizers:`
				`  MyCognitoAuth:`
				`  UserPoolArn:`
				`arn:aws:cognito-idp:ap-northeast-1:0123456789012:userpool/ap-northeast-1_HtXewla1P`

AccessLogSetting	CloudFormation AccessLogSetting Property	API Gateway のアクセスログの記録を行う場合に指定する。DestinationArn に CloudWatch Logs のロググループの ARN を指定し、Format に $contexts 変数を入れる。https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/api-gateway-mapping-template-reference.html#context-variable-reference	任意	`AccessLogSetting:`
				`  DestinationArn:`
				`arn:aws:logs:ap-northeast-1:0123456789012:log-group:MyLog`
				`  Format: "{'request_id': '$context.requestId'}"`

AWS::Serverless::Api プロパティ一覧 (4/4)

プロパティ	型	内容	指定	書き方の例
CanarySetting	CloudFormation CanarySetting Property	API Gateway のカナリアデプロイに関する設定を行う。CloudFormation の同名プロパティと同値であるため、下記を参照：https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/aws-properties-apigateway-stage-canarysetting.html	任意	`CanarySetting:` `  PercentTraffic: 40`

Models	JSON もしくは YAML 形式	API Gateway で使うモデルの内容を定義する	任意	`Models:`
				`  User:`
				`  type: object`
				`  properties:`
				`  username:`
				`  type: string`

GatewayResponses	Gateway Response Object	Unauthorized や Bad Request 等の際に、API Gateway から返すレスポンスの詳細を定義する。詳細はこちらの URL も参照：https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#gateway-response-object	任意	`GatewayResponses:`
				`  UNAUTHORIZED:`
				`  StatusCode: 401`

SAM テンプレートのリソースタイプ – 5 種類

- AWS::Serverless::Function
- AWS::Serverless::Api
- AWS::Serverless::SimpleTable
- AWS::Serverless::LayerVersion
- AWS::Serverless::Application

リソース タイプ – AWS::Serverless::SimpleTable

DynamoDB のテーブルをデプロイするリソースタイプ。CloudFormation の AWS::DynamoDB::Table タイプと同等だが、より簡潔に書くことができる

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  DdbTable:
    Type: 'AWS::Serverless::SimpleTable'
    Properties:
      TableName: my-table
      PrimaryKey:
        Name: id
        Type: String
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
      Tags:
        Department: Engineering
        AppType: Serverless
      SSESpecification:
        SSEEnabled: true
```

プロパティ一覧

プロパティ	型	内容	指定
TableName	String	DynamoDB のテーブル名	任意
PrimaryKey	Primary Key Object	Name (String) にパーティションキーの名前、Type (String) にパーティションキーの型を指定。指定しないと Id という String 型のパーティションキーになる	任意
Provisioned Throughput	Provisioned Throughput Object	ReadCapacityUnits (Number) に読み取りキャパシティユニットを、WriteCapacityUnits に書き込みキャパシティユニットの数を指定。指定しないとオンデマンドキャパシティになる	任意
Tags	Map (String to String)	DynamoDB のテーブルに割り当てるタグを、キーと値の形で指定	任意
SSESpecification	DynamoDB SSESpecification	SSEEnabled (Boolean) にサーバー側の暗号化を行うかどうかを指定	任意

SAM テンプレートのリソースタイプ – 5 種類

- AWS::Serverless::Function
- AWS::Serverless::Api
- AWS::Serverless::SimpleTable
- AWS::Serverless::LayerVersion
- AWS::Serverless::Application

リソース タイプ – AWS::Serverless::LayerVersion

Lambda レイヤーをデプロイするリソースタイプ

例

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  LambdaLayer:
    Type: 'AWS::Serverless::LayerVersion'
    Properties:
      LayerName: MyLayer
      Description: Layer description
      ContentUri:
        Bucket: xxx-bucket
        Key: xxx.zip
        Version: 1
      CompatibleRuntimes:
        - nodejs8.10
        - nodejs10.x
      LicenseInfo: 'MIT'
      RetentionPolicy: Retain
```

プロパティ一覧

プロパティ	型	内容	指定
LayerName	String	Lambda レイヤーの名称	任意
Description	String	Lambda レイヤーの説明	任意
ContentUri	String もしくは S3 Location Object	Lambda レイヤーとしてデプロイするプログラムが配置されている S3 の場所を指定。Bucket (String)、Key (String)、Version (Number) で指定する方法と、S3 の URI (s3://xxx-bucket/xxx.zip) を直に指定する方法がある。	**必須**
Compatible Runtimes	List (String)	Lambda レイヤーが互換性のある Lambda のランタイムの種類。複数指定可能	任意
LicenseInfo	String	Lambda レイヤーのライセンス情報	任意
RetentionPolicy	String	Retain もしくは Delete を指定。Delete を指定すると、ひとつ前の同じ Lambda レイヤーのデプロイバージョンを削除する。Retain の場合は削除せず、古いバージョンを残す	任意

ContentUri には、AWS::Serverless::Function と同様に、ローカルのパス (./src 等) を記載し、aws cloudformation package コマンドでパッケージングすることも可能。

SAM テンプレートのリソースタイプ – 5 種類

- AWS::Serverless::Function
- AWS::Serverless::Api
- AWS::Serverless::SimpleTable
- AWS::Serverless::LayerVersion
- AWS::Serverless::Application

リソース タイプ – AWS::Serverless::Application

Serverless Application Repository や特定の SAM テンプレートから、アプリケーションをデプロイする。実際のアプリケーションのデプロイは Nested Stack の中で行われる

例

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  MyApplication:
    Type: AWS::Serverless::Application
    Properties:
      Location:
        ApplicationId:
'arn:aws:serverlessrepo:us-east-
1:0123456789012:applications/xxx'
        SemanticVersion: 0.0.1
      NotificationARNs:
        - 'arn:aws:sns:us-east-
1:0123456789012:xxx'
      Tags:
        Version: 0.0.1
      TimeoutInMinutes: 60
      Parameters:
        TableName: MyTable
```

プロパティ一覧

プロパティ	型	内容	指定
Location	String もしくは Application Location Object	String の場合は SAM のテンプレート ファイルの置き場所を指定 (https://my-bucket.s3-ap-northeast-1.amazonaws.com/template.yaml 等) Application Location Object の場合、Serverless Application Repository の ARN とバージョンを指定する	**必須**
Notification ARNs	List (String)	Nested Stack に関するイベントが発生した場合に通知する SNS トピックを指定	任意
Tags	Map (String to String)	Nested Stack に付与するタグのキーと値を指定	任意
TimeoutInMinutes	Number	Nested Stack のデプロイを、タイムアウトで失敗扱いにするまでの時間を分単位で指定	任意
Parameters	Map (String to String)	Serverless Application Repository のアプリケーションや SAM テンプレートにわたすパラメータを指定	任意

SAM テンプレート – グローバル セクション

各リソースタイプに適用するプロパティのデフォルト値を定義することができる

```yaml
Globals:
  Function:
    Timeout: 180
  Api:
    BinaryMediaTypes:
      - image~1gif
  SimpleTable:
    SSESpecification:
      SSEEnabled: true
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: app.lambdaHandler
      Runtime: nodejs10.x
(...省略...)
```

AWS::Serverless::Function
AWS::Serverless::Api
AWS::Serverless::SimpleTable
の 3 種類のリソースが対応。

明示的にプロパティを指定しない場合のデフォルト値を決めることができる

非対応のプロパティなどの詳細は以下の URL を参照：

https://github.com/awslabs/serverless-application-model/blob/master/docs/globals.rst

アジェンダ

- AWS SAM とは
- AWS SAM の機能と文法
- AWS SAM Command Line Interface
- その他考慮事項

AWS SAM Command Line Interface (CLI)

- 開発者のローカル環境で Lambda や API のエンドポイントを起動し、実行をテストすることができる
  - Lambda をローカル実行するにあたり Docker コンテナを利用するため、Docker のインストールが事前に必要
- SAM を利用したサーバーレス アプリケーションの雛形を作ることができる
- SAM テンプレートの文法が正しいか検証することが可能
- アプリケーションのビルド、パッケージング、デプロイを行うことができる
- 現在はベータ版としての提供

雛形の生成 (sam init)

```
playground ) sam init --runtime nodejs
[+] Initializing project structure...

Project generated: ./sam-app

Steps you can take next within the project folder
===================================================
[*] Invoke Function: sam local invoke HelloWorldFunction --event event.json
[*] Start API Gateway locally: sam local start-api

Read sam-app/README.md for further instructions

[*] Project initialization is now complete
playground ) tree -C sam-app
sam-app
├── README.md
├── event.json
├── hello-world
│   ├── app.js    ・・・ Lambda 関数
│   ├── package.json
│   └── tests
│       └── unit
│           └── test-handler.js  ・・・ 単体テストコード
└── template.yaml ・・・ SAM テンプレート

3 directories, 6 files
```

sam init コマンドで雛形を作成

--runtime オプションに実装する言語を指定できる

実行すると、Lambda へデプロイするコードや、単体テストのコード、さらに関数をデプロイするための SAM テンプレートなどが生成される

SAM テンプレートの検証 (sam validate)

```
sam-app ) sam validate --template template.yaml
2019-07-30 18:17:23 Found credentials in shared credentials
file: ~/.aws/credentials
[redacted]/playground/sam-app/templat
e.yaml is a valid SAM Template
```

sam validate コマンドで SAM テンプレートが正しく CloudFormation の文法に変換できるか検証できる

```
sam-app ) sam validate --template template.yaml
2019-07-30 18:50:07 Found credentials in shared credentials
file: ~/.aws/credentials
Template provided at '[redacted]/play
ground/sam-app/template.yaml' was invalid SAM Template.
Error: [InvalidResourceException('HelloWorldFunction', 'prop
erty Hoge not defined for resource of type AWS::Serverless::
Function')] ('HelloWorldFunction', 'property Hoge not define
d for resource of type AWS::Serverless::Function')
```

SAM テンプレートの変換に失敗した場合、エラーが出力される

この場合、SAM テンプレートとして定義されていないプロパティ 'Hoge' が含まれているという内容のエラーとわかる

```
13 Resources:
14   HelloWorldFunction:
15     Type: AWS::Serverless::Function
   ons/2016-10-31.md#awsserverlessfunct
16     Properties:
17       Handler: app.lambdaHandler
18       Runtime: nodejs8.10
19       Hoge: fuga
```

エラー原因を発見

Lambda 関数のビルド (sam build)

```
sam-app ) sam build --template template.yaml
2019-07-30 20:22:20 Building resource 'HelloWorldFunction'
2019-07-30 20:22:20 Running NodejsNpmBuilder:NpmPack
2019-07-30 20:22:22 Running NodejsNpmBuilder:CopyNpmrc
2019-07-30 20:22:22 Running NodejsNpmBuilder:CopySource
2019-07-30 20:22:22 Running NodejsNpmBuilder:NpmInstall
2019-07-30 20:22:23 Running NodejsNpmBuilder:CleanUpNpmrc

Build Succeeded

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

Commands you can use next
=========================
[*] Invoke Function: sam local invoke
[*] Package: sam package --s3-bucket <yourbucket>
```

sam build コマンドでアプリケーションをビルドできる。ビルドされたアプリケーションは .aws-sam/build ディレクトリ内に作成される

--use-container オプションをつけた場合、Lambda の動作環境と同等のコンテナ内でビルドすることができるため、ネイティブ モジュールのビルドに役立つ

<u>以下のランタイムに対応 (2019/08/14 時点)</u>

- Python 2.7, 3.6, 3.7 (ビルドに PIP を使用)
- Node.js 10.x, 8.10, 6.10 (ビルドに NPM を使用)
- Ruby 2.5 (ビルドに Bundler を使用)
- Java 8 (ビルドに Gradle を使用)
- .NET Core 2.0, 2.1 (ビルドに Dotnet CLI を使用)

Lambda 関数のローカル実行 (sam local invoke)

```
sam-app ) sam local invoke --event event.json
2019-07-30 19:48:39 Found credentials in shared credentials
file: ~/.aws/credentials
2019-07-30 19:48:39 Invoking app.lambdaHandler (nodejs8.10)

Fetching lambci/lambda:nodejs8.10 Docker container image....
..
2019-07-30 19:48:44 Mounting [redacted]
[redacted]/playground/sam-app/.aws-sam/build/HelloWorldFunction as /
var/task:ro,delegated inside runtime container
START RequestId: 6ecd6a37-c7c6-12f4-ae8a-ba3b02778a06 Versio
n: $LATEST
END RequestId: 6ecd6a37-c7c6-12f4-ae8a-ba3b02778a06
REPORT RequestId: 6ecd6a37-c7c6-12f4-ae8a-ba3b02778a06  Dura
tion: 8.66 ms   Billed Duration: 100 ms Memory Size: 128 MBM
ax Memory Used: 30 MB

{"statusCode":200,"body":"{\"message\":\"hello world\"}"}
```

sam local invoke コマンドで、Lambda ランタイムが動作する Docker コンテナの内部で、ビルドした Lambda 関数を動かすことができる

--event オプションに、Lambda 関数に入力として渡す JSON ファイル名を指定する

```
13 Resources:
14   HelloWorldFunction:
15     Type: AWS::Serverless::Function
   ons/2016-10-31.md#awsserverlessfunct
16     Properties:
17       CodeUri: hello-world/
18       Handler: app.lambdaHandler
19       Runtime: nodejs8.10
```

ローカル実行時にどのランタイムで動かすか、また Lambda 関数のハンドラについては、ビルド時のテンプレートに指定する

テスト用イベントの生成 (sam local generate-event)

```
sam-app ) sam local generate-event s3 put --bucket my-bucket
 --key uploaded.json
{
  "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": "EXAMPLE123/5678abcdefghijklambdaisawe
some/mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "my-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::my-bucket"
        },
        "object": {
          "key": "uploaded.json",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
```

sam local generate-event コマンドで、Lambda 関数の入力に使うテスト用 JSON データを、サービスごとに出力できる。 左記は S3 への PUT イベントの例となるが、どのバケットやファイルで発生したかなどが CLI のオプションとして指定できる

以下のサービスに対応 (2019/08/14 時点)

- alexa-skills-kit
- alexa-smart-home
- apigateway
- batch
- cloudformation
- cloudfront
- cloudwatch
- codecommit
- codepipeline
- cognito
- config
- dynamodb
- kinesis
- lex
- rekognition
- s3
- ses
- sns
- sqs
- stepfunctions

ローカルエンドポイントの作成 (sam local start-lambda)

```
sam-app ) sam local start-lambda
2019-07-31 20:12:52 Found credentials in shared credentials f
ile: ~/.aws/credentials
2019-07-31 20:12:52 Starting the Local Lambda Service. You ca
n now invoke your Lambda Functions defined in your template t
hrough the endpoint.
2019-07-31 20:12:52  * Running on http://127.0.0.1:3001/ (Pre
ss CTRL+C to quit)
2019-07-31 20:12:57 Invoking app.lambdaHandler (nodejs8.10)

Fetching lambci/lambda:nodejs8.10 Docker container image.....
.
2019-07-31 20:13:00 Mounting [redacted]
/playground/sam-app/.aws-sam/build/HelloWorldFunction as /va
r/task:ro,delegated inside runtime container
START RequestId: 68996023-f272-19dc-a716-484c314813c3 Version
: $LATEST
END RequestId: 68996023-f272-19dc-a716-484c314813c3
REPORT RequestId: 68996023-f272-19dc-a716-484c314813c3  Durat
ion: 11.23 ms   Billed Duration: 100 ms Memory Size: 128 MB M
ax Memory Used: 30 MB
2019-07-31 20:13:02 127.0.0.1 - - [31/Jul/2019 20:13:02] "POS
T /2015-03-31/functions/HelloWorldFunction/invocations HTTP/1
.1" 200 -
```

sam local start-lambda コマンドで、AWS CLI や SDK など外部から、ローカルの Lambda 関数を実行するエンドポイントを立ち上げることができる

ユースケースとしては、外部ツールと連携してテストを行いたい場合などに利用する

```
sam-app ) aws lambda invoke --function-name HelloWorldFunctio
n --endpoint-url http://127.0.0.1:3001 /dev/stdout
{"statusCode":200,"body":"{\"message\":\"hello world\"}"}{
    "StatusCode": 200
}
```

CLI や SDK などから Lambda 関数を実行する際、エンドポイントを指定するオプションに対して、払い出された URL を指定する

aws

API エンドポイントの作成 (sam local start-api)

```
sam-app ) sam local start-api
2019-07-31 21:37:46 Found credentials in shared credentials f
ile: ~/.aws/credentials
2019-07-31 21:37:46 Mounting HelloWorldFunction at http://127
.0.0.1:3000/hello [GET]
2019-07-31 21:37:46 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 refl
ected instantly/automatically. You only need to restart SAM C
LI if you update your AWS SAM template
2019-07-31 21:37:46  * Running on http://127.0.0.1:3000/ (Pre
ss CTRL+C to quit)
2019-07-31 21:37:49 Invoking app.lambdaHandler (nodejs8.10)

Fetching lambci/lambda:nodejs8.10 Docker container image.....
.
2019-07-31 21:37:51 Mounting [redacted]
/playground/sam-app/.aws-sam/build/HelloWorldFunction as /va
r/task:ro,delegated inside runtime container
START RequestId: d1758460-e467-1c86-7904-f4b68bc9b84f Version
: $LATEST
END RequestId: d1758460-e467-1c86-7904-f4b68bc9b84f
REPORT RequestId: d1758460-e467-1c86-7904-f4b68bc9b84f  Durat
ion: 7.90 ms    Billed Duration: 100 ms Memory Size: 128 MB M
ax Memory Used: 30 MB
2019-07-31 21:37:53 No Content-Type given. Defaulting to 'app
lication/json'.
2019-07-31 21:37:53 127.0.0.1 - - [31/Jul/2019 21:37:53] "GET
 /hello HTTP/1.1" 200 -
```

```
sam-app ) curl http://127.0.0.1:3000/hello
{"message":"hello world"}
```

sam local start-api コマンドで、API Gateway をイベントソースにして起動する Lambda 関数を試すことができる

SAM テンプレートで定義している内容をもとに、関数へアクセスするパスが生成される

```
12 Resources:
13   HelloWorldFunction:
14     Type: AWS::Serverless::Function
   om/awslabs/serverless-application-mo
   unction
15     Properties:
16       CodeUri: hello-world/
17       Handler: app.lambdaHandler
18       Runtime: nodejs8.10
19       Events:
20         HelloWorld:
21           Type: Api # More info abou
   less-application-model/blob/master/v
22           Properties:
23             Path: /hello
24             Method: get
```

GET /hello というパスが生成 される

パッケージングとデプロイ (sam package / sam deploy)

```
sam-app ) sam package --s3-bucket [redacted]bucket --output-temp
late-file output.yml
Uploading to 81ad9dd6bdd8d111b196ff22d421dbaf  7291 / 7291.0
  (100.00%)
Successfully packaged artifacts and wrote output template to
 file output.yml.
Execute the following command to deploy the packaged templat
e
aws cloudformation deploy --template-file [redacted]
[redacted]/playground/sam-app/output.yml --stack-name <
YOUR STACK NAME>
```

sam package コマンドでアプリケーションを ZIP 形式にパッケージング。S3 にアプリケーションをアップロードする

(aws cloudformation package コマンドと同様の動作)

```
sam-app ) sam deploy --template-file output.yml --stack-name
 MyStack --capabilities CAPABILITY_IAM

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - MyStack
```

sam deploy コマンドで CloudFormation へデプロイ
(aws cloudformation deploy コマンドと同様の動作)

デプロイした関数のログ確認 (sam logs)

```
sam-app ) sam logs --name HelloWorldFunction --stack-name MyS
tack --tail
2019-07-31 20:31:26 Found credentials in shared credentials f
ile: ~/.aws/credentials
2019/07/31/[$LATEST]23020fb3337a457b80079554d281ab3c 2019-07-
31T11:31:18.727000 START RequestId: 9c16e253-f515-40f3-bab6-4
7296d7c964d Version: $LATEST
2019/07/31/[$LATEST]23020fb3337a457b80079554d281ab3c 2019-07-
31T11:31:18.741000 END RequestId: 9c16e253-f515-40f3-bab6-472
96d7c964d
2019/07/31/[$LATEST]23020fb3337a457b80079554d281ab3c 2019-07-
31T11:31:18.742000 REPORT RequestId: 9c16e253-f515-40f3-bab6-
47296d7c964d    Duration: 14.63 ms      Billed Duration: 100
ms      Memory Size: 128 MB     Max Memory Used: 56 MB
```

```
13 Resources:
14   HelloWorldFunction:
15     Type: AWS::Serverless::Function
   om/awslabs/serverless-application-mo
   unction
16     Properties:
17       CodeUri: hello-world/
18       Handler: app.lambdaHandler
19       Runtime: nodejs8.10
```

sam logs コマンドを使うと AWS にデプロイした Lambda 関数のログを出力することができる

--stack-name オプションに、デプロイした CloudFormation のスタック名を指定

--name オプションの指定には SAM テンプレートの中で定義した Lambda 関数の論理 ID を指定する

完成したアプリケーションの保管 (sam publish) (1/2)

```
10 Metadata:
11   AWS::ServerlessRepo::Application:
12     Author: Yuta Imamura
13     Description: HelloWorld
14     Name: my-app
15 Resources:
16   HelloWorldFunction:
17     Properties:
18       CodeUri: s3://[redacted]/74f28a4910b914710db9e4de
   b1d95739
```

SAM テンプレートに Metadata というセクションを入れる

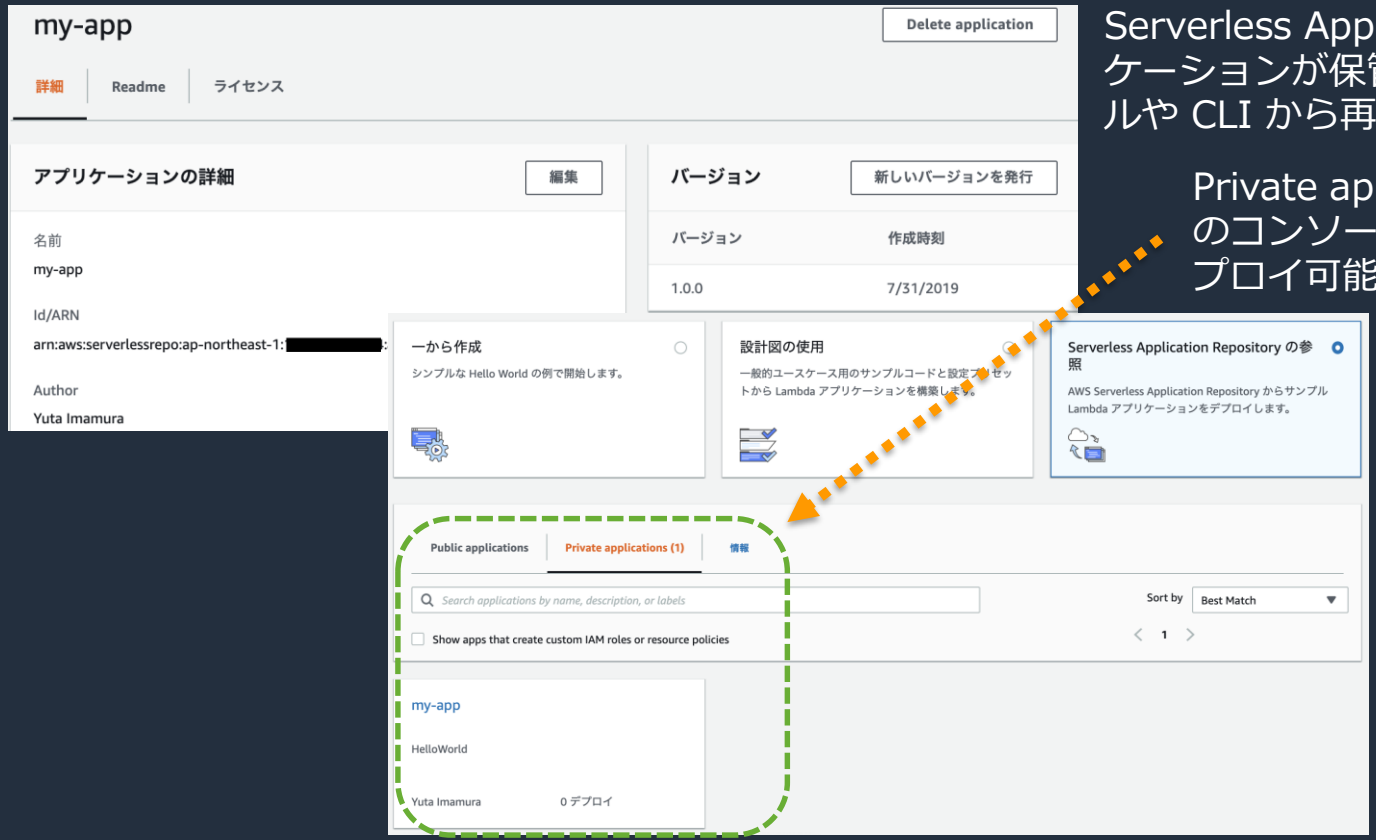
Author に作者、Description にアプリケーションの説明、Name にアプリケーション名を指定しておく

```
sam-app ) sam publish --region ap-northeast-1 --template outp
ut.yml --semantic-version 1.0.0
2019-07-31 21:27:48 Found credentials in shared credentials f
ile: ~/.aws/credentials
Publish Succeeded
The following metadata of application "arn:aws:serverlessrepo
:ap-northeast-1:[redacted]:applications/my-app" has been up
dated:
{
  "Author": "Yuta Imamura",
  "Description": "HelloWorld"
}
Click the link below to view your application in AWS console:
https://console.aws.amazon.com/serverlessrepo/home?region=ap-
northeast-1#/published-applications/arn:aws:serverlessrepo:ap
-northeast-1:[redacted]:applications~my-app
```

sam publish コマンドで Serverless Application Repository へアプリケーションを発行する

---region に発行先のリージョンを、-template にパッケージング後の SAM テンプレートを、--semantic-version に発行するアプリケーションのバージョンを指定する

完成したアプリケーションの保管 (sam publish) (2/2)



Serverless Application Repository にアプリケーションが保管され、Lambda のコンソールや CLI から再びデプロイできるようになる

Private application として Lambda のコンソールに表示され、すぐに再デプロイ可能となる

SAM CLI のインストール

SAM CLI は Linux, Mac, Windows に対応。

- 事前に AWS Command Line Interface (AWS CLI) のインストールが必要
- テストのために Lambda 関数をローカルで実行する場合、事前に Docker のインストールが必要
- SAM CLI は最新版のインストールを推奨
  - SAM の最新の文法への対応などが入るため

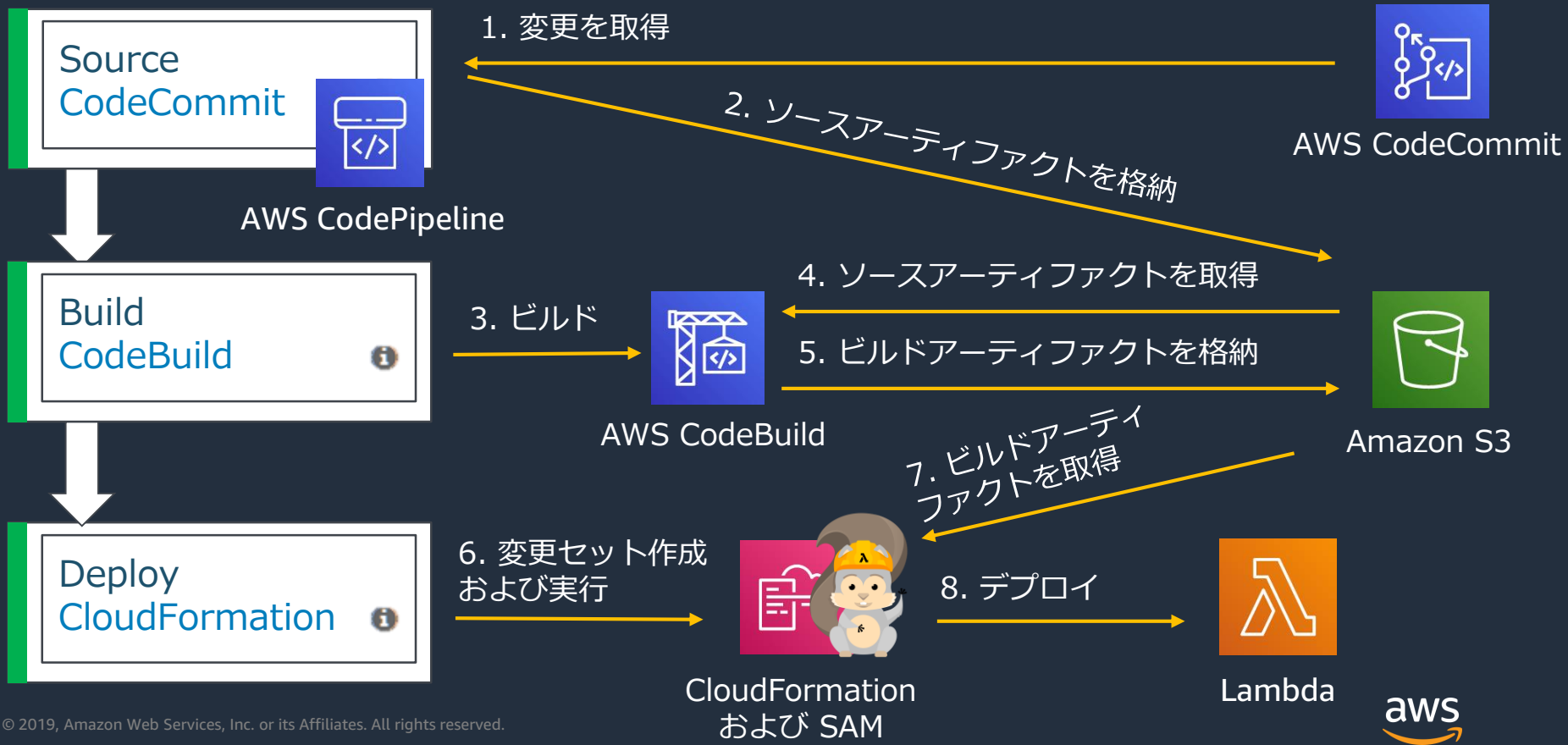
Linux、Mac、Windows など環境ごとにインストール手順は異なるため、詳細は AWS の公式ドキュメントを参照：

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html

アジェンダ

- AWS SAM とは
- AWS SAM の機能と文法
- AWS SAM Command Line Interface
- その他考慮事項

SAM と CI/CD パイプライン (1/2)



SAM と CI/CD パイプライン (2/2)

AWS CodeStar を使うことで、CI/CD パイプラインを**自動作成可能**

AWS CodeStar ▸ プロジェクトの作成

テンプレートの選択 ツールのセットアップ コーディングの開始

フィルター

アプリケーションのカテゴリ
- ウェブアプリケーション
- ウェブサービス
- Alexa スキル
- 静的ウェブサイト
- AWS Config ルール

プログラミング言語
- C#
- Go
- HTML 5
- Java
- Node.js
- PHP
- Python
- Ruby

AWS サービス
- AWS Elastic Beanstalk
- Amazon EC2
- AWS Lambda

プロジェクトのテンプレートを選択

プロジェクトテンプレートを使用して数分で AWS で新しいソフトウェアプロジェクトを始められます。選択のヘルプ

- Go: ウェブアプリケーション / AWS Lambda (サーバーレスで実行)
- Node.js: ウェブアプリケーション / AWS Lambda (サーバーレスで実行)
- Python: ウェブサービス / AWS Lambda (サーバーレスで実行)
- Express.js: ウェブサービス / AWS Lambda (サーバーレスで実行)
- Java Spring: ウェブサービス / AWS Lambda (サーバーレスで実行)
- Hello World Skill: Alexa スキル / AWS Lambda (サーバーレスで実行)
- Hello World Skill: Alexa スキル / AWS Lambda (サーバーレスで実行)
- Python: AWS Config ルール / AWS Lambda (サーバーレスで実行)

任意の言語の AWS Lambda のプロジェクト テンプレートを選択

類似ツールとの比較

	ユースケース
CloudFormation + SAM	・YAML/JSON で宣言的にインフラ管理を行いたい場合 ・開発者のローカル環境でテストを行いたい場合 ・Serverless Application Repository でアプリケーションを公開したい場合
AWS Cloud Development Kit (CDK)	・TypeScript や Python 等のプログラミング言語で管理を行いたい場合 ・管理対象となるサーバーレス以外のサービスが多く存在する場合 ・コード補完や文法検証 (lint) 機能を最大限利用したい場合
AWS Chalice	・Python に特化した開発を行いたい場合 ・開発者のローカル環境でテストを行いたい場合 ・ アプリの開発者とインフラエンジニアが同一である場合
AWS Amplify	・CLI で対話的にインフラのセットアップを行いたい場合 ・バックエンドについて複雑な設定が不要な場合 ・ AWS AppSync とバックエンドを統合したい場合

SAM のベストプラクティス (1/2)

- ひとつのサーバーレス アプリケーションを構築するリソースの中で、ライフサイクルが同じものは、同じテンプレートにまとめる
  - ライフサイクルが同じ = リソースが更新・削除されるタイミングが同一であること
- Lambda と、その呼び出し元となるリソース (S3 や API Gateway 等) は、同じテンプレートにまとめる
  - AWS SAM では、既存の API Gateway や S3 をトリガーとして Lambda 関数を呼び出すことをサポートしていない
- Dev / Test / Prod といった環境は、CloudFormation スタックで分離する
  - AWS SAM では複数の API Gateway のステージを利用することができない
  - ただし、テンプレートとしては共通化し、パラメータで環境を切り替えられるように

SAM のベストプラクティス (2/2)

- Lambda の実装内容や API Gateway の API 定義を更新する際はカナリア リリース (段階的デプロイ) を活用する
  - テストや CI/CD の仕組みが完全でも見落としは有り得る。万が一に備え、商用環境への適用は段階的に行い、エラーが発生しないか監視する
- 汎用的に利用するサーバーレス アプリケーションは、Serverless Application Repository を使い他の AWS アカウントへ共有する
  - アプリケーションとしてパッケージングされた資材を、あらかじめ認可した対象者に対してそのまま配布できるので、責任分界点が明確化できる
- AWS SAM CLI を使い、ローカル環境でビルド、デバッグ、SAM テンプレート検証を行う
  - Lambda を再現した Docker コンテナが立ち上がるので、本番同等の環境で検証ができる

Meet SAM!

AWS SAM :

https://github.com/awslabs/serverless-application-model

AWS SAM CLI :

https://github.com/awslabs/aws-sam-cli

Q&A

いただいたご質問については
AWS Japan Blog 「https://aws.amazon.com/jp/blogs/news/」にて
後日掲載します。

AWS の日本語資料の場所「AWS 資料」で検索

https://amzn.to/JPArchive

AWS Well-Architected 個別技術相談会

毎週”W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に対策などを相談することも可能
- 申込みはイベント告知サイトから

(https://aws.amazon.com/jp/about-aws/events/)

AWS イベント で[検索]

ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive