



AWS ソリューション

# AWS ソリューション構造



# AWS ソリューション構造: AWS ソリューション

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

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスと関連付けてはならず、また、お客様に混乱を招くような形や Amazon の信用を傷つけたり失わせたりする形で使用することはできません。Amazon が所有しない商標はすべてそれぞれの所有者に所属します。所有者は必ずしも Amazon と提携していたり、関連しているわけではありません。また、Amazon 後援を受けているとはかぎりません。

# Table of Contents

概要 .....	1
AWS Solutions Constructs とは何ですか? .....	1
AWS ソリューションコンストラクトを使用する理由 .....	1
はじめに .....	3
前提条件 .....	3
AWS CDK のインストール .....	4
AWS Solutions Constructs の使用 .....	4
ウォークスルー-パート 1 .....	5
Hello Constrents .....	5
アプリケーションディレクトリの作成と AWS CDK の初期化 .....	6
プロジェクトベースの依存関係の更新 .....	7
Lambda ハンドコード .....	9
AWS CDK をインストールし、AWS ソリューションを構築する依存関係 .....	10
Amazon API ゲートウェイ/AWS Lambda パターンをスタックに追加する .....	12
CDK デプロイ .....	18
スタック出力 .....	19
アプリのテスト .....	19
ウォークスルー-パート 2 .....	19
ヒットカウンタ Lambda コード .....	20
新しい依存関係をインストールする .....	22
リソースを定義します。 .....	23
変更の確認 .....	36
CDK デプロイ .....	37
スタック出力 .....	38
アプリのテスト .....	38
サンプルユースケース .....	39
AWS 静的 S3 ウェブサイト .....	40
AWS シンプルサーバーレスイメージハンドラー .....	40
AWS サーバーレスウェブアプリ .....	40
API リファレンス .....	42
モジュール .....	42
Module の内容 .....	42
aws-apigateway-ダイナモッド .....	43
概要 .....	43

イニシャライザ .....	44
パターン構成プロパティ .....	44
パターンプロパティ .....	45
デフォルト設定 .....	46
アーキテクチャ .....	47
GitHub .....	47
aws-apigateway-IoT .....	48
概要 .....	48
初期化 .....	49
パターン構成プロパティ .....	49
パターンプロパティ .....	50
デフォルト設定 .....	51
アーキテクチャ .....	54
例 .....	54
GitHub .....	56
aws-apigateway-キネシスストリーム .....	56
概要 .....	57
初期化 .....	57
パターン構成プロパティ .....	58
パターンプロパティ .....	59
API の使用例 .....	60
デフォルト設定 .....	61
アーキテクチャ .....	62
GitHub .....	62
aws-apigateway-ラムダ .....	62
概要 .....	63
初期化 .....	63
パターン構成プロパティ .....	64
パターンプロパティ .....	65
デフォルト設定 .....	65
アーキテクチャ .....	66
GitHub .....	66
aws-apigateway-sagemakerendpoint .....	67
概要 .....	67
初期化 .....	68
パターン構成プロパティ .....	68

パターンプロパティ .....	70
API の使用例 .....	60
デフォルト設定 .....	71
アーキテクチャ .....	72
GitHub .....	72
aws-apigateway-sqs .....	72
概要 .....	73
初期化 .....	73
パターン構成プロパティ .....	74
パターンプロパティ .....	76
API の使用例 .....	60
デフォルト設定 .....	77
アーキテクチャ .....	78
GitHub .....	78
aws-クラウドフロント-アピゲートウェイ .....	78
概要 .....	79
初期化 .....	80
パターン構成プロパティ .....	80
パターンプロパティ .....	81
デフォルト設定 .....	81
アーキテクチャ .....	82
GitHub .....	83
aws-クラウドフロント-アピゲートウェイ-ラムダ .....	83
概要 .....	83
初期化 .....	84
パターン構成プロパティ .....	84
パターンプロパティ .....	86
デフォルト設定 .....	87
アーキテクチャ .....	88
GitHub .....	88
aws-クラウドフロントメディアストア .....	88
概要 .....	89
初期化 .....	89
パターン構成プロパティ .....	90
パターンプロパティ .....	90
デフォルト設定 .....	91

アーキテクチャ .....	93
GitHub .....	93
aws-クラウドフロント-3 .....	93
概要 .....	94
イニシャライザ .....	94
パターン構成プロパティ .....	95
パターンプロパティ .....	96
デフォルト設定 .....	96
アーキテクチャ .....	97
GitHub .....	97
aws-コグニート-アピガテウェイ-ラムダ .....	98
概要 .....	79
初期化 .....	99
パターン構成プロパティ .....	100
パターンプロパティ .....	101
デフォルト設定 .....	102
アーキテクチャ .....	103
GitHub .....	103
aws-Dynamodb-stream lambda .....	103
概要 .....	104
初期化 .....	105
パターン構成プロパティ .....	105
パターンプロパティ .....	106
Lambda 関数 .....	106
デフォルト設定 .....	106
アーキテクチャ .....	107
GitHub .....	108
aws-ダイナモッド-ストリーム-ラムダ-弾性検索-キバナ .....	108
概要 .....	109
イニシャライザ .....	109
パターン構成プロパティ .....	110
パターンプロパティ .....	111
Lambda 関数 .....	112
デフォルト設定 .....	112
アーキテクチャ .....	114
GitHub .....	114

aws-events-ルール-キネシフアイアホース-3 .....	114
概要 .....	115
初期化 .....	116
パターン構成プロパティ .....	116
パターンプロパティ .....	117
デフォルト設定 .....	118
アーキテクチャ .....	119
GitHub .....	119
aws-events-ルール-キネシスストリーム .....	119
概要 .....	120
初期化 .....	121
パターン構成プロパティ .....	121
パターンプロパティ .....	122
デフォルト設定 .....	122
アーキテクチャ .....	123
GitHub .....	123
aws-events-ルール-ラムダ .....	123
概要 .....	124
初期化 .....	125
パターン構成プロパティ .....	125
パターンプロパティ .....	126
デフォルト設定 .....	126
アーキテクチャ .....	127
GitHub .....	127
aws-イベント-ルール-sns .....	127
概要 .....	128
初期化 .....	129
パターン構成プロパティ .....	129
パターンプロパティ .....	130
デフォルト設定 .....	131
アーキテクチャ .....	131
GitHub .....	132
aws-イベント-ルール-sqs .....	132
概要 .....	132
初期化 .....	133
パターン構成プロパティ .....	134

パターンプロパティ .....	135
デフォルト設定 .....	136
アーキテクチャ .....	137
GitHub .....	137
aws-イベント-ルール-ステップ関数 .....	137
概要 .....	138
イニシャライザ .....	139
パターン構成プロパティ .....	139
パターンプロパティ .....	140
デフォルト設定 .....	140
アーキテクチャ .....	141
GitHub .....	141
aws-iot-キネシシフアイアホース-3 .....	141
概要 .....	142
初期化 .....	143
パターン構成プロパティ .....	143
パターンプロパティ .....	144
デフォルト設定 .....	145
アーキテクチャ .....	146
GitHub .....	146
aws-イオット-ラムダ .....	146
概要 .....	147
初期化 .....	148
パターン構成プロパティ .....	148
パターンプロパティ .....	149
デフォルト設定 .....	149
アーキテクチャ .....	150
GitHub .....	150
aws-iot-ラムダ-ダイナモブ .....	150
概要 .....	151
初期化 .....	152
パターン構成プロパティ .....	152
パターンプロパティ .....	153
デフォルト設定 .....	153
アーキテクチャ .....	154
GitHub .....	155



aws-キネシスファイアホース-3 .....	155
概要 .....	155
イニシャライザ .....	156
パターン構成プロパティ .....	156
パターンプロパティ .....	157
デフォルト設定 .....	158
アーキテクチャ .....	159
GitHub .....	159
aws-キネシスファイアホース-S3-アンドキネシス解析 .....	159
概要 .....	160
初期化 .....	161
パターン構成プロパティ .....	161
パターンプロパティ .....	162
デフォルト設定 .....	163
アーキテクチャ .....	164
GitHub .....	164
aws-キネシスストリーム-gluejob .....	165
概要 .....	165
初期化 .....	166
パターン構成プロパティ .....	167
シンクデータストアプロップス .....	168
シンクストアタイプ .....	169
デフォルト設定 .....	169
アーキテクチャ .....	171
GitHub .....	171
aws-キネシスストリーム-キネシスファイアホース-3 .....	171
概要 .....	172
イニシャライザ .....	172
パターン構成プロパティ .....	173
パターンプロパティ .....	174
デフォルト設定 .....	175
アーキテクチャ .....	176
GitHub .....	176
aws-キネシスストリーム-ラムダ .....	177
概要 .....	177
初期化 .....	178

パターン構成プロパティ .....	178
パターンプロパティ .....	179
デフォルト設定 .....	180
アーキテクチャ .....	181
GitHub .....	181
aws-lambda-dynamodb .....	181
概要 .....	182
初期化子 .....	183
パターン構成プロパティ .....	183
パターンプロパティ .....	186
デフォルト設定 .....	187
アーキテクチャ .....	188
GitHub .....	188
aws-lambda-Elasticsearch- .....	188
概要 .....	189
イニシャライザ .....	190
パターン構成プロパティ .....	190
パターンプロパティ .....	191
Lambda 関数 .....	192
デフォルト設定 .....	192
アーキテクチャ .....	193
GitHub .....	194
aws-ラムダ-s3 .....	194
概要 .....	194
初期化子 .....	195
パターン構成プロパティ .....	195
パターンプロパティ .....	199
デフォルト設定 .....	200
アーキテクチャ .....	201
GitHub .....	201
aws-ラムダ-ssmstringパラメータ .....	201
概要 .....	202
初期化子 .....	202
パターン構成プロパティ .....	203
パターンプロパティ .....	207
デフォルト設定 .....	207

アーキテクチャ .....	208
GitHub .....	208
aws-ラムダ-サゲマケレンドポイント .....	208
概要 .....	209
イニシャライザ .....	210
パターン構成プロパティ .....	210
パターンプロパティ .....	214
デフォルト設定 .....	214
アーキテクチャ .....	215
GitHub .....	216
aws-ラムダ-セクレツマネージャ .....	216
概要 .....	216
初期化子 .....	217
パターン構成プロパティ .....	217
パターンプロパティ .....	220
デフォルト設定 .....	221
アーキテクチャ .....	222
GitHub .....	222
aws-lambda-sns .....	222
概要 .....	223
初期化子 .....	223
パターン構成プロパティ .....	224
パターンプロパティ .....	227
デフォルト設定 .....	228
アーキテクチャ .....	229
GitHub .....	229
aws-ラムダ-sqs .....	229
概要 .....	230
初期化 .....	230
パターン構成プロパティ .....	231
パターンプロパティ .....	234
デフォルト設定 .....	235
アーキテクチャ .....	236
GitHub .....	236
aws-ラムダ-sqs-ラムダ .....	236
概要 .....	237

イニシャライザ .....	238
パターン構成プロパティ .....	238
パターンプロパティ .....	240
デフォルト設定 .....	241
アーキテクチャ .....	242
GitHub .....	242
aws-ラムダ-ステップ関数 .....	242
概要 .....	243
初期化 .....	244
パターン構成プロパティ .....	244
パターンプロパティ .....	245
デフォルト設定 .....	246
アーキテクチャ .....	247
GitHub .....	247
aws-s3-ラムダ .....	247
概要 .....	248
初期化 .....	249
パターン構成プロパティ .....	249
パターンプロパティ .....	250
デフォルト設定 .....	250
アーキテクチャ .....	251
GitHub .....	252
aws-s3-sqs .....	252
概要 .....	252
初期化 .....	253
パターン構成プロパティ .....	253
パターンプロパティ .....	255
デフォルト設定 .....	256
アーキテクチャ .....	257
GitHub .....	257
aws-s3ステップ関数 .....	257
概要 .....	258
初期化 .....	259
パターン構成プロパティ .....	259
パターンプロパティ .....	260
デフォルト設定 .....	262

アーキテクチャ .....	263
GitHub .....	263
aws-sns-ラムダ .....	263
概要 .....	264
イニシャライザ .....	264
パターン構成プロパティ .....	265
パターンプロパティ .....	266
デフォルト設定 .....	266
アーキテクチャ .....	267
GitHub .....	267
aws-sns-sqs .....	267
概要 .....	268
初期化 .....	269
パターン構成プロパティ .....	269
パターンプロパティ .....	271
デフォルト設定 .....	271
アーキテクチャ .....	272
GitHub .....	272
aws-sqs-ラムダ .....	272
概要 .....	273
イニシャライザ .....	274
パターン構成プロパティ .....	274
パターンプロパティ .....	275
デフォルト設定 .....	276
アーキテクチャ .....	277
GitHub .....	277
core .....	277
AWS CDK コンストラクトのデフォルトプロパティ .....	278
既定のプロパティを上書きする .....	278
プロパティのオーバーライドに関する警告 .....	279
ドキュメントの改訂 .....	280
注意 .....	285
.....	cclxxxvi

# AWS Solutions の構造

発行日: 2021年5月([ドキュメントの改訂](#))

## AWS Solutions Constructs とは何ですか？

AWS ソリューションコンストラクト (コンストラクト) は、[AWS Development Kit \(AWS CDK\)](#) は、予測可能で反復可能なインフラストラクチャを作成するために、コード内のソリューションを迅速に定義するための、マルチサービス、優れたアーキテクチャのパターンを提供します。目標は、開発者がアーキテクチャのパターンベースの定義を使用して、あらゆる規模のソリューションを構築するエクスペリエンスを加速することです。

AWS ソリューション構成を使用して、使い慣れたプログラミング言語でソリューションを定義します。AWS ソリューションコンストラクトでは、現時点では TypeScript、JavaScript、Python、および Java がサポートされています。

AWS ソリューション構築パターンの完全なカタログを参照するには、[ここをクリックしてください](#)。

## AWS ソリューションコンストラクトを使用する理由

クラウドプロバイダーの革新のスピードにより、ベストプラクティスを把握して理解し、ソリューション全体に正しく実装されるようにすることは、大変な作業です。コンストラクトを使用すると、事前に構築された、適切に設計されたパターンと、スケーラブルで安全な方法でクラウドサービスを使用して一般的なアクションを実行するユースケースを組み合わせることができます。Constructs は最新のプログラミング言語用のライブラリを提供するため、既存の開発スキルや使い慣れたツールを、ソリューションに合わせて適切に設計されたクラウドインフラストラクチャを構築するタスクに適用できます。

AWS ソリューション構築のその他の利点は次のとおりです。

- AWS Cloud Development Kit (AWS CDK) のオープンソースソフトウェア開発フレームワークに基づいて構築されています。
- ソリューションインフラストラクチャを定義するときは、ロジック ( if文、for-loopなど ) を使用します。
- オブジェクト指向のテクニックを使用して、システムのモデルを作成します。
- 高レベルの抽象化を定義し、共有し、チーム、会社、コミュニティに公開します。

- ソリューションを論理モジュールに整理します。
- ソリューションを共有し、ライブラリとして再利用します。
- 業界標準のプロトコルを使用してインフラストラクチャコードをテストします。
- 既存のコードレビューワークフローを使用します。

AWS Solutions Constructs の目的は、AWS でソリューション目標を達成するために、よく設計された一般的なパターンを統合する際に必要となる複雑さを軽減し、ロジックを接着することです。

# AWS Solutions Constructs の開始方法

このトピックでは、AWS Cloud Development Kit ( AWS CDK )、AWS ソリューションコンストラクトをインストールおよび設定し、AWS ソリューションコンストラクトパターンを使用して最初の AWS CDK アプリケーションを作成する方法について説明します。

## Note

AWS CDK バージョン 1.46.0 以上で AWS Solutions Constructs がサポートされています。

## Tip

深く掘り下げたいですか？ 試してみましょう [CDK ワークショップ](#) を参照して、現実世界のプロジェクトのより詳細なツアーをご覧ください。

## Tip

AWS Cloud Development Kit (AWS CDK) の開始方法の詳細については、[AWS CDK 開発者ガイド](#)。




## Prerequisites

AWS ソリューションコンストラクトは AWS CDK に基づいて構築されているため、Node.js (  $\geq 10.3.0$  ) をインストールする必要があります。これは、TypeScript や JavaScript 以外の言語で動作しているものでも同様です。これは、[AWS CDK](#) と AWS ソリューションコンストラクトは TypeScript で開発され、Node.js 上で実行されます。サポートされている他の言語のバインディングでは、このバックエンドとツールセットが使用されます。

「認証情報およびリージョンの指定」の説明に従って、AWS CDK CLI を使用するには、認証情報と AWS リージョンを指定する必要があります。

その他の前提条件は、次のように、開発言語によって異なります。



言語	前提条件
	Python > = 3.6 P
 t	TypeScript > = 2.7 T
	Java > = 1.8 J;

## AWS CDK のインストール

AWS CDK をインストールおよび設定するには、『AWS CDK 開発者ガイド』を参照してください。[AWS CDK のインストール](#)。

## AWS Solutions Constructs の使用

AWS ソリューションコンストラクトを使用するとき新しいアプリケーションを作成する一般的なワークフローは、AWS CDK と同じアプローチに従います。

1. app ディレクトリを作成します。
2. アプリの初期化
3. AWS ソリューション構築パターンの依存関係を追加します。
4. アプリにコードを追加します。
5. 必要に応じて、アプリケーションをコンパイルします。
6. アプリで定義されているリソースをデプロイします。
7. アプリのテスト

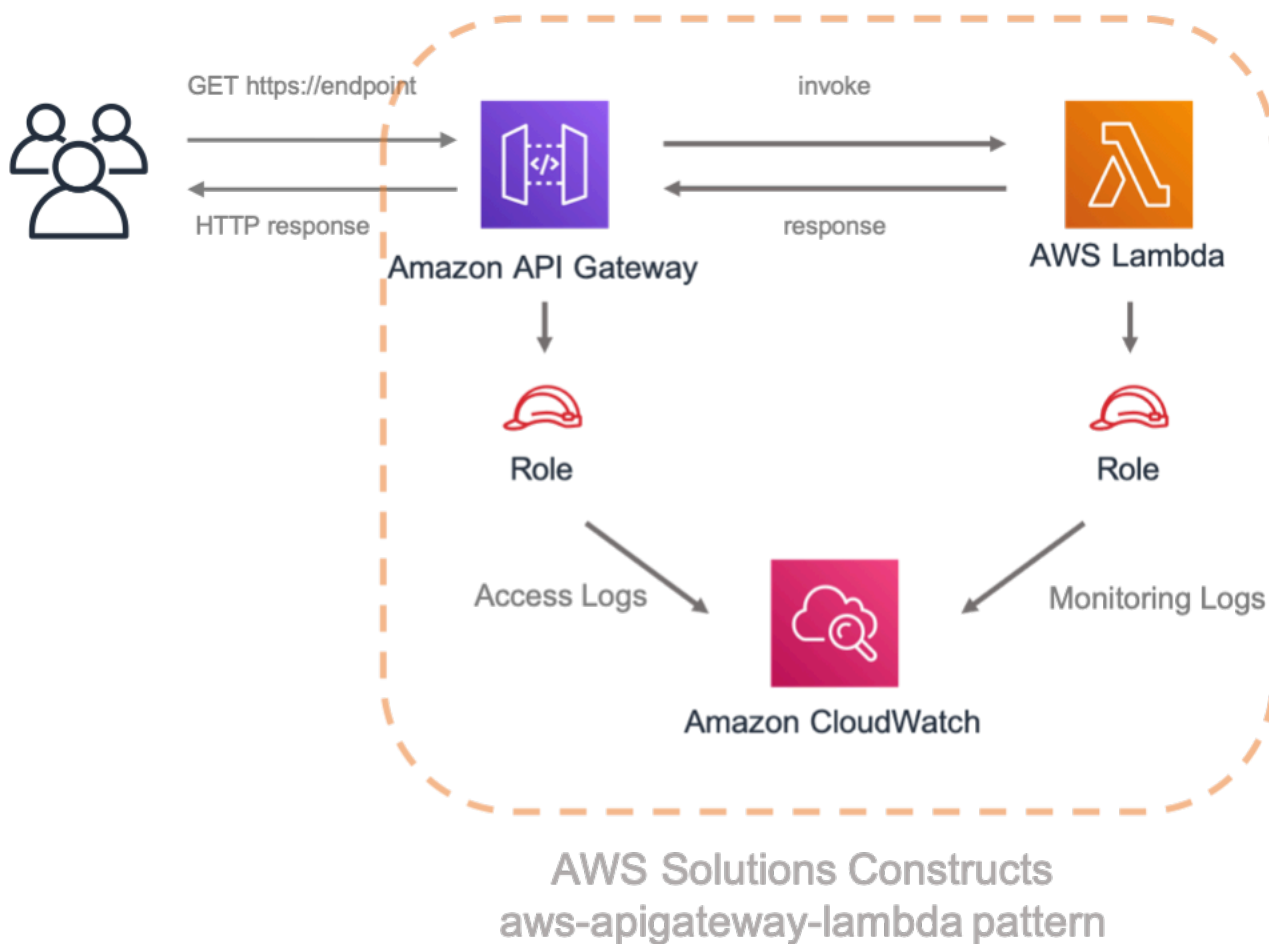
問題がある場合は、変更、コンパイル ( 必要な場合 )、デプロイ、およびテストを繰り返しループします。

# ウォークスルー-パート 1

## Note

AWS CDK バージョン 1.46.0 以上 AWS ソリューション構成がサポートされています。

このチュートリアルでは、プロジェクトの初期化から結果の AWS CloudFormation テンプレートのデプロイまで、AWS ソリューション構築のパターンを使用する、シンプルな「Hello Constructs」AWS CDK アプリケーションを作成してデプロイする方法について説明します。Hello Constructs アプリは、次の簡単なソリューションを作成します。



## Hello Constructs

パターンベースの開発を使用して、最初の AWS CDK アプリケーションの構築を始めましょう。

**Note**

これは、Hello CDK!からの[CDK ワークショップ](#)。AWS CDK を初めて使用する場合は、このワークショップから実践的なウォークスルーを行い、CDK を活用して実世界のプロジェクトを構築する方法をおすすめします。

## アプリケーションディレクトリの作成と AWS CDK の初期化

CDK アプリケーション用のディレクトリを作成し、そのディレクトリに AWS CDK アプリケーションを作成します。

### TypeScript

```
mkdir hello-constructs
cd hello-constructs
cdk init --language typescript
```

### Python

```
mkdir hello-constructs
cd hello-constructs
cdk init --language python
```

**Tip**

今度は、お気に入りの IDE でプロジェクトを開いて探索する良い時期です。プロジェクト構造の詳細については、適切なリンクを選択します。

- [TypeScript](#)
- [Python](#)

## プロジェクトベースの依存関係の更新

### ⚠ Warning

適切な機能を確保するために、AWS ソリューションコンストラクトと AWS CDK パッケージはプロジェクト内で同じバージョン番号を使用する必要があります。たとえば、AWS ソリューション構成 v.1.52.0 を使用している場合は、AWS CDK v.1.52.0 も使用する必要があります。

### ℹ Tip

AWS ソリューション構築物の最新バージョンを書き留め、そのバージョン番号をVERSION\_NUMBERプレースホルダを以下のステップで使用します ( AWS ソリューションコンストラクトと AWS CDK パッケージの両方 )。Constructsライブラリのすべてのパブリックリリースをチェックするには、[ここをクリックしてください](#) を選択します。

## TypeScript

編集package.jsonファイルを編集します。

```
"devDependencies": {
  "@aws-cdk/assert": "VERSION_NUMBER",
  "@types/jest": "^24.0.22",
  "@types/node": "10.17.5",
  "jest": "^24.9.0",
  "ts-jest": "^24.1.0",
  "aws-cdk": "VERSION_NUMBER",
  "ts-node": "^8.1.0",
  "typescript": "~3.7.2"
},
"dependencies": {
  "@aws-cdk/core": "VERSION_NUMBER",
  "source-map-support": "^0.5.16"
}
```

## Python

編集setup.pyファイルを編集します。

```
install_requires=[  
    "aws-cdk.core==VERSION_NUMBER",  
],
```

プロジェクトの基本依存関係をインストールします。

## TypeScript

```
npm install
```

## Python

```
source .venv/bin/activate  
pip install -r requirements.txt
```

アプリケーションをビルドして実行し、空のスタックが作成されていることを確認します。

## TypeScript

```
npm run build  
cdk synth
```

## Python

```
cdk synth
```

次のようなスタックが表示されます。CDK-VERSIONはCDKのバージョンです。(出力は、ここに示されているものと若干異なる場合があります)。

## TypeScript

```
Resources:
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Modules: aws-cdk=CDK-VERSION,@aws-cdk/core=VERSION_NUMBER,@aws-cdk/cx-
api=VERSION_NUMBER,jsii-runtime=node.js/10.17.0
```

## Python

```
Resources:
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Modules: aws-cdk=CDK-VERSION,@aws-cdk/core=VERSION_NUMBER,@aws-cdk/cx-
api=VERSION_NUMBER,jsii-runtime=Python/3.7.7
```

## Lambda ハンドコード

AWS Lambda ハンドラーコードから始めます。

ディレクトリを作成するlambdaプロジェクトツリーのルートに移動します。

## TypeScript

という名前のファイルを追加します。lambda/hello.js項目の変更は以下のとおりです。

```
exports.handler = async function(event) {
  console.log("request:", JSON.stringify(event, null, 2));
  return {
    statusCode: 200,
    headers: { "Content-Type": "text/plain" },
```

```
    body: `Hello, AWS Solutions Constructs! You've hit ${event.path}\n`
  };
};
```

## Python

という名前のファイルを追加します。lambda/hello.py項目の変更は以下のとおりです。

```
import json

def handler(event, context):
    print('request: {}'.format(json.dumps(event)))
    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'text/plain'
        },
        'body': 'Hello, CDK! You have hit {}\n'.format(event['path'])
    }
```

これは単純なLambda 関数で、「こんにちは、コンストラクト！[url path]」とヒットしました。関数の出力には、HTTP ステータスコードと HTTP ヘッダーも含まれます。これらは、API Gateway によって、ユーザーへの HTTP 応答を策定するために使用されます。

このLambda は、JavaScriptで提供されています。選択した言語で Lambda 関数を記述する方法の詳細については、[AWS Lambda ドキュメント](#)。

## AWS CDK をインストールし、AWS ソリューションを構築する依存関係

AWS ソリューションコンストラクトには、コンストラクトの広範なライブラリが付属しています。ライブラリは、適切に設計されたパターンごとに1つずつ、モジュールに分割されています。例えば、Amazon API Gateway レスト API を AWS Lambda 関数に定義する場合、aws-apigateway-lambdaパターンライブラリ。

また、AWS CDK から AWS Lambda および Amazon API Gateway 構築ライブラリを追加する必要があります。

AWS Lambda モジュールとそのすべての依存関係をプロジェクトにインストールします。

**Note**

AWS ソリューション構築と AWS CDK の両方で使用する正しい一致するバージョンをVERSION\_NUMBER各コマンドのプレースホルダフィールドです。パッケージ間でバージョンが一致しないと、エラーが発生する可能性があります。

## TypeScript

```
npm install -s @aws-cdk/aws-lambda@VERSION_NUMBER
```

## Python

```
pip install aws_cdk.aws_lambda==VERSION_NUMBER
```

次に、Amazon API Gateway モジュールとそのすべての依存関係をプロジェクトにインストールします。

## TypeScript

```
npm install -s @aws-cdk/aws-apigateway@VERSION_NUMBER
```

## Python

```
pip install aws_cdk.aws_apigateway==VERSION_NUMBER
```

最後に、AWS ソリューションコンストラクトをインストールしますaws-apigateway-lambdaモジュールとそのすべての依存関係をプロジェクトに追加します。



## TypeScript

```
npm install -s @aws-solutions-constructs/aws-apigateway-lambda@VERSION_NUMBER
```

## Python

```
pip install aws_solutions_constructs.aws_apigateway_lambda==VERSION_NUMBER
```

## Amazon API ゲートウェイ/AWS Lambda パターンをスタックに追加する

それでは、AWS Lambda プロキシを使用して Amazon API Gateway を実装するための AWS ソリューション構築パターンを定義しましょう。

## TypeScript

ファイルを編集します。lib/hello-constructs.ts項目の変更は次のとおりです。

```
import * as cdk from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';
import * as api from '@aws-cdk/aws-apigateway';
import { ApiGatewayToLambda, ApiGatewayToLambdaProps } from '@aws-solutions-constructs/aws-apigateway-lambda';

export class HelloConstructsStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The code that defines your stack goes here
    const api_lambda_props: ApiGatewayToLambdaProps = {
      lambdaFunctionProps: {
        code: lambda.Code.fromAsset('lambda'),
        runtime: lambda.Runtime.NODEJS_12_X,
        handler: 'hello.handler'
      },
    },
```

```
        apiGatewayProps: {
            defaultMethodOptions: {
                authorizationType: api.AuthorizationType.NONE
            }
        }
    };

    new ApiGatewayToLambda(this, 'ApiGatewayToLambda', api_lambda_props);
}
}
```

## Python

ファイルを編集します。hello\_constructs/hello\_constructs\_stack.py項目の変更は次のとおりです。

```
from aws_cdk import (
    aws_lambda as _lambda,
    aws_apigateway as apigw,
    core,
)

from aws_solutions_constructs import (
    aws_apigateway_lambda as apigw_lambda
)

class HelloConstructsStack(core.Stack):

    def __init__(self, scope: core.Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # The code that defines your stack goes here

        apigw_lambda.ApiGatewayToLambda(
            self, 'ApiGatewayToLambda',
            lambda_function_props=_lambda.FunctionProps(
                runtime=_lambda.Runtime.PYTHON_3_7,
                code=_lambda.Code.asset('lambda'),
                handler='hello.handler',
            ),
            api_gateway_props=apigw.RestApiProps(
```

```

        default_method_options=apigw.MethodOptions(
            authorization_type=apigw.AuthorizationType.NONE
        )
    )
)

```

それだ これは、すべてのリクエストを AWS Lambda 関数にプロキシする API Gateway を定義するために必要なすべてです。新しいスタックを元のスタックと比較しましょう：

## TypeScript

```

npm run build
cdk diff

```

## Python

```

cdk diff

```

出力は次のようになります。

```

Stack HelloConstructsStack
IAM Statement Changes
#####
# # Resource # Effect # Action # Principal
# # # Condition #
#####
# + # ${LambdaFunction.Arn} # Allow # lambda:InvokeFunction #
# # Service:apigateway.amazonaw # "ArnLike": { #
# # # # # # s.com
# # # "AWS:SourceArn": "arn:${AW #
# # # # # #
# # # S::Partition}:execute-api:${ #
# # # # # #
# # # AWS::Region}:${AWS::AccountI #

```

```

# # # # #
# # # d}:${RestApi0C43BF4B}/${Rest #
# # # # #
# # # Api/DeploymentStage.prod}/*/ #
# # # # #
# # # {proxy+}" #
# # # # #
# # # } #
# + # ${LambdaFunction.Arn} # Allow # lambda:InvokeFunction #
Service:apigateway.amazonaw # "ArnLike": { #
# # # # # # s.com
# # # "AWS:SourceArn": "arn:${AW #
# # # # # #
# # # S::Partition}:execute-api:${ #
# # # # # #
# # # AWS::Region}:${AWS::AccountI #
# # # # # #
# # # d}:${RestApi0C43BF4B}/test-i #
# # # # # #
# # # nvoke-stage/*/{proxy+}" #
# # # # # #
# # # } #
# + # ${LambdaFunction.Arn} # Allow # lambda:InvokeFunction #
Service:apigateway.amazonaw # "ArnLike": { #
# # # # # # s.com
# # # "AWS:SourceArn": "arn:${AW #
# # # # # #
# # # S::Partition}:execute-api:${ #
# # # # # #
# # # AWS::Region}:${AWS::AccountI #
# # # # # #
# # # d}:${RestApi0C43BF4B}/${Rest #
# # # # # #
# # # Api/DeploymentStage.prod}/*/ #
# # # # # #
# # # " #
# # # # # #
# # # # # #
# + # ${LambdaFunction.Arn} # Allow # lambda:InvokeFunction #
Service:apigateway.amazonaw # "ArnLike": { #
# # # # # # s.com
# # # "AWS:SourceArn": "arn:${AW #
# # # # # #
# # # S::Partition}:execute-api:${ #

```

```

# # # # #
# # # AWS::Region}:${AWS::AccountI #
# # # # #
# # # d}:${RestApi0C43BF4B}/test-i #
# # # # #
# # # nvoke-stage/*/" #
# # # # #
# # # # #
#####
# + # ${LambdaFunctionServiceRole # Allow # sts:AssumeRole #
Service:lambda.amazonaws.co # #
# # .Arn} # # # # m
# # # # #
#####
# + # ${LambdaRestApiCloudWatchRo # Allow # sts:AssumeRole #
Service:apigateway.amazonaw # #
# # le.Arn} # # # # s.com
# # # # #
#####
# + # arn:aws:logs:${AWS::Region} # Allow # logs:CreateLogGroup # AWS:
${LambdaRestApiCloudWat # #
# # :${AWS::AccountId}:* # # logs:CreateLogStream # chRole}
# # # # #
# # # # # logs:DescribeLogGroups #
# # # # #
# # # # # logs:DescribeLogStreams #
# # # # #
# # # # # logs:FilterLogEvents #
# # # # #
# # # # # logs:GetLogEvents #
# # # # # logs:PutLogEvents #
# # # # #
#####
# + # arn:aws:logs:${AWS::Region} # Allow # logs:CreateLogGroup # AWS:
${LambdaFunctionService # #
# # :${AWS::AccountId}:log-grou # # logs:CreateLogStream # Role}
# # # # #
# # # p:/aws/lambda/* # # logs:PutLogEvents #
# # # # #
#####

```

(NOTE: There may be security-related changes not in this list. See <https://github.com/aws/aws-cdk/issues/1299>)

## Parameters

[+] Parameter AssetParameters/

ba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340a/S3Bucket

AssetParametersba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340aS3Bucket9780A3B

{"Type":"String","Description":"S3 bucket for asset

\\"ba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340a\\""}

[+] Parameter AssetParameters/

ba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340a/S3VersionKey

AssetParametersba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340aS3VersionKey37F

{"Type":"String","Description":"S3 key for asset version

\\"ba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340a\\""}

[+] Parameter AssetParameters/

ba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340a/ArtifactHash

AssetParametersba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340aArtifactHash801

{"Type":"String","Description":"Artifact hash for asset

\\"ba91444ebd644d9419e8cfee417f3aaa728507dd428788a2fc40574646c4340a\\""}

## Conditions

[+] Condition CDKMetadataAvailable: {"Fn::Or":[{"Fn::Or":[{"Fn::Equals":

[{"Ref":"AWS::Region"},"ap-east-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"ap-

northeast-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"ap-northeast-2"]}, {"Fn::Equals":

[{"Ref":"AWS::Region"},"ap-south-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"ap-

southeast-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"ap-southeast-2"]}, {"Fn::Equals":

[{"Ref":"AWS::Region"},"ca-central-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"cn-

north-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"cn-northwest-1"]},

{"Fn::Equals":[{"Ref":"AWS::Region"},"eu-central-1"]}], {"Fn::Or":[{"Fn::Equals":

[{"Ref":"AWS::Region"},"eu-north-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"eu-

west-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"eu-west-2"]}, {"Fn::Equals":

[{"Ref":"AWS::Region"},"eu-west-3"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"me-

south-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"sa-east-1"]}, {"Fn::Equals":

[{"Ref":"AWS::Region"},"us-east-1"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"us-

east-2"]}, {"Fn::Equals":[{"Ref":"AWS::Region"},"us-west-1"]}, {"Fn::Equals":

[{"Ref":"AWS::Region"},"us-west-2"]}]}}}]

## Resources

[+] AWS::Logs::LogGroup ApiGatewayToLambda/ApiAccessLogGroup

ApiGatewayToLambdaApiAccessLogGroupE2B41502

[+] AWS::IAM::Role LambdaFunctionServiceRole LambdaFunctionServiceRole0C4CDE0B

[+] AWS::Lambda::Function LambdaFunction LambdaFunctionBF21E41F

[+] AWS::ApiGateway::RestApi RestApi RestApi0C43BF4B

[+] AWS::ApiGateway::Deployment RestApi/Deployment

RestApiDeployment180EC503d2c6df3c8dc8b7193b98c1a0bff4e677

[+] AWS::ApiGateway::Stage RestApi/DeploymentStage.prod

RestApiDeploymentStageprod3855DE66

```
[+] AWS::ApiGateway::Resource RestApi/Default/{proxy+} RestApiproxyC95856DD
[+] AWS::Lambda::Permission RestApi/Default/{proxy+}/ANY/
ApiPermission.HelloConstructsStackRestApiFDB18C2E.ANY..{proxy+}
RestApiproxyANYApiPermissionHelloConstructsStackRestApiFDB18C2EANYproxyE43D39B3
[+] AWS::Lambda::Permission RestApi/Default/{proxy+}/ANY/
ApiPermission.Test.HelloConstructsStackRestApiFDB18C2E.ANY..{proxy+}
RestApiproxyANYApiPermissionTestHelloConstructsStackRestApiFDB18C2EANYproxy0B23CDC7
[+] AWS::ApiGateway::Method RestApi/Default/{proxy+}/ANY RestApiproxyANY1786B242
[+] AWS::Lambda::Permission RestApi/Default/ANY/
ApiPermission.HelloConstructsStackRestApiFDB18C2E.ANY..
RestApiANYApiPermissionHelloConstructsStackRestApiFDB18C2EANY5684C1E6
[+] AWS::Lambda::Permission RestApi/Default/ANY/
ApiPermission.Test.HelloConstructsStackRestApiFDB18C2E.ANY..
RestApiANYApiPermissionTestHelloConstructsStackRestApiFDB18C2EANY81DBDF56
[+] AWS::ApiGateway::Method RestApi/Default/ANY RestApiANYA7C1DC94
[+] AWS::ApiGateway::UsagePlan RestApi/UsagePlan RestApiUsagePlan6E1C537A
[+] AWS::Logs::LogGroup ApiAccessLogGroup ApiAccessLogGroupCEA70788
[+] AWS::IAM::Role LambdaRestApiCloudWatchRole LambdaRestApiCloudWatchRoleF339D4E6
[+] AWS::ApiGateway::Account LambdaRestApiAccount LambdaRestApiAccount
```

#### Outputs

```
[+] Output RestApi/Endpoint RestApiEndpoint0551178A: {"Value":{"Fn::Join":["",
["https://",{"Ref":"RestApi0C43BF4B"},".execute-api.",{"Ref":"AWS::Region"},".",
{"Ref":"AWS::URLSuffix"}],"/",{"Ref":"RestApiDeploymentStageprod3855DE66"},"/" ]}}
```

いいね この単純な例では、AWS Solutions Constructs の 1 つの優れたアーキテクチャのパターンを使用して、スタックに 21 個の新しいリソースが追加されました。

## CDK デプロイ

### Tip

Lambda 関数を含む最初の AWS CDK アプリケーションをデプロイする前に、AWS 環境をブートストラップする必要があります。これにより、AWS CDK がアセットを含むスタックをデプロイするために使用するステージングバケットが作成されます。AWS CDK を使用してアセットをデプロイするのが初めての場合は、[cdk bootstrap](#)をクリックして、CDK ツールキットスタックを AWS 環境にデプロイします。

デプロイの準備が整いました？

```
cdk deploy
```

## スタック出力

デプロイが完了すると、次の行が表示されます。

```
Outputs:  
HelloConstructsStack.RestApiEndpoint0551178A = https://xxxxxxxxxx.execute-api.us-  
east-1.amazonaws.com/prod/
```

これは、AWS Solutions Constructs パターンによって自動的に追加されるスタック出力で、API Gateway エンドポイントの URL が含まれます。

## アプリのテスト

このエンドポイントをcurl。URLをコピーして実行します ( 接頭辞と地域が異なる可能性があります )。

```
curl https://xxxxxxxxxx.execute-api.us-east-1.amazonaws.com/prod/
```

出力は次のようになります。

```
Hello, AWS Solutions Constructs! You've hit /
```

これがあなたが受け取った出力であれば、あなたのアプリは動作します！

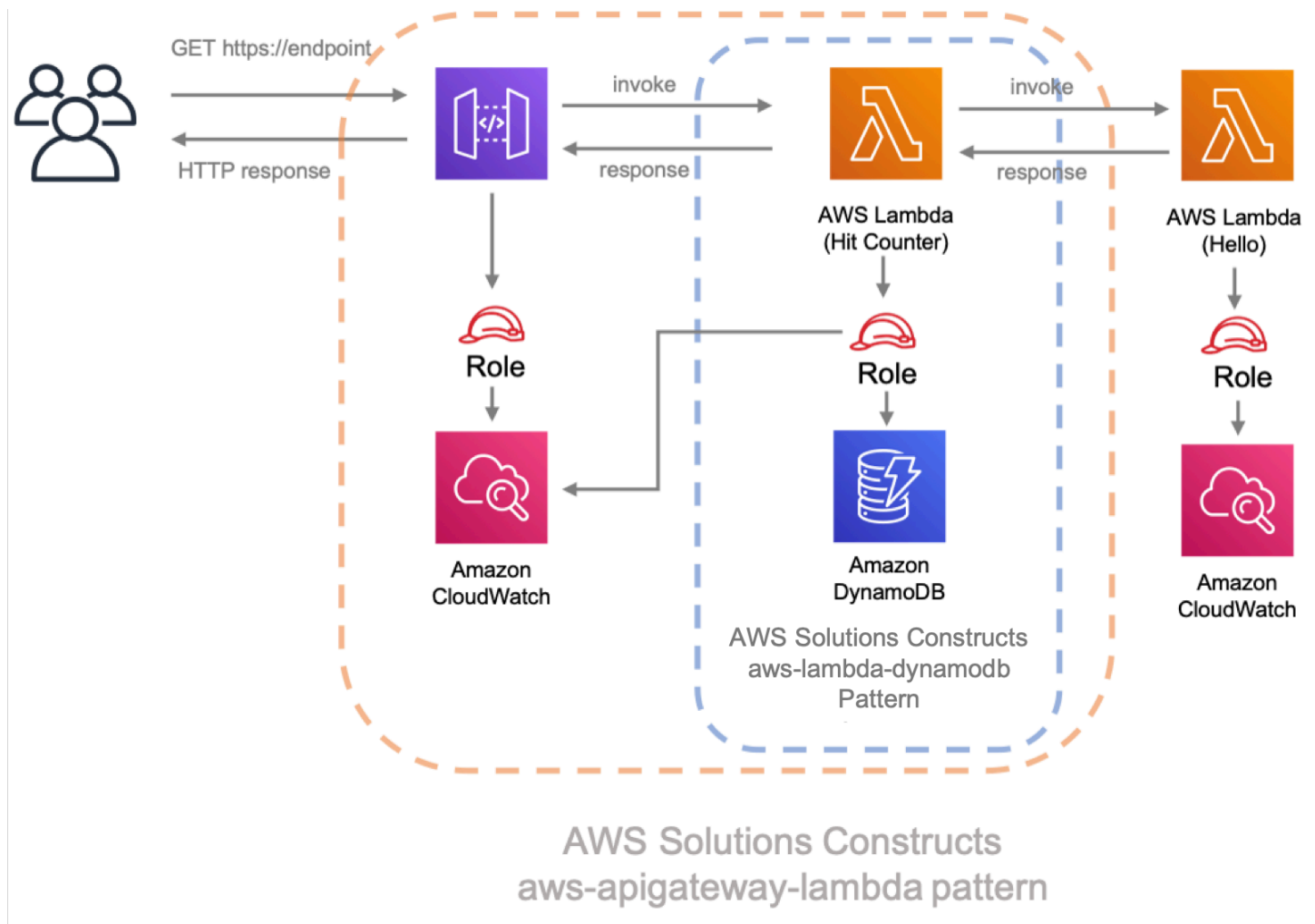
## ウォークスルー-パート 2

### Note

AWS ソリューションコンストラクトは、AWS CDK バージョン 1.46.0 以上でサポートされています。



このチュートリアルでは、で作成された「Hello Constructs」アプリを変更する方法について説明します。[パート 1](#)。変更により、AWS ソリューションコンストラクトから AWS Lambda から DynamoDB へのパターンを使用してサイトヒットカウンタが追加されます。Hello コンストラクトアプリを変更すると、次の解決策が得られます。



## ヒットカウンタ Lambda コード

まず、ヒットカウンタ AWS Lambda 関数のコードを記述します。この関数は、以下を実行します。

- は、Amazon DynamoDB テーブルの API パスに関連するカウンタをインクリメントします。
- ダウンストリームの Hello AWS Lambda 関数を呼び出します。
- を返し、エンドユーザーにレスポンスを返します。

## TypeScript

という名前のファイルを追加するlambda/hitcounter.js項目の変更後:

```
const { DynamoDB, Lambda } = require('aws-sdk');

exports.handler = async function(event) {
  console.log("request:", JSON.stringify(event, undefined, 2));

  // create AWS SDK clients
  const dynamo = new DynamoDB();
  const lambda = new Lambda();

  // update dynamo entry for "path" with hits++
  await dynamo.updateItem({
    TableName: process.env.DDB_TABLE_NAME,
    Key: { path: { S: event.path } },
    UpdateExpression: 'ADD hits :incr',
    ExpressionAttributeValues: { ':incr': { N: '1' } }
  }).promise();

  // call downstream function and capture response
  const resp = await lambda.invoke({
    FunctionName: process.env.DOWNSTREAM_FUNCTION_NAME,
    Payload: JSON.stringify(event)
  }).promise();

  console.log('downstream response:', JSON.stringify(resp, undefined, 2));

  // return response back to upstream caller
  return JSON.parse(resp.Payload);
};
```

## Python

という名前のファイルを追加するlambda/hitcounter.py項目の変更後:

```
import json
import os
import boto3
```

```
ddb = boto3.resource('dynamodb')
table = ddb.Table(os.environ['DDB_TABLE_NAME'])
_lambda = boto3.client('lambda')

def handler(event, context):
    print('request: {}'.format(json.dumps(event)))
    table.update_item(
        Key={'path': event['path']},
        UpdateExpression='ADD hits :incr',
        ExpressionAttributeValues={':incr': 1}
    )

    resp = _lambda.invoke(
        FunctionName=os.environ['DOWNSTREAM_FUNCTION_NAME'],
        Payload=json.dumps(event),
    )

    body = resp['Payload'].read()

    print('downstream response: {}'.format(body))
    return json.loads(body)
```

## 新しい依存関係をインストールする

### Note

AWS ソリューション構築と AWS CDK の両方で使用する正しい一致するバージョンを `VERSION_NUMBER` 各コマンドのプレースホルダフィールドです。これは、このウォークスルーの最初の部分で依存関係に使用されるバージョン番号と同じである必要があります。パッケージ間でバージョンが一致しないと、エラーが発生する可能性があります。

いつものように、まずソリューションのアップデートに必要な依存関係をインストールする必要があります。まず、DynamoDB コンストラクティブライブラリをインストールします。

### TypeScript

```
npm install -s @aws-cdk/aws-dynamodb@VERSION_NUMBER
```

## Python

```
pip install aws_cdk.aws_dynamodb==VERSION_NUMBER
```

最後に、AWS ソリューションコンストラクトをインストールします。aws-lambda-dynamodbモジュールとそのすべての依存関係をプロジェクトに追加します。

## TypeScript

```
npm install -s @aws-solutions-constructs/aws-lambda-dynamodb@VERSION_NUMBER
```

## Python

```
pip install aws_solutions_constructs.aws_lambda_dynamodb==VERSION_NUMBER
```

## リソースを定義します。

それでは、新しいアーキテクチャに対応するためにスタックコードを更新してみましょう。

まず、新しい依存関係をインポートし、「Hello」関数をaws-apigateway-lambdaパート1で作成したパターンです。

## TypeScript

ファイルを編集します。lib/hello-constructs.ts項目の変更後:

```
import * as cdk from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';
import * as api from '@aws-cdk/aws-apigateway';
import * as dynamodb from '@aws-cdk/aws-dynamodb';
```

```
import { ApiGatewayToLambda, ApiGatewayToLambdaProps } from '@aws-solutions-constructs/aws-apigateway-lambda';
import { LambdaToDynamoDB, LambdaToDynamoDBProps } from '@aws-solutions-constructs/aws-lambda-dynamodb';

export class HelloConstructsStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The code that defines your stack goes here

    const helloFunc = new lambda.Function(this, 'HelloHandler', {
      runtime: lambda.Runtime.NODEJS_12_X,
      code: lambda.Code.fromAsset('lambda'),
      handler: 'hello.handler'
    });

    const api_lambda_props: ApiGatewayToLambdaProps = {
      lambdaFunctionProps: {
        code: lambda.Code.fromAsset('lambda'),
        runtime: lambda.Runtime.NODEJS_12_X,
        handler: 'hello.handler'
      },
      apiGatewayProps: {
        defaultMethodOptions: {
          authorizationType: api.AuthorizationType.NONE
        }
      }
    };

    new ApiGatewayToLambda(this, 'ApiGatewayToLambda', api_lambda_props);
  }
}
```

## Python

ファイルを編集します。hello\_constructs/hello\_constructs\_stack.py項目の変更後:

```
from aws_cdk import (
    aws_lambda as _lambda,
    aws_apigateway as apigw,
    aws_dynamodb as ddb,
```

```
    core,
)

from aws_solutions_constructs import (
    aws_apigateway_lambda as apigw_lambda,
    aws_lambda_dynamodb as lambda_ddb
)

class HelloConstructsStack(core.Stack):

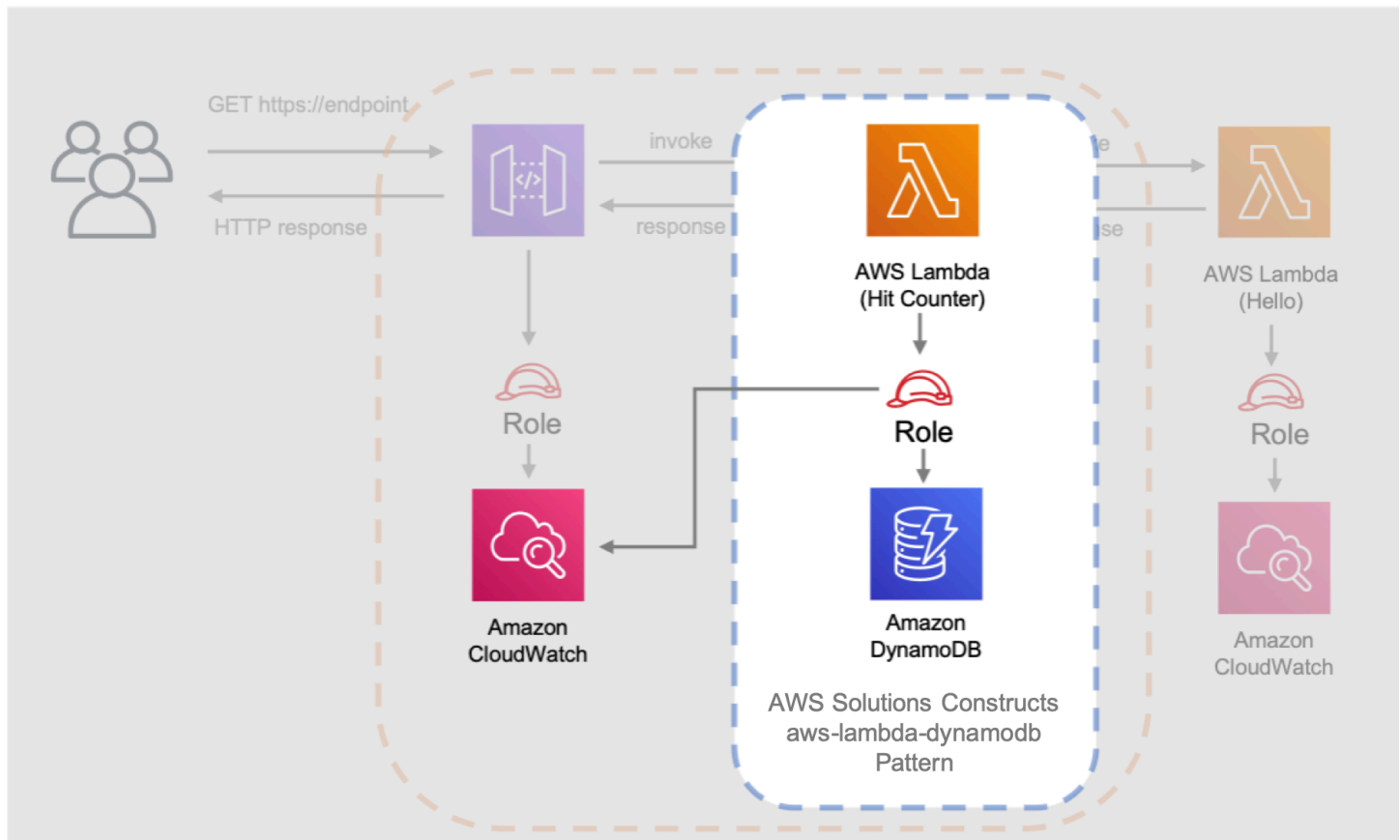
    def __init__(self, scope: core.Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # The code that defines your stack goes here

        self._handler = _lambda.Function(
            self, 'HelloHandler',
            runtime=_lambda.Runtime.PYTHON_3_7,
            handler='hello.handler',
            code=_lambda.Code.asset('lambda'),
        )

        apigw_lambda.ApiGatewayToLambda(
            self, 'ApiGatewayToLambda',
            lambda_function_props=_lambda.FunctionProps(
                runtime=_lambda.Runtime.PYTHON_3_7,
                code=_lambda.Code.asset('lambda'),
                handler='hello.handler',
            ),
            api_gateway_props=apigw.RestApiProps(
                default_method_options=apigw.MethodOptions(
                    authorization_type=apigw.AuthorizationType.NONE
                )
            )
        )
)
```

次に、我々は追加しようとしていますaws-lambda-dynamodbパターンを使用して、更新されたアーキテクチャのヒットカウンターサービスを構築します。



### AWS Solutions Constructs aws-apigateway-lambda pattern

以下の次の更新では、`aws-lambda-dynamodb`パターンを使用して AWS Lambda 関数をヒットカウンターハンドラで定義します。さらに、Amazon DynamoDB テーブルはHitsと、パーティションキーpath。

#### TypeScript

ファイルを編集します。lib/hello-constructs.ts項目の変更後:

```
import * as cdk from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';
import * as api from '@aws-cdk/aws-apigateway';
import * as dynamodb from '@aws-cdk/aws-dynamodb';
import { ApiGatewayToLambda, ApiGatewayToLambdaProps } from '@aws-solutions-constructs/aws-apigateway-lambda';
import { LambdaToDynamoDB, LambdaToDynamoDBProps } from '@aws-solutions-constructs/aws-lambda-dynamodb';
```

```
export class HelloConstructsStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The code that defines your stack goes here

    const helloFunc = new lambda.Function(this, 'HelloHandler', {
      runtime: lambda.Runtime.NODEJS_12_X,
      code: lambda.Code.fromAsset('lambda'),
      handler: 'hello.handler'
    });

    // hit counter, aws-lambda-dynamodb pattern
    const lambda_ddb_props: LambdaToDynamoDBProps = {
      lambdaFunctionProps: {
        code: lambda.Code.asset(`lambda`),
        runtime: lambda.Runtime.NODEJS_12_X,
        handler: 'hitcounter.handler',
        environment: {
          DOWNSTREAM_FUNCTION_NAME: helloFunc.functionName
        }
      },
      dynamoTableProps: {
        tableName: 'Hits',
        partitionKey: { name: 'path', type: dynamodb.AttributeType.STRING }
      }
    };

    const hitcounter = new LambdaToDynamoDB(this, 'LambdaToDynamoDB',
    lambda_ddb_props);

    const api_lambda_props: ApiGatewayToLambdaProps = {
      lambdaFunctionProps: {
        code: lambda.Code.fromAsset('lambda'),
        runtime: lambda.Runtime.NODEJS_12_X,
        handler: 'hello.handler'
      },
      apiGatewayProps: {
        defaultMethodOptions: {
          authorizationType: api.AuthorizationType.NONE
        }
      }
    };
  };
};
```



```
    new ApiGatewayToLambda(this, 'ApiGatewayToLambda', api_lambda_props);
  }
}
```

## Python

ファイルを編集します。hello\_constructs/hello\_constructs\_stack.py項目の変更後:

```
from aws_cdk import (
    aws_lambda as _lambda,
    aws_apigateway as apigw,
    aws_dynamodb as ddb,
    core,
)

from aws_solutions_constructs import (
    aws_apigateway_lambda as apigw_lambda,
    aws_lambda_dynamodb as lambda_ddb
)

class HelloConstructsStack(core.Stack):

    def __init__(self, scope: core.Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # The code that defines your stack goes here

        self.hello_func = _lambda.Function(
            self, 'HelloHandler',
            runtime=_lambda.Runtime.PYTHON_3_7,
            handler='hello.handler',
            code=_lambda.Code.asset('lambda'),
        )

        # hit counter, aws-lambda-dynamodb pattern
        self.hit_counter = lambda_ddb.LambdaToDynamoDB(
            self, 'LambdaToDynamoDB',
            lambda_function_props=_lambda.FunctionProps(
                runtime=_lambda.Runtime.PYTHON_3_7,
                code=_lambda.Code.asset('lambda'),
                handler='hitcounter.handler',
```

```
        environment={
            'DOWNSTREAM_FUNCTION_NAME': self.hello_func.function_name
        }
    ),
    dynamo_table_props=ddb.TableProps(
        table_name='Hits',
        partition_key={
            'name': 'path',
            'type': ddb.AttributeType.STRING
        }
    )
)

apigw_lambda.ApiGatewayToLambda(
    self, 'ApiGatewayToLambda',
    lambda_function_props=_lambda.FunctionProps(
        runtime=_lambda.Runtime.PYTHON_3_7,
        code=_lambda.Code.asset('lambda'),
        handler='hello.handler',
    ),
    api_gateway_props=apigw.RestApiProps(
        default_method_options=apigw.MethodOptions(
            authorization_type=apigw.AuthorizationType.NONE
        )
    )
)
```

次に、Hit Counter 関数を付与する必要があります。aws-lambda-dynamodbパターンは、Hello 関数を呼び出すための権限の上に追加されました。

## TypeScript

ファイルを編集します。lib/hello-constructs.ts項目の変更後:

```
import * as cdk from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';
import * as api from '@aws-cdk/aws-apigateway';
import * as dynamodb from '@aws-cdk/aws-dynamodb';
import { ApiGatewayToLambda, ApiGatewayToLambdaProps } from '@aws-solutions-constructs/aws-apigateway-lambda';
```

```
import { LambdaToDynamoDB, LambdaToDynamoDBProps } from '@aws-solutions-constructs/
aws-lambda-dynamodb';

export class HelloConstructsStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The code that defines your stack goes here

    // hello function responding to http requests
    const helloFunc = new lambda.Function(this, 'HelloHandler', {
      runtime: lambda.Runtime.NODEJS_12_X,
      code: lambda.Code.fromAsset('lambda'),
      handler: 'hello.handler'
    });

    // hit counter, aws-lambda-dynamodb pattern
    const lambda_ddb_props: LambdaToDynamoDBProps = {
      lambdaFunctionProps: {
        code: lambda.Code.asset(`lambda`),
        runtime: lambda.Runtime.NODEJS_12_X,
        handler: 'hitcounter.handler',
        environment: {
          DOWNSTREAM_FUNCTION_NAME: helloFunc.functionName
        }
      },
      dynamoTableProps: {
        tableName: 'Hits',
        partitionKey: { name: 'path', type: dynamodb.AttributeType.STRING }
      }
    };

    const hitcounter = new LambdaToDynamoDB(this, 'LambdaToDynamoDB',
lambda_ddb_props);

    // grant the hitcounter lambda role invoke permissions to the hello function
    helloFunc.grantInvoke(hitcounter.lambdaFunction);

    const api_lambda_props: ApiGatewayToLambdaProps = {
      lambdaFunctionProps: {
        code: lambda.Code.fromAsset('lambda'),
        runtime: lambda.Runtime.NODEJS_12_X,
        handler: 'hello.handler'
      },
```

```
    apiGatewayProps: {
      defaultMethodOptions: {
        authorizationType: api.AuthorizationType.NONE
      }
    }
  };

  new ApiGatewayToLambda(this, 'ApiGatewayToLambda', api_lambda_props);
}
}
```

## Python

ファイルを編集します。hello\_constructs/hello\_constructs\_stack.py項目の変更後:

```
from aws_cdk import (
    aws_lambda as _lambda,
    aws_apigateway as apigw,
    aws_dynamodb as ddb,
    core,
)

from aws_solutions_constructs import (
    aws_apigateway_lambda as apigw_lambda,
    aws_lambda_dynamodb as lambda_ddb
)

class HelloConstructsStack(core.Stack):

    def __init__(self, scope: core.Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # The code that defines your stack goes here

        self.hello_func = _lambda.Function(
            self, 'HelloHandler',
            runtime=_lambda.Runtime.PYTHON_3_7,
            handler='hello.handler',
            code=_lambda.Code.asset('lambda'),
        )

        # hit counter, aws-lambda-dynamodb pattern
```

```
self.hit_counter = lambda_ddb.LambdaToDynamoDB(
    self, 'LambdaToDynamoDB',
    lambda_function_props=_lambda.FunctionProps(
        runtime=_lambda.Runtime.PYTHON_3_7,
        code=_lambda.Code.asset('lambda'),
        handler='hitcounter.handler',
        environment={
            'DOWNSTREAM_FUNCTION_NAME': self.hello_func.function_name
        }
    ),
    dynamo_table_props=ddb.TableProps(
        table_name='Hits',
        partition_key={
            'name': 'path',
            'type': ddb.AttributeType.STRING
        }
    )
)

# grant the hitcounter lambda role invoke permissions to the hello function
self.hello_func.grant_invoke(self.hit_counter.lambda_function)

apigw_lambda.ApiGatewayToLambda(
    self, 'ApiGatewayToLambda',
    lambda_function_props=_lambda.FunctionProps(
        runtime=_lambda.Runtime.PYTHON_3_7,
        code=_lambda.Code.asset('lambda'),
        handler='hello.handler',
    ),
    api_gateway_props=apigw.RestApiProps(
        default_method_options=apigw.MethodOptions(
            authorization_type=apigw.AuthorizationType.NONE
        )
    )
)
```

最後に、オリジナルのaws-apigateway-lambdaパターンを使用してプロビジョニングされた新しいヒットカウンター関数を利用するには、aws-lambda-dynamodbpattern

## TypeScript

ファイルを編集します。lib/hello-constructs.ts項目の変更後:

```
import * as cdk from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';
import * as api from '@aws-cdk/aws-apigateway';
import * as dynamodb from '@aws-cdk/aws-dynamodb';
import { ApiGatewayToLambda, ApiGatewayToLambdaProps } from '@aws-solutions-constructs/aws-apigateway-lambda';
import { LambdaToDynamoDB, LambdaToDynamoDBProps } from '@aws-solutions-constructs/aws-lambda-dynamodb';

export class HelloConstructsStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The code that defines your stack goes here

    // hello function responding to http requests
    const helloFunc = new lambda.Function(this, 'HelloHandler', {
      runtime: lambda.Runtime.NODEJS_12_X,
      code: lambda.Code.fromAsset('lambda'),
      handler: 'hello.handler'
    });

    // hit counter, aws-lambda-dynamodb pattern
    const lambda_ddb_props: LambdaToDynamoDBProps = {
      lambdaFunctionProps: {
        code: lambda.Code.asset(`lambda`),
        runtime: lambda.Runtime.NODEJS_12_X,
        handler: 'hitcounter.handler',
        environment: {
          DOWNSTREAM_FUNCTION_NAME: helloFunc.functionName
        }
      },
      dynamoTableProps: {
        tableName: 'Hits',
        partitionKey: { name: 'path', type: dynamodb.AttributeType.STRING }
      }
    };
  }
};
```

```
const hitcounter = new LambdaToDynamoDB(this, 'LambdaToDynamoDB',
lambda_ddb_props);

// grant the hitcounter lambda role invoke permissions to the hello function
helloFunc.grantInvoke(hitcounter.lambdaFunction);

const api_lambda_props: ApiGatewayToLambdaProps = {
  existingLambdaObj: hitcounter.lambdaFunction,
  apiGatewayProps: {
    defaultMethodOptions: {
      authorizationType: api.AuthorizationType.NONE
    }
  }
};

new ApiGatewayToLambda(this, 'ApiGatewayToLambda', api_lambda_props);
}
```

## Python

ファイルを編集します。hello\_constructs/hello\_constructs\_stack.py項目の変更後:

```
from aws_cdk import (
    aws_lambda as _lambda,
    aws_apigateway as apigw,
    aws_dynamodb as ddb,
    core,
)

from aws_solutions_constructs import (
    aws_apigateway_lambda as apigw_lambda,
    aws_lambda_dynamodb as lambda_ddb
)

class HelloConstructsStack(core.Stack):

    def __init__(self, scope: core.Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # The code that defines your stack goes here
```

```
self.hello_func = _lambda.Function(
    self, 'HelloHandler',
    runtime=_lambda.Runtime.PYTHON_3_7,
    handler='hello.handler',
    code=_lambda.Code.asset('lambda'),
)

# hit counter, aws-lambda-dynamodb pattern
self.hit_counter = lambda_ddb.LambdaToDynamoDB(
    self, 'LambdaToDynamoDB',
    lambda_function_props=_lambda.FunctionProps(
        runtime=_lambda.Runtime.PYTHON_3_7,
        code=_lambda.Code.asset('lambda'),
        handler='hitcounter.handler',
        environment={
            'DOWNSTREAM_FUNCTION_NAME': self.hello_func.function_name
        }
    ),
    dynamo_table_props=ddb.TableProps(
        table_name='Hits',
        partition_key={
            'name': 'path',
            'type': ddb.AttributeType.STRING
        }
    )
)

# grant the hitcounter lambda role invoke permissions to the hello function
self.hello_func.grant_invoke(self.hit_counter.lambda_function)

apigw_lambda.ApiGatewayToLambda(
    self, 'ApiGatewayToLambda',
    existing_lambda_obj=self.hit_counter.lambda_function,
    api_gateway_props=apigw.RestApiProps(
        default_method_options=apigw.MethodOptions(
            authorization_type=apigw.AuthorizationType.NONE
        )
    )
)
```



## 変更の確認

プロジェクトを構築し、これを展開するときに発生するリソースの変更を確認してみましょう。

```
npm run build
cdk diff
```

出力は次のようになります。

```
Stack HelloConstructsStack
IAM Statement Changes
#####
# # Resource # Effect # Action #
Principal # Condition #
#####
# + # ${HelloHandler.Arn} # Allow # lambda:InvokeFunction #
AWS:${LambdaFunctionServiceRole} # #
#####
# + # ${HelloHandler/ServiceRole.Arn} # Allow # sts:AssumeRole #
Service:lambda.amazonaws.com # #
#####
# + # ${LambdaToDynamoDB/DynamoTable.Ar # Allow # dynamodb:BatchGetItem #
AWS:${LambdaFunctionServiceRole} # #
# # n} # # dynamodb:BatchWriteItem #
# # # # dynamodb:DeleteItem #
# # # # dynamodb:GetItem #
# # # # dynamodb:GetRecords #
# # # # dynamodb:GetShardIterator #
# # # # dynamodb:PutItem #
# # # # dynamodb:Query #
# # # # dynamodb:Scan #
# # # # dynamodb:UpdateItem #
# # # #
```

```

#####
IAM Policy Changes
#####
# # Resource # Managed Policy ARN
#
#####
# + # ${HelloHandler/ServiceRole} # arn:${AWS::Partition}:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole #
#####
(NOTE: There may be security-related changes not in this list. See https://github.com/
aws/aws-cdk/issues/1299)

Resources
[+] AWS::IAM::Role HelloHandler/ServiceRole HelloHandlerServiceRole11EF7C63
[+] AWS::Lambda::Function HelloHandler HelloHandler2E4FBA4D
[+] AWS::DynamoDB::Table LambdaToDynamoDB/DynamoTable
LambdaToDynamoDBDynamoTable53C1442D
[+] AWS::IAM::Policy LambdaFunctionServiceRole/DefaultPolicy
LambdaFunctionServiceRoleDefaultPolicy126C8897
[~] AWS::Lambda::Function LambdaFunction LambdaFunctionBF21E41F
## [+] Environment
# ## {"Variables":{"DOWNSTREAM_FUNCTION_NAME":
{"Ref":"HelloHandler2E4FBA4D"},"DDB_TABLE_NAME":
{"Ref":"LambdaToDynamoDBDynamoTable53C1442D"}}}
## [~] Handler
# ## [-] hello.handler
# ## [+] hitcounter.handler
## [~] DependsOn
## @@ -1,3 +1,4 @@
[ ] [
[+] "LambdaFunctionServiceRoleDefaultPolicy126C8897",
[ ] "LambdaFunctionServiceRole0C4CDE0B"
[ ] ]

```

## CDK デプロイ

さて、デプロイの準備が整いました？

```
cdk deploy
```

## スタック出力

デプロイが完了すると、次の行が表示されます。

```
Outputs:  
HelloConstructsStack.RestApiEndpoint0551178A = https://xxxxxxxxxx.execute-api.us-  
east-1.amazonaws.com/prod/
```

## アプリのテスト

カールでこのエンドポイントをヒットしようとしましょう。URLをコピーして実行します (プレフィックスとリージョンが異なる可能性があります)。

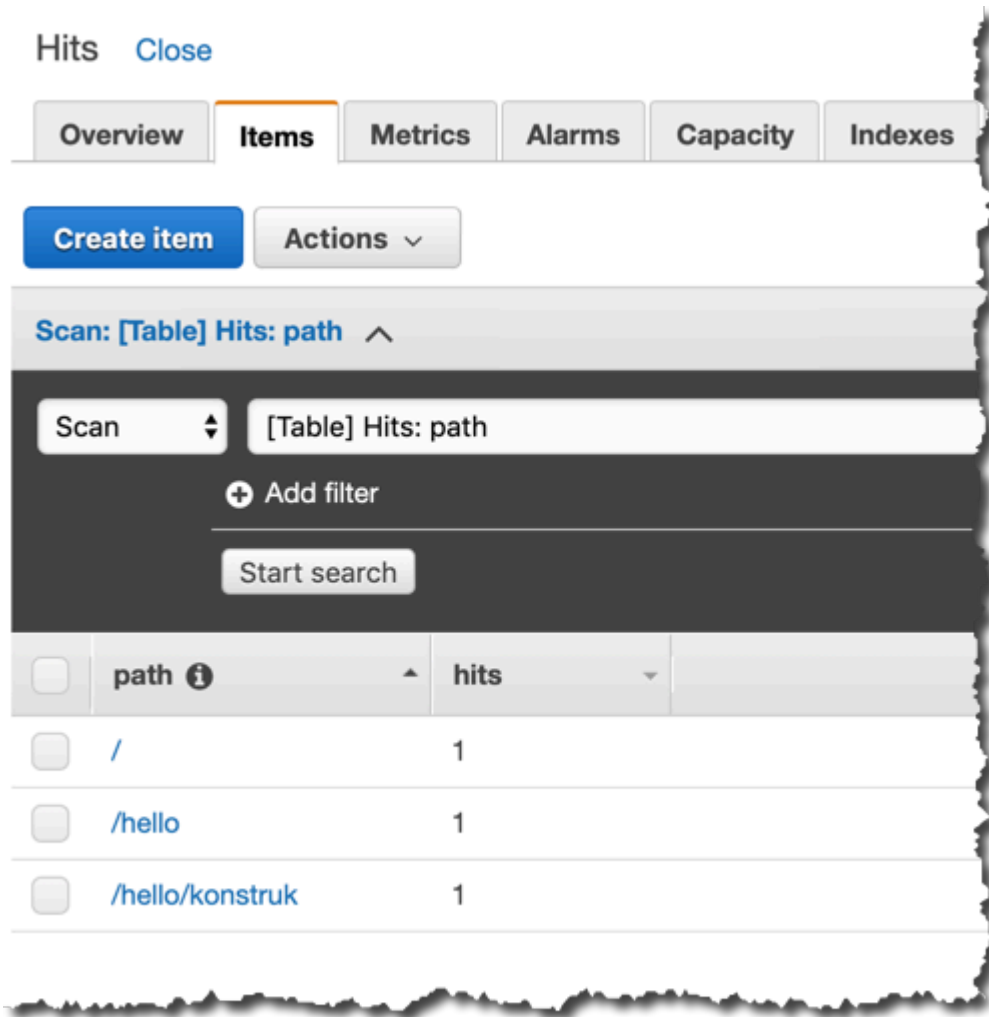
```
curl https://xxxxxxxxxx.execute-api.us-east-1.amazonaws.com/prod/
```

出力は次のようになります。

```
Hello, AWS Solutions Constructs! You've hit /
```

さて、以下を実行します。HitsAmazon DynamoDB テーブル。

1. DynamoDB コンソールに移動します。
2. テーブルを作成したリージョンにいることを確認します。
3. Selectテーブル項目の変更後、Hitsテーブル。
4. テーブルを開き、「アイテム」を選択します。
5. パスごとにヒット数が表示されるはずですが。



6. 新しいパスを押して、項目ビューを更新してみてください。新しい項目が表示されます。hits1 のカウント。

これがあなたが受け取った出力であれば、あなたのアプリは動作します！

## サンプルユースケース

このライブラリには、Constructs アーキテクチャパターンの使用方法を示す機能ユースケース実装のコレクションが含まれています。これらはアーキテクチャパターンと同じ方法で使用ことができ、それらのパターンの追加の「高レベル」抽象化として概念化することができます。次のユースケースは、機能的な例として提供されています。

## AWS 静的 S3 ウェブサイト

このユースケースパターン ( aws-s3-static-website ) は、Amazon CloudFront デイストリビューション、Amazon S3 バケット、および AWS Lambda ベースのカスタムリソースを実装して、Wild Rydes デモウェブサイトの静的ウェブサイトコンテンツ ( aws-serverless-web-app実装

① ソースコード ( aws-s3-静的ウェブサイト )

[https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use\\_cases/aws-s3-static-website](https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use_cases/aws-s3-static-website)

## AWS シンプルサーバーレスイメージハンドラー


このユースケースパターン ( aws-serverless-image-handler ) は、Amazon CloudFront デイストリビューション、Amazon API Gateway REST API、AWS Lambda 関数、およびデプロイアカウント内の 1 つ以上の Amazon S3 バケットからイメージコンテンツを提供する機能イメージハンドラー API をプロビジョニングするために必要な権限/ロジックを実装します。

① ソースコード ( aws-サーバーレスイメージハンドラー )

[https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use\\_cases/aws-serverless-image-handler](https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use_cases/aws-serverless-image-handler)

## AWS サーバーレスウェブアプリ

このユースケースパターン ( aws-serverless-web-app ) は、ユーザーがWild Rydesフリートからユニコーン乗り物をリクエストできるようにするシンプルなサーバーレス Web アプリケーションを実装しています。アプリケーションは、取得したい場所を示すHTMLベースのユーザーインターフェイスをユーザーに提示し、リクエストを送信して近くのユニコーンをディスパッチするための RESTful Webサービスとバックエンドにインターフェースします。また、このアプリケーションは、ユーザーがサービスに登録し、乗り物をリクエストする前にログインするための施設を提供します。

 ソースコード ( aws-サーバーレス-web-app )

[https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use\\_cases/aws-serverless-web-app](https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use_cases/aws-serverless-web-app)

# API リファレンス

AWS Solutions Constructs ( Constructs ) は、AWS Cloud Development Kit ( AWS CDK ) のオープンソースの拡張機能で、マルチサービスの優れたアーキテクチャパターンを提供し、コード内でソリューションを迅速に定義し、予測可能で反復可能なインフラストラクチャを作成します。Constructsの目標は、開発者がアーキテクチャのパターンベースの定義を使用して任意のサイズのソリューションを構築するためのエクスペリエンスを加速することです。

Constructs で定義されるパターンは、高度な AWS CDK コンストラクトのマルチサービス抽象化であり、適切に設計されたベストプラクティスに基づくデフォルト設定を持ちます。ライブラリは、各アーキテクチャパターンモデルを作成するためのオブジェクト指向技術を使用して論理モジュールに編成されています。

CDK は次の言語で利用可能です。

- JavaScript, TypeScript (Node.js  $\geq$  10.3.0)
- Python (Python  $\geq$  3.6)
- Java (Java  $\geq$  1.8)

## Modules

AWS ソリューション構成は、いくつかのモジュールで構成されています。彼らは次のように命名されています：

- `aws-xxx`: 示されたサービスのためのよく設計されたパターンパッケージ。このパッケージには、指定されたパターンを設定するための複数の AWS CDK サービスモジュールを含むコンストラクトが含まれます。
- `xxx`: 起動しないパッケージ」aws「は、パターンライブラリ内で使用されるサービスのベストプラクティスのデフォルトを設定するために使用されるコアモジュールを構築します。

## Module の内容

Module には、次のタイプが含まれています。

- パターン-このライブラリ内のすべての上位レベル、マルチサービス構造。
- その他のタイプ-パターンをサポートするために存在するすべての非構造クラス、インターフェース、構造体および列挙型。

パターンはコンストラクタ内の ( 入力 ) プロパティのセットを取ります。プロパティのセット ( および必要なもの ) は、パターンのドキュメントページで見ることができます。

パターンのドキュメントページには、呼び出し可能なメソッドと、インスタンス化後にパターンに関する情報を取得するために使用できるプロパティもリストされています。


## aws-apigateway-ダイナモッド

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョン管理](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_apigateway_dynamodb</code>
 TypeScript	<code>@aws-solutions-constructs/aws-apigateway-dynamodb</code>
 Java	<code>software.amazon.awsconstructs.services.apigatewaydynamodb</code>

## Overview

この AWS ソリューション構築物は、Amazon DynamoDB テーブルに接続された Amazon API Gateway REST API を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。



```
import { ApiGatewayToDynamoDBProps, ApiGatewayToDynamoDB } from "@aws-solutions-constructs/aws-apigateway-dynamodb";

new ApiGatewayToDynamoDB(this, 'test-api-gateway-dynamodb-default', {});
```

## Initializer

```
new ApiGatewayToDynamoDB(scope: Construct, id: string, props:
  ApiGatewayToDynamoDBProps);
```

### パラメータ

- `scope` [Construct](#)
- `id` `string`
- `props` [ApiGatewayToDynamoDBProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ダイナモテーブル	<a href="#">dynamodb.TableProps</a>	DynamoDB テーブルのデフォルトの小道具をオーバーライドするオプションのユーザー提供の小道具です
ApigateWayProps ?	<a href="#">api.RestApiProps</a>	API Gateway のデフォルトの小道具を上書きするオプションのユーザー提供の小道具です。
作成操作の許可	boolean	DynamoDB テーブルで Create オペレーションの API Gateway メソッドをデプロイするかどうか。

名前	タイプ	説明
リクエストテンプレートの作成	string	作成メソッドの API Gateway 要求テンプレート。AllowCreateOperation が true に設定されている場合に必要です
読み取り操作を許可	boolean	DynamoDB テーブルで API Gateway の読み取り操作メソッドをデプロイするかどうか。
更新操作の許可	boolean	DynamoDB テーブルで、更新用の API Gateway メソッドをデプロイするかどうか。
更新要求テンプレート	string	更新メソッドの API Gateway 要求テンプレート。 AllowUpdateOperation が true に設定されている場合に必要
削除操作の許可	boolean	DynamoDB テーブルで、削除操作の API Gateway メソッドをデプロイするかどうか。
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループのデフォルト小道具を上書きする、オプションのユーザー指定の小道具です。

## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#">api.RestApi</a>	パターンによって作成された API Gateway REST API のインスタンスを返します。

名前	タイプ	説明
APIGatewayクラウドウォッチ ロール	<a href="#">iam.Role</a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロールのインスタンスを返します。
APIGateWayLogGroup	<a href="#">logs.LogGroup</a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。
APIGateWayRole	<a href="#">iam.Role</a>	API Gateway REST API のパターンによって作成された IAM ロールのインスタンスを返します。
ダイナモテーブル	<a href="#">dynamodb.Table</a>	パターンによって作成された DynamoDB テーブルのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

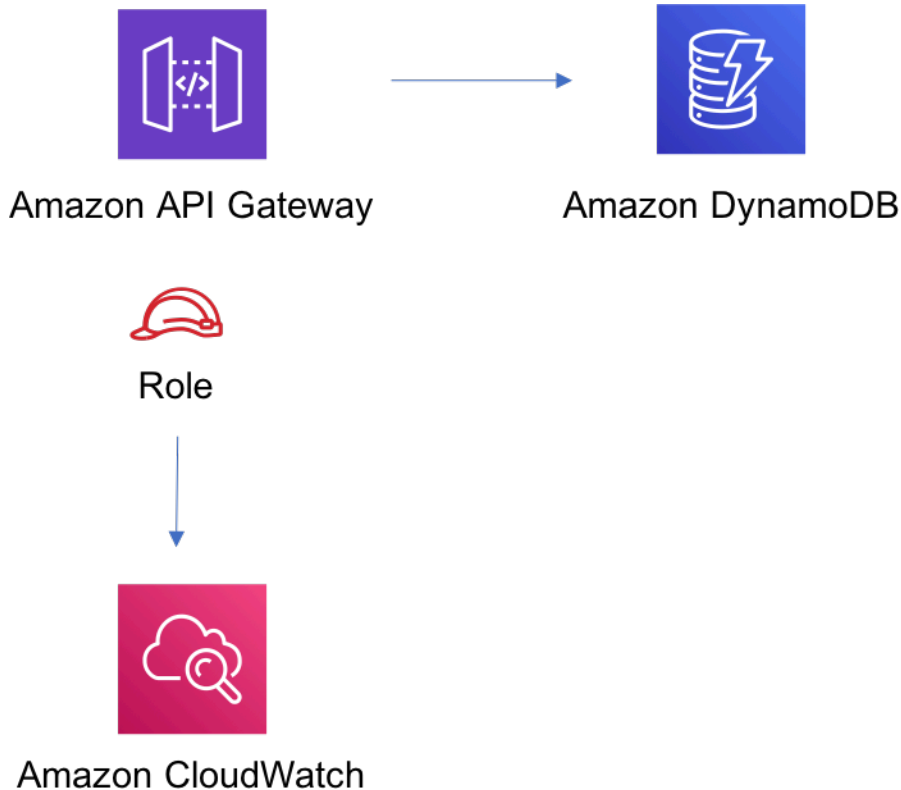
### Amazon API Gateway

- エッジ最適化 API エンドポイントのデプロイ
- API Gateway での CloudWatch によるロギングを有効にする
- API Gateway の最小権限アクセス IAM ロールを設定する
- すべての API メソッドのデフォルトの authorizationType を IAM に設定する
- X-Ray トレースを有効にする

## Amazon DynamoDB テーブル

- DynamoDB テーブルの請求モードをオンデマンドに設定する ( リクエストごとの支払い )
- AWS マネージド KMS キーを使用した DynamoDB テーブルのサーバー側の暗号化の有効化
- DynamoDB テーブルの 'id' という名前のパーティションキーを作成します。
- CloudFormation スタックを削除するときにテーブルを保持する
- 継続的なバックアップおよびポイントインタイムリカバリを有効にします

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-constructs/aws-apigateway-dynamodb](https://github.com/aws-solutions-constructs/aws-apigateway-dynamodb)



# aws-apigateway-iot

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理モデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_apigateway_iot</code>
 TypeScript	<code>@aws-solutions-constructs/aws-apigateway-iot</code>
 Java	<code>software.amazon.awsconstructs.services.apigatewayiot</code>

## Overview

この AWS ソリューション構築物は、AWS IoT パターンに接続された Amazon API Gateway REST API を実装します。

この構造体は、API Gateway と AWS IoT の間にスケーラブルな HTTPS プロキシを作成します。これは、MQTT または MQTT/WebSocket プロトコルをサポートしていないレガシーデバイスが AWS IoT プラットフォームと対話できるようにする場合に便利です。

この実装により、特定の MQTT トピックに書き込み専用メッセージが公開されるようになります。また、デバイスレジストリーで許可されているものに対する HTTPS デバイスのシャドウ更新もサポートされます。これは、メッセージをプロキシするための Lambda 関数を必要とせず、JSON

メッセージとバイナリメッセージの両方をサポートする直接 API Gateway から AWS IoT 統合に依存しています。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { ApiGatewayToIot } from '@aws-solutions-constructs/aws-apigateway-iot';

new ApiGatewayToIot(this, 'ApiGatewayToIotPattern', {
  iotEndpoint: 'a1234567890123-ats'
});
```

## Initializer

```
new ApiGatewayToIot(scope: Construct, id: string, props: ApiGatewayToIotProps);
```

## パラメータ

- [scopeConstruct](#)
- [idstring](#)
- [propsApiGatewayToIotProps](#)

## パターン構成プロパティ

名前	タイプ	説明
IoTendPoint	string	API Gateway を統合する AWS IoT エンドポイントサブドメイン ( a1234567890123-ats など )。
APIGateWayCreateApiKey ?	boolean	がに設定されている場合 true をクリックすると、API キーが作成され、U sagePlan に関連付けられます。RestApi にアクセスして

名前	タイプ	説明
		いる間、ユーザーは`x-api-key`ヘッダを指定する必要があります。デフォルト値はfalse。
ApigatewayExecutionRole ?	<a href="#">iam.Role</a>	AWS IoT にアクセスするために API Gateway によって使用される IAM ロール。指定しない場合、すべてのトピックと Thing へのワイルドカード (*) アクセス権を持つデフォルトのロールが作成されます。
ApigatewayProps ?	<a href="#">api.restApiProps</a>	API Gateway REST API のデフォルトの小道具をオーバーライドするオプションのユーザー提供の小道具です。
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループのデフォルトプロップを上書きする、オプションのユーザー指定のプロップ。

## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#">api.RestApi</a>	パターンによって作成された API Gateway REST API のインスタンスを返します。
APIGatewayクラウドウォッチ ロール	<a href="#">iam.Role</a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロール

名前	タイプ	説明
		ルのインスタンスを返します。
APIGatewayLogGroup	<a href="#">logs.LogGroup</a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。
APIGatewayRole	<a href="#">iam.Role</a>	API Gateway REST API のパターンによって作成された IAM ロールのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon API Gateway

- エッジ最適化 API エンドポイントのデプロイ
- API リソースをPOSTIoT トピックにメッセージを発行する方法
- API リソースをPOSTメッセージの発行方法ThingShadowおよびNamedShadows
- API Gateway での CloudWatch ログイングを有効にする
- すべてのトピックと Thing にアクセスできる API Gateway の IAM ロールを設定する
- すべての API メソッドのデフォルトの authorizationType を IAM に設定する
- X-Ray トレースを有効にする
- UsagePlan を作成し、prodstage

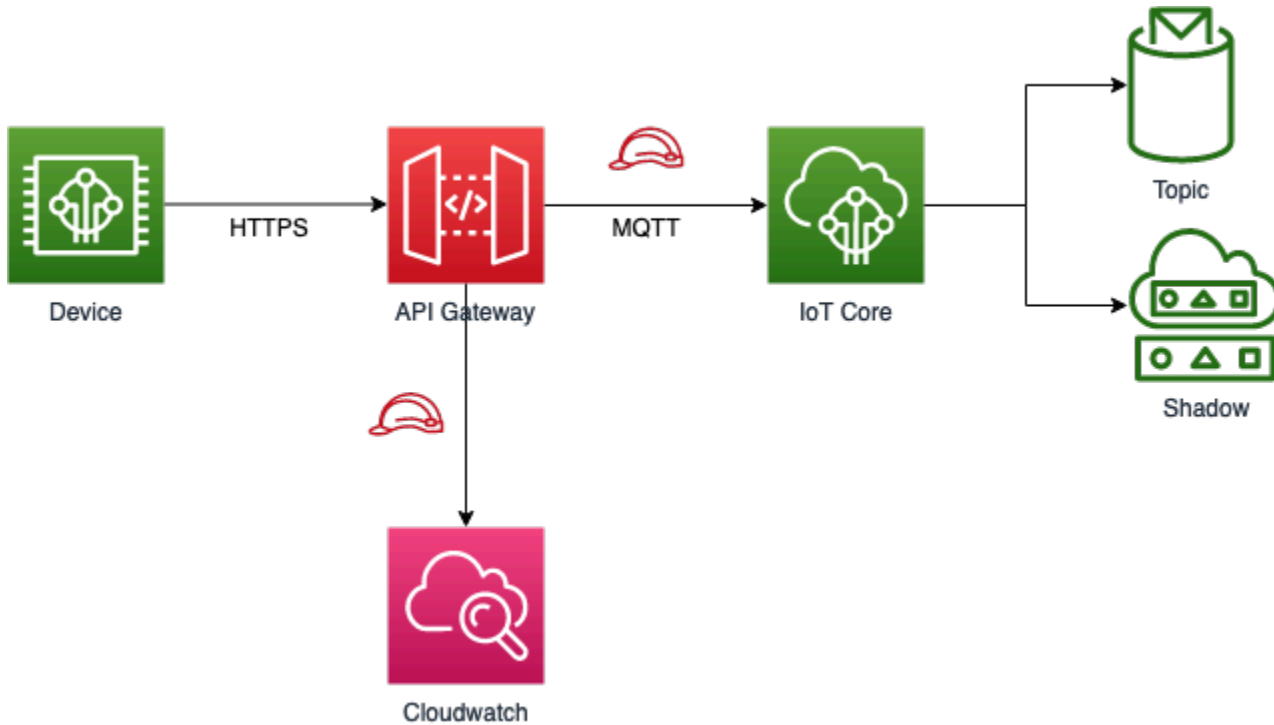
以下は、コンストラクトのデプロイ後に API Gateway によって公開されるさまざまなリソースとメソッドの説明です。フレームワークの使用の詳細については、[例](#)を使用してこれらのエンドポイントを簡単にテストする方法の詳細については、[curl](#)。



方法	リソース	クエリーパラメータ	戻りコード	説明
POST	/message/ <topics>	qos	200/403/500	このエンドポイントを呼び出すことで、公開したいトピックを渡す必要があります (例: <code>`/message/device/fo o `</code> )。
POST	/shadow/<thingName>	なし	200/403/500	このルートは、Thing のシャドードキュメントを更新することを可能にします。thingName 名前のない (クラシック) シャドウタイプを使用 ボディは、構成する標準的な影の漆喰を遵守しなければならないstateノードおよび関連するdesiredおよびreportedノードフレームワークの使用の詳細については、 <a href="#">デバイスシャドウの更新例</a> について

方法	リソース	クエリーパラメータ	戻りコード	説明
				ては、セクションを参照ください。
POST	/shadow/<thingName>/<shadowName>	なし	200/403/500	このルートは、Thing の名前付きシャドウドキュメントを更新することを可能にします。thingName とshadowName [名前の付いたシャドウ] タイプを使用します。ボディは、構成する標準的な影の漆喰を遵守しなければならないstateノードおよび関連するdesiredおよびreportedノードフレームワークの使用の詳細については、 <a href="#">名前の付いたシャドウの更新例</a> については、セクションを参照ください。

## Architecture



## Examples

以下の例は、でのみ機能します。API\_KEY認証タイプを使用する場合、IAM 認証では Sigv4 トークンも指定する必要があるため、`apiGatewayCreateApiKey`プロパティが`true`を呼び出す必要があります。そうしないと、以下の例は機能しません。

### メッセージの発行

次は、使用できます。curlを使用して、HTTPS API を使用して異なる MQTT トピックにメッセージを公開します。以下の例では、上のメッセージを投稿します`device/foo`トピック。

```
curl -XPOST https://<stage-id>.execute-api.<region>.amazonaws.com/prod/message/device/fo  
o -H "x-api-key: <api-key>" -H "Content-Type: application/json" -d '{"Hello":  
"World"}'
```

注意: 置き換え`stage-id,region`, および`api-key`パラメーターをデプロイメント値に置き換えます。

URL でトピック名を連結できます。API では、公開できるサブトピックが最大 7 つまで受け付けられます。例えば、以下の例では、トピックにメッセージをパブリッシュします device/foo/bar/abc/xyz。

```
curl -XPOST https://<stage-id>.execute-api.<region>.amazonaws.com/prod/message/device/
foo/bar/abc/xyz -H "x-api-key: <api-key>" -H "Content-Type: application/json" -d
'{"Hello": "World"}
```

## デバイスシャドウの更新

特定の Thing に関連付けられているシャドウドキュメントを更新するには、Thing 名を使用してシャドウステートリクエストを発行します。Thing Shadow を更新する方法の例を参照ください。

```
curl -XPOST https://<stage-id>.execute-api.<region>.amazonaws.com/prod/shadow/device1 -
H "x-api-key: <api-key>" -H "Content-Type: application/json" -d '{"state": {"desired":
{ "Hello": "World" }}}'
```

## 名前の付いたシャドウの更新

特定の Thing の名前付き shadow に関連付けられたシャドウドキュメントを更新するには、Thing 名とシャドウ名を使用してシャドウ状態リクエストを発行します。名前付きシャドウを更新する方法については、次の例を参照してください。

```
curl -XPOST https://<stage-id>.execute-api.<region>.amazonaws.com/prod/shadow/device1/
shadow1 -H "x-api-key: <api-key>" -H "Content-Type: application/json" -d '{"state":
{"desired": { "Hello": "World" }}}'
```

## バイナリペイロードの送信

バイナリペイロードをプロキシ API に送信し、AWS IoT サービスに送信することができます。次の例では、コンテンツを送信する README.md ファイル (バイナリデータとして扱われる) を device/foo を使用して、application/octet-stream コンテンツタイプ。

```
curl -XPOST https://<stage-id>.execute-api.<region>.amazonaws.com/prod/message/device/
foo/bar/baz/qux -H "x-api-key: <api-key>" -H "Content-Type: application/octet-stream"
--data-binary @README.md
```

注意: このコマンドをこのプロジェクトのディレクトリで実行します。その後、ファイルシステムから他のタイプのバイナリファイルの送信をテストできます。

## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。





[@aws-solutions-構築/aws-apigateway- IoT](#)


## aws-apigateway-キネシスストリーム

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	aws_solutions_constructs.aws_apigateway_kinesisstreams
 TypeScript	@aws-solutions-constructs/aws-apigateway-kinesisstreams

言語	パッケージ
 Java	software.amazon.awsconstructs.services.apigatewaykinesisstreams

## Overview

このパターンは、Amazon Kinesis データストリームに接続された Amazon API Gateway REST API を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { ApiGatewayToKinesisStreams, ApiGatewayToKinesisStreamsProps } from '@aws-solutions-constructs/aws-apigateway-kinesisstreams';

new ApiGatewayToKinesisStreams(this, 'test-apigw-kinesis', {});
```

## Initializer

```
new ApiGatewayToKinesisStreams(scope: Construct, id: string, props: ApiGatewayToKinesisStreamsProps);
```

## パラメータ

- scope [Construct](#)
- id `string`
- props [ApiGatewayToKinesisStreamsProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ApigateWayProps ?	<a href="#">api.RestApiProps</a>	API Gateway REST API のデフォルトの小道具をオーバーライドするオプションのユーザー提供の小道具です。
putRecordRequestTemplate?	string	PutRecord アクションの API Gateway 要求テンプレート。指定されていない場合は、デフォルトのものが使用されます。
putRecordRequestModel ?	<a href="#">api.ModelOptions</a>	PutRecord アクションの API Gateway 要求モデル。指定されていない場合は、デフォルトのものが作成されます。
PutRecordsRequestTemplate?	string	PutRecords アクションの API Gateway 要求テンプレート。指定されていない場合は、デフォルトのものが使用されます。
putRecordRequestModel ?	<a href="#">api.ModelOptions</a>	PutRecords アクションの API Gateway 要求モデル。指定されていない場合は、デフォルトのものが作成されます。
ExistingStreamObj ?	<a href="#">kinesis.Stream</a>	Kinesis ストリームの既存のインスタンスで、これと <code>kinesisStreamProps</code> はエラーを発生させます。
KinesisStreamProps ?	<a href="#">kinesis.StreamProps</a>	Kinesis ストリームのデフォルトのプロップを上書きするオ

名前	タイプ	説明
		プシヨンのユーザー指定のプロップ。
LogGroupPropsかな？	<a href="#"><u>logs.LogGroupProps</u></a>	CloudWatch Logs ロググループのデフォルトの小道具を上書きする、オプションのユーザー指定の小道具です。

## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#"><u>api.RestApi</u></a>	パターンによって作成された API Gateway REST API のインスタンスを返します。
APIGatewayRole	<a href="#"><u>iam.Role</u></a>	API Gateway REST API のパターンによって作成された IAM ロールのインスタンスを返します。
APIGatewayクラウドウォッチ ロール	<a href="#"><u>iam.Role</u></a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロールのインスタンスを返します。
APIGatewayLogGroup	<a href="#"><u>logs.LogGroup</u></a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。



名前	タイプ	説明
KinesisStream	<a href="#">kinesis.Stream</a>	パターンによって作成された Kinesis ストリームのインスタンスを返します。

## API の使用例

方法	リクエストパス	リクエストボディ	キューアクション	説明
POST	/record	<pre>{   "data":   "Hello   World!",   "partitionKey":   "pk001" }</pre>	kinesis:PutRecord	1つのデータレコードをストリームに書き込みます。
POST	/records	<pre>{   "records":   [     { "data":       "abc",       "partitionKey":       "pk001"     },     { "data":       "xyz",       "partitionKey":       "pk001"     }   ] }</pre>	kinesis:PutRecords	1回の呼び出しで複数のデータレコードをストリームに書き込みます。

方法	リクエストパス	リクエストボディ	キューアクション	説明
		<pre>       }     ]   } </pre>		

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

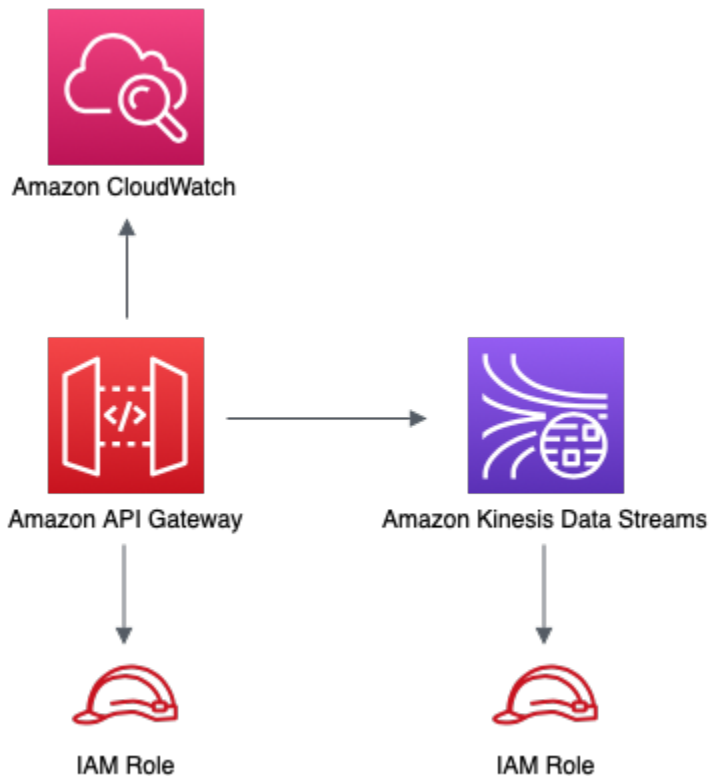
### Amazon API Gateway

- エッジ最適化 API エンドポイントをデプロイする。
- API Gateway の CloudWatch ロギングを有効にします。
- API Gateway の最小権限アクセス IAM ロールを設定します。
- すべての API メソッドのデフォルトの AuthorizationType を IAM に設定します。
- X-Ray トレースを有効にします。
- Kinesis にデータを渡す前にリクエスト本文を検証します。

### Amazon Kinesis Data Stream

- Kinesis ストリーム用の最小権限アクセス IAM ロールを設定します。
- AWS マネージド KMS キーを使用して Kinesis Stream のサーバー側の暗号化を有効にします。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-constructs/aws-apigateway-kinesisstreams](https://github.com/aws-solutions-constructs/aws-apigateway-kinesisstreams)

## aws-apigateway-ラムダ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_apigateway_lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-apigateway-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.apigatewaylambda</code>

## Overview

この AWS ソリューション構築物は、AWS Lambda 関数に接続された Amazon API Gateway REST API を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { ApiGatewayToLambda } from '@aws-solutions-constructs/aws-apigateway-lambda';

new ApiGatewayToLambda(this, 'ApiGatewayToLambdaPattern', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

## Initializer

```
new ApiGatewayToLambda(scope: Construct, id: string, props: ApiGatewayToLambdaProps);
```

## パラメータ

- scope [Construct](#)
- id `string`
- props [ApiGatewayToLambdaProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj が提供される。
ApiGatewayProps ?	<a href="#">api.LambdaRestApiProps</a>	オプションのユーザー提供の小道具で、API のデフォルトの小道具をオーバーライドします。
LogGroupPropsかな ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループのデフォルト小道具を上書きする、オプションのユーザー提供の小道具です。

## パターンプロパティ

名前	タイプ	説明
APIGatewayクラウドウォッチ ロール	<a href="#">iam.Role</a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロールのインスタンスを返します。
APIGateWayLogGroup	<a href="#">logs.LogGroup</a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
apigateway	<a href="#">api.LambdaRestApi</a>	パターンによって作成された API Gateway REST API のインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon API Gateway

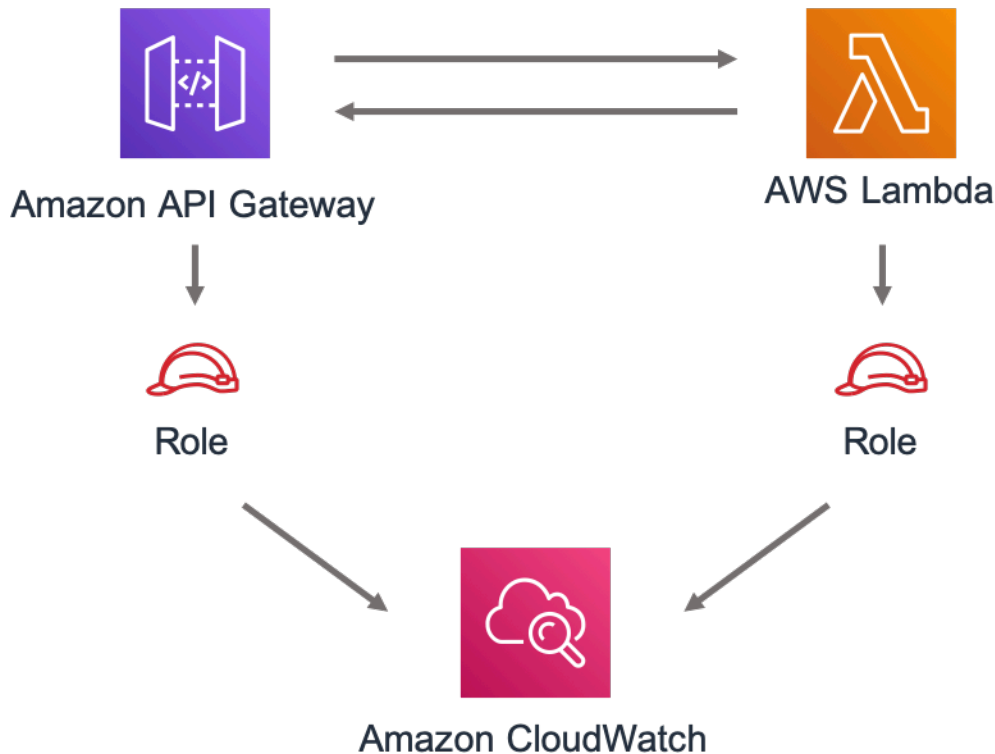
- エッジ最適化 API エンドポイントのデプロイ
- API Gateway での CloudWatch によるロギングの有効化
- API Gateway の最小権限アクセス IAM ロールを設定する
- すべての API メソッドのデフォルトの認証タイプを IAM に設定する
- X-Ray トレースを有効にする
- 環境変数の設定:

- `AWS_NODEJS_CONNECTION_REUSE_ENABLED` ( ノード10.x以上の機能の場合 )

## AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定する
- NodeJS Lambda 関数のキープアライブを使用して接続を再利用できるようにする
- X-Ray トレースを有効にする

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-constructs/aws-apigateway-lambda](https://github.com/aws-solutions-constructs/aws-apigateway-lambda)

# aws-apigateway-sagemakerendpoint

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理モデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_apigateway_sagemakerendpoint</code>
 TypeScript	<code>@aws-solutions-constructs/aws-apigateway-sagemakerendpoint</code>
 Java	<code>software.amazon.awsconstructs.services.apigatewaysagemakerendpoint</code>

## Overview

この AWS ソリューション構築物は、Amazon SageMaker エンドポイントに接続された Amazon API Gateway REST API を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { ApiGatewayToSageMakerEndpoint, ApiGatewayToSageMakerEndpointProps } from
  '@aws-solutions-constructs/aws-apigateway-sagemakerendpoint';

// Below is an example VTL (Velocity Template Language) mapping template for mapping
  the Api GET request to the Sagemaker POST request
```



```
const requestTemplate =
`{
  "instances": [
    #set( $user_id = $input.params("user_id") )
    #set( $items = $input.params("items") )
    #foreach( $item in $items.split(",") )
      {"in0": [$user_id], "in1": [$item]}#if( $foreach.hasNext ),#end
      $esc.newline
    #end
  ]
}`;

// Replace 'my-endpoint' with your Sagemaker Inference Endpoint
new ApiGatewayToSageMakerEndpoint(this, 'test-apigw-sagemakerendpoint', {
  endpointName: 'my-endpoint',
  resourcePath: '{user_id}',
  requestMappingTemplate: requestTemplate
});
```

## Initializer

```
new ApiGatewayToSageMakerEndpoint(scope: Construct, id: string, props:
  ApiGatewayToSageMakerEndpointProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [ApiGatewayToSageMakerEndpointProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ApigateWayProps ?	<a href="#">api.RestApiProps</a>	API Gateway REST API のデフォルトの小道具をオーバー

名前	タイプ	説明
		ライドするオプションのユーザー提供の小道具です。
ApigateWayExecutionRole ?	<a href="#">iam.Role</a>	SageMaker エンドポイントを呼び出すために API Gateway で使用される IAM ロール。指定しない場合、デフォルトのロールは <code>endpointName</code> 。
EndpointName	string	デプロイされた SageMaker 推論エンドポイントの名前。
resourceName	string	GET メソッドを使用できるオプションのリソース名。
resourcePath	string	GET メソッドのリソースパス。ここで定義された変数は <code>requestMappingTemplate</code> 。
リクエストMappingTemplate	string	REST API で受信した GET リクエストを SageMaker エンドポイントで期待される POST リクエストに変換するためのマッピングテンプレート。
ResponseMappingTempl	string	SageMaker エンドポイントから受信した応答を変換するためのオプションのマッピングテンプレート。
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループのデフォルト小道具を上書きする、オプションのユーザー提供の小道具です。

## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#">api.LambdaRestApi</a>	パターンによって作成された API Gateway REST API のインスタンスを返します。
APIGatewayRole	<a href="#">iam.Role</a>	API Gateway REST API のパターンによって作成された IAM ロールのインスタンスを返します。
APIGatewayクラウドウォッチ ロール	<a href="#">iam.Role</a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロールのインスタンスを返します。
APIGatewayLogGroup	<a href="#">logs.LogGroup</a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。

## API の使用例

注意: 各 SageMaker エンドポイントは一意であり、API からの応答はデプロイされたモデルによって異なります。下記の例は、からのサンプルを想定しています[このブログ投稿](#)。それがどのように実装されるかについての参考文献については、[integ.apigateway-sagemakerendpoint-上書き.ts](#)。

方法	リクエストパス	クエリ文字列	SageMaker アクション	説明
GET	/321	items=101,131,162	sagemaker:InvokeEndpoint	特定のユーザーとアイテムの予測を取得します。

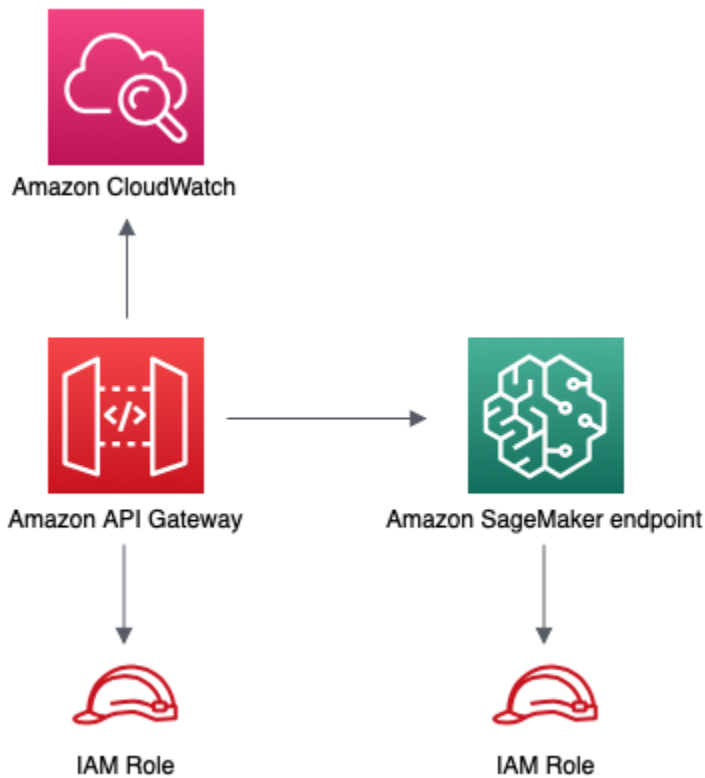
## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon API Gateway

- エッジ最適化 API エンドポイントのデプロイ
- API Gateway での CloudWatch によるロギングの有効化
- API Gateway の最小権限アクセス IAM ロールを設定する
- すべての API メソッドのデフォルトの authorizationType を IAM に設定する
- X-Ray トレースを有効にする
- SageMaker にデータを渡す前にリクエストパラメータを検証する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-constructs/aws-apigateway-sagemakerendpoint](https://github.com/aws-solutions-constructs/aws-apigateway-sagemakerendpoint)

## aws-apigateway-sqs

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_apigateway_sqs</code>
 TypeScript	<code>@aws-solutions-constructs/aws-apigateway-sqs</code>
 Java	<code>software.amazon.awsconstructs.services.apigatewaysqs</code>

## Overview

この AWS ソリューション構築物は、Amazon SQS キューに接続された Amazon API Gateway REST API を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { ApiGatewayToSqs, ApiGatewayToSqsProps } from "@aws-solutions-constructs/aws-apigateway-sqs";

new ApiGatewayToSqs(this, 'ApiGatewayToSqsPattern', {});
```

## Initializer

```
new ApiGatewayToSqs(scope: Construct, id: string, props: ApiGatewayToSqsProps);
```

## パラメータ

- `scope`[Construct](#)

- idstring
- props[ApiGatewayToSqsProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ApigateWayProps ?	<a href="#">api.RestApiProps</a>	API Gateway のデフォルトの小道具を上書きするオプションのユーザー提供の小道具です。
QueueProp?	<a href="#">sqs.QueueProps</a>	オプションのユーザー提供の小道具で、キューのデフォルトの小道具を上書きします。
デプロイデッドレターキュー?	boolean	デッドレターキューとして使用するセカンダリキューを展開するかどうか。デフォルトは true です。
maxReceiveCount	number	デッドレターキューに移動する前に、メッセージがデッドキューに失敗した回数。
作成操作を許可しますか?	boolean	キューに Create 操作の API Gateway メソッドをデプロイするかどうか (SQS: SendMessage)。
リクエストテンプレートを作成しますか?	string	Create メソッドのデフォルトの API Gateway リクエストテンプレートを上書きしません (allowCreateOperation は、に設定されます。true)。

名前	タイプ	説明
操作を許可しますか？	boolean	読み込み操作の API Gateway メソッドをキューにデプロイするかどうか (SQS: ReceiveMessage)。
readRequestテンプレートですか？	string	Read メソッドのデフォルトの API Gateway リクエストテンプレートをオーバーライドします (allowRead Operation は、に設定されます。true)。
削除操作を許可しますか？	boolean	削除操作のための API Gateway メソッドをキューにデプロイするかどうか (SQS: DeleteMessage)。
要求テンプレートを削除しますか？	string	Delete メソッドのデフォルトの API Gateway リクエストテンプレートを上書きします (allowDeleteOperation は、に設定されます。true)。
LogGroupPropsかな？	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループのデフォルトの小道具を上書きする、オプションのユーザー指定の小道具です。



## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#">api.RestApi</a>	パターンによって作成された API Gateway REST API のインスタンスを返します。
APIGatewayクラウドウォッチ ロール	<a href="#">iam.Role</a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロールのインスタンスを返します。
APIGatewayLogGroup	<a href="#">logs.LogGroup</a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。
APIGatewayRole	<a href="#">iam.Role</a>	API Gateway REST API のパターンによって作成された IAM ロールのインスタンスを返します。
デッドレターキュー？	<a href="#">sqs.Queue</a>	パターンによって作成されたデッドレターキューのインスタンスを返します ( デプロイされている場合 )。
SQUEUE	<a href="#">sqs.Queue</a>	パターンによって作成された SQS キューのインスタンスを返します。

## API の使用例

方法	リクエストパス	リクエストボディ	キューアクション	説明
GET	/		sqs::ReceiveMessage	キューからメッセージを取得します。
POST	/	{ "data": "Hello World!" }	sqs::SendMessage	メッセージをキューに配信します。
DELETE	/message?receiptHandle=[value]		sqs::DeleteMessage	指定されたメッセージをキューから削除します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

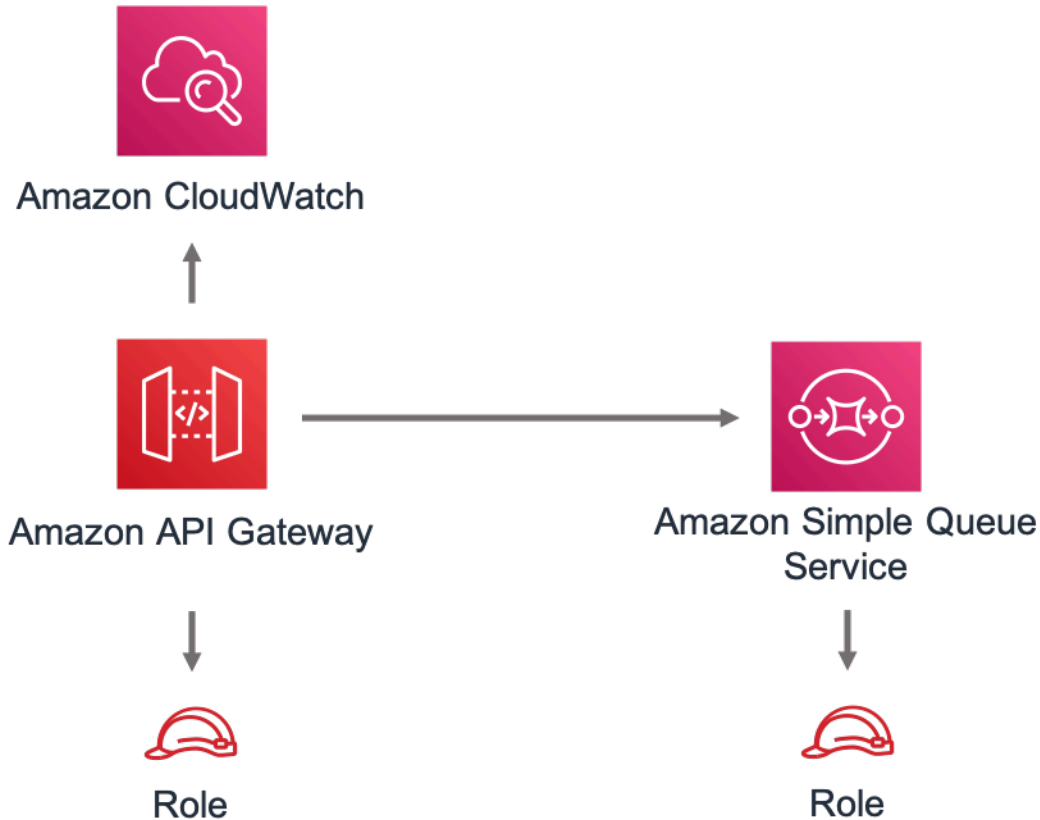
### Amazon API Gateway

- エッジ最適化 API エンドポイントのデプロイ
- API Gateway での CloudWatch によるロギングの有効化
- API Gateway の最小権限アクセス IAM ロールを設定する
- すべての API メソッドのデフォルトの認証タイプを IAM に設定する
- X-Ray トレースを有効にする

### Amazon SQS キュー

- ソース SQS キューの SQS デッドレターキューのデプロイ
- AWS マネージド KMS キーを使用したソース SQS キューのサーバー側の暗号化を有効にする
- 転送時のデータの暗号化を強制する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-apigateway-sqs](#)

## aws-クラウドフロント-アピゲートウェイ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_cloudfront_apigateway</code>
 TypeScript	<code>@aws-solutions-constructs/aws-cloudfront-apigateway</code>
 Java	<code>software.amazon.awsconstructs.services.cloudfrontapigateway</code>

## Overview

この AWS ソリューション構築物は、Amazon API Gateway REST API の前で Amazon CloudFront デイストリビューションを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import * as api from '@aws-cdk/aws-apigateway';
import * as lambda from "@aws-cdk/aws-lambda";
import { CloudFrontToApiGateway } from '@aws-solutions-constructs/aws-cloudfront-apigateway';

const lambdaProps: lambda.FunctionProps = {
  code: lambda.Code.fromAsset(`${__dirname}/lambda`),
  runtime: lambda.Runtime.NODEJS_12_X,
  handler: 'index.handler'
};

const lambdafunction = new lambda.Function(this, 'LambdaFunction', lambdaProps);

const apiGatewayProps: api.LambdaRestApiProps = {
  handler: lambdafunction,
```

```

    endpointConfiguration: {
      types: [api.EndpointType.REGIONAL]
    },
    defaultMethodOptions: {
      authorizationType: api.AuthorizationType.NONE
    }
  };

  const apiGateway = new api.LambdaRestApi(this, 'LambdaRestApi', apiGatewayProps);

  new CloudFrontToApiGateway(this, 'test-cloudfront-apigateway', {
    existingApiGatewayObj: apiGateway
  });

```

## Initializer

```

new CloudFrontToApiGateway(scope: Construct, id: string, props:
  CloudFrontToApiGatewayProps);

```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [CloudFrontToApiGatewayProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingApigateWayObj	<a href="#">api.RestApi</a>	CloudFront の前面となるリージョン API Gateway
CloudFrontDistributionProps ?	<a href="#">cloudfront.DistributionProps</a>	CloudFront デイストリビューションのデフォルトの小道具を上書きするオプションのユーザー提供の小道具です。

名前	タイプ	説明
TTPセキュリティヘッダーを挿入しますか？	boolean	CloudFront からのすべての応答でベストプラクティス HTTP セキュリティヘッダーの自動インジェクションをオン/オフするためのオプションのユーザー提供の小道具

## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#">api.RestApi</a>	パターンによって作成された API Gateway REST API のインスタンスを返します。
CloudFrontLoggingBucket ?	<a href="#">s3.Bucket</a>	CloudFront ウェブディストリビューションのパターンによって作成されたロギングバケットのインスタンスを返します。
クラウドフロントウェブディストリビューション	<a href="#">cloudfront.CloudFrontWebDistribution</a>	パターンによって作成された CloudFront ウェブディストリビューションのインスタンスを返します。
EdgeLambdaFunction Version ?	<a href="#">lambda.Version</a>	パターンによって作成された Lambda エッジ関数バージョンのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

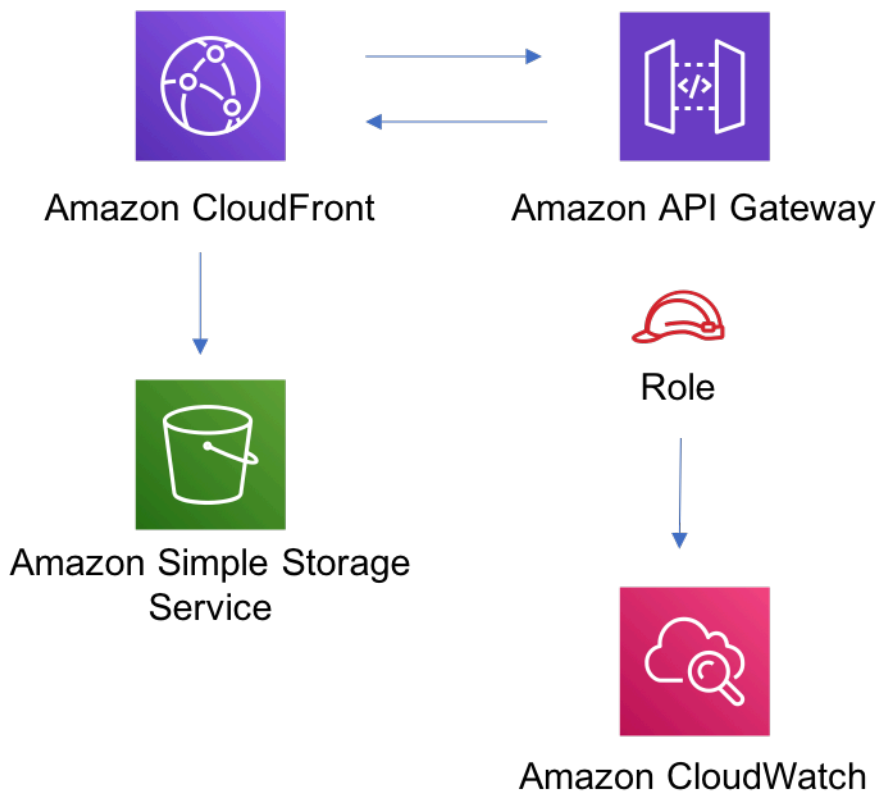
## Amazon CloudFront

- CloudFront ウェブディストリビューションのアクセスログの設定
- CloudFront WebDistribution からのすべてのレスポンスでベストプラクティス HTTP セキュリティヘッダーの自動インジェクションを有効にする

## Amazon API Gateway

- ユーザー提供の API Gateway オブジェクトはそのまま使用されます。
- X-Ray トレースを有効にする

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-構築/aws-cloudfront-api-gateway](#)



## aws-クラウドフロント-アピゲートウェイ-ラムダ

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	aws_solutions_constructs.aws_cloudfront_apigateway_lambda
 TypeScript	@aws-solutions-constructs/aws-cloudfront-apigateway-lambda
 Java	software.amazon.awsconstructs.services.cloudfrontapigatewaylambda

## Overview

この AWS ソリューション構築物は、Amazon API Gateway ラムダでバックアップされた REST API の前で Amazon CloudFront デイストリビューションを実装します。



TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { CloudFrontToApiGatewayToLambda } from '@aws-solutions-constructs/aws-cloudfront-apigateway-lambda';

new CloudFrontToApiGatewayToLambda(this, 'test-cloudfront-apigateway-lambda', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

## Initializer

```
new CloudFrontToApiGatewayToLambda(scope: Construct, id: string, props:
  CloudFrontToApiGatewayToLambdaProps);
```

## パラメータ

- scope [Construct](#)
- id [string](#)
- props [CloudFrontToApiGatewayToLambdaProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。

名前	タイプ	説明
LambdaFunctionProps ?	<a href="#"><u>lambda.FunctionProps</u></a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されずexistingLambdaObj はにあります。
ApiGatewayProps ?	<a href="#"><u>api.LambdaRestApiProps</u></a>	API Gateway のデフォルトの小道具をオーバーライドするオプションのユーザーが提供した小道具を
CloudFrontDistributionProps ?	<a href="#"><u>cloudfront.DistributionProps</u></a>	CloudFront ディストリビューションのデフォルトの小道具を上書きするオプションのユーザー提供の小道具です。
TTPセキュリティヘッダーを挿入しますか？	boolean	CloudFront からのすべての応答でベストプラクティス HTTP セキュリティヘッダーの自動インジェクションをオン/オフするためのオプションのユーザー提供の小道具
LogGroupPropsかな？	<a href="#"><u>logs.LogGroupProps</u></a>	CloudWatch Logs ロググループのデフォルトの小道具を上書きするオプションのユーザー指定の小道具です。

## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#">api.RestApi</a>	パターンによって作成された API Gateway REST API のインスタンスを返します。
APIGatewayクラウドウォッチ ロール	<a href="#">iam.Role</a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロールのインスタンスを返します。
APIGatewayLogGroup	<a href="#">logs.LogGroup</a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。
CloudFrontLoggingBucket ?	<a href="#">s3.Bucket</a>	CloudFront ウェブディストリビューションのパターンによって作成されたロギングバケットのインスタンスを返します。
クラウドフロントウェブディ ストリビューション	<a href="#">cloudfront.CloudFrontWebDistribution</a>	パターンによって作成された CloudFront ウェブディストリビューションのインスタンスを返します。
EdgeLambdaFunction Version ?	<a href="#">lambda.Version</a>	パターンによって作成された Lambda エッジ関数バージョンのインスタンスを返します。

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon CloudFront

- CloudFront ウェブディストリビューションのアクセスログの設定
- CloudFront WebDistribution からのすべてのレスポンスでベストプラクティス HTTP セキュリティヘッダーの自動インジェクションを有効にする

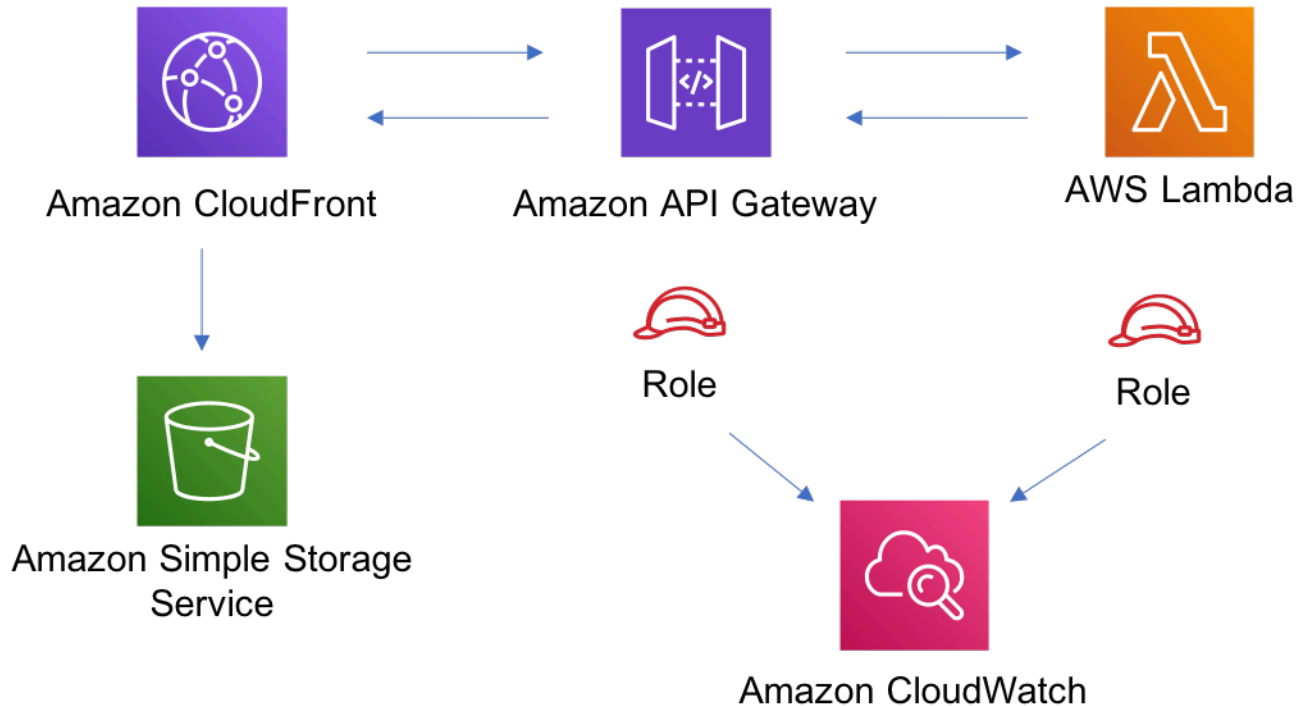
### Amazon API Gateway

- リージョン API エンドポイントのデプロイ
- API Gateway での CloudWatch によるログの有効化
- API Gateway の最小権限アクセス IAM ロールを設定する
- すべての API メソッドのデフォルトの認証タイプを IAM に設定する
- X-Ray トレースを有効化

### AWS Lambda 関数

- Lambda 関数の制限付きアクセスアクセスの IAM ロールを設定する
- NodeJS Lambda 関数のキープアライブを使用して接続を再利用できるようにする
- X-Ray トレースを有効化
- 環境変数の設定:
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。





[@aws-solutions-構築/aws-cloudfront-api-gateway-ラムダ](#)

## aws-クラウドフロントメディアストア

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_cloudfront_mediastore</code>
 TypeScript	<code>@aws-solutions-constructs/aws-cloudfront-mediastore</code>
 Java	<code>software.amazon.awsconstructs.services.cloudfrontmediastore</code>

## Overview

この AWS ソリューション構築物は、AWS Elemental MediaStore コンテナに接続された Amazon CloudFront デイストリビューションを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { CloudFrontToMediaStore } from '@aws-solutions-constructs/aws-cloudfront-mediastore';

new CloudFrontToMediaStore(this, 'test-cloudfront-mediastore-default', {});
```

## Initializer

```
new CloudFrontToMediaStore(scope: Construct, id: string, props: CloudFrontToMediaStoreProps);
```

## パラメータ

- `scope`[Construct](#)
- `id``string`

- props[CloudFrontToMediaStoreProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingMediaStoreContainer OBJ?	<a href="#">mediastore.CfnContainer</a>	オプションのユーザーが指定した MediaStore コンテナを使用して、デフォルトの MediaStore コンテナを上書きします。
MediaStoreContainerProps?	<a href="#">mediastore.CfnContainerProps</a>	MediaStore コンテナの既定の小道具を上書きするオプションのユーザー提供の小道具です。
CloudFrontDistributionProps?	<a href="#">cloudfront.DistributionProps</a>   any	オプションのユーザー提供の小道具で、CloudFront デイストリビューションのデフォルトの小道具を上書きします。
TTPセキュリティヘッダーを挿入しますか?	boolean	CloudFront からのすべての応答でベストプラクティス HTTP セキュリティヘッダーの自動インジェクションをオン/オフにするオプションのユーザー提供の小道具です。

## パターンプロパティ

名前	タイプ	説明
クラウドフロントウェブディストリビューション	<a href="#">cloudfront.CloudFrontWebDistribution</a>	パターンによって作成された CloudFront ウェブディストリ

名前	タイプ	説明
		ビューシヨンのインスタンスを返します。
メディアストアコンテナ	<a href="#">mediastore.CfnContainer</a>	パターンによって作成された MediaStore コンテナのインスタンスを返します。
クラウドフロントログバケツト	<a href="#">s3.Bucket</a>	CloudFront ウェブディストリビューシヨンのパターンによって作成されたロギングバケツトのインスタンスを返します。
クラウドフロントオリジンリクエストポリシー	<a href="#">cloudfront.OriginRequestPolicy</a>	CloudFront ウェブディストリビューシヨンのパターンによって作成された CloudFront オリジンリクエストポリシーのインスタンスを返します。
CloudFrontOriginAccessIdentity かな	<a href="#">cloudfront.OriginAccessIdentity</a>	CloudFront ウェブディストリビューシヨンのパターンによって作成された CloudFront オリジンアクセスアイデンティティのインスタンスを返します。
EdgeLambdaFunctionVersion	<a href="#">lambda.Version</a>	パターンによって作成された Lambda エツジ関数バージョンのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。



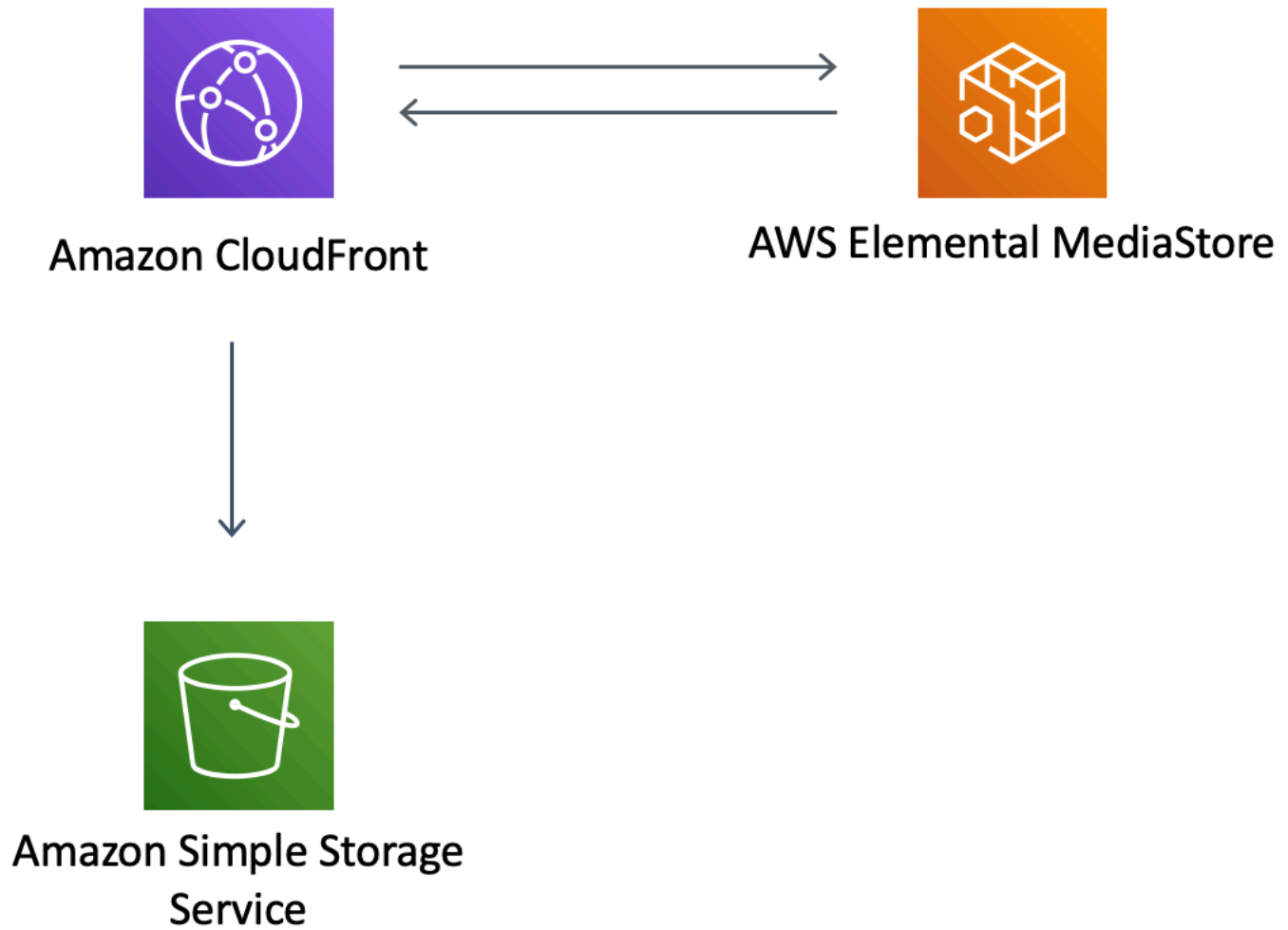
## Amazon CloudFront

- CloudFront ウェブディストリビューションのアクセスログの設定
- AWS Elemental MediaStore コンテナの CloudFront オリジンリクエストポリシーを有効にする
- 設定User-AgentCloudFront オリジンアクセスアイデンティティを持つカスタムヘッダー
- CloudFront ウェブディストリビューションからのすべてのレスポンスでベストプラクティス HTTP セキュリティヘッダーの自動インジェクションを有効にする

## AWS Elemental MediaStore

- リソースを保持するように削除ポリシーを設定する
- CloudFormation スタック名でコンテナ名を設定する
- デフォルトを設定[コンテナのクロスオリジンリソース共有 \(CORS\) ポリシー](#)
- デフォルトを設定[オブジェクトのライフサイクルポリシー](#)
- デフォルトを設定[コンテナポリシー](#)のみを許可するaws:UserAgentCloudFront オリジンアクセスアイデンティティを持つ
- デフォルトを設定[メトリクスポリシー](#)
- アクセスログの作成の有効化

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。





[@aws-solutions-構築/aws-クラウドフロント-メディアストア](#)

## aws-クラウドフロント-3

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョンニング](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_cloudfront_s3</code>
 TypeScript	<code>@aws-solutions-constructs/aws-cloudfront-s3</code>
 Java	<code>software.amazon.awsconstructs.services.cloudfronts3</code>

## Overview

この AWS ソリューション構築は Amazon S3 バケットの前に Amazon CloudFront デイストリビューションを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { CloudFrontToS3 } from '@aws-solutions-constructs/aws-cloudfront-s3';  
  
new CloudFrontToS3(this, 'test-cloudfront-s3', {});
```

## Initializer

```
new CloudFrontToS3(scope: Construct, id: string, props: CloudFrontToS3Props);
```

## パラメータ

- scope [Construct](#)
- id `string`
- props [CloudFrontToS3Props](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingBucketObj ?	<a href="#">s3.Bucket</a>	S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。
BucketProps ?	<a href="#">s3.BucketProps</a>	オプションのユーザー提供のプロパティ。バケットのデフォルトプロパティを上書きします。の場合は無視されます。existingBucketObj が提供される。
CloudFrontDistributionProps ?	<a href="#">cloudfront.DistributionProps</a>	CloudFront ディストリビューションのデフォルトの小道具を上書きするオプションのユーザー提供の小道具です。
TTPSecurityHeadersを挿入しますか？	boolean	CloudFront からのすべての応答でベストプラクティス HTTP セキュリティヘッダーの自動インジェクションをオン/オフするためのオプションのユーザー提供の小道具

## パターンプロパティ

名前	タイプ	説明
クラウドフロントウェブディストリビューション	<a href="#">cloudfront.CloudFrontWebDistribution</a>	パターンによって作成された CloudFront ウェブディストリビューションのインスタンスを返します。
S3bucket?	<a href="#">s3.Bucket</a>	パターンによって作成された S3 バケットのインスタンスを返します。
s3loggingBucket ?	<a href="#">s3.Bucket</a>	S3 バケットのパターンによって作成されたロギングバケットのインスタンスを返します。
EdgeLambdaFunction Version ?	<a href="#">lambda.Version</a>	パターンによって作成された Lambda エッジ関数バージョンのインスタンスを返します。
CloudFrontLoggingBucket ?	<a href="#">s3.Bucket</a>	CloudFront ウェブディストリビューションのパターンによって作成されたロギングバケットのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

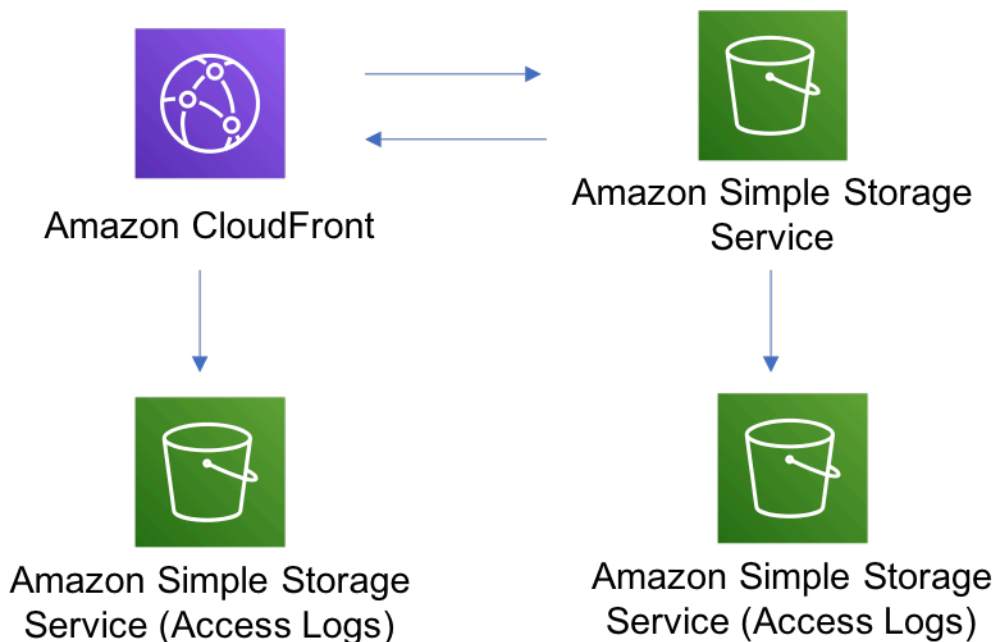
### Amazon CloudFront

- CloudFront ウェブディストリビューションのアクセスログの設定
- CloudFront WebDistribution からのすべてのレスポンスでベストプラクティスの HTTP セキュリティヘッダーの自動インジェクションを有効にする

## Amazon S3 バケット

- S3 バケットのアクセスログの設定
- AWS マネージド KMS キーを使用した S3 バケットのサーバー側の暗号化の有効化
- S3 バケットのバージョニングを有効にする
- S3 バケットのパブリックアクセスを許可しない
- CloudFormation スタックを削除するときに S3 バケットを保持する
- 転送時のデータの暗号化を強制する
- ライフサイクルルールを適用して、90 日後に最新でないオブジェクトバージョンを Glacier ストレージに移動する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-cloudfront-S3](#)




# aws-コグニート-アピガテウェイ-ラムダ

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理モデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_cognito_apigateway_lambda</code>
 活字体	<code>@aws-solutions-constructs/aws-cognito-apigateway-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.cognitoapigatewaylambda</code>

## Overview

この AWS ソリューション構築物は、Amazon API Gateway ラムダでバックアップされた REST API を保護する Amazon Cognito を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { CognitoToApiGatewayToLambda } from '@aws-solutions-constructs/aws-cognito-apigateway-lambda';

new CognitoToApiGatewayToLambda(this, 'test-cognito-apigateway-lambda', {
```

```
    lambdaFunctionProps: {
      runtime: lambda.Runtime.NODEJS_14_X,
      // This assumes a handler function in lib/lambda/index.js
      code: lambda.Code.fromAsset(`${__dirname}/lambda`),
      handler: 'index.handler'
    }
  });
```

APIでリソースとメソッドを定義している場合 (例: proxy = false) を呼び出す必要があります。addAuthorizers()API が完全に定義された後にメソッドを呼び出します。これにより、APIのすべてのメソッドが保護されます。

TypeScript tの例を次に示します。

```
import { CognitoToApiGatewayToLambda } from '@aws-solutions-constructs/aws-cognito-apigateway-lambda';

const construct = new CognitoToApiGatewayToLambda(this, 'test-cognito-apigateway-lambda', {
  lambdaFunctionProps: {
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    runtime: lambda.Runtime.NODEJS_12_X,
    handler: 'index.handler'
  },
  apiGatewayProps: {
    proxy: false
  }
});

const resource = construct.apiGateway.root.addResource('foobar');
resource.addMethod('POST');

// Mandatory to call this method to Apply the Cognito Authorizers on all API methods
construct.addAuthorizers();
```

## Initializer



```
new CognitoToApiGatewayToLambda(scope: Construct, id: string, props:
  CognitoToApiGatewayToLambdaProps);
```

## パラメータ

- scope [Construct](#)
- id [string](#)
- props [CognitoToApiGatewayToLambdaProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Function オブジェクトの既存のインスタンス。これと <code>lambdaFunctionProps</code> はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。場合は、無視されます。existingLambdaObj が提供される。
ApiGatewayProps ?	<a href="#">api.LambdaRestApiProps</a>	API Gateway のデフォルトの小道具をオーバーライドするオプションのユーザーが提供した小道具を
CognitoUserPoolProps ?	<a href="#">cognito.UserPoolProps</a>	Cognito ユーザープールのデフォルトのプロップを上書きするオプションのユーザー提供の小道具
CognitoUserPoolClientProps ?	<a href="#">cognito.UserPoolClientProps</a>	Cognito ユーザープールクライアントの既定のプロップを

名前	タイプ	説明
		上書きするオプションのユーザー提供の小道具
LogGroupProps ?	<a href="#"><u>logs.LogGroupProps</u></a>	CloudWatch Logs ロググループのデフォルトの小道具を上書きする、オプションのユーザー提供の小道具です。

## パターンプロパティ

名前	タイプ	説明
apigateway	<a href="#"><u>api.RestApi</u></a>	パターンによって作成された API Gateway REST API のインスタンスを返します。
LambdaFunction	<a href="#"><u>lambda.Function</u></a>	パターンによって作成された Lambda 関数のインスタンスを返します。
UserPool	<a href="#"><u>cognito.UserPool</u></a>	パターンによって作成された Cognito ユーザープールのインスタンスを返します。
ユーザーPoolClient	<a href="#"><u>cognito.UserPoolClient</u></a>	パターンによって作成された Cognito ユーザープールクライアントのインスタンスを返します。
APIGatewayクラウドウォッチロール	<a href="#"><u>iam.Role</u></a>	API Gateway REST API から CloudWatch へのアクセスロギングを有効にするパターンによって作成された IAM ロールのインスタンスを返します。

名前	タイプ	説明
APIGatewayLogGroup	<a href="#">logs.LogGroup</a>	API Gateway REST API アクセスログが送信されるパターンによって作成されたロググループのインスタンスを返します。
APIGatewayAuthorizer	<a href="#">api.CfnAuthorizer</a>	パターンによって作成された API Gateway 認証のインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon Cognito

- UserPools のパスワードポリシーを設定する
- ユーザープールに高度なセキュリティモードを適用する

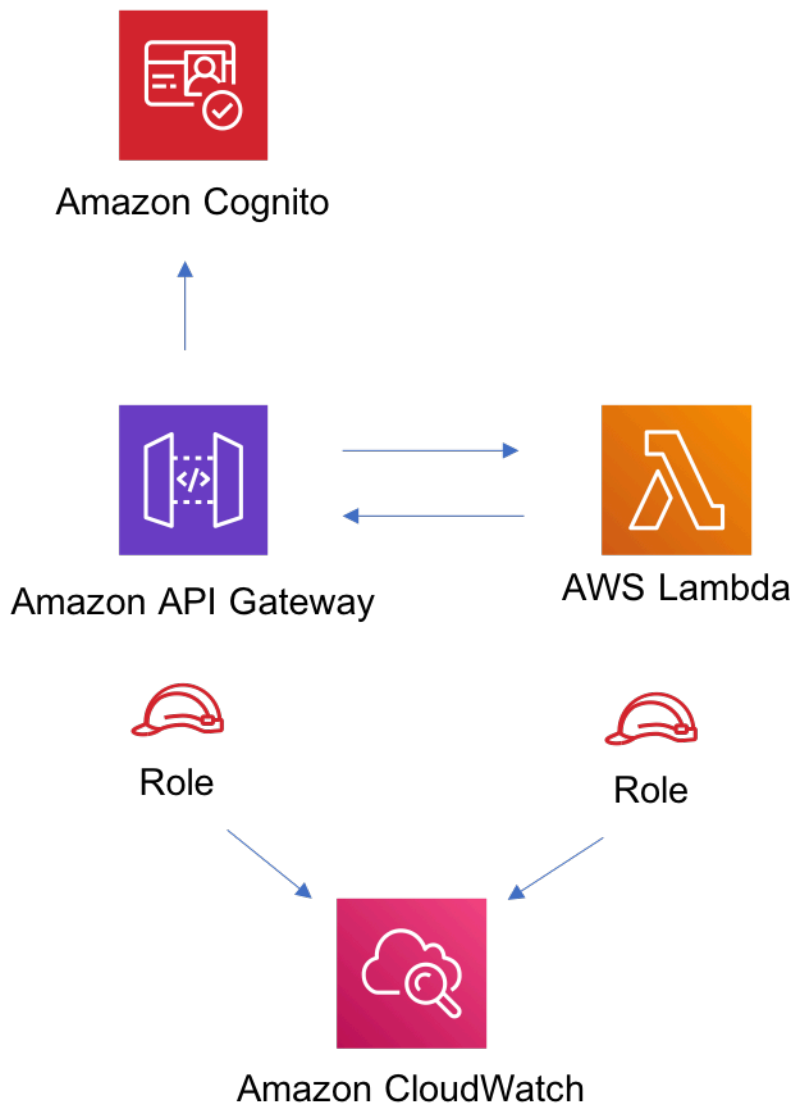
### Amazon API Gateway

- エッジ最適化 API エンドポイントのデプロイ
- API Gateway での CloudWatch によるロギングを有効にする
- API Gateway の最小権限アクセス IAM ロールを設定する
- すべての API メソッドのデフォルトの authorizationType を IAM に設定する
- X-Ray トレースを有効にする

### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定する
- NodeJS Lambda 関数のキープアライブを使用して接続を再利用できるようにする
- X-Ray トレースを有効にする
- 環境変数の設定:
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-constructs/aws-cognito-apigateway-lambda](https://github.com/aws-solutions-constructs/aws-cognito-apigateway-lambda)



## aws-Dynamodb-stream lambda

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョン管理](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_dynamodb_stream_lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-dynamodb-stream-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.dynamodbstreamlambda</code>

## Overview

この AWS ソリューション構成では、Amazon DynamoDB テーブルをストリームで実装し、権限が最も低い権限で AWS Lambda 関数を呼び出します。

最小限のデプロイ可能なパターン定義を次に示します。

```
import { DynamoDBStreamToLambdaProps, DynamoDBStreamToLambda } from '@aws-solutions-constructs/aws-dynamodb-stream-lambda';

new DynamoDBStreamToLambda(this, 'test-dynamodb-stream-lambda', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

```
    },
  });
```

## Initializer

```
new DynamoDBStreamToLambda(scope: Construct, id: string, props:
  DynamoDBStreamToLambdaProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [DynamoDBStreamToLambdaProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。次の場合は無視されます。existingLambdaObj が提供される。
DynamoTableProps ?	<a href="#">dynamodb.TableProps</a>	DynamoDB テーブルのデフォルトの小道具をオーバーライドするオプションのユーザー提供の小道具です

名前	タイプ	説明
ExistingTableObj ?	<a href="#">dynamodb.Table</a>	DynamoDB テーブルオブジェクトの既存のインスタンス。これとdynamoTableProps はエラーを発生させます。
DynamoEventSourceProps ?	<a href="#">aws-lambda-event-sources.DynamoEventSourceProps</a>	DynamoDB イベントソースのデフォルトのプロップを上書きするオプションのユーザー提供の小道具

## パターンプロパティ

名前	タイプ	説明
ダイナモテーブル	<a href="#">dynamodb.Table</a>	パターンによって作成された DynamoDB テーブルのインスタンスを返します。
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。

## Lambda 関数

このパターンには、DynamoDB ストリームから Elasticsearch サービスにデータを投稿できる Lambda 関数が必要です。サンプル関数が用意されています [ここ](#)。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon DynamoDB テーブル

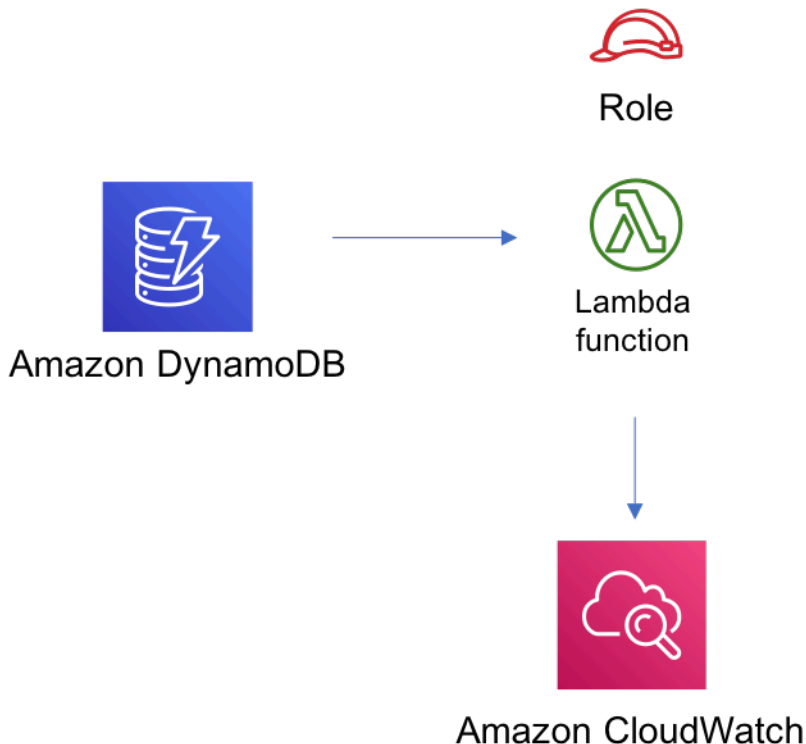
- DynamoDB テーブルの請求モードをオンデマンドに設定する (リクエストごとの支払い)

- AWS マネージド KMS キーを使用した DynamoDB テーブルのサーバー側の暗号化の有効化
- DynamoDB テーブルの 'id' という名前のパーティションキーを作成します。
- CloudFormation スタックを削除するときにテーブルを保持する
- 継続的なバックアップとポイントインタイムリカバリを実現

## AWS Lambda 関数

- Lambda 関数用に制限された特権アクセス IAM ロールを設定する
- NodeJS Lambda 関数のキープアライブを使用して接続を再利用できるようにする
- X-Ray トレースを有効にする
- 障害処理機能の有効化:関数エラーの bisect の有効化、デフォルトの最大レコード有効期間 (24 時間) の設定、デフォルトの最大再試行回数 (500) の設定、障害発生時の宛先として SQS デッドレターキューのデプロイ
- 環境変数の設定:
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Architecture





# GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-構築/aws-dynamodb-stream-ラムダ](#)

## aws-ダイナモッド-ストリーム-ラムダ-弾性検索-キバナ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンング](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_dynamodb_stream_lambda_elasticsearch_kibana</code>
 TypeScript	<code>@aws-solutions-constructs/aws-dynamodb-stream-lambda-elasticsearch-kibana</code>
 Java	<code>software.amazon.awsconstructs.services.dynamodbstreamlambdaelasticsearchkibana</code>

## Overview

この AWS ソリューション構築物は、ストリームを含む Amazon DynamoDB テーブル、AWS Lambda 関数、および最も権限の低いアクセス許可を持つ Amazon Elasticsearch Service 実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { DynamoDBStreamToLambdaToElasticSearchAndKibana,
  DynamoDBStreamToLambdaToElasticSearchAndKibanaProps } from '@aws-solutions-constructs/
aws-dynamodb-stream-lambda-elasticsearch-kibana';
import { Aws } from "@aws-cdk/core";

const props: DynamoDBStreamToLambdaToElasticSearchAndKibanaProps = {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  },
  domainName: 'test-domain',
  // TODO: Ensure the Cognito domain name is globally unique
  cognitoDomainName: 'globallyuniquedomain' + Aws.ACCOUNT_ID;
};

new DynamoDBStreamToLambdaToElasticSearchAndKibana(this, 'test-dynamodb-stream-lambda-
elasticsearch-kibana', props);
```

## Initializer

```
new DynamoDBStreamToLambdaToElasticSearchAndKibana(scope: Construct, id: string, props:
DynamoDBStreamToLambdaToElasticSearchAndKibanaProps);
```

## パラメータ

- `scope`[Construct](#)
- `id``string`

- [propsDynamoDBStreamToLambdaToElasticSearchAndKibanaProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。Ignored if an existingLambdaObj が提供される。
DynamoTablePropsかな？	<a href="#">dynamodb.TableProps</a>	DynamoDB テーブルのデフォルトのプロップを上書きするオプションのユーザー提供の小道具
ExistingTableObj ?	<a href="#">dynamodb.Table</a>	DynamoDB テーブルオブジェクトの既存のインスタンス。これとdynamoTableProps はエラーを発生させます。
DynamoEventSourceProps ?	<a href="#">aws-lambda-event-sources.DynamoEventSourceProps</a>	DynamoDB イベントソースのデフォルトのプロップを上書きするオプションのユーザー提供の小道具
ESDomainProps ?	<a href="#">elasticsearch.CfnDomainProps</a>	Amazon Elasticsearch Service デフォルトの小道具を上書き

名前	タイプ	説明
		するためのオプションのユーザー提供の小道具です
domainName	string	Cognito および Amazon Elasticsearch Service ドメイン名
クラウド・ワット・チャラームズ	boolean	推奨される CloudWatch アラームを作成するかどうか。

## パターンプロパティ

名前	タイプ	説明
CloudWatchAlarms ?	<a href="#"><u>cloudwatch.Alarm[]</u></a>	パターンによって作成された 1 つ以上の CloudWatch アラームのリストを返します。
ダイナモテーブル	<a href="#"><u>dynamodb.Table</u></a>	パターンによって作成された DynamoDB テーブルのインスタンスを返します。
弾性検索ドメイン	<a href="#"><u>elasticsearch.CfnDomain</u></a>	パターンによって作成された Elasticsearch ドメインのインスタンスを返します。
IdentityPool	<a href="#"><u>cognito.CfnIdentityPool</u></a>	パターンによって作成された Cognito ID プールのインスタンスを返します。
LambdaFunction	<a href="#"><u>lambda.Function</u></a>	パターンによって作成された Lambda 関数のインスタンスを返します。

名前	タイプ	説明
UserPool	<a href="#">cognito.UserPool</a>	パターンによって作成された Cognito ユーザープールのインスタンスを返します。
UserPoolCli	<a href="#">cognito.UserPoolClient</a>	パターンによって作成された Cognito ユーザープールクライアントのインスタンスを返します。

## Lambda 関数

このパターンには、DynamoDB ストリームから Elasticsearch サービスにデータを投稿できる Lambda 関数が必要です。サンプル関数が提供されています [ここ](#)。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

## Amazon DynamoDB テーブル

- DynamoDB テーブルの請求モードをオンデマンドに設定する ( リクエストごとの支払い )
- AWS マネージド KMS キーを使用した DynamoDB テーブルのサーバー側の暗号化の有効化
- DynamoDB テーブルの 'id' という名前のパーティションキーを作成します。
- CloudFormation スタックを削除するときにテーブルを保持する
- 継続的なバックアップとポイントインタイムリカバリを有効にします

## AWS Lambda 関数

- Lambda 関数の制限付き特権アクセスの IAM ロールを設定する
- NodeJS Lambda 関数のキープアライブを使用して接続を再利用できるようにする
- X-Ray トレースを有効にする
- 障害処理機能の有効化:関数エラーの bisect の有効化、デフォルトの最大レコード有効期間 (24 時間) の設定、デフォルトの最大再試行回数 (500) の設定、障害発生時の宛先として SQS デッドレターキューのデプロイ
- 環境変数の設定:

- `AWS_NODEJS_CONNECTION_REUSE_ENABLED` ( ノード10.x以上の機能の場合 )

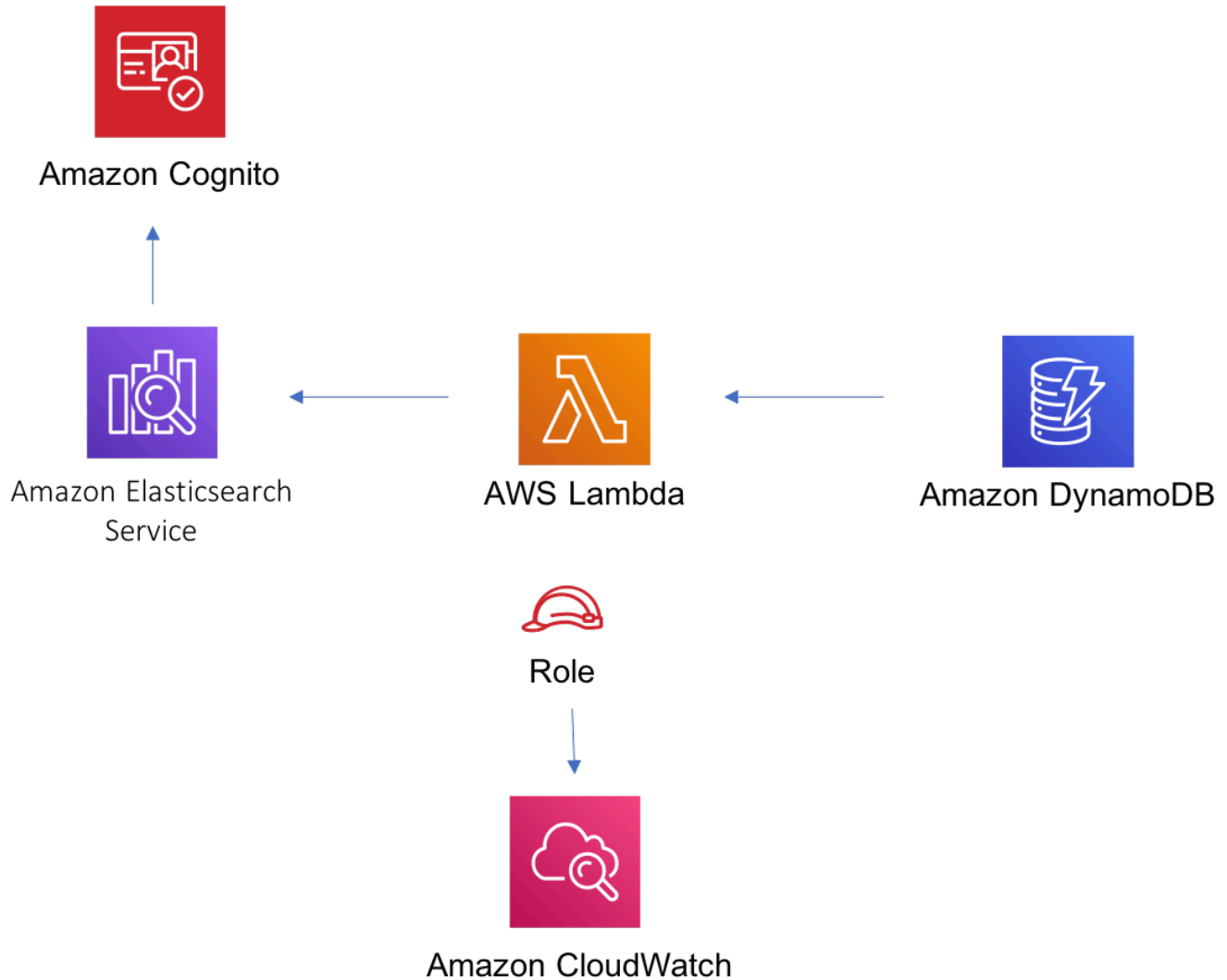
## Amazon Cognito

- UserPoolのパスワードポリシーを設定する
- ユーザープールの高度なセキュリティモードを適用する

## Amazon Elasticsearch Service

- Elasticsearch ドメイン用のベストプラクティス CloudWatch アラームのデプロイ
- Cognito ユーザープールを使用した Kibana ダッシュボードのアクセスの保護
- AWS マネージド KMS キーを使用した Elasticsearch ドメインのサーバー側の暗号化の有効化
- Elasticsearchドメインのノード間の暗号化を有効にする
- Amazon ES ドメインのクラスターを設定する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-ソリューション-構築/aws-dynamodb-stream-lambda-elasticsearch-kibana](#)

## aws-events-ルール-キネシファイアホース-3

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンング](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_events_rule_kinesisfirehose_s3</code>
 TypeScript	<code>@aws-solutions-constructs/aws-events-rule-kinesisfirehose-s3</code>
 Java	<code>software.amazon.awsconstructs.services.eventsrulekinesisfirehoses3</code>

## Overview

この AWS ソリューション構築は、Amazon S3 バケットに接続された Amazon Kinesis データ Firehose 配信ストリームにデータを送信する Amazon CloudWatch Events ルールを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import * as cdk from '@aws-cdk/core';
import { EventsRuleToKinesisFirehoseToS3, EventsRuleToKinesisFirehoseToS3Props } from
 '@aws-solutions-constructs/aws-events-rule-kinesisfirehose-s3';

const eventsRuleToKinesisFirehoseToS3Props: EventsRuleToKinesisFirehoseToS3Props = {
  eventRuleProps: {
    schedule: events.Schedule.rate(cdk.Duration.minutes(5))
  }
};
```



```
new EventsRuleToKinesisFirehoseToS3(this, 'test-events-rule-firehose-s3',
  eventsRuleToKinesisFirehoseToS3Props);
```

## Initializer

```
new EventsRuleToKinesisFirehoseToS3(scope: Construct, id: string, props:
  EventsRuleToKinesisFirehoseToS3Props);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [EventsRuleToKinesisFirehoseToS3Props](#)

### パターン構成プロパティ

名前	タイプ	説明
EventRuleProps	<a href="#">events.RuleProps</a>	CloudWatch イベントルールのデフォルトプロパティを上書きするユーザー指定のプロパティ。
KineSisFireHoseprops ?	<a href="#">aws-kinesisfirehose.CfnDeliveryStreamProps</a>	Kinesis Firehose 配信ストリームのデフォルトの小道具を上書きするオプションのユーザーが提供した小道具です。
ExistingBucketObj ?	<a href="#">s3.IBucket</a>	S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。

名前	タイプ	説明
BucketProps ?	<a href="#"><u>s3.BucketProps</u></a>	オプションのユーザー提供の小道具で、S3 バケットのデフォルトの小道具を上書きします。
LogGroupProps ?	<a href="#"><u>logs.LogGroupProps</u></a>	CloudWatch Logs ロググループのデフォルト小道具を上書きする、オプションのユーザー指定の小道具です。

## パターンプロパティ

名前	タイプ	説明
イベントルール	<a href="#"><u>events.Rule</u></a>	パターンによって作成されたイベントルールのインスタンスを返します。
キネシファイアホース	<a href="#"><u>kinesisfirehose.CfnDeliveryStream</u></a>	パターンによって作成された Kinesis Firehose 配信ストリームのインスタンスを返します。
bucket	<a href="#"><u>s3.Bucket</u></a>	パターンによって作成された S3 バケットのインスタンスを返します。
s3loggingBucket ?	<a href="#"><u>s3.Bucket</u></a>	S3 バケットのパターンによって作成されたロギングバケットのインスタンスを返します。
EventsRole?	<a href="#"><u>iam.Role</u></a>	CloudWatch イベントルールのコンストラクトによって作

名前	タイプ	説明
		成されたロールのインスタンスを返します。
キネシファイアホセロール	<a href="#">iam.Role</a>	Kinesis Firehose 配信ストリームのパターンによって作成された IAM ロールのインスタンスを返します。
キネシファイアホースロググループ	<a href="#">logs.LogGroup</a>	Kinesis Firehose アクセスログの送信先のパターンによって作成されたロググループのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon CloudWatch Events Events

- イベントルールの最小権限アクセス IAM ロールを設定し、Kinesis Firehose 配信ストリームに発行します。

### Amazon Kinesis Firehose

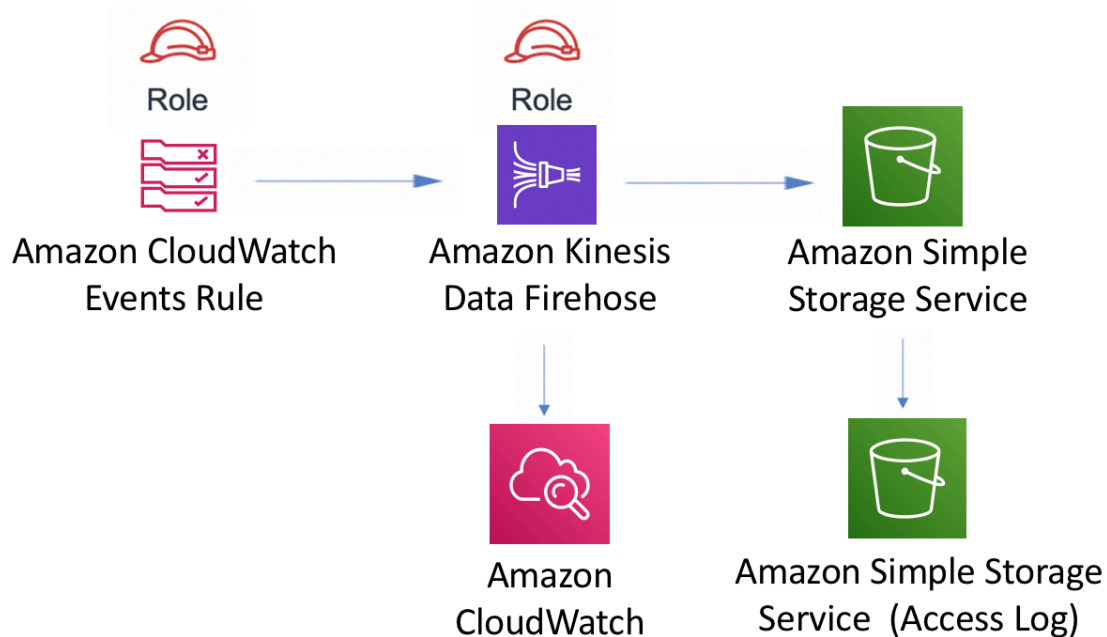
- Kinesis Firehose の CloudWatch ロギングを有効にします。
- Amazon Kinesis Firehose Firehose Firehose

### Amazon S3 バケット

- バケットのアクセスログを設定します。
- AWS マネージド KMS キーを使用して、バケットのサーバー側の暗号化を有効にします。
- バケットのバージョニングを有効にします。
- バケットのパブリックアクセスを許可しないでください。
- CloudFormation スタックを削除するときはバケットを保持します。

- 90 日後に Glacier ストレージに最新でないオブジェクトバージョンを移動するライフサイクルルールを適用します。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-constructs/aws-events-ルールキネシファイアホース-3](https://github.com/aws-solutions-constructs/aws-events-ルールキネシファイアホース-3)



## aws-events-ルールキネシスストリーム

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンング](#)モデル。つまり、これらのパッケージ

を使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_events_rule_kinesisstream</code>
 TypeScript	<code>@aws-solutions-constructs/aws-events-rule-kinesisstreams</code>
 Java	<code>software.amazon.awsconstructs.services.eventsrulekinesisstream</code>

## Overview

この AWS ソリューション構築では、Amazon CloudWatch Events ルールを実装して Amazon Kinesis データストリームにデータを送信します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import * as cdk from '@aws-cdk/core';
import {EventsRuleToKinesisStreams, EventsRuleToKinesisStreamsProps} from "@aws-solutions-constructs/aws-events-rule-kinesisstreams";

const props: EventsRuleToKinesisStreamsProps = {
  eventRuleProps: {
    schedule: events.Schedule.rate(Duration.minutes(5)),
  }
};

new EventsRuleToKinesisStreams(this, 'test-events-rule-kinesis-stream', props);
```

## Initializer

```
new EventsRuleToKinesisStreams(scope: Construct, id: string, props:
  EventsRuleToKinesisStreamsProps);
```

### パラメータ

- scope [Construct](#)
- id `string`
- props [EventsRuleToKinesisStreamsProps](#)

### パターン構成プロパティ

名前	タイプ	説明
EventRuleProps	<a href="#">events.RuleProps</a>	CloudWatch イベントルールのデフォルトプロパティを上書きするユーザー指定のプロパティ。
ExistingStreamObj?	<a href="#">kinesis.Stream</a>	Kinesis ストリームの既存のインスタンスで、これと <code>kinesisStreamProps</code> はエラーを発生させます。
KinesisStreamProps?	<a href="#">kinesis.StreamProps</a>	Kinesis ストリームのデフォルトのプロップを上書きするオプションのユーザー指定のプロップ。
クラウド・ワット・チャラームズ	<code>boolean</code>	推奨される CloudWatch アラームを作成するかどうか。

## パターンプロパティ

名前	タイプ	説明
イベントルール	<a href="#">events.Rule</a>	パターンによって作成されたイベントルールのインスタンスを返します。
KinesisStream	<a href="#">kinesis.Stream</a>	パターンによって作成された Kinesis ストリームのインスタンスを返します。
EventsRole?	<a href="#">iam.Role</a>	CloudWatch イベントルールのコンストラクトによって作成されたロールのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

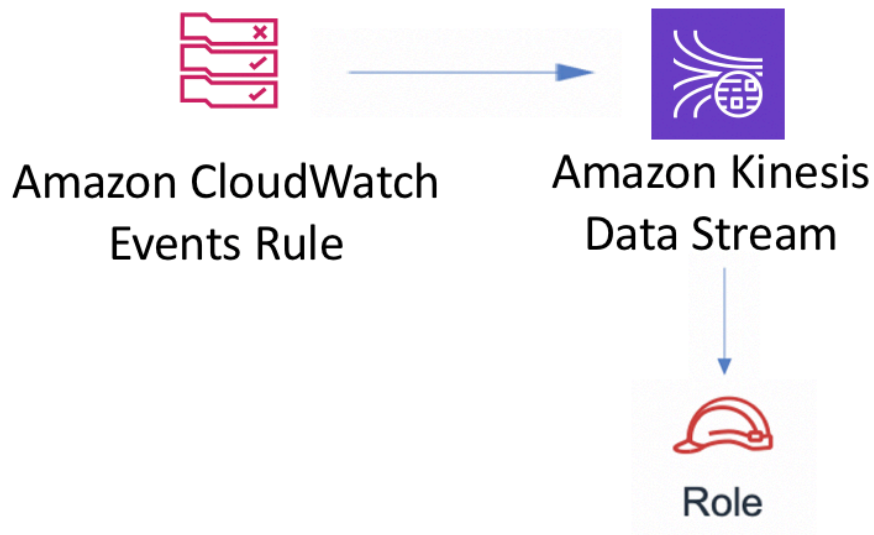
### Amazon CloudWatch Events ルール

- Kinesis データストリームにパブリッシュするイベントルールの最小権限アクセス IAM ロールを設定します。

### Amazon Kinesis Stream

- AWS マネージド KMS キーを使用して、Kinesis データストリームのサーバー側の暗号化を有効にします。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-constructions/aws-events-ルールキネシスストリーム](#)




## aws-events-ルール-ラムダ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョン管理](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。



注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_events_rule_lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-events-rule-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.eventsrulelambda</code>

## Overview

この AWS ソリューション構築は、AWS Events ルールと AWS Lambda 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
const { EventsRuleToLambdaProps, EventsRuleToLambda } from '@aws-solutions-constructs/aws-events-rule-lambda';

const props: EventsRuleToLambdaProps = {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  },
  eventRuleProps: {
    schedule: events.Schedule.rate(Duration.minutes(5))
  }
};

new EventsRuleToLambda(this, 'test-events-rule-lambda', props);
```

## Initializer

```
new EventsRuleToLambda(scope: Construct, id: string, props: EventsRuleToLambdaProps);
```

### パラメータ

- scope [Construct](#)
- id `string`
- props [EventsRuleToLambdaProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
ラムダファンクション	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj が提供される。
EventRuleProps	<a href="#">events.RuleProps</a>	デフォルトを上書きするためにユーザーが指定したEventRuleProps

## パターンプロパティ

名前	タイプ	説明
イベントルール	<a href="#">events.Rule</a>	パターンによって作成されたイベントルールのインスタンスを返します。
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。

### デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

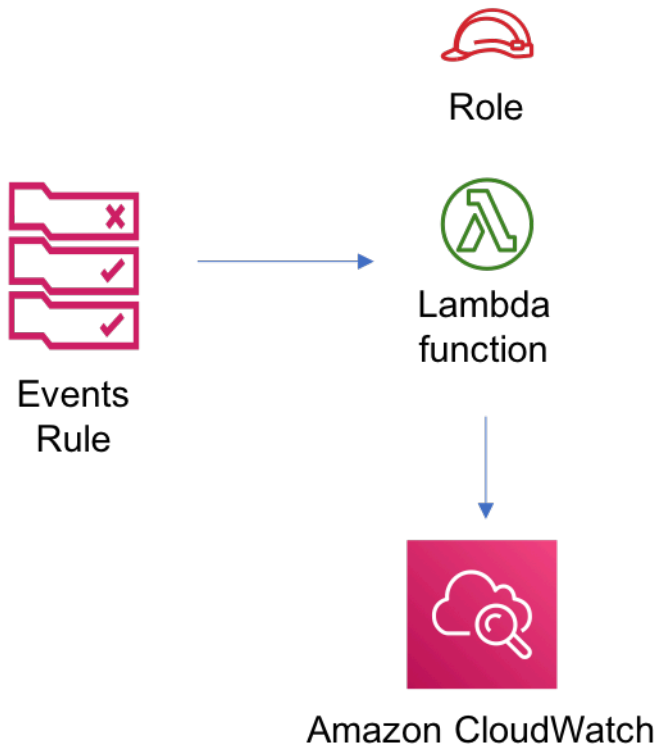
#### Amazon CloudWatch Events Events ルール

- Lambda 関数をトリガーするための CloudWatch イベントへの最小権限の付与

#### AWS Lambda 関数

- Lambda 関数用に制限された特権アクセス IAM ロールを設定する
- NodeJS Lambda 関数のキープアライブを使用して接続を再利用できるようにする
- X-Ray トレースを有効化
- 環境変数の設定:
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-ソリューション-構築/aws-events-ルール-ラムダ](#)

## aws-イベント-ルール-sns

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_events_rule_sns</code>
 TypeScript	<code>@aws-solutions-constructs/aws-events-rule-sns</code>
 Java	<code>software.amazon.awsconstructs.services.eventsrulesns</code>

## Overview

このパターンは、Amazon SNS トピックに関連付けられている Amazon CloudWatch Events ルールを実装します。

最小限のデプロイ可能なパターン定義を次に示します。

```
import { Duration } from '@aws-cdk/core';
import * as events from '@aws-cdk/aws-events';
import * as iam from '@aws-cdk/aws-iam';
import { EventsRuleToSnsProps, EventsRuleToSns } from "@aws-solutions-constructs/aws-events-rule-sns";

const props: EventsRuleToSnsProps = {
  eventRuleProps: {
    schedule: events.Schedule.rate(Duration.minutes(5)),
  }
};

const constructStack = new EventsRuleToSns(this, 'test-construct', props);

// Grant yourself permissions to use the Customer Managed KMS Key
const policyStatement = new iam.PolicyStatement({
  actions: ["kms:Encrypt", "kms:Decrypt"],
  effect: iam.Effect.ALLOW,
  principals: [ new iam.AccountRootPrincipal() ],
```

```
resources: [ "*" ]
});

constructStack.encryptionKey?.addToResourcePolicy(policyStatement);
```

## Initializer

```
new EventsRuleToSNS(scope: Construct, id: string, props: EventsRuleToSNSProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [EventsRuleToSnsProps](#)

## パターン構成プロパティ

名前	タイプ	説明
EventRuleProps	<a href="#">events.RuleProps</a>	CloudWatch イベントルールのデフォルトプロパティを上書きするユーザー指定のプロパティ。
ExistingTopicObj ?	<a href="#">sns.Topic</a>	SNS トピックオブジェクトの既存のインスタンス。これと topicProps はエラーを発生させます。
TopicProps ?	<a href="#">sns.TopicProps</a>	SNS トピックのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。場合は、無視されず existingTopicObj が提供される。

名前	タイプ	説明
顧客管理キーによる暗号化の有効化	boolean	カスタマー管理型の暗号化キーを使用するかどうか。このCDKアプリによって管理されるか、インポートされず。暗号化キーをインポートする場合は、暗号化キーを <code>encryptionKey</code> プロパティです。
<code>encryptionKey</code>	<a href="#">kms.Key</a>	デフォルトの暗号化キーの代わりに使用する、オプションの既存の暗号化キー。
暗号化キープロップ？	<a href="#">kms.KeyProps</a>	オプションのユーザー指定のプロパティで、暗号化キーのデフォルトプロパティを上書きします。

## パターンプロパティ

名前	タイプ	説明
イベントルール	<a href="#">events.Rule</a>	パターンによって作成されたイベントルールのインスタンスを返します。
<code>snsTopic</code>	<a href="#">sns.Topic</a>	パターンによって作成された SNS トピックのインスタンスを返します。
<code>encryptionKey</code>	<a href="#">kms.Key</a>	パターンによって作成された暗号化キーのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

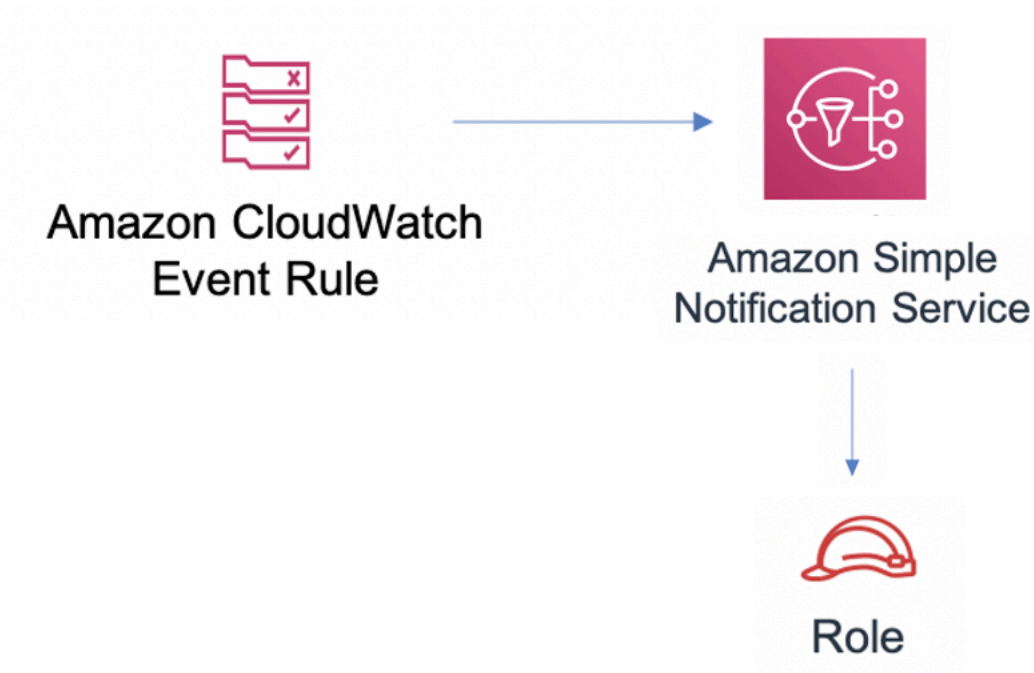
### Amazon CloudWatch Events

- SNS トピックにパブリッシュする CloudWatch イベントに、最小限の権限を付与します。

### Amazon SNS トピック

- SNS トピックの最小権限アクセス権限を設定します。
- カスタマー管理型の AWS KMS キーを使用して、SNS トピックのサーバー側の暗号化を可能にします。
- 転送時のデータの暗号化を強制する

## Architecture





## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-構築/aws-events-ルール-sns](#)

## aws-イベント-ルール-sqs

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョン管理](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	aws_solutions_constructs.aws_events_rule_sqs
 TypeScript	@aws-solutions-constructs/aws-events-rule-sqs
 Java	software.amazon.awsconstructs.services.eventsrulesqs

## Overview

このパターンは、Amazon SQS キューに接続された Amazon CloudWatch Events ルールを実装します。

最小限のデプロイ可能なパターン定義を次に示します。

```
import { Duration } from '@aws-cdk/core';
import * as events from '@aws-cdk/aws-events';
import * as iam from '@aws-cdk/aws-iam';
import { EventsRuleToSqsProps, EventsRuleToSqs } from "@aws-solutions-constructs/aws-
events-rule-sqs";

const props: EventsRuleToSqsProps = {
  eventRuleProps: {
    schedule: events.Schedule.rate(Duration.minutes(5))
  }
};

const constructStack = new EventsRuleToSqs(this, 'test-construct', props);

// Grant yourself permissions to use the Customer Managed KMS Key
const policyStatement = new iam.PolicyStatement({
  actions: ["kms:Encrypt", "kms:Decrypt"],
  effect: iam.Effect.ALLOW,
  principals: [ new iam.AccountRootPrincipal() ],
  resources: [ "*" ]
});

constructStack.encryptedKey?.addToResourcePolicy(policyStatement);
```

## Initializer

```
new EventsRuleToSqs(scope: Construct, id: string, props: EventsRuleToSqsProps);
```

### パラメータ

- `scope` [Construct](#)
- `id` `string`
- `props` [EventsRuleToSqsProps](#)

## パターン構成プロパティ

名前	タイプ	説明
EventRuleProps	<a href="#">events.RuleProps</a>	CloudWatch イベントルールのデフォルトプロパティを上書きするユーザー指定のプロパティ。
ExistingQueueObj?	<a href="#">sqs.Queue</a>	デフォルトキューの代わりに使用されるオプションの既存の SQS キュー。これと queueProps はエラーを発生させます。
QueueProp	<a href="#">sqs.QueueProps</a>	SQS キューのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティです。次の場合は無視されます。existingQueueObj が提供される。
キューのパージを有効化しますか?	boolean	SQS キューの削除を可能にする Lambda 関数に追加のアクセス許可を付与するかどうか。デフォルトは false です。
DeployDeadleterQueue	boolean	デッドレターキューとして使用するセカンダリキューを作成するかどうか。デフォルトは true です。
DeadletterQueueProps?	<a href="#">sqs.QueueProps</a>	デッドレターキューのデフォルト小道具を上書きするオプションのユーザー提供の小道具です。場合にのみ使用されません。deployDeadLetterQu

名前	タイプ	説明
		eue プロパティが true に設定された場合。
maxReceiveCount?	number	デッドレターキューに移動する前に、メッセージがデキューに失敗した回数。デフォルトは 15 です。
顧客管理キーによる暗号化の有効化	boolean	カスタマー管理型の暗号化キーを使用するかどうか。このCDKアプリによって管理されるか、インポートされます。暗号化キーをインポートする場合、暗号化キーを encryptionKey プロパティです。
encryptionKey	<a href="#">kms.Key</a>	デフォルトの暗号化キーの代わりに使用する、オプションの既存の暗号化キー。
EncryptionKeyProps ?	<a href="#">kms.KeyProps</a>	オプションのユーザー指定のプロパティで、暗号化キーのデフォルトプロパティを上書きします。

## パターンプロパティ

名前	タイプ	説明
イベントルール	<a href="#">events.Rule</a>	パターンによって作成されたイベントルールのインスタンスを返します。

名前	タイプ	説明
SQUEUE	<a href="#">sqs.Queue</a>	パターンによって作成された SQS キューのインスタンスを返します。
encryptionKey	<a href="#">kms.Key</a>	パターンによって作成された暗号化キーのインスタンスを返します。
デッドレターキューですか？	<a href="#">sqs.Queue</a>	パターンによって作成されたデッドレターキューのインスタンスを返します ( デプロイされている場合 )。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

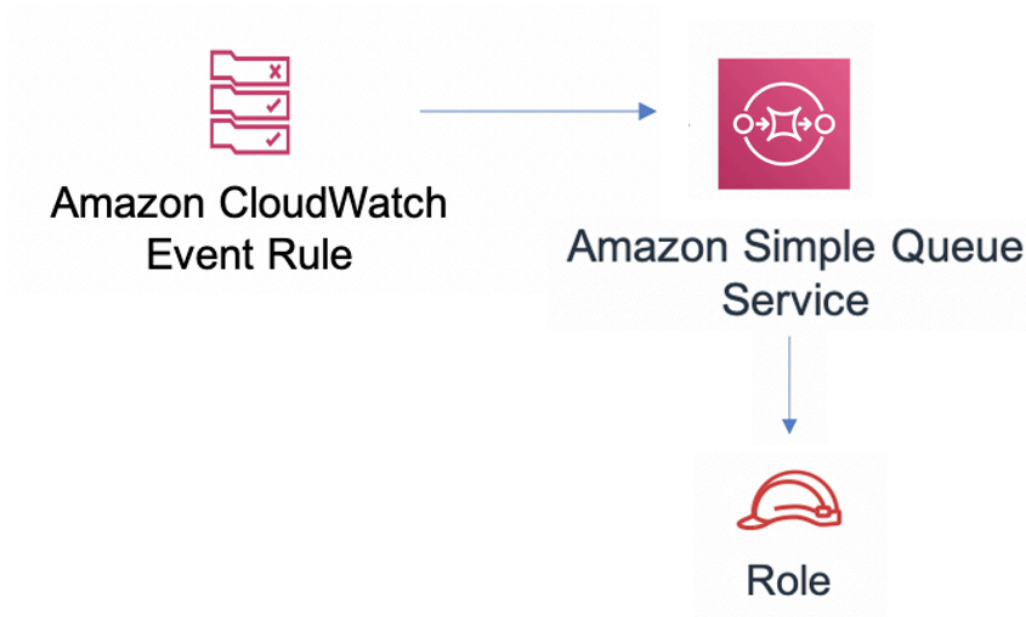
### Amazon CloudWatch Events ルール

- SQS キューに発行する CloudWatch イベントに最小限の権限を付与します。

### Amazon SQS キュー

- ソースキューのデッドレターキューをデプロイする。
- カスタマー管理型の AWS KMS キーを使用して、ソースキューのサーバー側の暗号化を可能にします。
- 転送時のデータの暗号化を強制する。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-イベント-ルール-sqs](#)




## aws-イベント-ルール-ステップ関数

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョニングモデル](#)。つまり、これらのパッケージ

を使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_events_rule_step_function</code>
 TypeScript	<code>@aws-solutions-constructs/aws-events-rule-step-function</code>
 Java	<code>software.amazon.awsconstructs.services.eventsrulestepfunction</code>

## Overview

この AWS ソリューション構築は、AWS イベントルールと AWS Step 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { EventsRuleToStepFunction, EventsRuleToStepFunctionProps } from '@aws-solutions-constructs/aws-events-rule-step-function';

const startState = new stepfunctions.Pass(this, 'StartState');

const props: EventsRuleToStepFunctionProps = {
  stateMachineProps: {
    definition: startState
  },
  eventRuleProps: {
    schedule: events.Schedule.rate(Duration.minutes(5))
  }
};
```

```
new EventsRuleToStepFunction(this, 'test-events-rule-step-function-stack', props);
```

## Initializer

```
new EventsRuleToStepFunction(scope: Construct, id: string, props:
  EventsRuleToStepFunctionProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [EventsRuleToStepFunctionProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ステートメントマシンプロップ	<a href="#">sfn.StateMachinePr ops</a>	sfn.stateMachineのデフォルト小道具を上書きするためのオプションのユーザ提供の小道具です
EventRuleProps	<a href="#">events.RuleProps</a>	デフォルトを上書きするためにユーザーが指定したEventRuleProps
クラウド・ワット・チャラームズ	boolean	推奨される CloudWatch アラームを作成するかどうか。
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループのデフォルトの小道具を上書きする、オプションのユーザー指定の小道具です。



## パターンプロパティ

名前	タイプ	説明
CloudWatchAlarms?	<a href="#">cloudwatch.Alarm[]</a>	パターンによって作成された1つ以上のCloudWatch Alarmのリストを返します。
イベントルール	<a href="#">events.Rule</a>	パターンによって作成されたイベントルールのインスタンスを返します。
StateMachine	<a href="#">sfn.StateMachine</a>	パターンによって作成されたステートマシンのインスタンスを返します。
ステートメントマシンロググループ	<a href="#">logs.LogGroup</a>	ステートマシンのパターンによって作成されたロググループのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

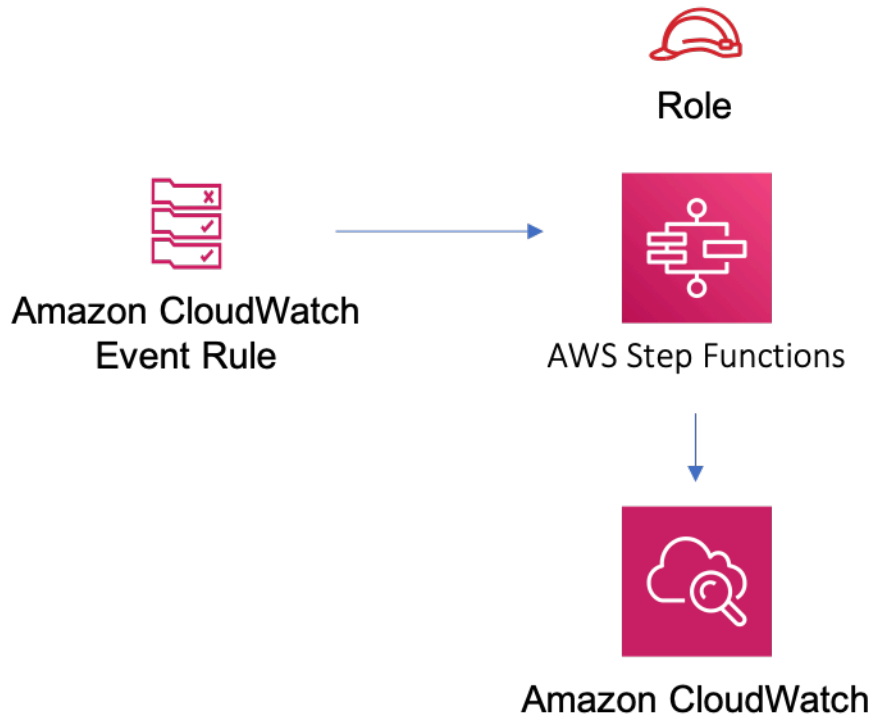
### Amazon CloudWatch Events ルール

- Lambda 関数をトリガーするための CloudWatch イベントへの最小権限の付与

### AWS ステップ関数

- API Gateway での CloudWatch によるロギングの有効化
- ステップ機能用のベストプラクティスの CloudWatch アラームのデプロイ

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-ソリューション-構築/aws-イベント-ルール-ステップ関数](#)

## aws-iot-キネシスファイアホース-3

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_iot_kinesisfirehose_s3</code>
 TypeScript	<code>@aws-solutions-constructs/aws-iot-kinesisfirehose-s3</code>
 Java	<code>software.amazon.awsconstructs.services.iotkinesisfirehoses3</code>

## Overview

この AWS ソリューション構築は、Amazon S3 バケットに接続された Amazon Kinesis データ Firehose 配信ストリームにデータを送信する AWS IoT MQTT トピックルールを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { IotToKinesisFirehoseToS3Props, IotToKinesisFirehoseToS3 } from '@aws-solutions-constructs/aws-iot-kinesisfirehose-s3';

const props: IotToKinesisFirehoseToS3Props = {
  iotTopicRuleProps: {
    topicRulePayload: {
      ruleDisabled: false,
      description: "Persistent storage of connected vehicle telematics data",
      sql: "SELECT * FROM 'connectedcar/telemetry/#'",
      actions: []
    }
  }
};

new IotToKinesisFirehoseToS3(this, 'test-iot-firehose-s3', props);
```

## Initializer

```
new IotToKinesisFirehoseToS3(scope: Construct, id: string, props:
  IotToKinesisFirehoseToS3Props);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [IotToKinesisFirehoseToS3Props](#)

### パターン構成プロパティ

名前	タイプ	説明
IOTTopicRuleProps	<a href="#">iot.CfnTopicRulePr ops</a>	デフォルトを上書きするためにユーザーが提供した CfnTopicRuleProps
KineSisFireHoseprops ?	<a href="#">kinesisfirehose.Cf nDeliveryStreamPro ps</a>	Kinesis Firehose 配信ストリームのデフォルトのプロップを上書きするオプションのユーザー提供の小道具
ExistingBucketObj ?	<a href="#">s3.Bucket</a>	S3 Bucket オブジェクトの既存のインスタンス。これと bucketProps はエラーを発生させます。
BucketProps ?	<a href="#">s3.BucketProps</a>	S3 バケットのデフォルトの小道具を上書きするために、ユーザーが提供した小道具です。これが提供されている場合は、bucketProps はエラーです。

名前	タイプ	説明
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループのデフォルト小道具を上書きする、オプションのユーザー指定の小道具です。

## パターンプロパティ

名前	タイプ	説明
IoTactionsRole	<a href="#">iam.Role</a>	IoT ルールのパターンによって作成された IAM ロールのインスタンスを返します。
IOTTopicRule	<a href="#">iot.CfnTopicRule</a>	パターンによって作成された IoT トピックルールのインスタンスを返します。
キネシファイアホース	<a href="#">kinesisfirehose.CfnDeliveryStream</a>	パターンによって作成された Kinesis Firehose 配信ストリームのインスタンスを返します。
キネシファイアホースロググループ	<a href="#">logs.LogGroup</a>	Kinesis Firehose アクセスログの送信先のパターンによって作成されたロググループのインスタンスを返します。
キネシファイアホースロール	<a href="#">iam.Role</a>	Kinesis Firehose 配信ストリームのパターンによって作成された IAM ロールのインスタンスを返します。
S3bucket?	<a href="#">s3.Bucket</a>	パターンによって作成された S3 バケットのインスタンスを返します。

名前	タイプ	説明
s3loggingBucket ?	<a href="#">s3.Bucket</a>	S3 バケットのパターンによって作成されたロギングバケットのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon IoT ルール

- Amazon IoT の最小権限アクセス IAM ロールを設定する

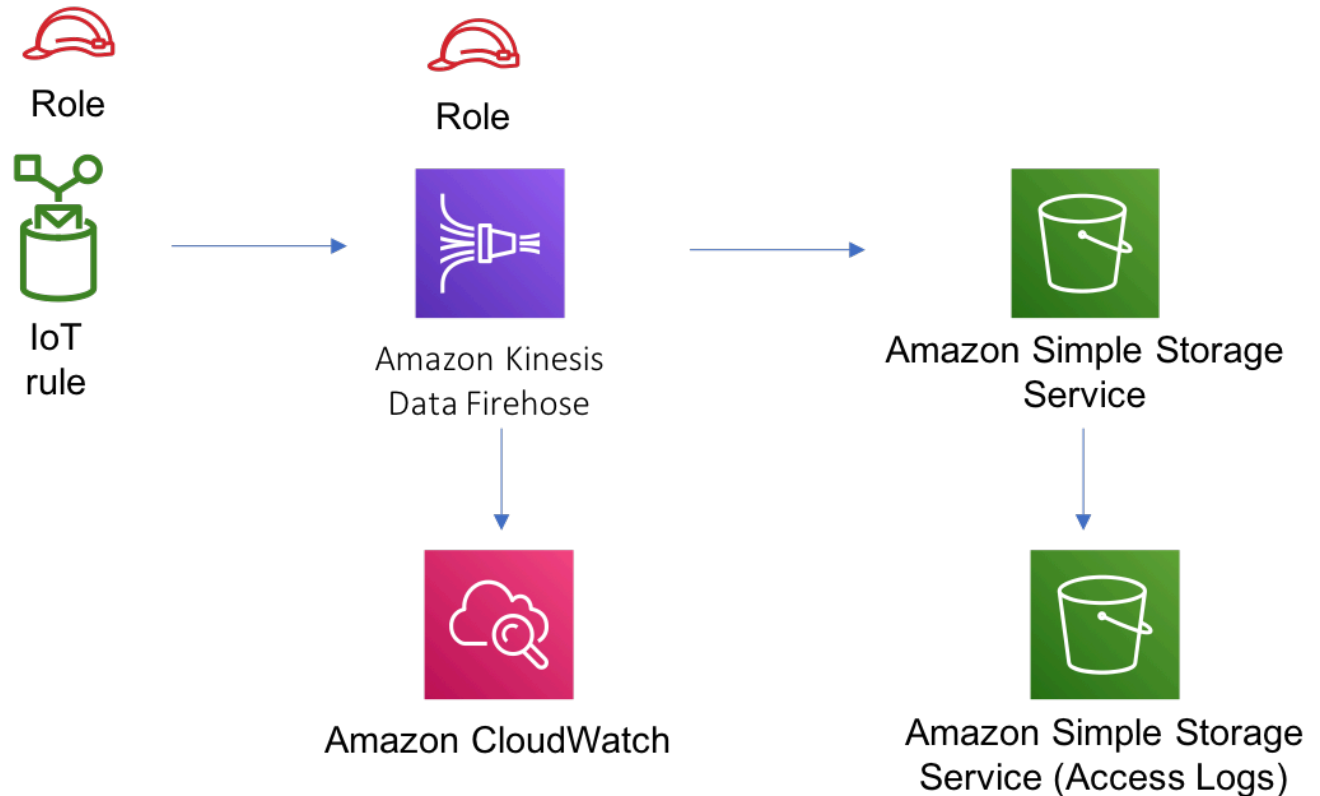
### Amazon Kinesis Firehose

- Kinesis Firehose の CloudWatch ログ記録を有効にする
- Amazon Kinesis Firehose の最小権限アクセス IAM ロールを設定する

### Amazon S3 バケットのパターン

- S3 バケットのアクセスログの設定
- AWS マネージド KMS キーを使用した S3 バケットのサーバー側の暗号化の有効化
- S3 バケットのバージョニングを有効にする
- S3 バケットのパブリックアクセスを許可しない
- CloudFormation スタックを削除するときに S3 バケットを保持する
- ライフサイクルルールを適用して、90 日後に最新でないオブジェクトバージョンを Glacier ストレージに移動する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-constructs/aws-iot-キネシシ  
ファイアホース-3](https://github.com/aws-solutions-constructs/aws-iot-キネシシファイアホース-3)

## aws-イオット-ラムダ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理モデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_iot_lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-iot-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.iotlambda</code>

## Overview

この AWS ソリューション構築パターンは、AWS IoT MQTT トピックルールと AWS Lambda 関数パターンを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { IotToLambdaProps, IotToLambda } from '@aws-solutions-constructs/aws-iot-lambda';

const props: IotToLambdaProps = {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  },
  iotTopicRuleProps: {
    topicRulePayload: {
      ruleDisabled: false,
      description: "Processing of DTC messages from the AWS Connected Vehicle Solution.",

```



```
        sql: "SELECT * FROM 'connectedcar/dtc/#'",
        actions: []
    }
}
};

new IotToLambda(this, 'test-iot-lambda-integration', props);
```

## Initializer

```
new IotToLambda(scope: Construct, id: string, props: IotToLambdaProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [IotToLambdaProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj が提供される。

名前	タイプ	説明
IOtopicRuleProps ?	<a href="#"><u>iot.CfnTopicRulePr ops</u></a>	デフォルトを上書きするためにユーザが提供したCFntopicRuleProps

## パターンプロパティ

名前	タイプ	説明
IOTTopicRule	<a href="#"><u>iot.CfnTopicRule</u></a>	パターンによって作成されたIoT トピックルールのインスタンスを返します。
LambdaFunction	<a href="#"><u>lambda.Function</u></a>	パターンによって作成されたLambda 関数のインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

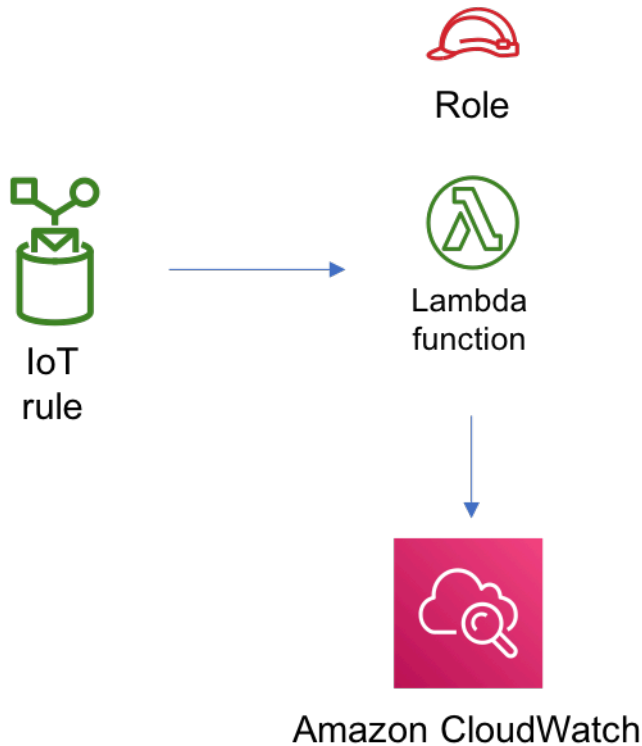
### Amazon IoT ルール

- Amazon IoT の最小権限アクセス IAM ロールを設定します。

### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にします。
- 環境変数の設定:
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-構築/aws-iot-lambda](#)

## aws-iot-ラムダ-ダイナモブ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョニング](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_iam_lambda_dynamodb</code>
 TypeScript	<code>@aws-solutions-constructs/aws-iot-lambda-dynamodb</code>
 Java	<code>software.amazon.awsconstructs.services.iotlambdadynamodb</code>

## Overview

この AWS ソリューション構築パターンは、AWS IoT トピックルール、AWS Lambda 関数、および Amazon DynamoDB テーブルを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { IotToLambdaToDynamoDBProps, IotToLambdaToDynamoDB } from '@aws-solutions-constructs/aws-iot-lambda-dynamodb';

const props: IotToLambdaToDynamoDBProps = {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  },
  iotTopicRuleProps: {
    topicRulePayload: {
      ruleDisabled: false,
      description: "Processing of DTC messages from the AWS Connected Vehicle Solution.",
      sql: "SELECT * FROM 'connectedcar/dtc/#'",
      actions: []
    }
  }
}
```

```

    }
};

new IotToLambdaToDynamoDB(this, 'test-iot-lambda-dynamodb-stack', props);

```

## Initializer

```

new IotToLambdaToDynamoDB(scope: Construct, id: string, props:
  IotToLambdaToDynamoDBProps);

```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [IotToLambdaToDynamoDBProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
ラムダファンクション	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj が提供される。
IOTTopicRuleProps	<a href="#">iot.CfnTopicRuleProps</a>	デフォルトの小道具を上書きするためにユーザーが提供した小道具の提供

名前	タイプ	説明
DynamoTableProps ?	<a href="#"><u>dynamodb.TableProps</u></a>	DynamoDB テーブルのデフォルトの小道具をオーバーライドするオプションのユーザー提供の小道具です
テーブルパーミッション	<a href="#"><u>string</u></a>	Lambda 関数に付与されるオプションのテーブルパーミッション。以下のいずれかのオプションを指定できません。All,Read,ReadWrite , またはWrite。

## パターンプロパティ

名前	タイプ	説明
ダイナモテーブル	<a href="#"><u>dynamodb.Table</u></a>	パターンによって作成された DynamoDB テーブルのインスタンスを返します。
IOTTopicRule	<a href="#"><u>iot.CfnTopicRule</u></a>	パターンによって作成された IoT トピックルールのインスタンスを返します。
LambdaFunction	<a href="#"><u>lambda.Function</u></a>	パターンによって作成された Lambda 関数のインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon IoT ルール

- Amazon IoT の最小権限アクセス IAM ロールを設定します。

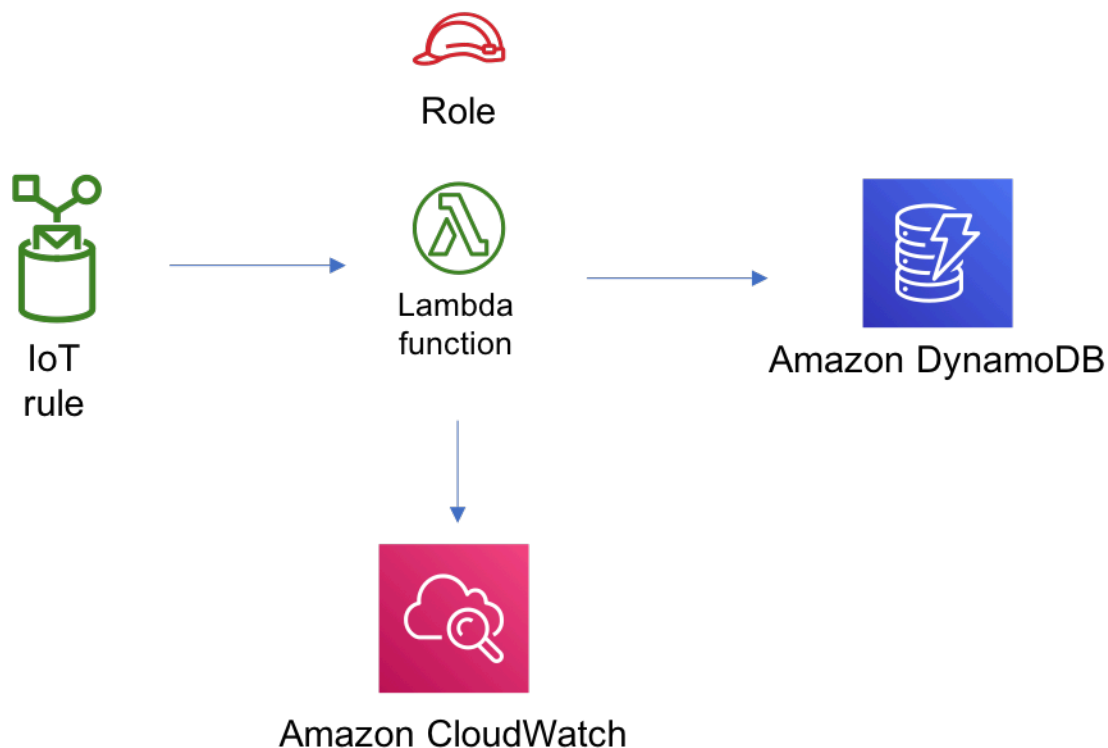
## AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にします。
- 環境変数の設定:
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED` ( ノード10.x以上の機能の場合 )

## Amazon DynamoDB テーブル

- DynamoDB テーブルの請求モードをオンデマンドに設定します ( リクエストごとの支払い )。
- AWS マネージド KMS キーを使用して DynamoDB テーブルのサーバー側の暗号化を有効にします。
- DynamoDB テーブルの 'id' という名前のパーティションキーを作成します。
- CloudFormation スタックを削除するときに、テーブルを保持します。
- 継続的なバックアップとポイントインタイムリカバリを可能にします。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。





[@aws-solutions-constructs/aws-iot-lambda-ダイナモブ](#)

## aws-キネシシファイアホース-3

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	aws_solutions_constructs.aws-kinesis-firehose-s3
 TypeScript	@aws-solutions-constructs/aws-kinesisfirehose-s3
 Java	software.amazon.awsconstructs.services.kinesisfirehoses3

## Overview

この AWS ソリューション構築は、Amazon S3 バケットに接続された Amazon Kinesis Data Firehose 配信ストリームを実装します。



TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { KinesisFirehoseToS3 } from '@aws-solutions-constructs/aws-kinesisfirehose-s3';

new KinesisFirehoseToS3(this, 'test-firehose-s3', {});
```

## Initializer

```
new KinesisFirehoseToS3(scope: Construct, id: string, props: KinesisFirehoseToS3Props);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [KinesisFirehoseToS3Props](#)

### パターン構成プロパティ

名前	タイプ	説明
BucketProps ?	<a href="#">s3.BucketProps</a>	S3 バケットのデフォルトの小道具を上書きするオプションのユーザーが提供した小道具です。
ExistingBucketObj ?	<a href="#">s3.IBucket</a>	オプションの S3 バケットの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。
ExistingLoggingBucketObj ?	<a href="#">s3.IBucket</a>	パターンによって作成された S3 バケットのログ S3 バケッ

名前	タイプ	説明
		トの既存のインスタンス ( オプション )。
KineSisFireHoseprops ?	<a href="#">kinesisfirehose.CfnDeliveryStreamProps</a>   any	Kinesis Firehose 配信ストリームのデフォルトの小道具を上書きするオプションのユーザーが提供した小道具です。
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatchLogs LogGroup のデフォルトの小道具を上書きするオプションのユーザー提供の小道具です。

## パターンプロパティ

名前	タイプ	説明
キネシファイアホース	<a href="#">kinesisfirehose.CfnDeliveryStream</a>	コンストラクトによって作成された KinesisFirehose.cf nDeliveryStream のインスタンスを返します。
キネシファイアホースロググループ	<a href="#">logs.LogGroup</a>	Kinesis Data Firehose hose 配信ストリームのコンストラクトによって作成された Logs.logGroup のインスタンスを返します。
キネシファイアホセロール	<a href="#">iam.Role</a>	Kinesis Data Firehose hose 配信ストリームのコンストラクトによって作成された IAM.role のインスタンスを返します。

名前	タイプ	説明
S3bucket ?	<a href="#">s3.Bucket</a>	コンストラクトによって作成された S3.bucket のインスタンスを返します。
s3loggingBucket ?	<a href="#">s3.Bucket</a>	コンストラクトによって作成された S3.bucket のインスタンスを、プライマリバケットのロギングバケットとして返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

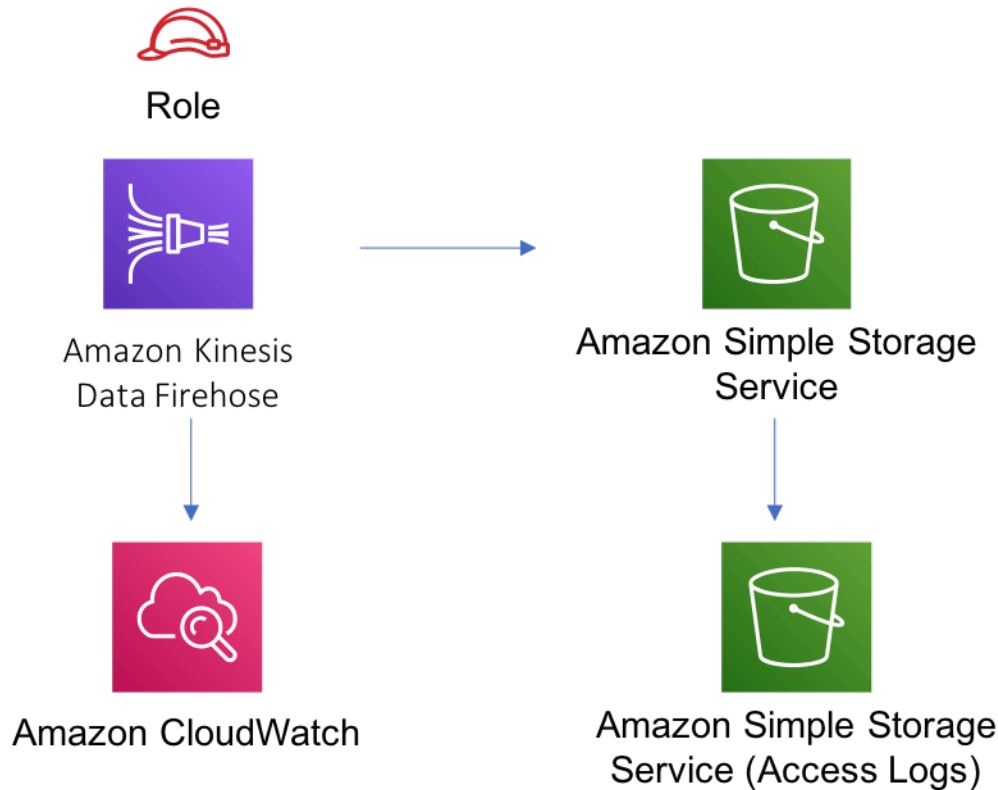
### Amazon Kinesis Firehose

- Kinesis Firehose の CloudWatch ログ記録を有効にする
- Amazon Kinesis Firehose の最小権限アクセス IAM ロールを設定する

### Amazon S3 バケット

- S3 バケットのアクセスログの設定
- AWS マネージド KMS キーを使用した S3 バケットのサーバー側の暗号化の有効化
- S3 バケットのバージョニングを有効にする
- S3 バケットのパブリックアクセスを許可しない
- CloudFormation スタックを削除するときに S3 バケットを保持する
- 転送時のデータの暗号化を強制する
- ライフサイクルルールを適用して、90 日後に最新でないオブジェクトバージョンを Glacier ストレージに移動する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-constructs/aws-kinesefirehose-s3](https://github.com/aws-solutions-constructs/aws-kinesefirehose-s3)

## aws-キネシスファイアホース-S3-アンドキネシス解析

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョンニング](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_kinesisfirehose_s3_and_kinesisanalytics</code>
 TypeScript	<code>@aws-solutions-constructs/aws-kinesisfirehose-s3-and-kinesisanalytics</code>
 Java	<code>software.amazon.awsconstructs.services.kinesisfirehose_s3kinesisanalytics</code>

## Overview

この AWS ソリューション構築は、Amazon S3 バケットに接続された Amazon Kinesis Firehose 配信ストリームと Amazon Kinesis Analytics アプリケーションを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { KinesisFirehoseToAnalyticsAndS3 } from '@aws-solutions-constructs/aws-kinesisfirehose-s3-and-kinesisanalytics';

new KinesisFirehoseToAnalyticsAndS3(this, 'FirehoseToS3AndAnalyticsPattern', {
  kinesisAnalyticsProps: {
    inputs: [{
      inputSchema: {
        recordColumns: [{
          name: 'ticker_symbol',
          sqlType: 'VARCHAR(4)',
          mapping: '$.ticker_symbol'
        }, {
          name: 'sector',
          sqlType: 'VARCHAR(16)',
          mapping: '$.sector'
        }
      ]
    }
  ]
}
```

```

    }, {
      name: 'change',
      sqlType: 'REAL',
      mapping: '$.change'
    }, {
      name: 'price',
      sqlType: 'REAL',
      mapping: '$.price'
    }
  ]],
  recordFormat: {
    recordFormatType: 'JSON'
  },
  recordEncoding: 'UTF-8'
},
namePrefix: 'SOURCE_SQL_STREAM'
]]
}
});

```

## Initializer

```

new KinesisFirehoseToAnalyticsAndS3(scope: Construct, id: string, props:
  KinesisFirehoseToAnalyticsAndS3Props);

```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [KinesisFirehoseToAnalyticsAndS3Props](#)

## パターン構成プロパティ

名前	タイプ	説明
KineSisFireHoseprops ?	<a href="#">kinesisFirehose.CfnDeliveryStreamProperties</a>	Kinesis Firehose 配信ストリームのデフォルトのプロップを

名前	タイプ	説明
		上書きするオプションのユーザー指定のプロップ。
KinesisAnalyticsProps?	<a href="#">kinesisAnalytics.CfnApplicationProps</a>	Kinesis Analytics アプリケーションのデフォルトのプロップを上書きするオプションのユーザー指定のプロップ。
ExistingBucketObj?	<a href="#">s3.IBucket</a>	S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。
BucketProps?	<a href="#">s3.BucketProps</a>	オプションのユーザー提供のプロパティ。バケットのデフォルトプロパティを上書きします。の場合は無視されます。existingBucketObj が提供される。
LogGroupProps?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループの CloudWatch Logs ロググループのデフォルト小道具を上書きする、ユーザー指定のオプションの小道具です。

## パターンプロパティ

名前	タイプ	説明
キネシス・アナリティクス	<a href="#">kinesisAnalytics.CfnApplication</a>	パターンによって作成された Kinesis Analytics アプリケーションのインスタンスを返します。

名前	タイプ	説明
キネシファイアホース	<a href="#">kinesisfirehose.CfnDeliveryStream</a>	パターンによって作成された Kinesis Firehose 配信ストリームのインスタンスを返します。
キネシファイアホースロググループ	<a href="#">logs.LogGroup</a>	Kinesis Firehose アクセスログの送信先のパターンによって作成されたロググループのインスタンスを返します。
キネシファイアホセロール	<a href="#">iam.Role</a>	Kinesis Firehose 配信ストリームのパターンによって作成された IAM ロールのインスタンスを返します。
S3bucket?	<a href="#">s3.Bucket</a>	パターンによって作成された S3 バケットのインスタンスを返します。
s3loggingBucket ?	<a href="#">s3.Bucket</a>	S3 バケットのパターンによって作成されたロギングバケットのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon Kinesis Firehose

- Kinesis Firehose の CloudWatch ログ記録を有効にする
- Amazon Kinesis Firehose の最小権限アクセス IAM ロールを設定する

### Amazon S3 バケット

- S3 バケットのアクセスログの設定



- AWS マネージド KMS キーを使用した S3 バケットのサーバー側の暗号化の有効化
- S3 バケットのバージョニングを有効にする
- S3 バケットのパブリックアクセスを許可しない
- CloudFormation スタックを削除するときに S3 バケットを保持する
- 転送時のデータの暗号化を強制する
- ライフサイクルルールを適用して、90 日後に最新でないオブジェクトバージョンを Glacier ストレージに移動する

## Amazon Kinesis Data Analytics

- Amazon Kinesis Analytics の最小権限アクセス IAM ロールを設定する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-構築/aws-kinesisfirehose-s3-and-kinesisanalytics](https://github.com/aws-solutions-構築/aws-kinesisfirehose-s3-and-kinesisanalytics)



# aws-キネシスストリーム-gluejob

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理モデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_kinesis_streams_gluejob</code>
 TypeScript	<code>@aws-solutions-constructs/aws-kinesisstreams-gluejob</code>
 Java	<code>software.amazon.awsconstructs.services.kinesisstreamsgluejob</code>

## Overview

この AWS ソリューション構成では、Amazon Kinesis データストリームをデプロイし、対話とセキュリティのために適切なリソース/プロパティを使用してカスタム ETL 変換を実行するように AWS Glue Job を設定します。また、AWS Glue Job 用の Python スクリプトをアップロードできる Amazon S3 バケットも作成されます。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import * as glue from '@aws-cdk/aws-glue';
import * as s3assets from '@aws-cdk/aws-s3-assets';
```

```
import { KinesisstreamsToGluejob } from '@aws-solutions-constructs/aws-kinesisstreams-gluejob';

const fieldSchema: glue.CfnTable.ColumnProperty[] = [
  {
    name: 'id',
    type: 'int',
    comment: 'Identifier for the record',
  },
  {
    name: 'name',
    type: 'string',
    comment: 'Name for the record',
  },
  {
    name: 'address',
    type: 'string',
    comment: 'Address for the record',
  },
  {
    name: 'value',
    type: 'int',
    comment: 'Value for the record',
  },
];

const customEtlJob = new KinesisstreamsToGluejob(this, 'CustomETL', {
  glueJobProps: {
    command: {
      name: 'gluestreaming',
      pythonVersion: '3',
      scriptLocation: new s3assets.Asset(this, 'ScriptLocation', {
        path: `${__dirname}/../etl/transform.py`,
      }).s3objectUrl,
    },
  },
  fieldSchema: fieldSchema,
});
```

## Initializer

```
new KinesisstreamsToGluejob(scope: Construct, id: string, props:
  KinesisstreamsToGluejobProps);
```

## パラメータ

- scope [Construct](#)
- id `string`
- props [KinesisstreamsToGluejobProps](#)

## パターン構成プロパティ

名前	タイプ	説明
KinesisStreamProps ?	<a href="#">kinesis.StreamProps</a>	Amazon Kinesis データストリームのデフォルトの小道具を上書きする、オプションのユーザー提供の小道具です。
ExistingStreamObj ?	<a href="#">kinesis.Stream</a>	Kinesis ストリームの既存のインスタンスで、これと <code>kinesisStreamProps</code> エラーを発生させます。
GlueJobProps ?	<a href="#">cfnJob.CfnJobProps</a>	AWS Glue ジョブのデフォルトの小道具を上書きするユーザー提供の小道具です。
ExistingGlueJob ?	<a href="#">cfnJob.CfnJob</a>	AWS Glue Job 既存のインスタンス。これと <code>glueJobProps</code> エラーを発生させます。
既存のデータベース?	<a href="#">CfnDatabase</a>	この構造体で使用する既存の AWS Glue データベース。これが設定されている場合、 <code>databaseProps</code> は無視されます。

名前	タイプ	説明
databaseProps?	<a href="#">CfnDatabaseProps</a>	AWS Glue データベースの作成に使用されるデフォルトの小道具を上書きするユーザー提供の小道具です。
ExistingTable?	<a href="#">CfnTable</a>	AWS Glue テーブルの既存のインスタンス。これが設定されている場合、tableProps および fieldSchema は無視されます。
TableProps?	<a href="#">CfnTableProps</a>	AWS Glue テーブルの作成に使用されるデフォルトの小道具を上書きするユーザー提供の小道具です。
フィールドスキーマ?	<a href="#">CfnTable.ColumnProperty[]</a>	AWS Glue テーブルを作成するためのユーザー指定のスキーマ構造。
出力データストア?	<a href="#">SinkDataStoreProps</a>	AWS Glue ジョブからの出力を格納する Amazon S3 バケットのユーザー提供の小道具。現時点では、出力データストアタイプとして Amazon S3 のみをサポートしています。

## SinkDataStoreProps

名前	タイプ	説明
existings3outputBucket?	<a href="#">Bucket</a>	データを書き込む必要のある S3 バケットの既存のインスタンス。これと outputBuc

名前	タイプ	説明
		ketProps エラーを発生させます。
出力バケットプロップ	<a href="#">BucketProps</a>	AWS Glue ジョブからの出力を保存するために使用する Amazon S3 バケットを作成するためのユーザー指定のバケットプロパティ。
データストアタイプ	<a href="#">SinkStoreType</a>	シンクデータストアタイプ。

## SinkStoreType

S3、DynamoDB、DocumentDB、RDS、または Redshift を含むデータストア型の列挙。現在の構造体実装は S3 のみをサポートしていますが、将来他の出力タイプを追加する可能性があります。

名前	タイプ	説明
S3	string	S3 ストレージタイプ

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon Kinesis Stream

- Amazon Kinesis データストリームの最小権限アクセス IAM ロールを設定します。
- AWS マネージド KMS キーを使用して、Amazon Kinesis Stream のサーバー側の暗号化を有効にします。
- Amazon Kinesis ストリームにベストプラクティスの Amazon CloudWatch アラームをデプロイします。

## Glue Job

- CloudWatch、Job ブックマーク、S3 の暗号化を設定する AWS Glue セキュリティ設定を作成します。CloudWatch と Job ブックマークは、AWS AWS Glue サービス用に作成された AWS マネージド KMS キーを使用して暗号化されます。S3 バケットは SSE-S3 暗号化モードで設定されます。
- AWS Glue が Amazon Kinesis Data Streams の読み取りを許可するサービスロールポリシーを設定します。

## Glue データベース

- AWS Glue データベースを作成します。AWS Glue テーブルがデータベースに追加されます。このテーブルは、Amazon Kinesis データストリームでバッファされるレコードのスキーマを定義します。

## Glue テーブル

- AWS Glue テーブルを作成します。テーブルスキーマ定義は、Amazon Kinesis データストリームにバッファされるレコードの JSON 構造に基づいています。

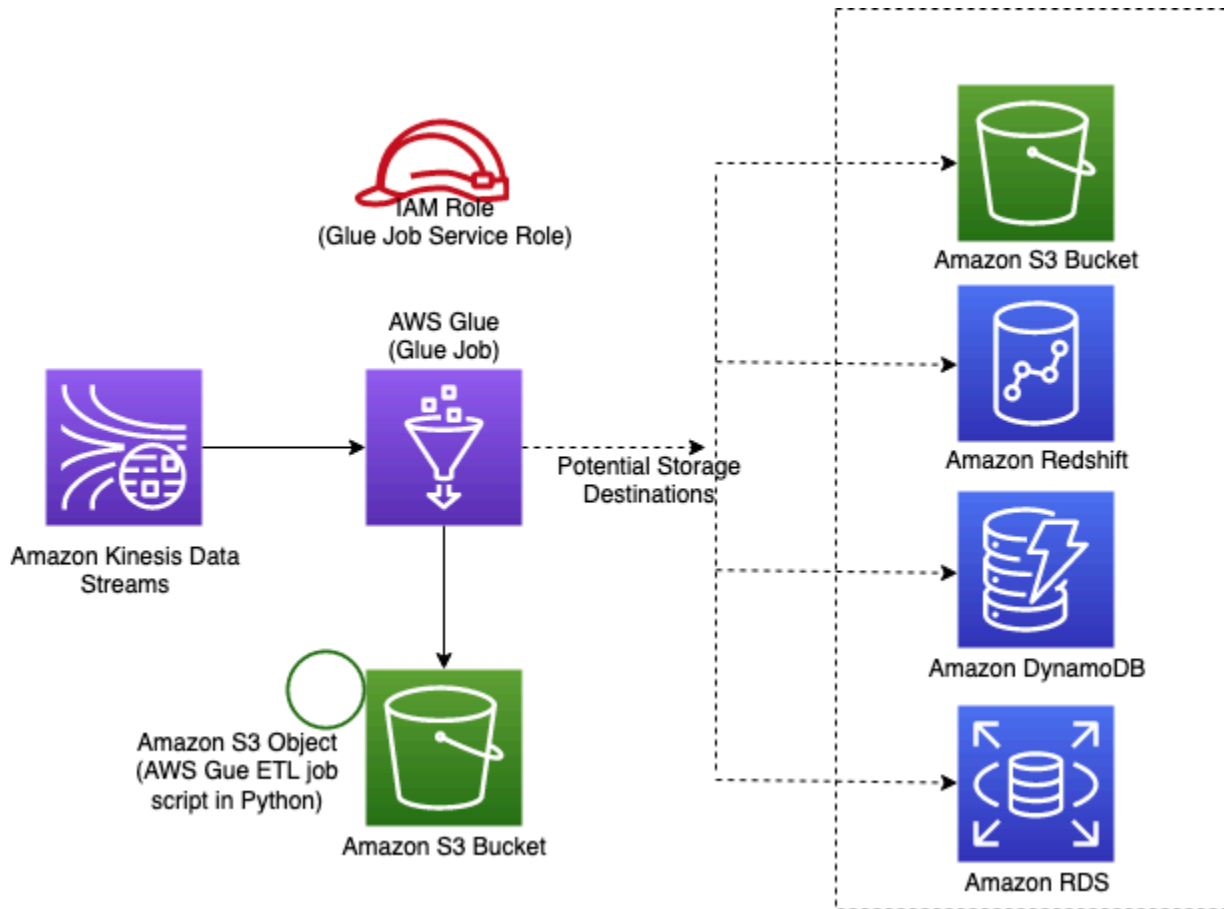
## IAM ロール

- 1) Amazon S3 バケットの場所から ETL スクリプトを読み込む、2) Amazon Kinesis データストリームからレコードを読み込む、3) Amazon Glue ジョブを実行する権限を持つジョブ実行ロール。

## 出力 S3 バケット

- ETL 変換のアウトプットを保存する Amazon S3 バケット。このバケットは、作成された AWS Glue ジョブに引数として渡され、ETL スクリプトでデータを書き込むために使用できます。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-構築/aws-kinesistreams-gluejob](#)




## aws-キネシスストリーム-キネシスファイアホース-3

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。



注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_kinesisstreams_kinesisfirehose_s3</code>
 活字体	<code>@aws-solutions-constructs/aws-kinesis-streams-kinesis-firehose-s3</code>
 Java	<code>software.amazon.awsconstructs.services.kinesisstreams_kinesisfirehoses3</code>

## Overview

この AWS ソリューション構成は、Amazon S3 バケットに接続された Amazon Kinesis データ Firehose ( KDF ) 配信ストリームに接続された Amazon Kinesis データストリーム ( KDS ) を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { KinesisStreamsToKinesisFirehoseToS3 } from '@aws-solutions-constructs/aws-kinesisstreams-kinesisfirehose-s3';

new KinesisStreamsToKinesisFirehoseToS3(this, 'test-stream-firehose-s3', {});
```

## Initializer

```
new KinesisStreamsToKinesisFirehoseToS3(scope: Construct, id: string, props: KinesisStreams...ToS3Props);
```

## パラメータ

- scope [Construct](#)
- idstring
- props [KinesisStreams...ToS3Props](#)

## パターン構成プロパティ

名前	タイプ	説明
BucketProps ?	<a href="#">s3.BucketProps</a>	S3 バケットのデフォルトの小道具を上書きするオプションのユーザーが提供した小道具です。
CreateCloudWatchalarms ?	boolean	推奨される CloudWatch アラームを作成するかどうか (オプション)。
ExistingBucketObj ?	<a href="#">s3.IBucket</a>	オプションの S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。
ExistingLoggingBucketObj ?	<a href="#">s3.IBucket</a>	パターンによって作成された S3 バケットのログ S3 Bucket オブジェクトのオプションの既存のインスタンス。
ExistingStreamObj ?	<a href="#">kinesis.Stream</a>	Kinesis ストリームの既存のインスタンスで、これとkinesisStreamProps はエラーを発生させます。
KineSisFireHoseprops ?	<a href="#">aws-kinesisfirehose.CfnDeliveryStreamProps</a>   any	Kinesis Firehose 配信ストリームのデフォルトの小道具を

名前	タイプ	説明
		上書きするオプションのユーザーが提供した小道具です。
KinesisStreamProps ?	<a href="#"><u>kinesis.StreamProps</u></a>	Kinesis ストリームのデフォルトのプロップを上書きするオプションのユーザーが提供したプロップ。
LogGroupPropsかな ?	<a href="#"><u>logs.LogGroupProps</u></a>	CloudWatchLogs ロググループのデフォルトの小道具を上書きするオプションのユーザー提供の小道具です。

## パターンプロパティ

名前	タイプ	説明
CloudWatchAlarms	<a href="#"><u>cloudwatch.Alarm[]</u></a>	コンストラクトによって作成された CloudWatch.alarm インスタンスのリストを返します。
キネシファイアホース	<a href="#"><u>kinesisfirehose.CfnDeliveryStream</u></a>	コンストラクトによって作成された KinesisFirehose.cfnDeliveryStream のインスタンスを返します。
キネシファイアホースロググループ	<a href="#"><u>logs.LogGroup</u></a>	Kinesis Data Firehose hose 配信ストリームのコンストラクトによって作成された Logs.logGroup のインスタンスを返します。
キネシファイアホースロール	<a href="#"><u>iam.Role</u></a>	Kinesis Data Firehose hose 配信ストリームのコンスト

名前	タイプ	説明
		ラクトによって作成された IAM.role のインスタンスを返します。
キネシスストリームロール	<a href="#">iam.Role</a>	Kinesis ストリームのコンストラクトによって作成された IAM.Role のインスタンスを返します。
s3bucket?	<a href="#">s3.Bucket</a>	コンストラクトによって作成された S3.bucket のインスタンスを返します。
s3loggingBucket ?	<a href="#">s3.Bucket</a>	コンストラクトによって作成された S3.bucket のインスタンスを、プライマリバケットのロギングバケットとして返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon Kinesis ストリーム

- Kinesis ストリームの最小権限アクセス IAM ロールを設定する
- AWS マネージド KMS キーを使用した Kinesis ストリームのサーバー側の暗号化の有効化
- Kinesis ストリームにベストプラクティスの CloudWatch アラームをデプロイする

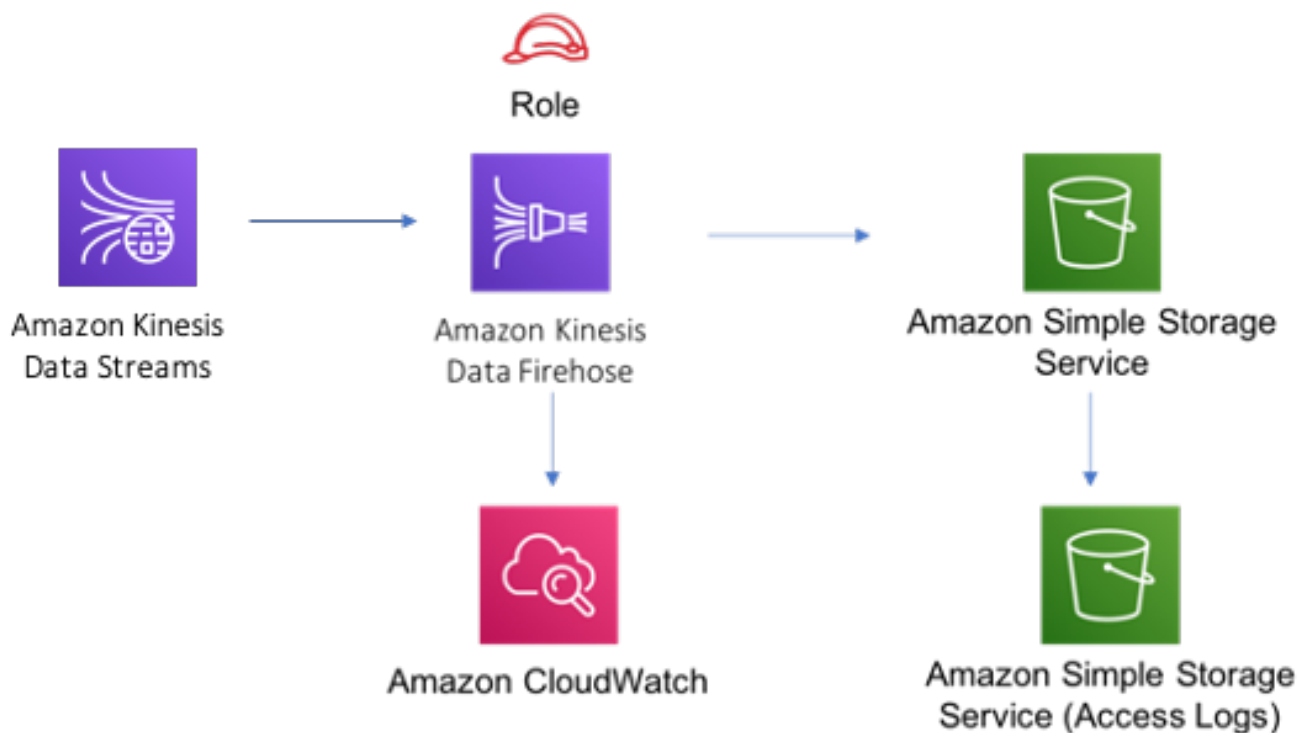
### Amazon Kinesis Firehose

- Kinesis Firehose の CloudWatch ログ記録を有効にする
- Amazon Kinesis Firehose の最小権限アクセス IAM ロールを設定する

## Amazon S3 バケットの数

- S3 バケットのアクセスログ記録の設定
- AWS マネージド KMS キーを使用した S3 バケットのサーバー側の暗号化の有効化
- 転送時のデータの暗号化を強制する
- バケットのバージョニングを有効にする
- S3 バケットのパブリックアクセスを許可しない
- CloudFormation スタックを削除するときに S3 バケットを保持する
- 90 日後にライフサイクルルールを適用して、最新でないオブジェクトバージョンを Glacier ストレージに移動する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-solutions-constructs/aws-キネシスストリーム-キネシスファイアホース-3](https://github.com/aws-solutions-constructs/aws-キネシスストリーム-キネシスファイアホース-3)

# aws-キネシスストリーム-ラムダ

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョン管理](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws-kinesis-streams-lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-kinesisstreams-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.kinesisstreamslambda</code>

## Overview

この AWS ソリューション構築では、インタラクションとセキュリティのための適切なリソース/プロパティを持つ Kinesis ストリームと Lambda 関数がデプロイされます。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { KinesisStreamsToLambda } from '@aws-solutions-constructs/aws-kinesisstreams-lambda';

new KinesisStreamsToLambda(this, 'KinesisToLambdaPattern', {
  kinesisEventSourceProps: {
    startingPosition: lambda.StartingPosition.TRIM_HORIZON,
```

```

        batchSize: 1
    },
    lambdaFunctionProps: {
        runtime: lambda.Runtime.NODEJS_14_X,
        // This assumes a handler function in lib/lambda/index.js
        code: lambda.Code.fromAsset(`${__dirname}/lambda`),
        handler: 'index.handler'
    }
});

```

## Initializer

```

new KinesisStreamsToLambda(scope: Construct, id: string, props:
    KinesisStreamsToLambdaProps);

```

### パラメータ

- scope [Construct](#)
- id string
- props [KinesisStreamsToLambdaProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されま

名前	タイプ	説明
		す existingLambdaObj が提供される。
KinesisStreamProps ?	<a href="#">kinesis.StreamProps</a>	Kinesis ストリームのデフォルトのプロップを上書きするオプションのユーザー指定のプロップ。
ExistingStreamObj ?	<a href="#">kinesis.Stream</a>	Kinesis ストリームの既存のインスタンスで、これと kinesisStreamProps はエラーを発生させます。
KinesisEventSourceProps ?	<a href="#">aws-lambda-event-sources.KinesisEventSourceProps</a>	オプションのユーザー提供の小道具で、Lambda イベントソースマッピングのデフォルトの小道具を上書きします。
クラウド・ワット・チャラームズ	boolean	推奨される CloudWatch アラームを作成するかどうか。

## パターンプロパティ

名前	タイプ	説明
KinesisStream	<a href="#">kinesis.Stream</a>	パターンによって作成された Kinesis ストリームのインスタンスを返します。
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
キネシス・ストリーム・ロール	<a href="#">iam.Role</a>	Kinesis ストリームのパターンによって作成された IAM



名前	タイプ	説明
		ロールのインスタンスを返します。
CloudWatchAlarms ?	<a href="#">cloudwatch.Alarm[]</a>	パターンによって作成される 1 つ以上の CloudWatch アラームのリストを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

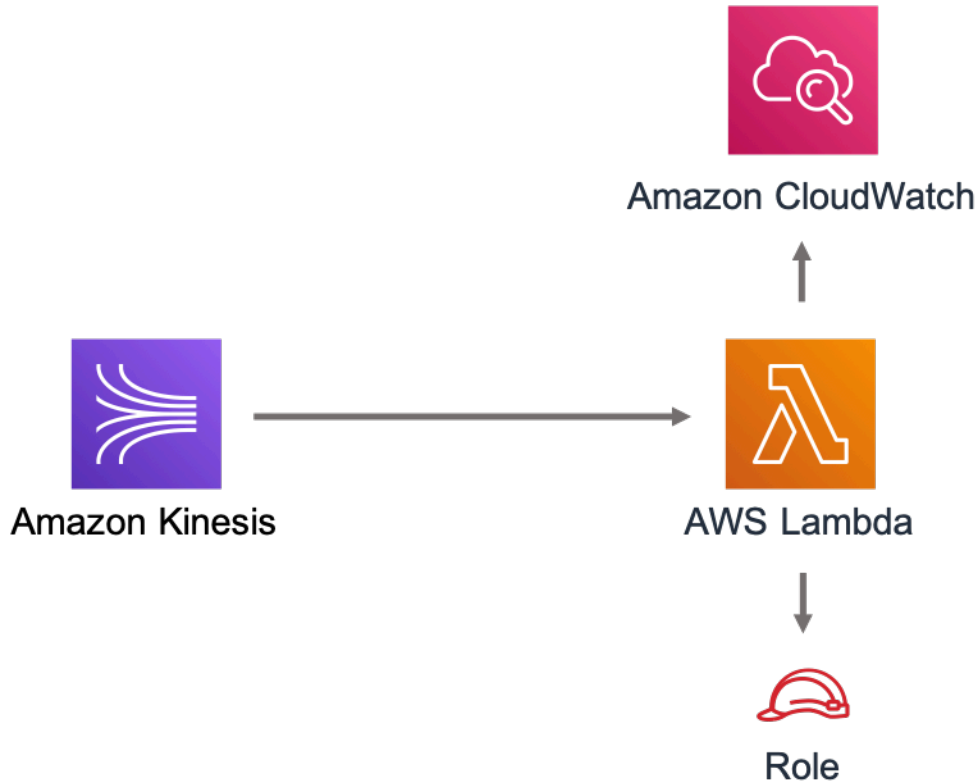
### Amazon Kinesis Stream

- Kinesis Stream の最小権限アクセス IAM ロールを設定します。
- AWS マネージド KMS キーを使用して、Kinesis Stream のサーバー側の暗号化を有効にします。
- Kinesis ストリームにベストプラクティスの CloudWatch アラームをデプロイします。

### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にします。
- 障害処理機能の有効化:関数エラーの bisect の有効化、デフォルトの最大レコード有効期間 (24 時間) の設定、デフォルトの最大再試行回数 (500) の設定、障害発生時の宛先として SQS デッドレターキューをデプロイします。
- 環境変数の設定:
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-構築/aws-kinesistreams-lambda](#)

## aws-lambda-dynamodb

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_dynamodb</code>
 TypeScript	<code>@aws-solutions-constructs/aws-lambda-dynamodb</code>
 Java	<code>software.amazon.awsconstructs.services.lambda_dynamodb</code>

## Overview

この AWS ソリューション構成では、AWS Lambda 関数と Amazon DynamoDB テーブルが最小限の権限で実装されています。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { LambdaToDynamoDBProps, LambdaToDynamoDB } from '@aws-solutions-constructs/aws-lambda-dynamodb';

const props: LambdaToDynamoDBProps = {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
};

new LambdaToDynamoDB(this, 'test-lambda-dynamodb-stack', props);
```

## Initializer

```
new LambdaToDynamoDB(scope: Construct, id: string, props: LambdaToDynamoDBProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [LambdaToDynamoDBProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj が提供される。
DynamoTableProps ?	<a href="#">dynamodb.TableProps</a>	DynamoDB テーブルのデフォルトのプロップを上書きするオプションのユーザー提供の小道具
ExistingTableObj ?	<a href="#">dynamodb.Table</a>	DynamoDB テーブルオブジェクトの既存のインスタンス。これとdynamoTab

名前	タイプ	説明
		leProps はエラーを発生させます。
テーブルパーミッション?	<a href="#">string</a>	Lambda 関数に付与されるオプションのテーブルパーミッション。以下のいずれかのオプションを指定できます。All,Read,ReadWrite , またはWrite。
表環境変数名ですか ?	string	Lambda 関数に設定された DynamoDB テーブル環境変数のオプション名。
既存のVPCかな ?	<a href="#">ec2.IVpc</a>	このパターンをデプロイするオプションの既存の VPC。VPC にデプロイされると、Lambda 関数は VPC 内の ENI を使用してネットワークリソースにアクセスし、ゲートウェイエンドポイントは Amazon DynamoDB 用の VPC 内に作成されます。既存の VPC が提供されている場合、deployVpc プロパティを true。これは <a href="#">ec2.IVpc</a> を使用して、クライアントがスタックの外部に存在する VPC を提供できるようにします。 <a href="#">ec2.Vpc.fromLookup()</a> 方法。

名前	タイプ	説明
vPCProps?	<a href="#">ec2.VpcProps</a>	新しい VPC のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。enableDns Hostnames ,enableDns Support ,natGateways ,およびsubnetConfiguration はパターンによって設定されるため、ここで指定されたプロパティの値はすべて上書きされます。もしdeployVpc ではありませんtrueの場合、このプロパティは無視されます。

名前	タイプ	説明
DeployVPC ?	boolean	<p>に基づいて新しい VPC を作成するかどうか <code>vpcProps</code> のこのパターンを展開します。これを <code>true</code> に設定すると、パターンを実行するために最小限のほとんどのプライベート VPC がデプロイされます。</p> <ul style="list-style-type: none"> <li>CDK プログラムで使用される各アベイラビリティゾーン内の 1 つに分離されたサブネット</li> <li><code>enableDnsHostnames</code> および <code>enableDnsSupport</code> はどちらも <code>true</code></li> </ul> <p>このプロパティが <code>true</code>、次に <code>existingVpc</code> は指定できません。デフォルトは <code>false</code> です。</p>

## パターンプロパティ

名前	タイプ	説明
ダイナモテーブル	<a href="#"><code>dynamodb.Table</code></a>	パターンによって作成された DynamoDB テーブルのインスタンスを返します。
LambdaFunction	<a href="#"><code>lambda.Function</code></a>	パターンによって作成された Lambda 関数のインスタンスを返します。

名前	タイプ	説明
vpcay	<a href="#">ec2.IVpc</a>	パターンによって使用される VPC 上のインターフェイスを返します ( 存在する場合 )。これは、パターンによって作成された VPC、またはパターンコンストラクタに提供された VPC です。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### AWS Lambda 関数

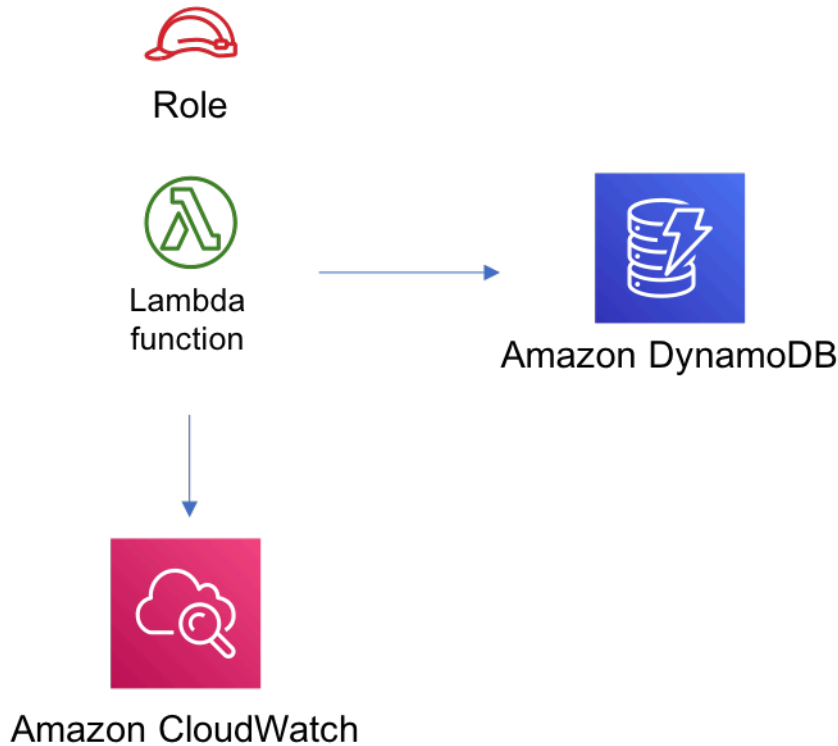
- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray によるトレースを有効にします。
- 環境変数の設定:
  - DDB\_TABLE\_NAME (デフォルト)
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

### Amazon DynamoDB テーブル

- DynamoDB テーブルの請求モードをオンデマンドに設定します ( リクエストごとの支払い )。
- AWS マネージド KMS キーを使用して DynamoDB テーブルのサーバー側の暗号化を有効にします。
- DynamoDB テーブルの 'id' という名前のパーティションキーを作成します。
- CloudFormation スタックを削除するときに、テーブルを保持します。
- 継続的なバックアップとポイントインタイムリカバリを可能にします。



## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-ソリューション-構築/aws-lambda-dynamodb](#)

## aws-lambda-Elasticsearch-

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョニング](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_elasticsearch_kibana</code>
 TypeScript	<code>@aws-solutions-constructs/aws-lambda-elasticsearch-kibana</code>
 Java	<code>software.amazon.awsconstructs.services.lambdaelasticsearchkibana</code>

## Overview

この AWS ソリューション構築は、AWS Lambda 関数と Amazon Elasticsearch Service ドメインを実装し、権限が最も低い権限を持ちます。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { LambdaToElasticSearchAndKibana } from '@aws-solutions-constructs/aws-lambda-elasticsearch-kibana';
import { Aws } from "@aws-cdk/core";

const lambdaProps: lambda.FunctionProps = {
  runtime: lambda.Runtime.NODEJS_14_X,
  // This assumes a handler function in lib/lambda/index.js
  code: lambda.Code.fromAsset(`${__dirname}/lambda`),
  handler: 'index.handler'
};

new LambdaToElasticSearchAndKibana(this, 'test-lambda-elasticsearch-kibana', {
  lambdaFunctionProps: lambdaProps,
  domainName: 'test-domain',
  // TODO: Ensure the Cognito domain name is globally unique
  cognitoDomainName: 'globallyuniquedomain' + Aws.ACCOUNT_ID;
});
```

## Initializer

```
new LambdaToElasticSearchAndKibana(scope: Construct, id: string, props:
  LambdaToElasticSearchAndKibanaProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [LambdaToElasticSearchAndKibanaProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合、無視されずexistingLambdaObj が提供される。
ESDomainProps ?	<a href="#">elasticsearch.CfnDomainProps</a>	Amazon Elasticsearch Service デフォルトの小道具を上書きするためのオプションのユーザー提供の小道具です
domainName	string	Cognito および Amazon Elasticsearch Service ドメイン名

名前	タイプ	説明
CognitoDomainName?	string	オプションの Cognito ドメインの名前。指定されている場合は、Cognito ドメインで使用され、domainName が Elasticsearch ドメインに使用されます。
クラウド・ワット・チャラームズ	boolean	推奨される CloudWatch アラームを作成するかどうか。
DomainEndPointEnvironmentVariableName?	string	Lambda 関数に設定された ElasticSearch ドメインエンドポイント環境変数のオプション名。

## パターンプロパティ

名前	タイプ	説明
CloudWatchAlarms?	<a href="#"><u>cloudwatch.Alarm[]</u></a>	パターンによって作成された 1 つ以上の CloudWatch アラームのリストを返します。
弾性検索ドメイン	<a href="#"><u>elasticsearch.CfnDomain</u></a>	パターンによって作成された Elasticsearch ドメインのインスタンスを返します。
ElasticSearchドメインロール	<a href="#"><u>iam.Role</u></a>	Elasticsearch ドメインのパターンによって作成された IAM ロールのインスタンスを返します。
IdentityPool	<a href="#"><u>cognito.CfnIdentityPool</u></a>	パターンによって作成された Cognito ID プールのインスタンスを返します。

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
UserPool	<a href="#">cognito.UserPool</a>	パターンによって作成された Cognito ユーザープールのインスタンスを返します。
UserLClient	<a href="#">cognito.UserPoolClient</a>	パターンによって作成された Cognito ユーザープールクライアントのインスタンスを返します。

## Lambda 関数

このパターンには、DynamoDB ストリームから Elasticsearch サービスにデータを投稿できる Lambda 関数が必要です。サンプル関数が提供される [ここ](#)。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray によるトレースを有効にします。
- 環境変数の設定:
  - DOMAIN\_ENDPOINT (デフォルト)
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

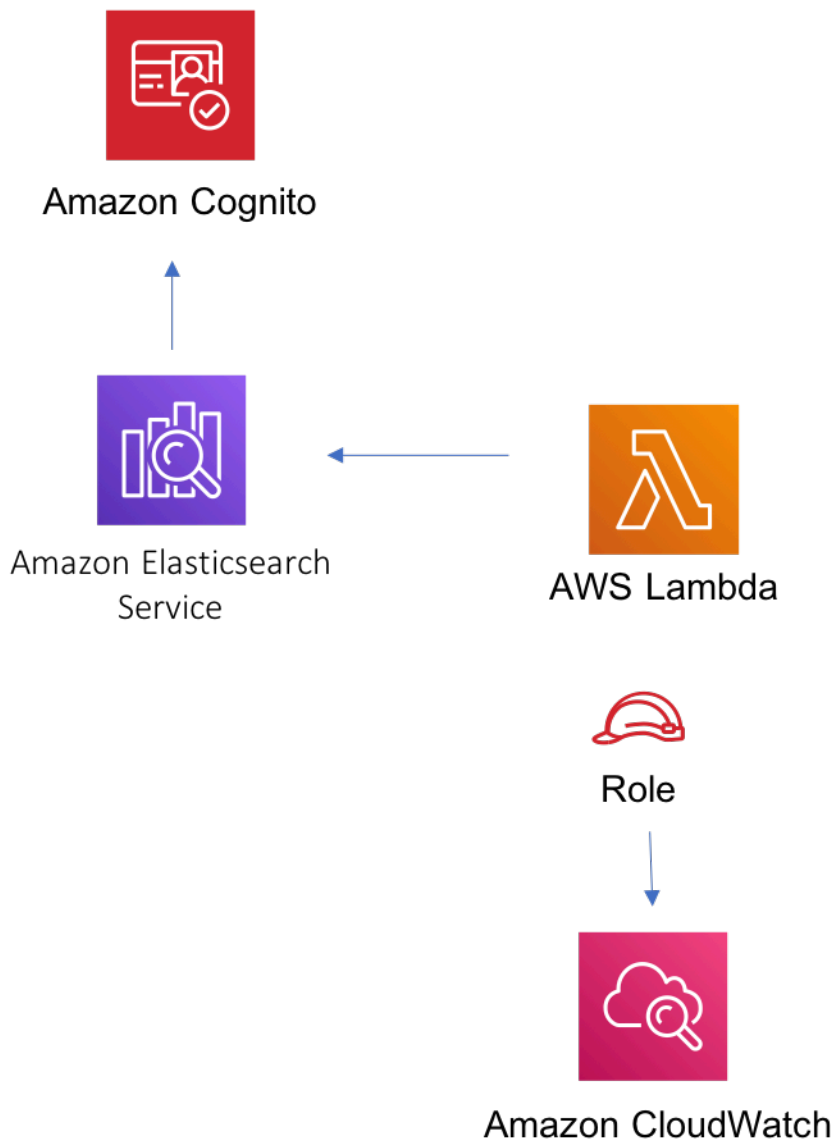
### Amazon Cognito

- ユーザープールのパスワードポリシーを設定します。
- ユーザープールの高度なセキュリティモードを適用します。

## Amazon Elasticsearch Service

- Elasticsearch ドメイン用のベストプラクティスの CloudWatch アラームをデプロイします。
- Cognito ユーザープールを使用して、Kibana ダッシュボードのアクセスを保護します。
- AWS マネージド KMS キーを使用して、Elasticsearch ドメインのサーバー側の暗号化を有効にします。
- Elasticsearch ドメインのノード間の暗号化を有効にします。
- Amazon ES ドメインのクラスターを設定します。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-ソリューション-構築/aws-lambda-elasticsearch-kibana](#)

## aws-ラムダ-s3

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_s3</code>
 TypeScript	<code>@aws-solutions-constructs/aws-lambda-s3</code>
 Java	<code>software.amazon.awsconstructs.services.lambdas3</code>

## Overview

この AWS ソリューション構築では、Amazon S3 バケットに接続された AWS Lambda 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { LambdaToS3 } from '@aws-solutions-constructs/aws-lambda-s3';

new LambdaToS3(this, 'LambdaToS3Pattern', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

## Initializer

```
new LambdaToS3(scope: Construct, id: string, props: LambdaToS3Props);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [LambdaToS3Props](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionPropsはエラーを発生させます。
LambdaFunctionProps?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定



名前	タイプ	説明
ExistingBucketObj?	<a href="#">s3.IBucket</a>	のプロパティ。の場合は無視されます。existingLambdaObj はにあります。 S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。
BucketProps?	<a href="#">s3.BucketProps</a>	オプションのユーザー提供のプロパティ。バケットのデフォルトプロパティを上書きします。の場合は無視されます。existingBucketObj はにあります。
BucketPermissions?	string[]	Lambda 関数に付与するオプションのバケットアクセス許可。次の 1 つ以上。Delete,Put,Read,ReadWrite,Write。

名前	タイプ	説明
既存のVPCかな？	<a href="#">ec2.IVpc</a>	<p>このパターンをデプロイするオプションの既存のVPC。VPC にデプロイされると、Lambda 関数は VPC 内の ENI を使用してネットワークリソースにアクセスし、インターフェイスエンドポイントは Amazon SQS 用の VPC 内に作成されます。既存の VPC が提供されている場合、<code>deployVpc</code> プロパティは、<code>true</code>。これは <code>ec2.IVpc</code> を使用して、クライアントがスタックの外部に存在する VPC を提供できるようにします。<a href="#">ec2.Vpc.fromLookup()</a> メソッド。</p>

名前	タイプ	説明
DeployVPC ?	boolean	<p>に基づいて新しい VPC を作成するかどうか <code>vpcProps</code> のこのパターンを展開します。これをに設定します。 <code>true</code> は、パターンを実行するために、最小限のほとんどのプライベート VPC をデプロイします。</p> <ul style="list-style-type: none"><li>• CDK プログラムによって使用されるアベイラビリティゾーンごとに 1 つずつ分離されたサブネット。</li><li>• <code>enableDnsHostnames</code> および <code>enableDnsSupport</code> はどちらも <code>true</code>。</li></ul> <p>このプロパティが <code>true</code>、次に <code>existingVpc</code> は指定できません。デフォルトは <code>false</code> です。</p>

名前	タイプ	説明
vPCProps?	<a href="#">ec2.VpcProps</a>	新しい VPC のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。enableDns Hostnames ,enableDns Support ,natGateways およびsubnetConfiguration はパターンによって設定されるため、ここで指定されたプロパティの値はすべて上書きされます。もしdeployVpc ではありませんtrueの場合、このプロパティは無視されます。
BucketEnvironmentVariableName?	string	Lambda 関数に設定された S3 バケット環境変数のオプション名。

## パターンプロパティ

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
S3bucket?	<a href="#">s3.Bucket</a>	パターンによって作成された S3 バケットのインスタンスを返します。
s3loggingBucket?	<a href="#">s3.Bucket</a>	S3 バケットのパターンによって作成されたロギングバケッ

名前	タイプ	説明
		トのインスタンスを返します。
VPC ?	<a href="#">ec2.IVpc</a>	パターンによって使用される VPC のインスタンスを返します (存在する場合)。これは、パターンによって作成された VPC、またはパターンコンストラクタに提供された VPC です。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にする
- 環境変数の設定:
  - S3\_BUCKET\_NAME (デフォルト)
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

### Amazon S3 バケット

- S3 バケットのアクセスログを設定します。
- AWS マネージド KMS キーを使用して S3 バケットのサーバー側の暗号化を有効にします。
- S3 バケットのバージョニングを有効にします。
- S3 バケットのパブリックアクセスを許可しません。
- CloudFormation スタックを削除するときは、S3 バケットを保持します。
- 転送時のデータの暗号化を強制する。
- 90 日後に Glacier ストレージに最新でないオブジェクトバージョンを移動するライフサイクルルールを適用します。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。




[@aws-ソリューション-構築/aws-lambda-S3](#)



## aws-ラムダ-ssmstringパラメータ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_ssm_string_parameter</code>

言語	パッケージ
 TypeScript	@aws-solutions-constructs/aws-lambda-ssmstringparameter
 Java	software.amazon.awsconstructs.services.lambdastringparameter

## Overview

この AWS ソリューション構成では、AWS Lambda 関数と AWS Systems Manager Parameter Store 文字列パラメーターが最小権限で実装されます。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
const { LambdaToSsmstringparameterProps, LambdaToSsmstringparameter } from '@aws-solutions-constructs/aws-lambda-ssmstringparameter';

const props: LambdaToSsmstringparameterProps = {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  },
  stringParameterProps: { stringValue: "test-string-value" }
};

new LambdaToSsmstringparameter(this, 'test-lambda-ssmstringparameter-stack', props);
```

## Initializer

```
new LambdaToSsmstringparameter(scope: Construct, id: string, props:
  LambdaToSsmstringparameterProps);
```

## パラメータ

- scope [Construct](#)
- idstring
- props [LambdaToSsmstringparameterProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj が提供される。
ExistingStringParameterObj ?	<a href="#">ssm.StringParameter</a>	SSM String パラメータオブジェクトの既存のインスタンス。これとstringParameterProps はエラーを発生させます。
StringParameterProps ?	<a href="#">ssm.StringParameterProps</a>	SSM String パラメータのデフォルトプロップを上書きするオプションのユーザー提供の小道具です。もしexistingStringParameterObj が設定されていない場合、stringParameterProps は必須です。はサポートされる唯一



名前	タイプ	説明
		の方法です。 <a href="#">ssm.StringParameterProps.type</a> 、 <a href="#">STRING</a> 別の値が指定されている場合は、オーバーライドされます。
StringParameterEnvironmentVariableName?	string	Lambda 関数に設定された SSM String パラメーター環境変数のオプション名。
既存のVPCかな？	<a href="#">ec2.IVpc</a>	このパターンをデプロイするオプションの既存の VPC。VPC にデプロイされると、Lambda 関数は VPC 内の ENI を使用してネットワークリソースにアクセスし、インターフェイスエンドポイントが VPC for AWS Systems Manager パラメータに作成されます。既存の VPC が提供されている場合、 <code>deployVpc</code> プロパティが、 <code>true</code> 。これは、 <code>ec2.IVpc</code> を使用して、クライアントがスタックの外部に存在する VPC を提供できるようにします。 <a href="#">ec2.Vpc.fromLookup()</a> 方法。

名前	タイプ	説明
vPCProps?	<a href="#">ec2.VpcProps</a>	新しい VPC のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。enableDns Hostnames ,enableDns Support ,natGateways およびsubnetConfiguration はパターンによって設定されるため、ここで指定されたプロパティの値はすべて上書きされます。もしdeployVpc ではありませんtrueの場合、このプロパティは無視されます。

名前	タイプ	説明
DeployVPC ?	boolean	<p>に基づいて新しい VPC を作成するかどうか <code>vpcProps</code> のパターンを展開します。これをに設定する <code>true</code> は、パターンを実行するために、最小限のほとんどのプライベート VPC をデプロイします。</p> <ul style="list-style-type: none"><li>• CDK プログラムで使用される各アベイラビリティゾーンに 1 つずつ分離されたサブネット。</li><li>• <code>enableDnsHostnames</code> および <code>enableDnsSupport</code> はどちらも <code>true</code>。</li></ul> <p>このプロパティがに設定されている場合 <code>true</code>、次に <code>existingVpc</code> は指定できません。デフォルトは <code>false</code> です。</p>
StringParameterPermissions	string	<p>Lambda 関数に付与するオプションの SSM String パラメーターのアクセス許可。次のいずれかを指定できます。<code>Read,ReadWrite</code>。</p>

## パターンプロパティ

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	のインスタンスを返します。lambda.Function 構成によって作成されます。
文字列Parameter	<a href="#">ssm.StringParameter</a>	のインスタンスを返します。ssm.StringParameter 構成によって作成されます。
vpc?	<a href="#">ec2.IVpc</a>	パターンによって使用される VPC 上のインターフェイスを返します (存在する場合)。これは、パターンによって作成された VPC、またはパターンコンストラクタに提供された VPC です。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

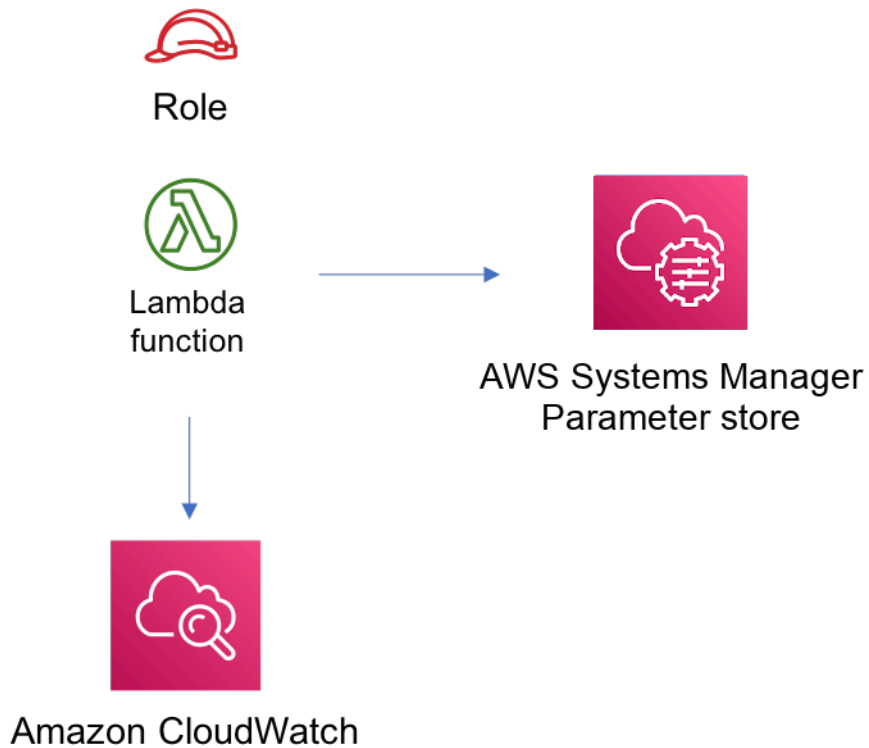
### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にします。
- 環境変数の設定:
  - SSM\_STRING\_PARAMETER\_NAME (デフォルト)
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED (ノード10.x以上の機能の場合)

## Amazon AWS Systems Manager パラメータストア文字列

- 関連付けられた AWS Lambda 関数の読み取り専用アクセスを有効にします。
- 指定された値を使用して新しい SSM String パラメータを作成します。
- CloudFormation スタックを削除するときは、SSM スtring パラメータを保持します。

### Architecture



### GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-ソリューション-構築/aws-lambda-ssmstringパラメータ](#)

## aws-ラムダ-サゲマケレンドポイント

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_sagemakerendpoint</code>
 TypeScript	<code>@aws-solutions-constructs/aws-lambda-sagemakerendpoint</code>
 Java	<code>software.amazon.awsconstructs.services.lambda.sagemakerendpoint</code>

## Overview

この AWS ソリューション構築物は、Amazon Sagemaker エンドポイントに接続された AWS Lambda 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { Duration } from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';
import {
  LambdaToSagemakerEndpoint,
  LambdaToSagemakerEndpointProps,
} from '@aws-solutions-constructs/aws-lambda-sagemakerendpoint';

const constructProps: LambdaToSagemakerEndpointProps = {
  modelProps: {
    primaryContainer: {
```

```

    image: '{{AccountId}}.dkr.ecr.{{region}}.amazonaws.com/linear-learner:latest',
    modelDataUrl: 's3://{{bucket-name}}/{{prefix}}/model.tar.gz',
  },
},
lambdaFunctionProps: {
  runtime: lambda.Runtime.PYTHON_3_8,
  // This assumes a handler function in lib/lambda/index.py
  code: lambda.Code.fromAsset(`${__dirname}/lambda`),
  handler: 'index.handler',
  timeout: Duration.minutes(5),
  memorySize: 128,
},
};

new LambdaToSagemakerEndpoint(this, 'LambdaToSagemakerEndpointPattern',
  constructProps);

```

## Initializer

```

new LambdaToSagemakerEndpoint(scope: Construct, id: string, props:
  LambdaToSagemakerEndpointProps);

```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [LambdaToSagemakerEndpointProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionPropsはエラーを発生させます。

名前	タイプ	説明
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。
ExistingSagemakerEndpointObj ?	<a href="#">sagemaker.CfnEndpoint</a>	使用する、オプションの既存のSagemaker Endpoint。これとendpointProps はエラーを発生させます。
ModelProps?	<a href="#">sagemaker.CfnModelProps</a>   any	Sagemaker モデルの既定のプロパティを上書きするユーザー指定のプロパティ。少なくともmodelProps.primaryContainer はモデルを作成するために提供する必要があります。デフォルトでは、パターンは最小限の権限でロールを作成しますが、クライアントはmodelProps.executionRoleArn 。
EndPointConfigProps?	<a href="#">sagemaker.CfnEndpointConfigProps</a>	Sagemaker エンドポイント設定のデフォルトプロパティを上書きする、ユーザー指定のオプションのプロパティです。
EndPointProps?	<a href="#">sagemaker.CfnEndpointProps</a>	Sagemaker エンドポイントのデフォルトプロパティを上書きする、ユーザー指定のオプションのプロパティです。



名前	タイプ	説明
既存のVPCかな？	<a href="#">ec2.IVpc</a>	オプションの既存の VPC で、このコンストラクトをデプロイします。VPC にデプロイされると、Lambda 関数と Sagemaker エンドポイントは VPC 内の ENI を使用してネットワークリソースにアクセスします。インターフェイスエンドポイントは、Amazon Sagemaker ランタイム用の VPC と Amazon S3 VPC エンドポイントに作成されます。既存の VPC が提供されている場合、 <code>deployVpc</code> プロパティは、 <code>true</code> 。
vPCProps?	<a href="#">ec2.VpcProps</a>	新しい VPC のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。 <code>enableDns Hostnames</code> 、 <code>enableDns Support</code> 、 <code>natGateways</code> および <code>subnetConfiguration</code> はコンストラクトによって設定されるため、ここで指定されたプロパティの値はすべて上書きされます。もし <code>deployVpc</code> ではありません <code>true</code> に設定されている場合、このプロパティは無視されます。

名前	タイプ	説明
DeployVPC ?	boolean	<p>VPC に基づいて、新しい VPC を作成するかどうか <code>vpcProps</code> このパターンを展開します。これをに設定します。 <code>true</code> は、パターンを実行するために、最小限のほとんどのプライベート VPC をデプロイします。</p> <ul style="list-style-type: none"><li>• CDK プログラムによって使用されるアベイラビリティゾーンごとに 1 つずつ分離されたサブネット。</li><li>• <code>enableDnsHostnames</code> および <code>enableDnsSupport</code> はどちらも <code>true</code>。</li></ul> <p>このプロパティが <code>true</code>、次に <code>existingVpc</code> は指定できません。デフォルトは <code>false</code> です。</p>
SagemakerEnvironmentVariableName ?	string	<p>Lambda 関数に設定された SageMaker エンドポイント環境変数のオプション名。</p>

## パターンプロパティ

名前	タイプ	説明
LambdaFunction	<a href="#"><u>lambda.Function</u></a>	パターンによって作成された Lambda 関数のインスタンスを返します。
サゲマケレンドポイント	<a href="#"><u>sagemaker.CfnEndpoint</u></a>	パターンによって作成された Sagemaker エンドポイントのインスタンスを返します。
SagemakerendPointConfig ?	<a href="#"><u>sagemaker.CfnEndpointConfig</u></a>	パターンによって作成された SageMaker EndpointConfig のインスタンスを返しません。existingSagemakerEndpointObj は提供されていません。
SageMakerModelかな？	<a href="#"><u>sagemaker.CfnModel</u></a>	パターンによって作成された Sagemakerモデルのインスタンスを返します。existingSagemakerEndpointObj は提供されていません。
vpcかな	ec2.IVpc	パターンによって作成された VPC のインスタンスを返しません。deployVpc、trueの場合、またはexistingVpc は、である。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### AWS Lambda 関数

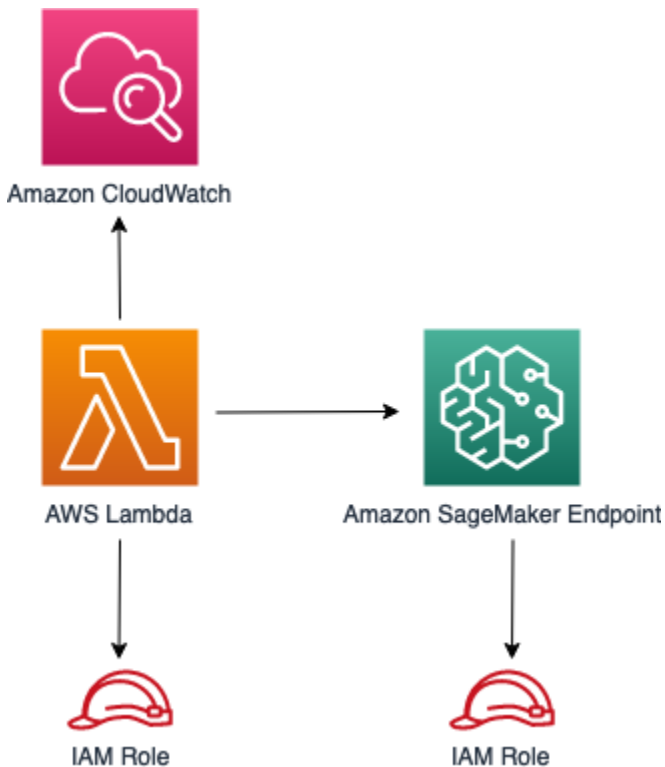
- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。

- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- 関数が推論の Sagemaker エンドポイントを呼び出すことを許可します。
- Sagemaker エンドポイントがデプロイされている VPC 内のリソースにアクセスするように関数を設定します。
- X-Ray トレースを有効にします。
- 環境変数の設定:
  - SAGEMAKER\_ENDPOINT\_NAME (デフォルト)
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Amazon SageMaker エンドポイント

- Sagemaker リソースを作成するための制限付き権限を設定します。
- Sagemaker モデル、EndPointConfig、およびエンドポイントを展開します。
- Sagemaker エンドポイントを VPC にデプロイするように設定します。
- S3 VPC エンドポイントと Sagemaker ランタイム VPC インターフェイスをデプロイします。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-constructs/aws-lambda-sagemakerendpoint](https://github.com/aws-solutions-constructs/aws-lambda-sagemakerendpoint)

## aws-ラムダ-セクレツマネージャ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンング](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	aws_solutions_constructs.aws_lambda_secretsmanager
 TypeScript	@aws-solutions-constructs/aws-lambda-secretsmanager
 Java	software.amazon.awsconstructs.services.lambda_secretsmanager

## Overview

この AWS ソリューション構築は、AWS Lambda 関数と AWS Secrets Manager のシークレットを、最も権限の低いアクセス権限で実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
const { LambdaToSecretsmanagerProps, LambdaToSecretsmanager } from '@aws-solutions-constructs/aws-lambda-secretsmanager';

const props: LambdaToSecretsmanagerProps = {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  },
};

new LambdaToSecretsmanager(this, 'test-lambda-secretsmanager-stack', props);
```

## Initializer

```
new LambdaToSecretsmanager(scope: Construct, id: string, props:
  LambdaToSecretsmanagerProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [LambdaToSecretsmanagerProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj?	<a href="#">lambda.Function</a>	Lambda Function オブジェクトの既存のインスタンス。これと <code>lambdaFunctionProps</code> はエラーを発生させます。

名前	タイプ	説明
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトの小道具をオーバーライドするためにユーザーが提供した小道具です。
SecretProps ?	<a href="#">secretsmanager.SecretProps</a>	オプションのユーザーが提供する小道具で、Secrets Manager のデフォルトの小道具を上書きします。
ExistingSecretObj ?	<a href="#">secretsmanager.Secret</a>	シークレットマネージャーのシークレットオブジェクトの既存のインスタンス、これが設定されている場合 secretProps は無視されます。
GrantWriteAccessかな	boolean	Lambda 関数のシークレットへのオプションの書き込みアクセス (デフォルトでは読み取り専用)。
SecretEnvironmentVariableName ?	string	Lambda 関数に設定された Secrets Manager のシークレット環境変数の省略可能な名前。

名前	タイプ	説明
既存のVPCかな？	<a href="#">ec2.IVpc</a>	このパターンをデプロイするオプションの既存のVPC。VPC にデプロイされると、Lambda 関数は VPC 内の ENI を使用してネットワークリソースにアクセスし、インターフェイスエンドポイントは AWS Secrets Manager 用 VPC 内に作成されます。既存の VPC が提供されている場合、 <code>deployVpc</code> プロパティは、 <code>true</code> 。これは <code>ec2.IVpc</code> を使用して、クライアントがスタックの外部に存在する VPC を提供できるようにします。 <a href="#">ec2.Vpc.fromLookup()</a> メソッド。
vPCProps?	<a href="#">ec2.VpcProps</a>	新しい VPC のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。 <code>enableDns Hostnames</code> , <code>enableDns Support</code> , <code>natGateways</code> , および <code>subnetConfiguration</code> はパターンによって設定されるため、ここで指定されたプロパティの値はすべて上書きされます。もし <code>deployVpc</code> ではありません <code>true</code> の場合、このプロパティは無視されます。



名前	タイプ	説明
DeployVPC ?	boolean	<p>に基づいて新しい VPC を作成するかどうか vpcProps このパターンを展開します。これをに設定する true は、パターンを実行するために、最小限のほとんどのプライベート VPC をデプロイします。</p> <ul style="list-style-type: none"> <li>CDK プログラムで使用されるアベイラビリティーゾーンごとに 1 つずつ独立したサブネット</li> <li>enableDnsHostnames および enableDnsSupport はどちらも true</li> </ul> <p>このプロパティが true、次に existingVpc は指定できません。デフォルトは false です。</p>

## パターンプロパティ

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	のインスタンスを返す lambda.Function 構成によって作成されます。
シークレット	<a href="#">secretsmanager.Secret</a>	のインスタンスを返す secretsmanager.Secret 構成によって作成されます。

名前	タイプ	説明
vpcかな	<a href="#">ec2.IVpc</a>	パターンによって使用される VPC 上のインターフェイスを返します ( 存在する場合 )。これは、パターンによって作成された VPC、またはパターンコンストラクタに提供された VPC です。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

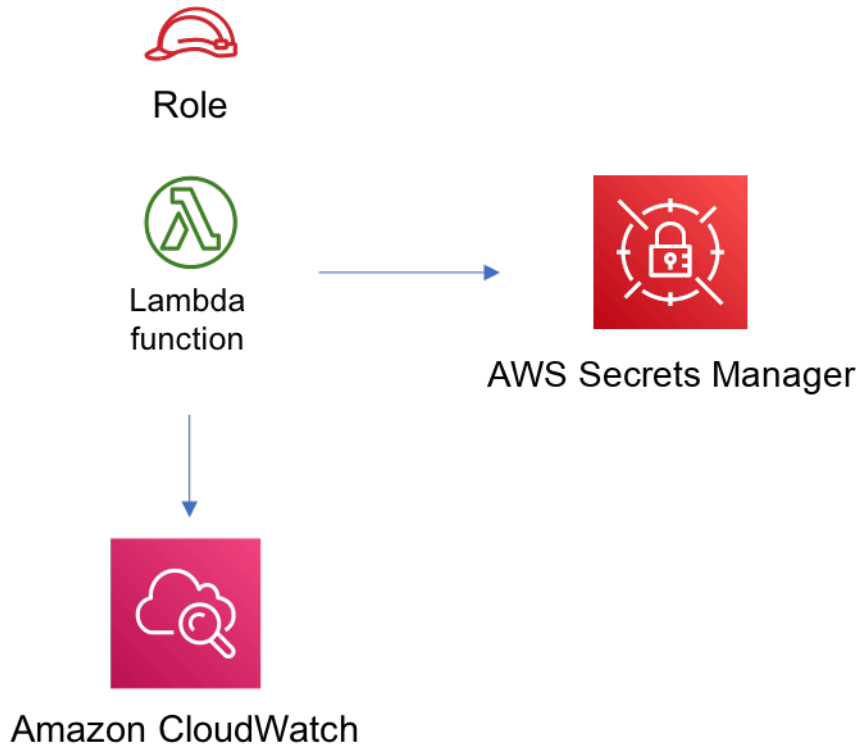
### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にします。
- 環境変数の設定:
  - (デフォルト) CDK が返すシークレットの ARN を含む SECRET\_ARN [SecretArn](#) プロパティ
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード 10.x 以上の機能の場合 )

### Amazon Secrets Manager のシークレット

- 関連付けられた AWS Lambda 関数の読み取り専用アクセスを有効にする
- アカウントとリージョンのデフォルトの KMS キーを使用してサーバー側の暗号化を有効にする
- 新しいシークレットを作成します。
  - (デフォルト) ランダムな名前
  - (デフォルト) ランダム値
- CloudFormation スタックを削除するときにシークレットを保持する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-構築/aws-lambda-secrets-manager](#)

## aws-lambda-sns

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョニングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_sns</code>
 TypeScript	<code>@aws-solutions-constructs/aws-lambda-sns</code>
 Java	<code>software.amazon.awsconstructs.services.lambdasns</code>

## Overview

この AWS ソリューション構築物は、Amazon SNS トピックに接続された AWS Lambda 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { LambdaToSns, LambdaToSnsProps } from "@aws-solutions-constructs/aws-lambda-sns";

new LambdaToSns(this, 'test-lambda-sns', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

## Initializer

```
new LambdaToSns(scope: Construct, id: string, props: LambdaToSnsProps);
```

## パラメータ

- [scopeConstruct](#)
- `idstring`
- [propsLambdaToSnsProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj が提供される。
ExistingTopicObj ?	<a href="#">sns.Topic</a>	SNS トピックオブジェクトの既存のインスタンス。これとtopicProps はエラーを発生させます。
TopicProps ?	<a href="#">sns.TopicProps</a>	SNS トピックのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。
既存のVPCかな ?	<a href="#">ec2.IVpc</a>	このパターンをデプロイするオプションの既存のVPC。VPC にデプロイされると、Lambda 関数は VPC

名前	タイプ	説明
		<p>内の ENI を使用してネットワークリソースにアクセスし、インターフェイスエンドポイントは Amazon SQS 用の VPC 内に作成されます。既存の VPC が提供されている場合、<code>deployVpc</code> プロパティは、できません。true。これは、を使用します。ec2.IVpcを使用して、クライアントがスタックの外部に存在する VPC を提供できるようにします。<a href="#">ec2.Vpc.fromLookup()</a> メソッド。</p>

名前	タイプ	説明
DeployVPC ?	boolean	<p>に基づいて新しい VPC を作成するかどうか <code>vpcProps</code> のパターンを展開します。これをに設定する <code>true</code> は、パターンを実行するために、最小限のほとんどのプライベート VPC をデプロイします。</p> <ul style="list-style-type: none"><li>• CDK プログラムによって使用されるアベイラビリティゾーンごとに 1 つずつ分離されたサブネット。</li><li>• <code>enableDnsHostnames</code> および <code>enableDnsSupport</code> はどちらも <code>true</code>。</li></ul> <p>このプロパティが <code>true</code>、次に <code>existingVpc</code> は指定できません。デフォルトは <code>false</code> です。</p>

名前	タイプ	説明
vPCProps?	<a href="#">ec2.VpcProps</a>	新しい VPC のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。enableDns Hostnames ,enableDns Support ,natGateways およびsubnetConfiguration はパターンによって設定されるため、ここで指定されたプロパティの値はすべて上書きされます。もしdeployVpc ではありませんtrueの場合、このプロパティは無視されます。
topicArnEnvironmentVariable Name	string	Lambda 関数に設定された SNS トピック ARN 環境変数のオプション名。
トピック名環境変数名ですか?	string	Lambda 関数に設定された SNS トピック名環境変数のオプション名。

## パターンプロパティ

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
snsTopic	<a href="#">sns.Topic</a>	パターンによって作成された SNS トピックのインスタンスを返します。



名前	タイプ	説明
vpcかな	<a href="#">ec2.IVpc</a>	パターンによって使用される VPC のインスタンスを返します ( 存在する場合 )。これは、パターンによって作成された VPC、またはパターンコンストラクタに提供された VPC です。

## デフォルト設定

オーバーライドなしでコンストラクトをすぐに使用すると、次のデフォルトが設定されます。

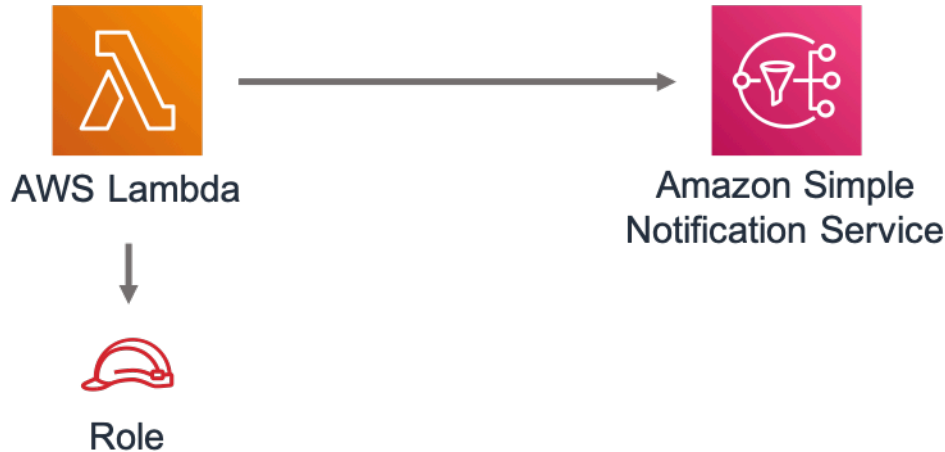
### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray によるトレースを有効にします。
- 環境変数の設定:
  - SNS\_TOPIC\_NAME (デフォルト)
  - SNS\_TOPIC\_ARN (デフォルト)
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

### Amazon SNS トピック

- SNS トピックの最小権限アクセス権限を設定します。
- AWS マネージド KMS キーを使用してサーバー側の暗号化を有効にします。
- 転送時のデータの暗号化を強制する。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。




[@aws-solutions-構築/aws-lambda-sns](#)



## aws-ラムダ-sqs

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_sq</code>

言語	パッケージ
 TypeScript	@aws-solutions-constructs/aws-lambda-sqs
 Java	software.amazon.awsconstructs.services.lambda.sqs

## Overview

この AWS ソリューション構成は、Amazon SQS キューに接続された AWS Lambda 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { LambdaToSqs, LambdaToSqsProps } from "@aws-solutions-constructs/aws-lambda-sqs";

new LambdaToSqs(this, 'LambdaToSqsPattern', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

## Initializer

```
new LambdaToSqs(scope: Construct, id: string, props: LambdaToSqsProps);
```

## パラメータ

- `scope`[Construct](#)

- idstring
- props[LambdaToSqsProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	デフォルト関数の代わりに使用されるオプションの既存の Lambda 関数。これと <code>lambdaFunctionProps</code> はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。
ExistingQueueObj ?	<a href="#">sqs.Queue</a>	デフォルトキューの代わりに使用されるオプションの既存の SQS キュー。これと <code>queueProps</code> はエラーを発生させます。
QueueProp?	<a href="#">sqs.QueueProps</a>	SQS キューのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティです。
キューのパージを有効化しますか？	boolean	SQS キューの削除を可能にする Lambda 関数に追加のアクセス許可を付与するかどうか。デフォルトは <code>false</code> です。
DeployDeadleterQueue	boolean	デッドレターキューとして使用する 2 次キューを作成する

名前	タイプ	説明
		かどうか。デフォルトは true です。
DeadletterQueueProps ?	<a href="#">sqs.QueueProps</a>	デッドレターキューのデフォルト小道具を上書きするオプションのユーザ提供の小道具です。場合にのみ使用されます。deployDeadLetterQueue プロパティが true に設定されます。
maxReceiveCount?	number	デッドレターキューに移動する前に、メッセージがデキューされた回数。デフォルトは 15 です。
既存のVPCかな？	<a href="#">ec2.IVpc</a>	このパターンをデプロイするオプションの既存のVPC。VPC にデプロイされると、Lambda 関数は VPC 内の ENI を使用してネットワークリソースにアクセスし、インターフェイスエンドポイントは Amazon SQS 用の VPC 内に作成されます。既存の VPC が提供されている場合、deployVpc プロパティは、true。あんec2.IVpcを使用して、クライアントがスタックの外部に存在する VPC を <a href="#">ec2.Vpc.fromLookup()</a> メソッド。

名前	タイプ	説明
DeployVPC ?	boolean	<p>に基づいて新しい VPC を作成するかどうか <code>vpcProps</code> のパターンを展開します。これをに設定する <code>true</code> は、パターンを実行するために、最小限のほとんどのプライベート VPC をデプロイします。</p> <ul style="list-style-type: none"><li>• CDK プログラムで使用されるアベイラビリティゾーンごとに 1 つずつサブネットを独立しました。</li><li>• <code>enableDnsHostnames</code> および <code>enableDnsSupport</code> はどちらも <code>true</code></li></ul> <p>このプロパティが <code>true</code>、次に <code>existingVpc</code> は指定できません。デフォルトは <code>false</code> です。</p>

名前	タイプ	説明
vPCProps?	<a href="#">ec2.VpcProps</a>	新しい VPC のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。enableDns Hostnames ,enableDns Support ,natGateways ,およびsubnetConfiguration はパターンによって設定されるため、ここで指定されたプロパティの値はすべてオーバーライドされます。もしdeployVpc ではありませんtrueに設定されている場合、このプロパティは無視されます。
キュー環境変数名ですか?	string	Lambda 関数に設定された SQS キュー URL 環境変数のオプション名。

## パターンプロパティ

名前	タイプ	説明
DeadLetterQueue	<a href="#">sqs.Queue</a>	パターンによって作成されたデッドレターキューのインスタンスを返します ( デプロイされている場合 )。
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。

名前	タイプ	説明
SQUEUE	<a href="#">sqs.Queue</a>	パターンによって作成された SQS キューのインスタンスを返します。
vpc?	<a href="#">ec2.IVpc</a>	パターンによって作成または使用される VPC のインスタンスを返します ( 存在する場合 )。これは、パターンによって作成された VPC、またはパターンコンストラクタに提供された VPC です。

## デフォルト設定

オーバーライドなしでコンストラクトをすぐに実装すると、次のデフォルトが設定されます。

### AWS Lambda 関数

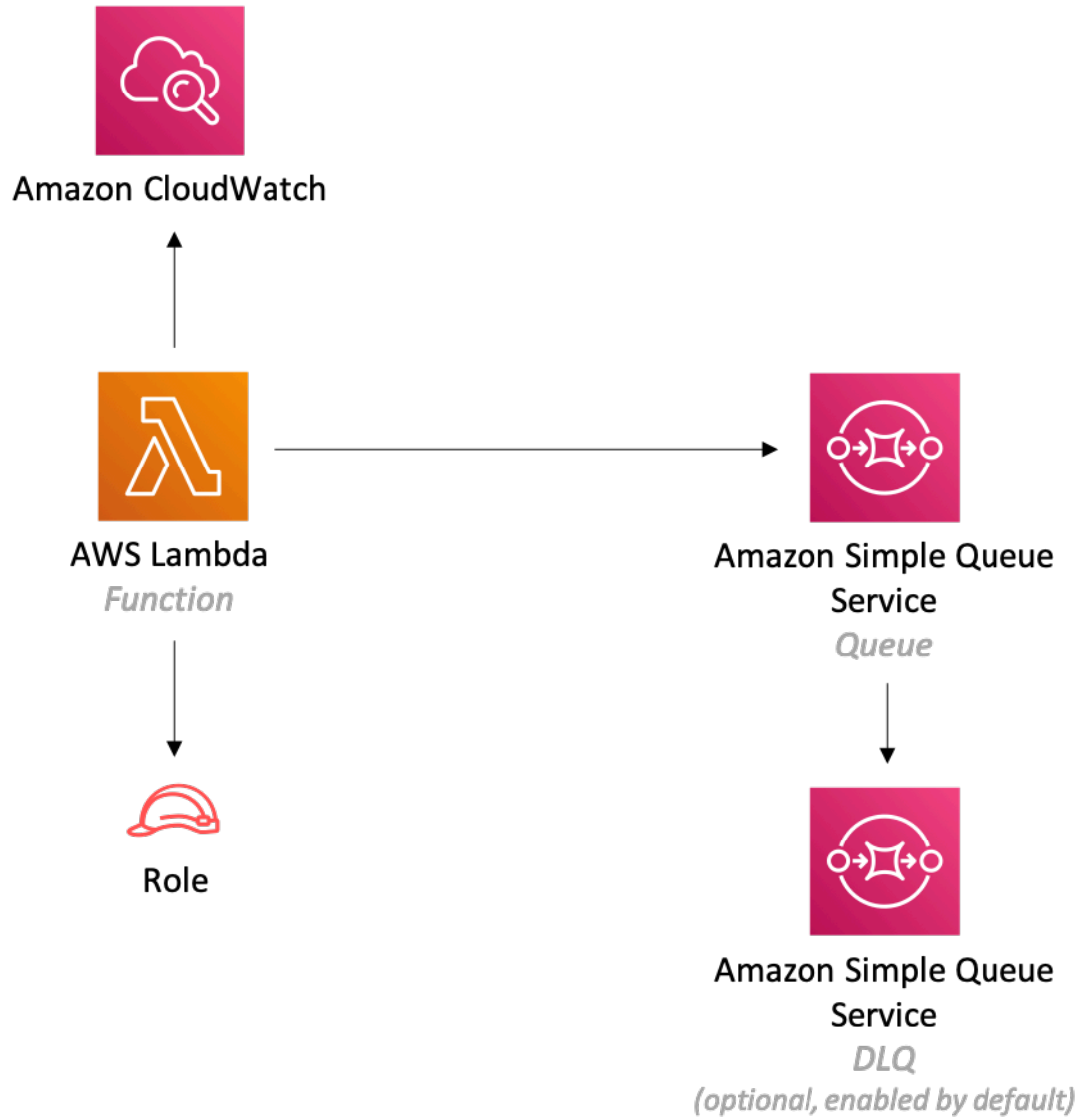
- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキーペアライブで接続を再利用できるようにします。
- 関数がキューへのメッセージの送信のみを許可する ( パージはenableQueuePurgeプロパティ)。
- X-Ray トレースを有効にする
- 環境変数の設定:
  - SQS\_QUEUE\_URL
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

### Amazon SQS キュー

- ソース SQS キューの SQS デッドレターキューをデプロイします。
- AWS マネージド KMS キーを使用して、ソース SQS キューのサーバー側の暗号化を可能にします。
- 転送時のデータの暗号化を強制する。



## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-lambda-sqs](#)




## aws-ラムダ-sqs-ラムダ

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンニング](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	aws_solutions_constructs.aws_lambda_sqs_lambda
 TypeScript	@aws-solutions-constructs/aws-lambda-sqs-lambda
 Java	software.amazon.awsconstructs.services.lambdasqslambda

## Overview

この AWS ソリューション構築パターンは、(1) キューにメッセージを送信するように設定された AWS Lambda 関数、(2) Amazon SQS キュー、(3) キューからメッセージを使用するように設定された AWS Lambda 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { LambdaToSqsToLambda, LambdaToSqsToLambdaProps } from "@aws-solutions-constructs/aws-lambda-sqs-lambda";

new LambdaToSqsToLambda(this, 'LambdaToSqsToLambdaPattern', {
  producerLambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/producer-function/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda/producer-function`),
```

```

    handler: 'index.handler'
  },
  consumerLambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/consumer-function/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda/consumer-function`),
    handler: 'index.handler'
  }
});

```

## Initializer

```
new LambdaToSqsToLambda(scope: Construct, id: string, props: LambdaToSqsToLambdaProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [LambdaToSqsToLambdaProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingProducerLambdaObj?	<a href="#">lambda.Function</a>	キューにメッセージを送信するためのデフォルト関数の代わりに使用する、オプションの既存の Lambda 関数。これとの両方を提供する producerLambdaFunctionProps はエラーを発生させます。
ProducerLambdaFunctionProps?	<a href="#">lambda.FunctionProps</a>	プロデューサーの Lambda 関数のデフォルトプロパティを

名前	タイプ	説明
		上書きするオプションのユーザー指定のプロパティ。
ExistingQueueObj ?	<a href="#">sqs.Queue</a>	デフォルトキューの代わりに使用されるオプションの既存の SQS キュー。これとの両方を提供するqueueProps はエラーを発生させます。
キュープロップ?	<a href="#">sqs.QueueProps</a>	SQS キューのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。これとの両方を提供するexistingQueueObj はエラーを発生させます。
デプロイデッドレターキュー?	boolean	デッドレターキューとして使用するセカンダリキューを作成するかどうか。デフォルトは true です。
DeadletterQueueProps ?	<a href="#">sqs.QueueProps</a>	デッドレターキューのデフォルト小道具を上書きするオプションのユーザー提供の小道具です。場合にのみ使用されません。deployDeadLetterQueue プロパティはtrue。
maxReceiveCount?	number	デッドレターキューに移動する前に、メッセージがデッドキューに失敗した回数。デフォルトは 15 です。

名前	タイプ	説明
ExistingConsumerLambdaObj?	<a href="#">lambda.Function</a>	キューからメッセージを受信/消費するためのデフォルト関数の代わりに使用されるオプションの既存の Lambda 関数。これとの両方を提供する consumerLambdaFunctionProps はエラーを発生させます。
ConsumerLambdaFunctionProps?	<a href="#">lambda.FunctionProps</a>	コンシューマー Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。
キュー環境変数名ですか?	string	プロデューサ Lambda 関数に設定された SQS キュー URL 環境変数のオプション名。

## パターンプロパティ

名前	タイプ	説明
コンシューマーラムダファンクション	<a href="#">lambda.Function</a>	パターンによって作成されたコンシューマ Lambda 関数のインスタンスを返します。
DeadLetterQueue	<a href="#">sqs.Queue</a>	パターンによって作成されたデッドレターキューのインスタンスを返します (デプロイされている場合)。
生産者ラムダファンクション	<a href="#">lambda.Function</a>	パターンによって作成されたプロデューサ Lambda 関数のインスタンスを返します。

名前	タイプ	説明
SQUEUE	<a href="#">sqs.Queue</a>	パターンによって作成された SQS キューのインスタンスを返します。

## デフォルト設定

このコンストラクトのすぐれた実装 ( オーバーライドされたプロパティなし ) は、次のデフォルトに従います。

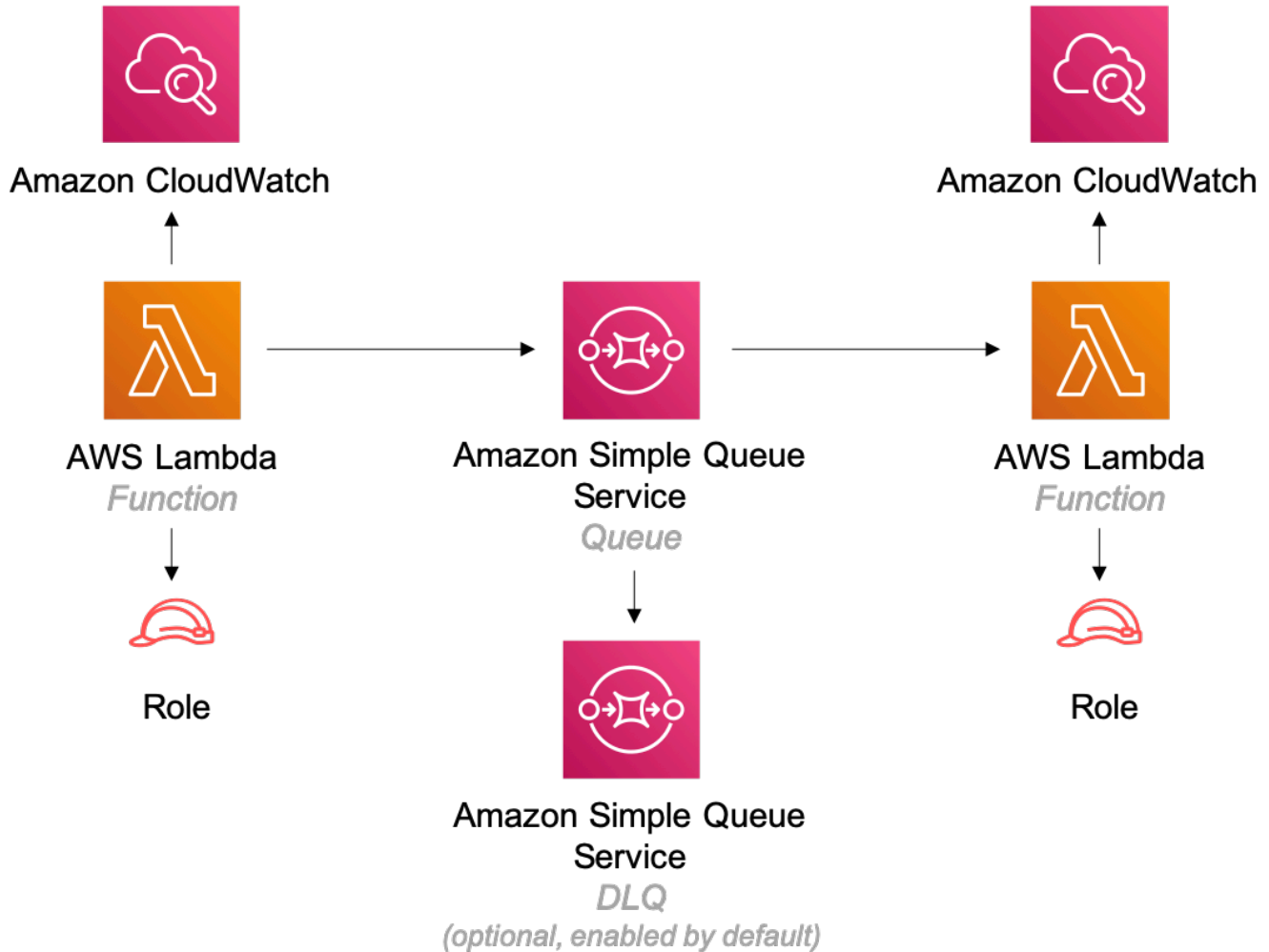
### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキーペアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にする
- 環境変数の設定:
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED` ( ノード10.x以上の機能の場合 )

### Amazon SQS キュー

- プライマリキューにデッドレターキューをデプロイします。
- AWS Managed KMS キーを使用して、プライマリキューに対してサーバー側の暗号化を有効にします。
- 転送時のデータの暗号化を強制する

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-lambda-sqs-lambda](https://github.com/@aws-ソリューション-構築/aws-lambda-sqs-lambda)




## aws-ラムダ-ステップ関数

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージ

を使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_lambda_step_function</code>
 TypeScript	<code>@aws-solutions-constructs/aws-lambda-step-function</code>
 Java	<code>software.amazon.awsconstructs.services.lambdastepfunction</code>

## Overview

この AWS ソリューション構築物は、AWS ステップ関数に接続された AWS Lambda 関数を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { LambdaToStepFunction } from '@aws-solutions-constructs/aws-lambda-step-function';
import * as stepfunctions from '@aws-cdk/aws-stepfunctions';

const startState = new stepfunctions.Pass(this, 'StartState');

new LambdaToStepFunction(this, 'LambdaToStepFunctionPattern', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```



```

    },
    stateMachineProps: {
      definition: startState
    }
  });

```

## Initializer

```

new LambdaToStepFunction(scope: Construct, id: string, props:
  LambdaToStepFunctionProps);

```

### パラメータ

- scope [Construct](#)
- id `string`
- props [LambdaToStepFunctionProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。次の場合は無視されますexistingLambdaObj が提供される。
ステートメントマシンプロップ	<a href="#">sfn.StateMachineProps</a>	sfn.stateMachineにユーザーが提供した小道具です。

名前	タイプ	説明
クラウド・ワット・チャラームズ	boolean	推奨される CloudWatch アラームを作成するかどうか。
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググループの CloudWatch Logs ロググループのデフォルトの小道具を上書きするオプションのユーザー指定の小道具です。
ステートメントマシン環境変数名	string	プロデューサーの Lambda 関数に設定された Step Functions 状態マシン環境変数のオプション名。

## パターンプロパティ

名前	タイプ	説明
CloudWatchAlarms?	<a href="#">cloudwatch.Alarm[]</a>	パターンによって作成される 1 つ以上の CloudWatch アラームのリストを返します。
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
StateMachine	<a href="#">sfn.StateMachine</a>	パターンによって作成されたステートマシンのインスタンスを返します。
ステートメントマシンロググループ	<a href="#">logs.LogGroup</a>	ステートマシンのパターンによって作成されたロググループのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

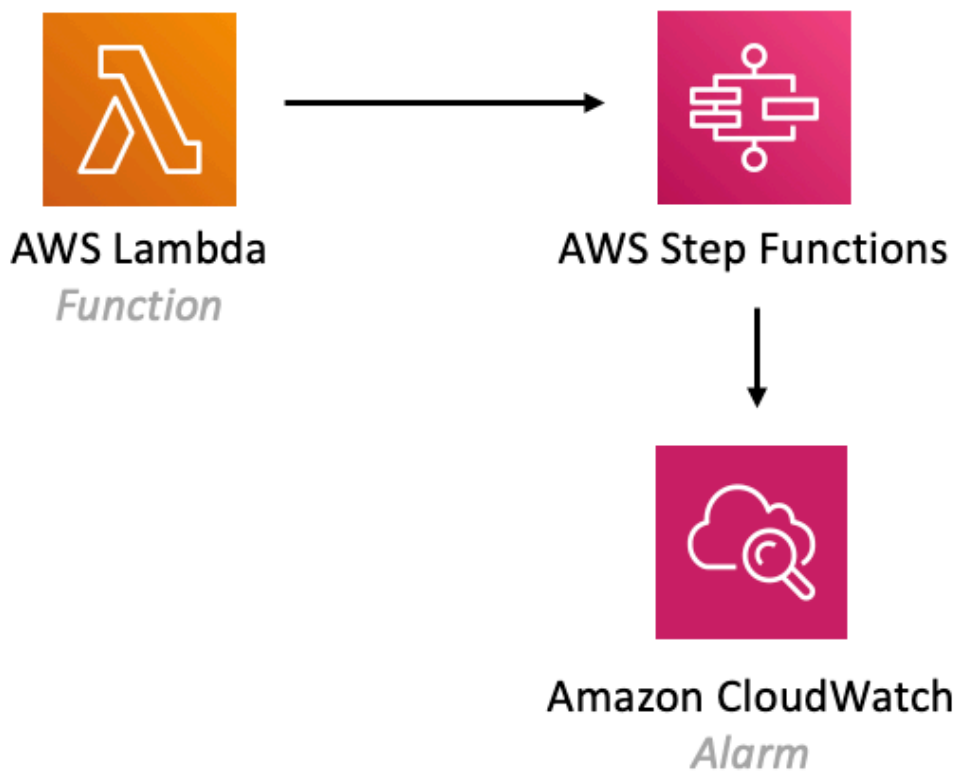
### AWS Lambda 関数

- Lambda 関数の制限付きアクセス権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray によるトレースを有効にします。
- 環境変数の設定:
  - STATE\_MACHINE\_ARN (デフォルト)
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

### AWS Step Functions ステート

- AWS Step Functions ステートマシン用のベストプラクティスの CloudWatch アラームをデプロイします。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-ラムダ-ステップ関数](#)




## aws-s3-ラムダ

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_s3_lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-s3-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.s3lambda</code>

## Overview

この AWS ソリューション構築物は、AWS Lambda 関数に接続された Amazon S3 バケットを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { S3ToLambdaProps, S3ToLambda } from '@aws-solutions-constructs/aws-s3-lambda';

new S3ToLambda(this, 'test-s3-lambda', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  },
});
```

## Initializer

```
new S3ToLambda(scope: Construct, id: string, props: S3ToLambdaProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [S3ToLambdaProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Function オブジェクトの既存のインスタンス。これと lambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj にはあります。
ExistingBucketObj ?	<a href="#">s3.Bucket</a>	S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。
BucketProps ?	<a href="#">s3.BucketProps</a>	オプションのユーザー提供のプロパティ。バケットの

名前	タイプ	説明
		デフォルトプロパティを上書きします。の場合は無視されます。existingBucketObj はにあります。
S3EventSourceProps ?	<a href="#">S3EventSourceProps</a>	S3EventSourceProps のデフォルトのプロップを上書きするオプションのユーザ提供の小道具

## パターンプロパティ

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
S3bucket?	<a href="#">s3.Bucket</a>	パターンによって作成された S3 バケットのインスタンスを返します。
s3loggingBucket ?	<a href="#">s3.Bucket</a>	S3 バケットのパターンによって作成されたロギングバケットのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon S3 バケット名

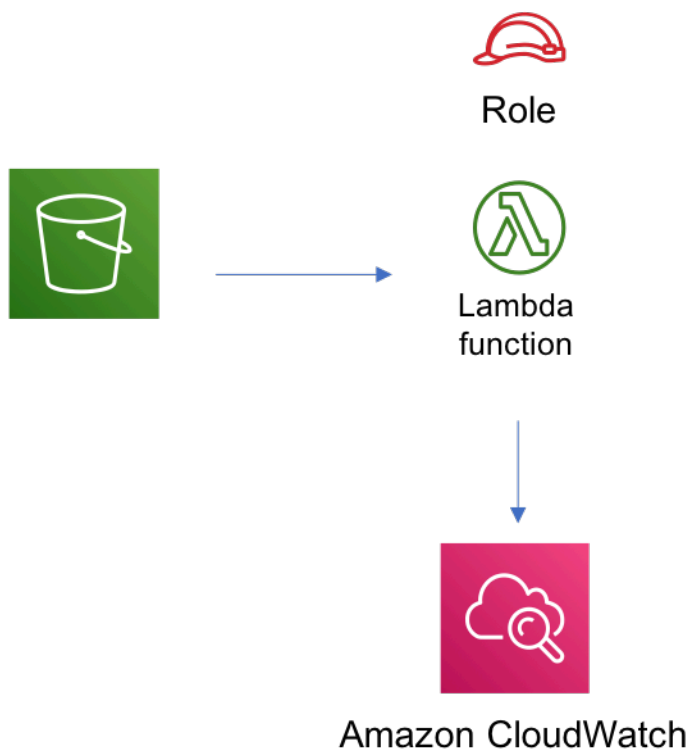
- S3 バケットのアクセスログを設定します。
- AWS マネージド KMS キーを使用して S3 バケットのサーバー側の暗号化を有効にします。

- S3 バケットのバージョニングを有効にします。
- S3 バケットのパブリックアクセスを許可しません。
- CloudFormation スタックを削除するときは、S3 バケットを保持します。
- 転送時のデータの暗号化を強制する。
- 90 日後に Glacier ストレージに最新でないオブジェクトバージョンを移動するライフサイクルルールを適用します。

## AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にします。
- 環境変数の設定:
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED` ( ノード10.x以上の機能の場合 )

## Architecture





## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-solutions-constructions/aws-s3-ラムダ](#)

## aws-s3-sqs

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンング](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	aws_solutions_constructs.aws_s3_sqs
 TypeScript	@aws-solutions-constructs/aws-s3-sqs
 Java	software.amazon.awsconstructs.services.s3sqs

## Overview

この AWS ソリューション構築は、Amazon SQS キューに通知を送信するように設定された Amazon S3 バケットを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { S3ToSqs } from "@aws-solutions-constructs/aws-s3-sqs";

new S3ToSqs(stack, 'S3ToSQSPattern', {});
```

## Initializer

```
new S3ToSqs(scope: Construct, id: string, props: S3ToSqsProps);
```

### パラメータ

- `scope`[Construct](#)
- `id``string`
- `props`[S3ToSqsProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingBucketObj ?	<a href="#">s3.Bucket</a>	S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、 <code>bucketProps</code> はエラーです。
BucketProps ?	<a href="#">s3.BucketProps</a>	オプションのユーザー提供の小道具で、S3 バケットのデフォルトの小道具を上書きします。
s3EventTypes ?	<a href="#">s3.EventType[]</a>	通知をトリガーする S3 イベントタイプ。デフォルト

名前	タイプ	説明
		は <code>s3.EventType.OBJECT_CREATED</code> です。
S3EventFilters ?	<a href="#">s3.NotificationKeyFilter[]</a>	このイベントをトリガーするオブジェクトを決定するための S3 オブジェクトキーフィルタ規則。指定しない場合、フィルタ規則は適用されません。
ExistingQueueObj ?	<a href="#">sqs.Queue</a>	デフォルトキューの代わりに使用されるオプションの既存の SQS キュー。これと <code>queueProps</code> はエラーを発生させます。SQS キューが暗号化されている場合、暗号化に使用する KMS キーは、カスタマー管理の CMK である必要があります。
QueueProp?	<a href="#">sqs.QueueProps</a>	SQS キューのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティです。場合は、無視されます。existingQueueObj が提供される。
DeadletterQueueProps ?	<a href="#">sqs.QueueProps</a>	デッドレターキューのデフォルト小道具を上書きするオプションのユーザー提供の小道具です。場合にのみ使用されません。deployDeadLetterQueue プロパティが true に設定されている場合にのみ実行されます。

名前	タイプ	説明
DeployDeadleterQueue	boolean	デッドレターキューとして使用するセカンダリキューを作成するかどうか。デフォルトは true です。
maxReceiveCount?	number	デッドレターキューに移動する前に、メッセージがデキューに失敗した回数。デフォルトは 15 です。
顧客管理キーによる暗号化の有効化	boolean	この CDK アプリで管理される KMS キーを使用するか、インポートされるかを指定します。暗号化キーをインポートする場合は、暗号化キーを encryptionKey プロパティです。
encryptionKey	<a href="#">kms.Key</a>	デフォルトの暗号化キーの代わりに使用する、オプションの既存の暗号化キー。
EncryptionKeyProps ?	<a href="#">kms.KeyProps</a>	オプションのユーザー指定のプロパティで、暗号化キーのデフォルトプロパティを上書きします。

## パターンプロパティ

名前	タイプ	説明
SQUEUE	<a href="#">sqs.Queue</a>	パターンによって作成された SQS キューのインスタンスを返します。

名前	タイプ	説明
DeadLetterQueue	<a href="#">sqs.Queue</a>	パターンによって作成されたデッドレターキューのインスタンスを返します ( デプロイされている場合 )。
encryptionKey	<a href="#">kms.IKey</a>	パターンによって作成された暗号化キーのインスタンスを返します。
S3bucket ?	<a href="#">s3.Bucket</a>	パターンによって作成された S3 バケットのインスタンスを返します。
s3loggingBucket ?	<a href="#">s3.Bucket</a>	S3 バケットのパターンによって作成されたロギングバケットのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

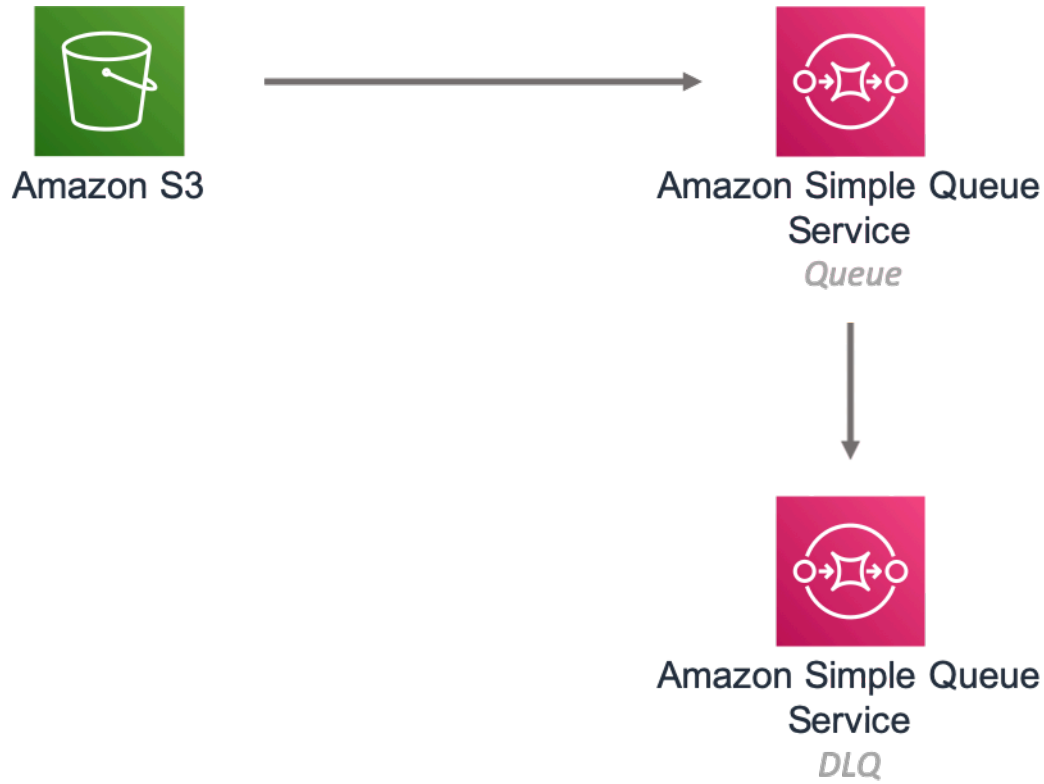
### Amazon S3 バケット

- S3 バケットのアクセスログの設定
- AWS マネージド KMS キーを使用した S3 バケットのサーバー側の暗号化の有効化
- S3 バケットのバージョニングを有効にする
- S3 バケットのパブリックアクセスを許可しない
- CloudFormation スタックを削除するときに S3 バケットを保持する
- 転送時のデータの暗号化を強制する
- ライフサイクルルールを適用して、90 日後に最新でないオブジェクトバージョンを Glacier ストレージに移動する

## Amazon SQS キュー

- SQS キューの最小権限アクセス権限の設定
- ソース SQS キューの SQS デッドレターキューのデプロイ
- カスタマー管理された KMS キーを使用した SQS キューのサーバー側の暗号化を有効にする
- 転送時のデータの暗号化を強制する

### Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-s3-sqs](#)




## aws-s3ステップ関数

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは対象外です [セマンティックバージョンング](#) モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_s3_step_function</code>
 TypeScript	<code>@aws-solutions-constructs/aws-s3-step-function</code>
 Java	<code>software.amazon.awsconstructs.services.s3stepfunction</code>

## Overview

この AWS ソリューション構築物は、AWS ステップ関数に接続された Amazon S3 バケットを実装します。

### Note

この構造体は、Amazon EventBridge ( Amazon CloudWatch Events ) を使用して AWS Step Functions をトリガーします。EventBridge はより柔軟ですが、S3 イベント通知を使用して Step Functions をトリガーするレイテンシーが少なく、費用対効果が高くなります。コストやレイテンシーに問題がある場合は、`aws-s3-lambda` および `aws-lambda-stepfunctions` この構成要素の代わりに。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { S3ToStepFunction, S3ToStepFunctionProps } from '@aws-solutions-constructs/aws-s3-step-function';
import * as stepfunctions from '@aws-cdk/aws-stepfunctions';

const startState = new stepfunctions.Pass(this, 'StartState');

new S3ToStepFunction(this, 'test-s3-step-function-stack', {
  stateMachineProps: {
    definition: startState
  }
});
```

## Initializer

```
new S3ToStepFunction(scope: Construct, id: string, props: S3ToStepFunctionProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [S3ToStepFunctionProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingBucketObj ?	<a href="#">s3.IBucket</a>	S3 Bucket オブジェクトの既存のインスタンス。これが提供されている場合は、bucketProps はエラーです。
BucketProps ?	<a href="#">s3.BucketProps</a>	オプションのユーザー提供のプロパティ。バケットのデフォルトプロパティを上書き



名前	タイプ	説明
ステートメントマシンプロップ	<a href="#">sfn.StateMachinePr ops</a>	します。の場合は無視されま ずexistingBucketObj が 提供される。  sfn.stateMachine のデフォル トプロップを上書きするオプ ションのユーザ提供の小道具 です。
EventRuleProps ?	<a href="#">events.RuleProps</a>	オプションのユーザーがデ フォルトを上書きするために EventRuleProps を提供しまし た。
デプロイクラウドトレイル ?	boolean	AWS CloudTrail でトレイルを デプロイして、Amazon S3 で API イベントを記録するかと うか。デフォルトは true で す。
クラウド・ワット・チャラー ムズ	boolean	推奨される CloudWatch ア ラームを作成するかどうか。
LogGroupProps ?	<a href="#">logs.LogGroupProps</a>	CloudWatch Logs ロググルー プのデフォルトの小道具を上 書きする、オプションのユー ザー提供の小道具です。

## パターンプロパティ

名前	タイプ	説明
cloudtrail?	<a href="#">cloudtrail.Trail</a>	パターンによって作成された Cloudtrailトレイルのインス タンスを返します。

名前	タイプ	説明
CloudTrailBucket ?	<a href="#"><u>s3.Bucket</u></a>	Cloudtrail トレイルデータを格納するためにパターンによって作成されたバケットのインスタンスを返します。
CloudTrailLoggingBucket ?	<a href="#"><u>s3.Bucket</u></a>	Cloudtrail トレイルで使用されるプライマリバケットのパターンによって作成されたロギングバケットのインスタンスを返します。
CloudWatchAlarms ?	<a href="#"><u>cloudwatch.Alarm[]</u></a>	パターンによって作成される 1 つまたは複数の CloudWatch アラームのリストを返します。
S3bucket ?	<a href="#"><u>s3.Bucket</u></a>	パターンによって作成された S3 バケットのインスタンスを返します。
s3loggingBucket ?	<a href="#"><u>s3.Bucket</u></a>	S3 バケットのパターンによって作成されたロギングバケットのインスタンスを返します。
StateMachine	<a href="#"><u>sfn.StateMachine</u></a>	パターンによって作成されたステートマシンのインスタンスを返します。
ステートメントマシンロググループ	<a href="#"><u>logs.LogGroup</u></a>	ステートマシンのパターンによって作成されたロググループのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

### Amazon S3 バケット

- S3 バケットのアクセスログを設定します。
- AWS マネージド KMS キーを使用して S3 バケットのサーバー側の暗号化を有効にします。
- S3 バケットのバージョニングを有効にします。
- S3 バケットのパブリックアクセスを許可しません。
- CloudFormation スタックを削除するときは、S3 バケットを保持します。
- 転送時のデータの暗号化を強制します。
- 90 日後に Glacier ストレージに最新でないオブジェクトバージョンを移動するライフサイクルルールを適用します。

### AWS CloudTrail

- AWS CloudTrail で証跡を設定し、構築によって作成されたバケットに関連する Amazon S3 の API イベントを記録します。

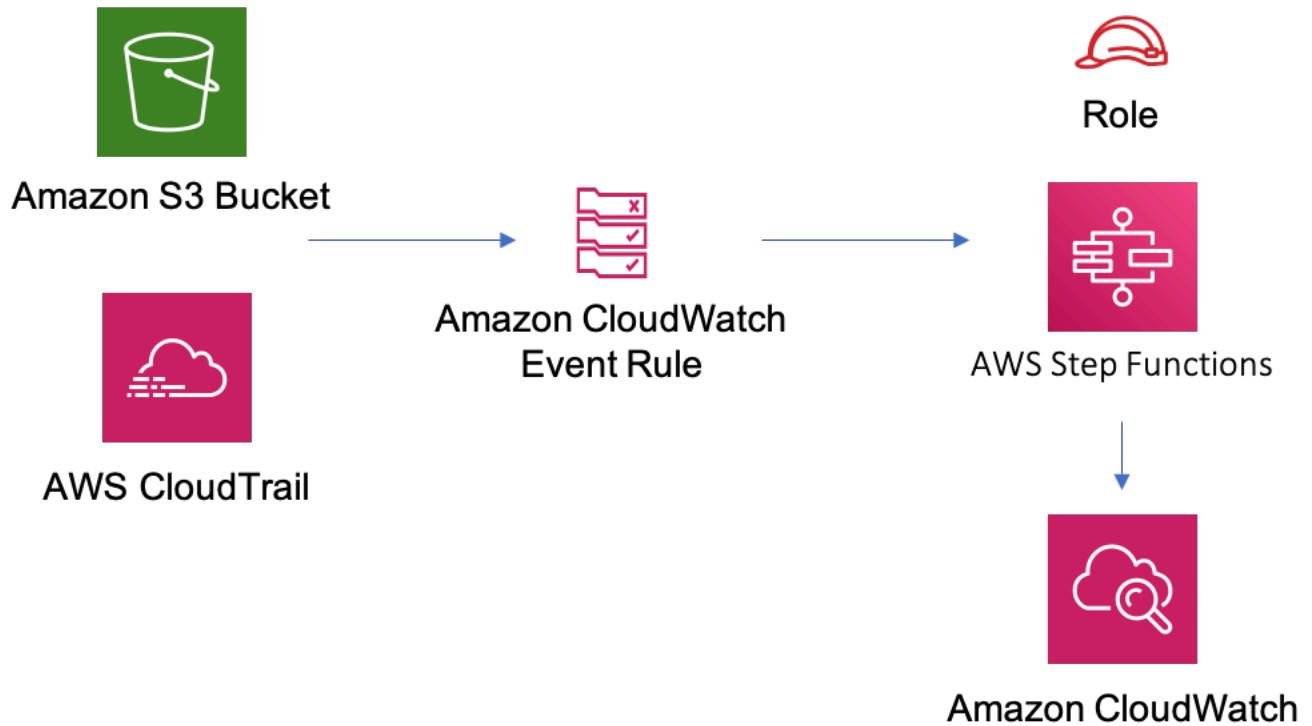
### Amazon CloudWatch Events Events ルール

- CloudWatch イベントに Lambda 関数をトリガーするための最小限の権限を付与します。

### AWS ステップ関数

- API Gateway の CloudWatch ログを有効にします。
- ベストプラクティスの CloudWatch アラームをステップ関数にデプロイします。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。






[@aws-ソリューション-構築/aws-s3-ステップ関数](#)

## aws-sns-ラムダ

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_sns_lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-sns-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.snslambda</code>

## Overview

この AWS ソリューション構築物は、AWS Lambda 関数に接続された Amazon SNS を実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { SnsToLambda, SnsToLambdaProps } from "@aws-solutions-constructs/aws-sns-lambda";

new SnsToLambda(this, 'test-sns-lambda', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

## Initializer

```
new SnsToLambda(scope: Construct, id: string, props: SnsToLambdaProps);
```

## パラメータ

- scope [Construct](#)
- idstring
- props [SnsToLambdaProps](#)

## パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj ?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps ?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj にはあります。
ExistingTopicobj ?	<a href="#">sns.Topic</a>	SNS トピックオブジェクトの既存のインスタンス。これとtopicProps はエラーを発生させます。
TopicProps ?	<a href="#">sns.TopicProps</a>	SNS トピックのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。

## パターンプロパティ

名前	タイプ	説明
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。
snsTopic	<a href="#">sns.Topic</a>	パターンによって作成された SNS トピックのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

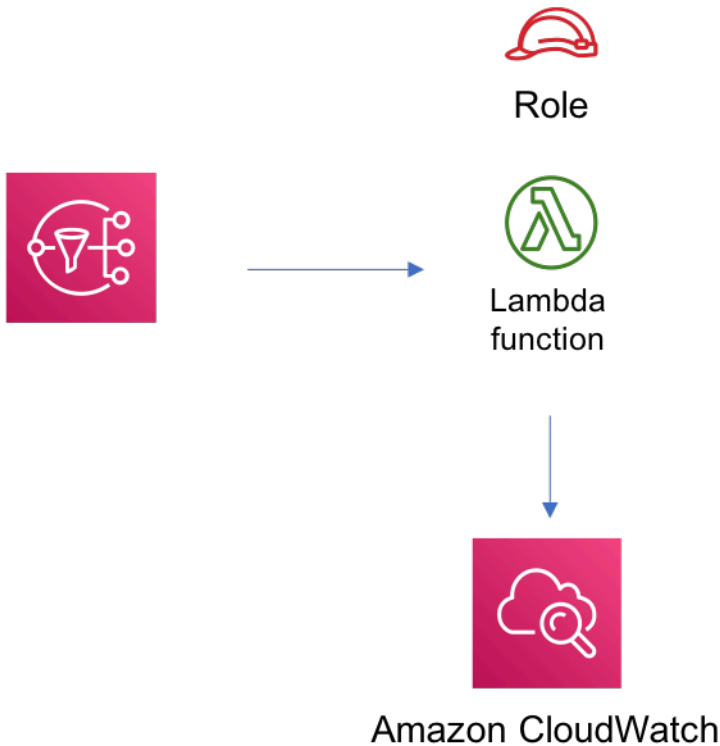
### Amazon SNS トピック

- SNS トピックの最小権限アクセス権限を設定します。
- AWS マネージド KMS キーを使用してサーバー側の暗号化を有効にします。
- 転送時のデータの暗号化を強制する。

### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にします。
- 環境変数の設定:
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED` ( ノード10.x以上の機能の場合 )

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-sns-lambda](#)




## aws-sns-sqs

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。



言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_sns_sqs</code>
 TypeScript	<code>@aws-solutions-constructs/aws-sns-sqs</code>
 Java	<code>software.amazon.awsconstructs.services.snssqs</code>

## Overview

この AWS ソリューション構築は、Amazon SQS キューに接続された Amazon SNS トピックを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
import { SnsToSqs, SnsToSqsProps } from "@aws-solutions-constructs/aws-sns-sqs";
import * as iam from '@aws-cdk/aws-iam';

const snsToSqsStack = new SnsToSqs(this, 'SnsToSqsPattern', {});

// Grant yourself permissions to use the Customer Managed KMS Key
const policyStatement = new iam.PolicyStatement({
  actions: ["kms:Encrypt", "kms:Decrypt"],
  effect: iam.Effect.ALLOW,
  principals: [ new iam.AccountRootPrincipal() ],
  resources: [ "*" ]
});

snsToSqsStack.encryptionKey?.addToResourcePolicy(policyStatement);
```

## Initializer

```
new SnsToSqs(scope: Construct, id: string, props: SnsToSqsProps);
```

### パラメータ

- scope [Construct](#)
- id `string`
- props [SnsToSqsProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingTopicObj ?	<a href="#">sns.Topic</a>	SNS トピックオブジェクトの既存のインスタンス。これと topicProps はエラーを発生させます。
TopicProps ?	<a href="#">sns.TopicProps</a>	SNS トピックのデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されません。existingTopicObj が提供される。
ExistingQueueObj ?	<a href="#">sqs.Queue</a>	デフォルトキューの代わりに使用されるオプションの既存の SQS キュー。これと queueProps はエラーを発生させます。
キュープロップ	<a href="#">sqs.QueueProps</a>	SQS キューのデフォルトプロパティを上書きするオプションのユーザー指定のプロ

名前	タイプ	説明
DeployDeadleterQueue	boolean	□プロパティです。の場合は無視されます。existingQueueObj が提供される。  デッドレターキューとして使用するセカンダリキューを作成するかどうか。デフォルトは true です。
DeadletterQueueProps?	<a href="#">sqs.QueueProps</a>	デッドレターキューのデフォルト小道具を上書きするオプションのユーザ提供の小道具です。場合にのみ使用されません。deployDeadLetterQueue プロパティが true に設定されています。
maxReceiveCount?	number	デッドレターキューに移動する前に、メッセージがデキューに失敗した回数。デフォルトは 15 です。
顧客管理キーによる暗号化の有効化	boolean	カスタマー管理型の暗号化キーを使用するかどうか。このCDKアプリによって管理されるか、インポートされます。暗号化キーをインポートする場合、暗号化キーをencryptionKey プロパティです。
encryptionKey	<a href="#">kms.Key</a>	デフォルトの暗号化キーの代わりに使用する、オプションの既存の暗号化キー。

名前	タイプ	説明
EncryptionKeyProps ?	<a href="#">kms.KeyProps</a>	オプションのユーザー指定のプロパティで、暗号化キーのデフォルトプロパティを上書きします。

## パターンプロパティ

名前	タイプ	説明
snsTopic	<a href="#">sns.Topic</a>	パターンによって作成された SNS トピックのインスタンスを返します。
encryptionKey	<a href="#">kms.Key</a>	パターンによって作成された暗号化キーのインスタンスを返します。
SQUEUE	<a href="#">sqs.Queue</a>	パターンによって作成された SQS キューのインスタンスを返します。
DeadLetterQueue	<a href="#">sqs.Queue</a>	パターンによって作成されたデッドレターキューのインスタンスを返します ( デプロイされている場合 )。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

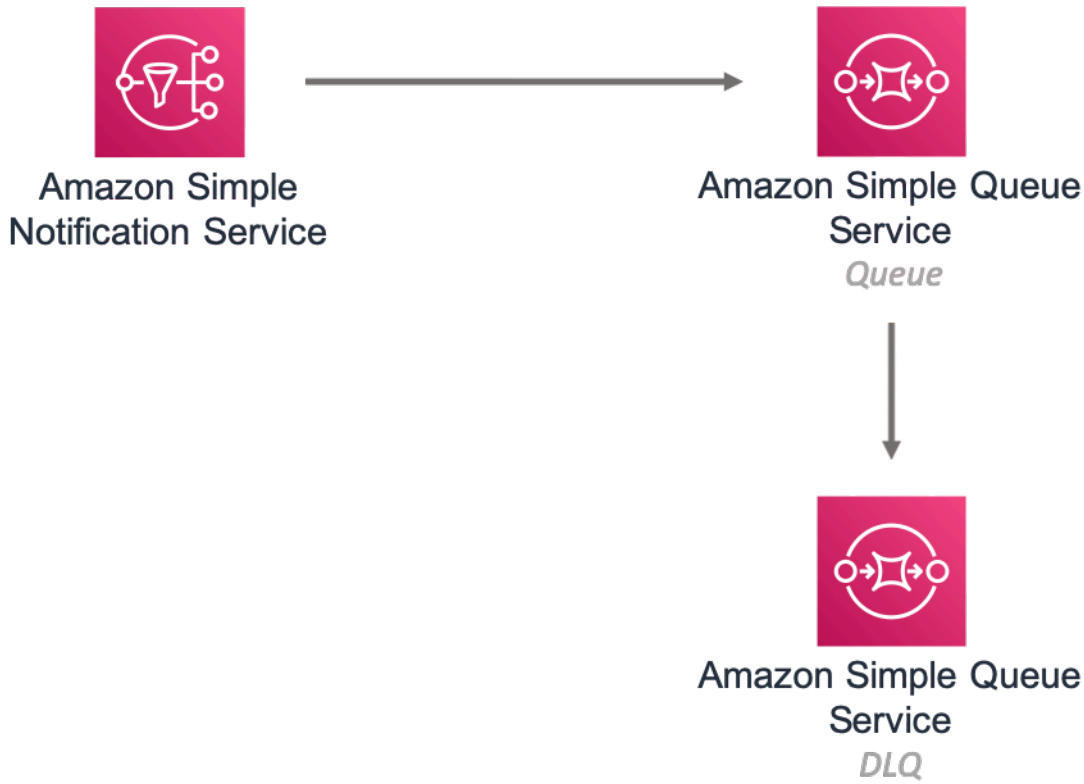
### Amazon SNS トピック

- SNS トピックの最小権限アクセス権限を設定します。
- AWS マネージド KMS キーを使用してサーバー側の暗号化を有効にします。
- 転送時のデータの暗号化を強制する。

## Amazon SQS キュー

- SQS キューの最小権限アクセス権限を設定します。
- ソース SQS キューのデッドレターキューをデプロイする。
- カスタマー管理の KMS キーを使用して、SQS キューのサーバー側の暗号化を可能にします。
- 転送時のデータの暗号化を強制する。

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-sns-sqs](#)




## aws-sqs-ラムダ

STABILITY

EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョン管理](#)モデル。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときソースコードを更新する必要があるかもしれません。

注意: 正しく機能するためには、プロジェクト内の AWS ソリューション構築パッケージと AWS CDK パッケージが同じバージョンである必要があります。

言語	パッケージ
 Python	<code>aws_solutions_constructs.aws_sqs_lambda</code>
 TypeScript	<code>@aws-solutions-constructs/aws-sqs-lambda</code>
 Java	<code>software.amazon.awsconstructs.services.sqslambda</code>

## Overview

この AWS ソリューション構築物は、AWS Lambda 関数に接続された Amazon SQS キューを実装します。

TypeScript の最小限のデプロイ可能なパターン定義は次のとおりです。

```
const { SqsToLambda } = require('@aws-solutions-constructs/aws-sqs-lambda');

new SqsToLambda(stack, 'SqsToLambdaPattern', {
  lambdaFunctionProps: {
    runtime: lambda.Runtime.NODEJS_14_X,
    // This assumes a handler function in lib/lambda/index.js
    code: lambda.Code.fromAsset(`${__dirname}/lambda`),
    handler: 'index.handler'
  }
});
```

## Initializer

```
new SqsToLambda(scope: Construct, id: string, props: SqsToLambdaProps);
```

### パラメータ

- scope [Construct](#)
- id [string](#)
- props [SqsToLambdaProps](#)

### パターン構成プロパティ

名前	タイプ	説明
ExistingLambdaObj?	<a href="#">lambda.Function</a>	Lambda Functionオブジェクトの既存のインスタンス。これとlambdaFunctionProps はエラーを発生させます。
LambdaFunctionProps?	<a href="#">lambda.FunctionProps</a>	Lambda 関数のデフォルトプロパティを上書きするオプションのユーザー指定のプロパティ。の場合は無視されます。existingLambdaObj にはあります。
ExistingQueueObj?	<a href="#">sqs.Queue</a>	デフォルトキューの代わりに使用されるオプションの既存の SQS キュー。これとqueueProps はエラーを発生させます。
QueueProp?	<a href="#">sqs.QueueProps</a>	SQS キューのデフォルトプロパティを上書きするオプ

名前	タイプ	説明
		シヨンのユーザー指定のプロパティ。の場合は無視されません。existingQueueObj はにあります。
DeployDeadleterQueue	boolean	デッドレターキューとして使用するセカンダリキューを作成するかどうか。デフォルトは true です。
DeadletterQueueProps ?	<a href="#">sqs.QueueProps</a>	デッドレターキューのデフォルト小道具を上書きするオプションのユーザ提供の小道具です。場合にのみ使用されません。deployDeadLetterQueue プロパティが true に設定された場合。
maxReceiveCount?	number	デッドレターキューに移動する前に、メッセージがデキューに失敗した回数。デフォルトは 15 です。

## パターンプロパティ

名前	タイプ	説明
DeadLetterQueue	<a href="#">sqs.Queue</a>	パターンによって作成されたデッドレターキューのインスタンスを返します ( デプロイされている場合 )。
LambdaFunction	<a href="#">lambda.Function</a>	パターンによって作成された Lambda 関数のインスタンスを返します。



名前	タイプ	説明
SQUEUE	<a href="#">sqs.Queue</a>	パターンによって作成された SQS キューのインスタンスを返します。

## デフォルト設定

オーバーライドなしでこのパターンをすぐに実装すると、次のデフォルトが設定されます。

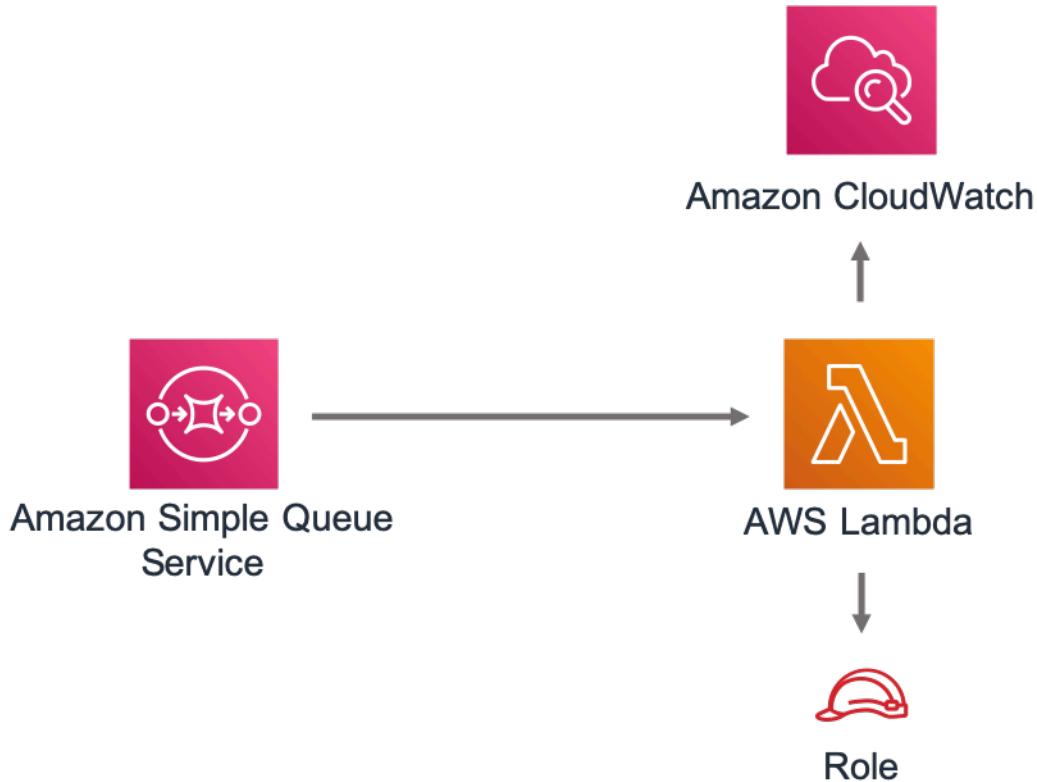
### Amazon SQS キュー

- ソース SQS キューの SQS デッドレターキューをデプロイします。
- AWS マネージド KMS キーを使用して、ソース SQS キューのサーバー側の暗号化を可能にします。
- 転送時のデータの暗号化を強制する

### AWS Lambda 関数

- Lambda 関数の制限付き特権アクセス IAM ロールを設定します。
- NodeJS Lambda 関数のキープアライブで接続を再利用できるようにします。
- X-Ray トレースを有効にする
- 環境変数の設定:
  - AWS\_NODEJS\_CONNECTION\_REUSE\_ENABLED ( ノード10.x以上の機能の場合 )

## Architecture



## GitHub

このパターンのコードを表示するには、問題を作成/表示し、プル要求などを行います。



[@aws-ソリューション-構築/aws-sqs-ラムダ](#)

## core

STABILITY EXPERIMENTAL

すべてのクラスは積極的に開発されており、将来のバージョンでは下位互換性がない変更または削除の対象となります。これらは、[セマンティックバージョンニングモデル](#)。つまり、これらのパッケージを使用するかもしれませんが、このパッケージの新しいバージョンにアップグレードするときにソースコードを更新する必要があるかもしれません。

コアライブラリには、AWS ソリューション構成の基本的な構成要素が含まれています。AWS ソリューションコンストラクトの残りの部分で使用されるコアクラスを定義します。

## AWS CDK コンストラクトのデフォルトプロパティ

コアライブラリは、AWS ソリューションコンストラクトで使用される AWS CDK コンストラクトのデフォルトのプロパティを設定します。

たとえば、AWS Solutions Constructs コンストラクトで作成された S3 Bucket コンストラクトのデフォルトプロパティのスニペットは次のとおりです。デフォルトでは、サーバー側の暗号化、バケットのバージョニングが有効になり、すべてのパブリックアクセスがブロックされ、S3 アクセスログがセットアップされます。

```
{
  encryption: s3.BucketEncryption.S3_MANAGED,
  versioned: true,
  blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
  removalPolicy: RemovalPolicy.RETAIN,
  serverAccessLogsBucket: loggingBucket
}
```

## 既定のプロパティを上書きする

コアライブラリによって設定されたデフォルトのプロパティは、ユーザーが指定したプロパティによって上書きできます。たとえば、特定の要件を満たすために、ユーザーは Amazon S3 のパブリックアクセスをブロックするプロパティを上書きできます。

```
const stack = new cdk.Stack();

const props: CloudFrontToS3Props = {
  bucketProps: {
    blockPublicAccess: {
      blockPublicAcls: false,
      blockPublicPolicy: true,
      ignorePublicAcls: false,
      restrictPublicBuckets: true
    }
  }
};
```

```
new CloudFrontToS3(stack, 'test-cloudfront-s3', props);

expect(stack).toHaveResource("AWS::S3::Bucket", {
  PublicAccessBlockConfiguration: {
    BlockPublicAcls: false,
    BlockPublicPolicy: true,
    IgnorePublicAcls: false,
    RestrictPublicBuckets: true
  },
});
```

## プロパティのオーバーライドに関する警告

コアライブラリのデフォルトプロパティがユーザー提供のプロパティによってオーバーライドされると、Constructsは変更を強調表示する1つ以上の警告メッセージをコンソールに放出します。これらのメッセージは、ユーザーに状況認識を提供し、セキュリティ上のリスクを招く可能性のある意図しない上書きを防止することを目的としています。これらのメッセージは、デプロイメント/ビルド関連のコマンドが実行されるたびに表示されます。cdk deploy, cdk synth, npm test、など

メッセージの例: `AWS_CONSTRUCTS_WARNING: An override has been provided for the property: BillingMode. Default value: 'PAY_PER_REQUEST'. You provided: 'PROVISIONED'.`

### 上書き警告の切り替え

上書き警告メッセージはデフォルトで有効になっていますが、`overrideWarningsEnabled` シェル変数。

- 明示的な実行をオフにする警告をオーバーライドする、実行 `export overrideWarningsEnabled=false`。
- 明示的な実行を有効化警告をオーバーライドする、実行 `export overrideWarningsEnabled=true`。
- デフォルトに戻すには `unset overrideWarningsEnabled`。

## ドキュメントの改訂

AWS ソリューション構成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します。

update-history-change	update-history-description	update-history-date
<a href="#">更新された内容</a>	aws-lambda-ssmstring パラメータパターンが追加されました。その他のマイナーなコンテンツの更新	2021 年 5 月 27 日
<a href="#">更新された内容</a>	aws-lambda-secretsmanager パターンが追加されました。その他のマイナーなコンテンツの更新	2021 年 5 月 12 日
<a href="#">更新された内容</a>	*-lambda パターンを選択するためのプロパティの更新 その他のマイナーなコンテンツの更新	2021 年 4 月 17 日
<a href="#">更新された内容</a>	Python ユーザー向けウォークスルーの問題を修正し、Lambda 関数を含むコンストラクトのプロパティ例を更新しました。	2021 年 3 月 30 日
<a href="#">更新された内容</a>	パターン小道具と選択パターンのデフォルト設定に対するマイナーな修正/更新	2021 年 3 月 8 日
<a href="#">更新された内容</a>	ウォークスルーコンテンツのマイナーな修正/更新	2021 年 3 月 4 日
<a href="#">更新された内容</a>	「」を追加aws-lambda-sagemakerendpoint パターンと、選択した Kinesis	2021 年 2 月 24 日

	Firehose パターンのプロパティを更新しました。	
<a href="#">更新された内容</a>	「」を追加aws-kinesisstreams-gluejob パターン、および Python ユーザ用のウォークスルーステップを更新しました。	2021 年 2 月 17 日
<a href="#">更新された内容</a>	の更新されたプロパティaws-cloudfront-* パターン	2021 年 2 月 9 日
<a href="#">更新された内容</a>	パターンごとにGitHub へのリンクを追加しました。	2021 年 2 月 5 日
<a href="#">更新された内容</a>	選択パターンのプロパティを更新しました。	2021 年 2 月 1 日
<a href="#">更新された内容</a>	選択したパターンのプロパティとデフォルト設定のドキュメントが更新されました。	2021 年 1 月 4 日
<a href="#">更新された内容</a>	新しいパターン : aws-クラウドフロントメディアストアとaws-s3-sqsが追加されました。	2020 年 12 月 20 日
<a href="#">更新された内容</a>	aws-lambda-sagemakerパターンを削除しました。	2020 年 11 月 17 日
<a href="#">更新された内容</a>	aws-events-rule-kinesisstreams、aws-events-rule-kinesisfirehose-S3、aws-lambda-sagemakerという新しいパターンが追加されました。	2020 年 10 月 27 日

<a href="#">更新された内容</a>	aws-events-rule-sns および aws-events-rule-sqs パターンの変更を反映するように更新されました。クラス名とインターフェイス名はパスカル大文字小文字に変更されました。	2020 年 10 月 22 日
<a href="#">更新された内容</a>	aws-apigateway-sagemakerendpoint と aws-キネシスストリーム-キネシスファイアホース-3 パターンが追加されました。既存のコンテンツに対するその他のマイナーな更新。	2020 年 10 月 20 日
<a href="#">更新された内容</a>	aws-apigateway-iot パターンを追加しました。既存のコンテンツに対するその他のマイナーな更新。	2020 年 10 月 7 日
<a href="#">更新された内容</a>	最小限のデプロイ可能なパターンコードスニペットとすべてのパターンのベストプラクティスのデフォルトを更新しました。	2020 年 10 月 5 日
<a href="#">更新された内容</a>	aws-kinesisstreams-lambda パターンのプロパティが更新され、ブレークする変更を反映しました。	2020 年 9 月 14 日
<a href="#">更新された内容</a>	ウォークスルーの 2 番目の部分へのマイナーな修正	2020 年 9 月 10 日

<a href="#">更新された内容</a>	aws-apigateway-kinesistreams、aws-events-rule-sns、aws-events-rule-sqs パターンが追加されました。	2020 年 9 月 10 日
<a href="#">更新された内容</a>	aws-sns-sqs パターンを追加。すべての SNS パターンの更新。誤植修正。	2020 年 9 月 2 日
<a href="#">更新された内容</a>	aws-sqs-ラムダパターンのモジュール名が修正されました。	2020 年 8 月 31 日
<a href="#">更新された内容</a>	aws-dynamodb-stream-lambda-elasticsearch-kibana パターンの Python モジュール名が修正	2020 年 8 月 31 日
<a href="#">更新された内容</a>	Lambda パターンのデフォルトが更新され、その他のマイナーな更新が行われました。	2020 年 8 月 27 日
<a href="#">更新された内容</a>	S3 パターンのパブリックプロパティが更新されました。DynamoDB パターンのデフォルトが更新されました。	2020 年 8 月 10 日
<a href="#">更新された内容</a>	転送中の暗号化のデフォルトの適用を強調するために、複数のパターンを更新しました。	2020 年 8 月 4 日



<a href="#">更新された内容</a>	aws-lambda-sqs-lambda パターンの追加、入門ガイドの設定手順の改善、パブリックプロパティを通じて追加のリソースを使用できるようにすべてのパターンを更新しました。	2020 年 7 月 27 日
<a href="#">更新された内容</a>	aws-lambda-sqs パターンが追加されました。その他のマイナーアップデート。	2020 年 7 月 20 日
<a href="#">更新された内容</a>	関連するパターンから deployLambda プロパティと deployBucket プロパティを削除しました。その他のマイナーな	2020年7月9日
<a href="#">更新された内容</a>	aws-lambda-step-function パターンを追加し、小さな誤植を修正しました。	2020 年 7 月 7 日
<a href="#">更新された内容</a>	ExistingTableObj が追加されました。プロパティを使用して DynamoDB パターンを選択します。	2020 年 6 月 25 日
<a href="#">更新された内容</a>	いくつかのテキストの修正と壊れたリンクの修正。	2020 年 6 月 23 日
<a href="#">初回リリース</a>	AWS ソリューション構築物が一般公開されました。	2020 年 6 月 22 日

## Notices

お客様には、このドキュメントの情報を個別に評価していただく責任があります。本文書:(a) は情報提供のみを目的としており、(b) 現在の AWS 製品および慣行を表し、これらは予告なく変更される可能性があります。(c) AWS およびその関連会社、サプライヤー、ライセンサーからのコミットメントや保証は作成しません。AWS の製品またはサービスは、明示または黙示を問わず、いかなる種類の保証、表明、または条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は AWS 契約によって規定されています。また、本文書は、AWS とお客様との間の契約に属するものではなく、また、当該契約が本文書によって修正されることもありません。

© 2020 Amazon Web Services, Inc. or its a its All rights reserved.

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。