AWS 解決方案

AWS 解決方案建構

AWS 解決方案建構: AWS 解決方案

Table of Contents

概觀 1
什麼是 AWS 解決方案建構？ 1
為什麼要使用 AWS 解決方案建構？ 1
入門 2
必要條件 2
安裝 AWS CDK 3
使用 AWS 解決方案建構 3
穿越-第 1 部分 3
Hello 建構 4
建立應用程式目錄並初始化 AWS CDK 4
更新專案基礎相依性 5
Lambda 處理常式代碼 8
安裝 AWS CDK 和 AWS 解決方案建構相依性 9
將亞馬遜 API 网關 /AWS Lambda 模式添加到您的堆棧 10
cdk 部署 16
堆疊輸出 17
測試您的應用程式 17
穿越-第 2 部分 17
命中計數器 Lambda 程式碼 18
安裝新的相依性 20
定義資源 21
檢閱變更 34
cdk 部署 35
堆疊輸出 36
測試您的應用程式 36
範例使用案例 37
AWS 靜態 S3 網站 37
AWS 簡單的無伺服器影像處理常式 38
AWS 無伺服器 Web 應用程式 38
API 參考 39
模組 39
模組內容 39
Aw-每個人的動作模式 40
概觀 40

初始化 41
模式建立道具 41
模式性質 42
預設設定 43
架構 44
GitHub 44
AW-每個人-物聯網 44
概觀 45
初始化程式 45
模式建立道具 46
模式性質 47
預設設設設定 47
架構 49
範例 49
GitHub 51
AW-每個人的運動流 51
概觀 52
初始化 52
模式建構道具 53
模式性質 54
API API 用量範例 54
預設設設設設 55
架構 56
GitHub 56
AW-每個人的拉姆達 56
概觀 57
初始化 58
模式建立道具 58
模式性質 59
預設定 59
架構 60
GitHub 60
AW-每個地方-邏輯點 60
概觀 61
初始化 62
模式建立道具 62

模式性質 63
範例 API 使用 54
預設設定 64
架構 65
GitHub 65
AW-每個人的-數據 65
概觀 66
初始化 66
模式建立道具 67
模式性質 68
範例 API 使用 54
預設設定 69
架構 70
GitHub 70
雲端前面的每個角落 70
概觀 71
初始化 72
模式建立道具 72
模式性質 73
預設設定 73
架構 74
GitHub 74
aws-云前面的每個角度-lambda 75
概觀 75
初始化 76
模式建立道具 76
模式性質 77
預設定定 78
架構 79
GitHub 79
雲端前端媒體存放區 79
概觀 80
初始化程式 80
模式建立道具 81
模式性質 81
預設設定 82

架構 83
GitHub 83
雲端前端 3 83
概觀 84
初始化 84
模式建立道具 85
模式性質 85
預設設定 86
架構 87
GitHub 87
aw-認識-意大利-拉姆達 87
概觀 71
初始化 89
模式建立道具 90
模式性質 90
預設設定 91
架構 92
GitHub 93
aws-dynamodb-流-lambda 93
概觀 93
初始設定 94
模式建立道具 94
模式性質 95
Lambda 功能 95
預設設設定 95
架構 96
GitHub 97
aws-Dynamodb 流-羊肉-彈性搜索-基班納 97
概觀 97
初始化 98
模式建立道具 99
模式性質 100
Lambda 功能 100
預設定 100
架構 102
GitHub 102

AWS-事件規則-運動防火軟管-3 102
概觀 103
初始化 104
模式建立道具 104
模式性質 105
預設定 105
架構 107
GitHub 107
aws--事件-規則-運動流 107
概觀 108
初始化 108
模式建立道具 109
模式性質 109
預設設定 110
架構 110
GitHub 111
AW-事件規則-拉姆達 111
概觀 111
初始化 112
模式建立道具 112
模式性質 113
預設設定 113
架構 114
GitHub 114
AWS-事件-規則-SN 114
概觀 115
初始化 116
模式建立道具 116
模式性質 117
預設設定 117
架構 118
GitHub 118
aws--事件-規則- 118
概觀 119
初始化程式 120
模式建立道具 120

模式性質 121
預設設定 122
架構 123
GitHub 123
aws 事件規則步驟功能 123
概觀 124
初始化 125
模式建立道具 125
模式性質 125
預設設定 126
架構 127
GitHub 127
Aws-io-運動防火軟管-3 127
概觀 128
初始化 129
模式建立道具 129
模式性質 130
預設設定 130
架構 131
GitHub 132
大概-蘭姆達 132
概觀 132
初始化 133
模式建立道具 134
模式性質 134
預設設定 134
架構 135
GitHub 135
Aw-io-羊肉-動態學調節模式 136
概觀 136
初始化 137
模式建立道具 137
模式性質 138
預設設定 138
架構 140
GitHub 140

AW-運動防火軟管-3 140
概觀 141
初始化 141
模式建立道具 142
模式屬性 142
預設定 143
架構 144
GitHub 144
運動防火軟管-3 和運動分析 144
概觀 145
初始化 146
模式建立道具 146
模式性質 147
預設設定 148
架構 149
GitHub 149
運動流-糖果工作 149
概觀 150
初始化 151
模式建立道具 152
匯流數據存儲道具 153
匯流器類型 153
預設設定 154
架構 155
GitHub 155
AW-運動流-運動防火軟管-3 155
概觀 156
初始化 156
模式建立道具 157
模式性質 158
預設設定 158
架構 159
GitHub 160
aw-運動流-拉姆達 160
概觀 160
初始化 161

模式建立道具 161
模式性質 162
預設設定 163
架構 164
GitHub 164
aws-lambda-dynamodb 164
概觀 165
初始化 165
模式建立道具 166
模式性質 168
預設設設設定 168
架構 169
GitHub 170
aws-lambda-彈性搜索-基班納 170
概觀 170
初始化器 171
模式建立道具 172
模式性質 172
Lambda 功能 173
預設設定 173
架構 175
GitHub 175
aws-lambda-3 176
概觀 176
初始化 177
模式建立道具 177
模式性質 179
預設定 180
架構 180
GitHub 181
aws-lambda 字符串參數 181
概觀 181
初始化 182
模式建立道具 182
模式屬性 185
預設設定 186

架構 187
GitHub 187
aws-lambda-區域分析點 187
概觀 188
初始化 189
模式建立道具 189
模式屬性 192
預設設定 192
架構 193
GitHub 193
aws-lambda 秘密管理器 194
概觀 194
初始化 195
模式建立道具 195
模式性質 197
預設設設設定 198
架構 199
GitHub 199
aws-lambda- 199
概觀 200
初始化 200
模式建立道具 201
模式性質 203
預設設定 203
架構 204
GitHub 204
aws-lambda-資料 204
概觀 205
初始化 205
模式建立道具 206
模式屬性 208
預設設定 209
架構 210
GitHub 210
aw-拉姆達-Q-拉姆達 210
概觀 211

初始化 212
模式建立道具 212
模式屬性 214
預設定 214
架構 215
GitHub 215
aws-lambda 步驟函數 215
概觀 216
初始化 217
模式建立道具 217
模式性質 218
預設定 218
架構 219
GitHub 220
差異-3 蘭姆達 220
概觀 220
初始化 221
模式建立道具 221
模式性質 222
預設設定 222
架構 223
GitHub 224
AWS-3-數據庫 224
概觀 224
初始化 225
模式建立道具 225
模式性質 227
預設設定 227
架構 228
GitHub 228
AWS-3 步驟函數 228
概觀 229
初始化程式 230
模式建立道具 230
模式性質 231
預設設定 232

架構 233
GitHub 233
差不多-蘭姆達 233
概觀 234
初始化 234
模式建立道具 235
模式性質 235
預設設定 236
架構 236
GitHub 237
反射-反射-反射 237
概觀 237
初始化 238
模式建立道具 238
模式性質 240
預設設定 240
架構 241
GitHub 241
Aws-秒-蘭姆達 241
概觀 242
初始化 242
模式建立道具 243
模式性質 244
預設設定 244
架構 245
GitHub 245
core 245
AWS CDK 建構的預設屬性 246
覆寫預設屬性 246
屬性覆寫警告 247
文件修訂 248
注意 252
..... ccliii

AWS 解決方案建構

Pubdate Date：2021 年五月(文件修訂)

什麼是 AWS 解決方案建構？

AWS 解決方案建構 (建構) 是AWS Cloud Development Kit (AWS CDK)，提供多服務、架構良好的模式，以便在程式碼中快速定義解決方案，以建立可預測且可重複的基礎架構。目標是加速開發人員使用模式型定義為架構，建置任何大小的解決方案的體驗。

使用 AWS 解決方案建構，以熟悉的程式設計語言定義您的解決方案。AWS 解決方案建構目前支援 TypeScript、JavaScript、Python 和 Java。

若要瀏覽 AWS 解決方案建構模式的完整目錄，請點選此處。

為什麼要使用 AWS 解決方案建構？

隨著雲端供應商的創新速度，了解並瞭解最佳實務，並確保其在整個解決方案中正確實作可能會令人生畏。構造允許您結合預先構建的，架構良好的模式和使用案例，以可擴展和安全的方式使用雲服務執行常見的操作。由於建構提供現代程式設計語言的程式庫，因此您可以將現有的開發技能和熟悉的工具應用於為您的解決方案建置架構良好的雲端基礎結構的工作。

AWS 解決方案建構的其他優點包括：

- 它建置在 AWS Cloud Development Kit (AWS CDK) 開放原始碼軟體開發架構上。
- 定義解決方案基礎設施時，使用邏輯（if 語句，for 循環等）。
- 使用面向對象的技術來創建系統的模型。
- 定義高層次的抽象，共用它們，然後將它們發佈到您的團隊、公司或社群。
- 將您的解決方案組織成邏輯模組。
- 共用並重複使用您的解決方案做為資料庫。
- 使用業界標準通訊協定測試基礎架構程式碼。
- 使用您現有的程式碼檢閱工作流程。

AWS 解決方案建構的目的是在整合常見架構良好的模式以在 AWS 上實現解決方案目標時，降低所需的複雜性和膠水邏輯。

AWS 解決方案建構入門

本主題說明如何安裝和設定 AWS Cloud Development Kit (AWS CDK)、AWS 解決方案建構，以及如何使用 AWS 解決方案建構模式建立您的第一個 AWS CDK 應用程式。

> **Note**
>
> AWS CDK 版支援 AWS 解決方案建構。

> **Tip**
>
> 想要深入挖掘嗎？ 嘗試使用CDK 研討會，進行更深入的真實世界專案導覽。

> **Tip**
>
> 如需開始使用 AWS Cloud Development Kit (AWS CDK) 的詳細資訊，請參閱AWS CDK 開發人員指南。

Prerequisites

AWS 解決方案結構是建立在 AWS CDK 上的，因此您需要安裝 Node.js (> = 10.3.0) ，即使是那些使用 TypeScript 或 JavaScript 以外的語言工作。這是因為AWS CDK和 AWS 解決方案結構是在 TypeScript 開發的，並在 Node.js 上運行。其他支持語言的綁定使用這個後端和工具集。

您必須提供登入資料和 AWS 區域才能使用 AWS CDK CLI，如指定登入資料和區域所述。

其他必要條件取決於您的開發語言，如下所示。

語言	必要條件
 P	Python 3.6

語言	必要條件
 T t	TypeScript >= 2.7
 J	Java >= 1.8

安裝 AWS CDK

若要安裝和設定 AWS CDK，請參閱 AWS CDK 開發人員指南-安裝 AWS CDK。

使用 AWS 解決方案建構

使用 AWS 解決方案建構時建立新應用程式的典型工作流程採用與 AWS CDK 相同的方法。

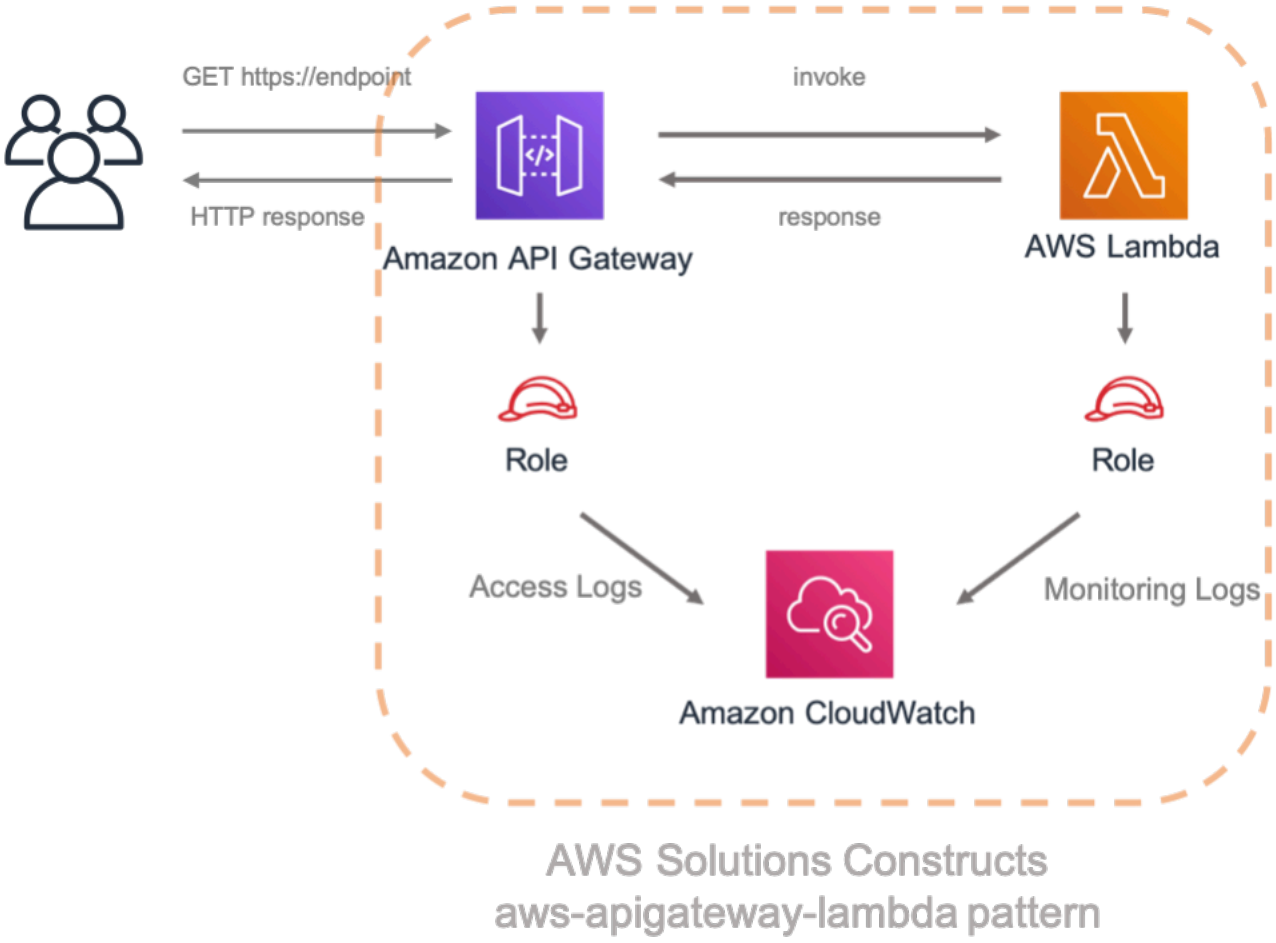
1. 建立應用程式目錄。
2. 初始化應用程式。
3. 新增 AWS 解決方案建構模式相依性。
4. 將其他程式碼新增至應用程式。
5. 視需要編譯應用程式。
6. 部署應用程式中定義的資源。
7. 測試應用程式。

如果有任何問題，循環修改，編譯（如有必要），部署和再次測試。

穿越-第 1 部分

> **Note**
>
> AWS CDK 版本和 1.46.0 支援 AWS 解決方案建構。

本教學將逐步引導您如何建立和部署使用 AWS 解決方案建構模式的簡單「Hello Protets」AWS CDK 應用程式，從初始化專案到部署產生的 AWS CloudFormation 範本。Hello 構造應用程序將創建以下簡單的解決方案：



Hello 建構

讓我們開始使用以模式為基礎的開發來建置我們的第一個 AWS CDK 應用程式。

> **Note**
>
> 這是一個範例修改`Hello CDK!`來自CDK 工作坊。如果這是您第一次使用 AWS CDK，我們建議您從本研討會開始實際操作逐步解說，以及如何利用 CDK 建置真實世界的專案。

建立應用程式目錄並初始化 AWS CDK

為您的 CDK 應用程式建立目錄，然後在該目錄中建立 AWS CDK 應用程式。

TypeScript

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

Python

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

> **Tip**
>
> 現在是在您最喜愛的 IDE 中開啟專案並探索的好時機。若要進一步了解專案結構，請選取適當的連結：
>
> - TypeScript
> - Python

更新專案基礎相依性

> **Warning**
>
> 為了確保功能正確，AWS 解決方案建構和 AWS CDK 套件必須在專案中使用相同的版本號碼。例如，如果您使用的是 AWS 解決方案建構 1.52.0 版，則也必須使用 AWS CDK 版 1.52.0 版。

> **Tip**
>
> 請注意 AWS 解決方案建構的最新版本，並將該版本號碼套用至VERSION_NUMBER預留位置 (適用於 AWS 解決方案建構和 AWS CDK 套件)。要檢查構造函數庫的所有公開版本，Click here。

TypeScript

編輯package.json檔案，並包含下列資訊：

```
  "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

編輯setup.py檔案，並包含下列資訊：

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

安裝項目基礎依賴關係。

TypeScript

```
npm install
```

Python

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

建置並執行應用程式，並確認它會建立空的堆疊。

TypeScript

```
npm run build
cdk synth
```

Python

```
cdk synth
```

你應該看到一個像下面這樣的堆棧，其中CDK-VERSION是 CDK 的版本。(您的輸出可能與此處顯示的內容略有不同。)

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 處理常式代碼

我們將從 AWS Lambda 處理常式程式碼開始。

建立目錄`lambda`在您的專案樹的根目錄中。

TypeScript

新增稱為的檔案`lambda/hello.js`，並包含：

```
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

新增稱為的檔案`lambda/hello.py`，並包含：

```
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'])
    }
```

這是一個簡單的 Lambda 函數，它返回文本「你好，構造！你已經點擊 [網址路徑]」。該函數的輸出還包括 HTTP 狀態碼和 HTTP 標頭。這些通過 API Gateway 用於制定 HTTP 響應給用戶。

這個 Lambda 寓式酒店在 JavaScript 中提供。如需使用您選擇的語言編寫 Lambda 函數的詳細資訊，請參閱AWS Lambda 文件。

安裝 AWS CDK 和 AWS 解決方案建構相依性

AWS 解決方案建構隨附廣泛的建構程式庫。該庫被分為模塊，每個架構良好的模式一個。例如，如果您想要將 Amazon API Gateway 休息 API 定義給 AWS Lambda 函數，我們需要使用`aws-apigateway-lambda`模式資源庫。

我們還需要從 AWS CDK 新增 AWS Lambda 和 Amazon API Gateway 建構程式庫。

將 AWS Lambda 模組及其所有相依性安裝到我們的專案中：

> **Note**
>
> 請記得將用於 AWS 解決方案構造和 AWS CDK 的正確相符版本替換為`VERSION_NUMBER`預留位置欄位。套件之間的版本不相符可能會導致錯誤。

TypeScript

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

Python

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

接下來，將 Amazon API Gateway 模組及其所有依賴項安裝到我們的專案中：

TypeScript

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

Python

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

最後，安裝 AWS 解決方案建構`aws-apigateway-lambda`模塊及其所有依賴關係到我們的項目中：

TypeScript

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

Python

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

將亞馬遜 API 网關 /AWS Lambda 模式添加到您的堆棧

現在，讓我們定義 AWS 解決方案建構模式，用於使用 AWS Lambda 代理實作 Amazon API Gateway。

TypeScript

編輯檔案`lib/hello-constructs.ts`，並包含：

```
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

編輯檔案hello_constructs/hello_constructs_stack.py，並包含：

```
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
                )
            )
        )
```

就是這樣 為了定義將所有請求代理到 AWS Lambda 函數的 API Gateway，您需要執行這項操作。讓我們將我們的新堆棧與原始堆棧進行比較：

TypeScript

```
npm run build
cdk diff
```

Python

```
cdk diff
```

輸出應如下所示：

```
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"},"/"]]}}
```

那很好 這個簡單的範例包含 AWS 解決方案建構中的一個架構良好的模式，為您的堆疊增加了 21 個新資源。

cdk 部署

> **Tip**
>
> 在部署包含 Lambda 函數的第一個 AWS CDK 應用程式之前，您必須先啟動 AWS 環境。這會建立一個臨時儲存貯體，AWS CDK 用來部署包含資產的堆疊。如果這是您第一次使用 AWS

CDK 部署資產，則需要執行`cdk bootstrap`，將 CDK 工具組堆疊部署到您的 AWS 環境中。

好，準備好部署嗎？

```
cdk deploy
```

堆疊輸出

部署完成後，您會注意到這一行：

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

這是由 AWS 解決方案建構模式自動新增的堆疊輸出，並包含 API Gateway 端點的 URL。

測試您的應用程式

讓我們嘗試用`curl`。複製 URL 並執行（您的前綴和地區可能不同）。

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

輸出應如下所示：

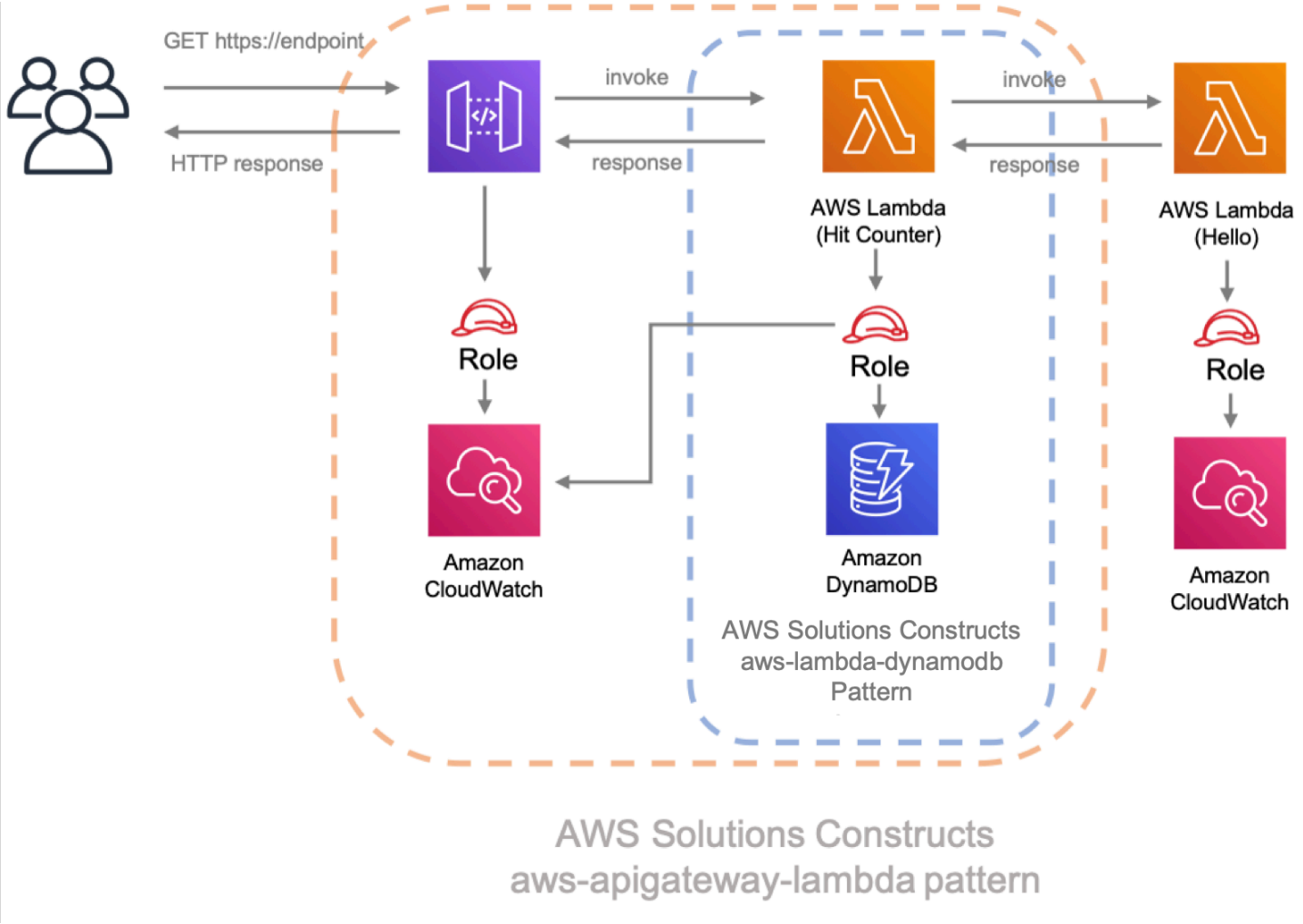
```
Hello, AWS Solutions Constructs! You've hit /
```

如果這是你收到的輸出，你的應用程序工作！

穿越-第 2 部分

> **Note**
>
> AWS CDK 版本和 1.46.0 支援 AWS 解決方案建構。

本教程將引導您完成如何修改中創建的「Hello 構造」應用程序第 1 部分。我們的修改將使用 AWS Lambda 新增網站點擊計數器到 AWS 解決方案建構中的 DynamoDB 模式。修改 Hello 建構應用程式會產生下列解決方案：



命中計數器 Lambda 程式碼

讓我們開始撰寫命中計數器 AWS Lambda 函數的程式碼。此函數將：

- 遞增與 Amazon DynamoDB 表格中 API 路徑相關的計數器，
- 調用下游 Hello AWS Lambda 函數，
- 並將響應返回給最終用戶。

TypeScript

新增稱為的檔案`lambda/hitcounter.js`，並包含：

```
  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

新增稱為的檔案`lambda/hitcounter.py`，並包含：

```
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)
```

安裝新的相依性

> **Note**
>
> 請記得將用於 AWS 解決方案構造和 AWS CDK 的正確相符版本替換為VERSION_NUMBER預留位置欄位。這應該與本逐步解說的第一個部分中用於相依性的版本號碼相同。套件之間的版本不符可能會導致錯誤。

像往常一樣，我們首先需要安裝解決方案更新所需的相依性。首先，我們需要安裝 DynamoDB 建構程式庫：

TypeScript

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

Python

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

最後，安裝 AWS 解決方案建構aws-lambda-dynamodb模塊及其所有依賴關係到我們的項目中：

TypeScript

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

Python

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

定義資源

現在，讓我們更新堆疊程式碼，以容納我們的新架構。

首先，我們要導入我們的新依賴關係，並將「Hello」函數移到aws-apigateway-lambda模式，我們在第 1 部分中創建。

TypeScript

編輯檔案lib/hello-constructs.ts，並包含：

```
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);
  }
}
```

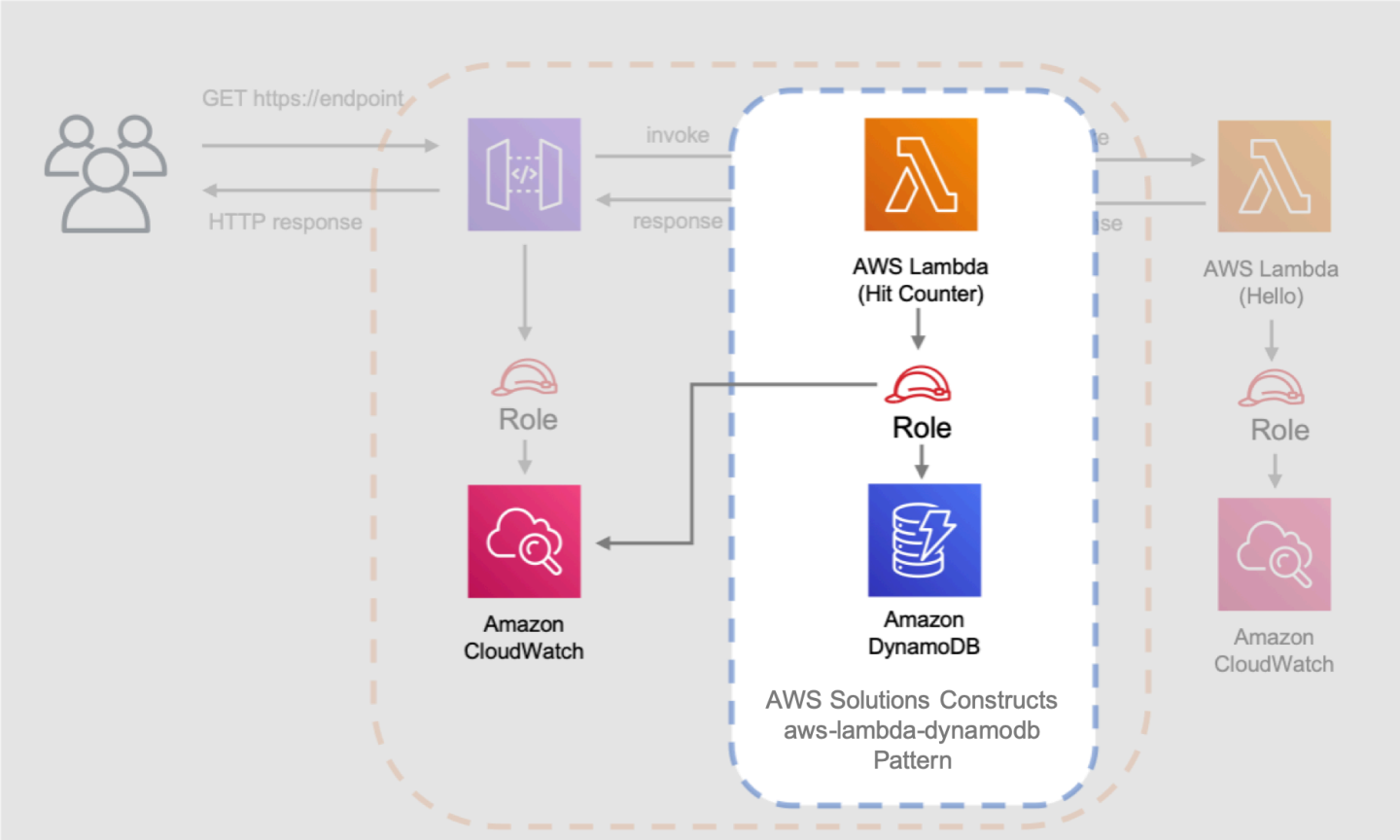
Python

編輯檔案hello_constructs/hello_constructs_stack.py，並包含：

```
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
                )
            )
        )
```

接下來，我們將添加aws-lambda-dynamodb模式，為我們更新的架構建立命中計數器服務。



AWS Solutions Constructs
aws-apigateway-lambda pattern

下面的下一個更新定義了`aws-lambda-dynamodb`模式，方法是使用命中計數器處理常式定義 AWS Lambda 函數。此外，Amazon DynamoDB 表格定義的名稱為`Hits`和分區索引鍵的`path`。

TypeScript

編輯檔案`lib/hello-constructs.ts`，並包含：

```
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

編輯檔案hello_constructs/hello_constructs_stack.py，並包含：

```
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
                )
            )
        )
```

接著，我們需要授予從創建的命中計數器函數aws-lambda-dynamodb模式添加上述權限來調用我們的 Hello 函數。

TypeScript

編輯檔案lib/hello-constructs.ts，並包含：

```
  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

編輯檔案hello_constructs/hello_constructs_stack.py，並包含：

```
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，我們需要更新我們的原始aws-apigateway-lambda模式來利用我們新的命中計數器函數，該函數配置為aws-lambda-dynamodb模式。

TypeScript

編輯檔案lib/hello-constructs.ts，並包含：

```
  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

編輯檔案hello_constructs/hello_constructs_stack.py，並包含：

```
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
                )
            )
        )
```

檢閱變更

讓我們構建我們的項目，並檢閱我們部署這個時會發生的資源變化：

```
npm run build
cdk diff
```

我們的輸出應如下所示：

```
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 部署

準備好部署了嗎？

```
cdk deploy
```

堆疊輸出

部署完成後，您會注意到這一行：

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

測試您的應用程式

讓我們嘗試用捲曲來打這個端點。複製 URL 並執行（您的前綴和地區可能不同）。

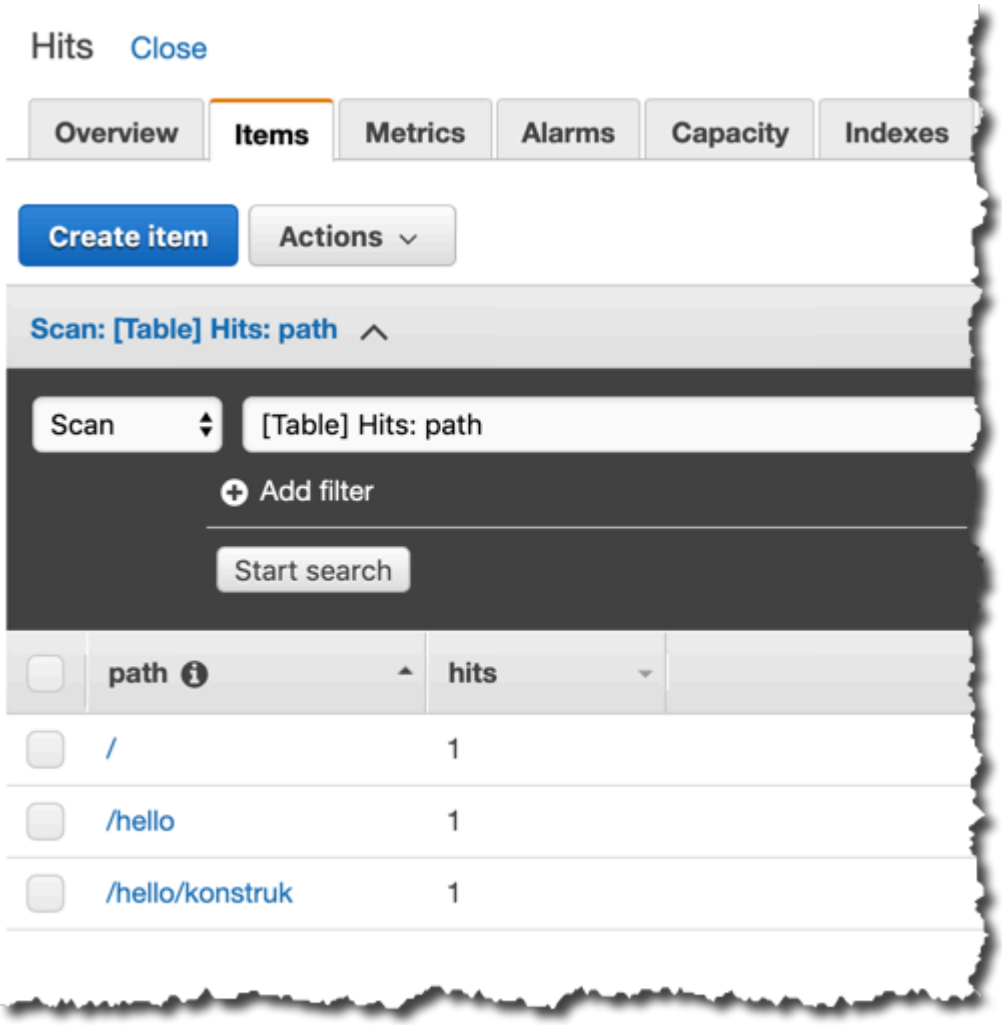
```
curl https://xxxxxxxxxx.execute-api.us-east-1.amazonaws.com/prod/
```

輸出應如下所示：

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

現在，讓我們檢閱HitsAmazon DynamoDB 資料表。

1. 前往 DynamoDB 主控台。
2. 請確認您位於您建立資料表的區域。
3. 選擇資料表，然後選取命中資料表。
4. 打開表格並選擇「項目」。
5. 你應該看到你得到了多少點擊每條路徑。



6. 嘗試點擊一個新的路徑並刷新項目視圖。您應該會看到一個新的項目，並包含：`hits`數一。

如果這是你收到的輸出，你的應用程序工作！

範例使用案例

這個庫包括功能用例實現的集合來演示構造體系結構模式的使用。這些可以用與建築模式相同的方式使用，並且可以概念化為這些模式的額外「更高級別」抽象。以下使用案例作為功能範例提供：

AWS 靜態 S3 網站

這個用例模式（`aws-s3-static-website`）實作 Amazon CloudFront 分發、Amazon S3 儲存貯體和以 AWS LambDA 為基礎的自訂資源，以複製 Wild Rydes 示範網站的靜態網站內容 (`aws-serverless-web-app`實作)。

源代碼（aws-s3 靜態網站）

https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use_cases/aws-s3-static-website

AWS 簡單的無伺服器影像處理常式

這個用例模式（`aws-serverless-image-handler`）實作 Amazon CloudFront 分發、Amazon API 閘道 REST API、AWS Lambda 函數，以及佈建功能影像處理常式 API 的必要權限/邏輯，以便從部署帳戶內的一個或多個 Amazon S3 儲存貯體提供映像內容。

源代碼（aws-無服務器-圖像處理程序）

https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use_cases/aws-serverless-image-handler

AWS 無伺服器 Web 應用程式

這個用例模式（`aws-serverless-web-app`）實作簡單的無伺服器 Web 應用程式，可讓使用者向 Wild Rydes 艦隊要求獨角獸騎乘。該應用程序將向用戶提供一個基於 HTML 的用戶界面，用於指示他們想要被拾取的位置，並將在後端與 RESTful Web 服務接口以提交請求並分派附近的獨角獸。此外，應用程式也會提供設施，讓使用者在預約乘車前登入服務。

源代碼（aws 無服務器-網絡應用程序）

https://github.com/awslabs/aws-solutions-constructs/tree/master/source/use_cases/aws-serverless-web-app

API 參考

AWS 解決方案建構 (建構) 是 AWS Cloud Development Kit (AWS CDK) 的開放原始碼延伸，提供多服務、架構良好的模式，以便在程式碼中快速定義解決方案，以建立可預測且可重複的基礎設施。構造的目標是加速開發人員使用基於模式的定義為他們的架構，構建任何規模的解決方案的體驗。

在構造中定義的模式是 AWS CDK 構造的高層次多服務抽象，這些構造具有基於架構良好的最佳實務的預設組態。該庫被組織成使用面向對象的技術來創建每個架構模式模型邏輯模塊。

CDK 提供以下語言：

- JavaScript，TypeScript（Node.js 10.3.0）
- 蟒蛇（蟒蛇和 3.6）
- 爪哇 (Java 及 1.8)

Modules

AWS 解決方案建構分為數個模組。他們是這樣命名的：

- aws-xxx：適用於指定服務的架構良好的模式套件。此套件將包含包含多個 AWS CDK 服務模組的建構，以設定指定模式。
- xxx：無法啟動的套件」aws-」是構造核心模塊，用於為模式庫中使用的服務配置最佳實踐默認值。

模組內容

模組包含下列類型：

- 模式-這個庫中的所有更高級別的多服務結構。
- 其他類型-存在以支持模式的所有非構造類，接口，結構和枚舉。

模式在其構造函數中採用一組（輸入）屬性; 可以在模式的文檔頁面上看到屬性集（以及需要哪些屬性）。

該模式的文檔頁面還列出了可用的方法來調用，以及可用於在模式已實例化後檢索有關模式的信息的屬性。

Aw-每個人的動作模式

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_apigateway_dynamodb`
 打字稿	`@aws-solutions-constructs/aws-apigateway-dynamodb`
 Java	`software.amazon.awsconstructs.services.apigatewaydynamodb`

Overview

這個 AWS 解決方案建構實作了一個連接到 Amazon DynamoDB 表格的 Amazon API Gateway REST API。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案ApiGatewayToDynamoDBProps

模式建立道具

名稱	類型	描述
Dynamote 表面道具	dynamodb.TableProps	選擇性的使用者提供的道具，用於覆寫 DynamoDB 表格的預設道具
阿比格特威道具？	api.RestApiProps	選用的使用者提供的道具，可覆寫 API Gateway 的預設道具。
允許作業	boolean	是否要在 DynamoDB 表格上部署建立作業的 API Gateway 方法。
建立請求範本	string	建立方法的 API Gateway 要求範本，如果允許作業設定為 True，則需要
允許讀取作業	boolean	是否要在 DynamoDB 表格上部署讀取作業的 API Gateway 方法。

名稱	類型	描述
允許更新作業	`boolean`	是否要在 DynamoDB 表上部署更新作業的 API Gateway 方法。
更新請求範本	`string`	更新方法的 API Gateway 要求範本，如果允許更新作業設定為 True，則需要
允許刪除作業	`boolean`	是否要在 DynamoDB 表上部署用於刪除操作的 API Gateway 方法。
記錄群組道具？	`logs.LogGroupProps`	使用者提供的選用道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
ApeGateway	`api.RestApi`	返回由模式創建的 API Gateway REST API 的實例。
雲端觀察角色	`iam.Role`	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。
應用路徑記錄群組	`logs.LogGroup`	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。
角色	`iam.Role`	傳回由 API Gateway REST API 模式所建立的 IAM 角色執行個體。

名稱	類型	描述
DynaMotion 表格	dynamodb.Table	傳回由樣式建立之 DynamoDB 表格的實體。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

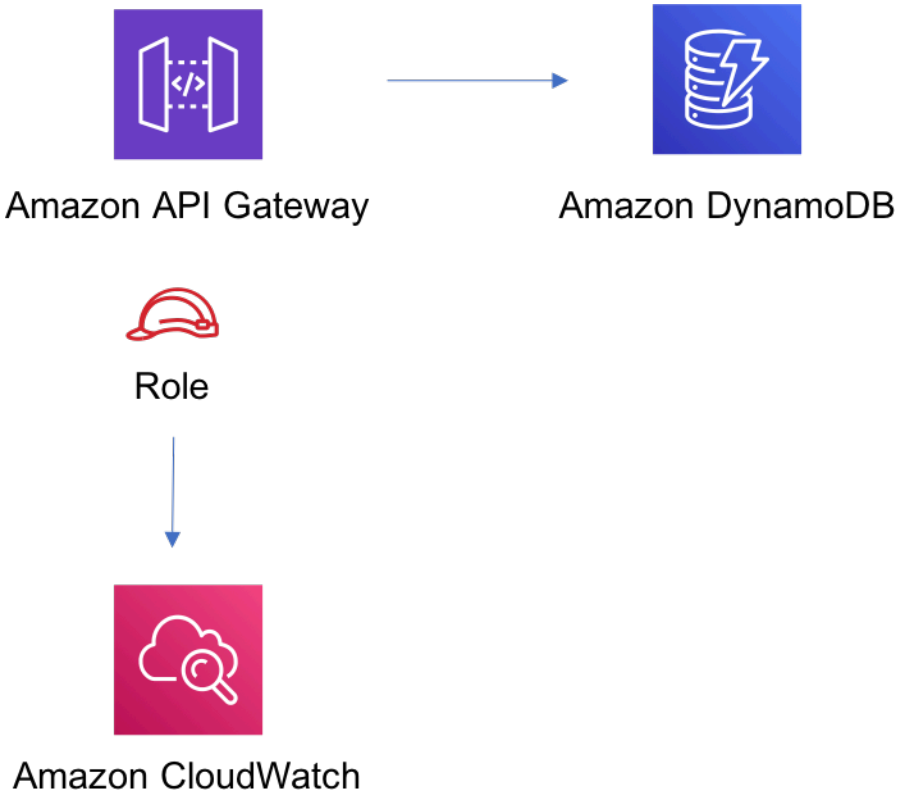
Amazon API Gateway

- 部署邊緣最佳化的 API 端點
- 啟用適用於 API Gateway 的 CloudWatch 日誌
- 設定 API Gateway 的最低權限存取 IAM 角色
- 將所有 API 方法的預設授權類型設定為 IAM
- 啟用 X-Ray 追蹤

Amazon DynamoDB 表

- 將 DynamoDB 表格的計費模式設定為隨選 (按請求付費)
- 使用 AWS 受管的 KMS 金鑰啟用 DynamoDB 表格的伺服器端加密
- 為 DynamoDB 表格建立名為「id」的分割區索引鍵
- 刪除 CloudFormation 堆棧時保留表
- 啟用持續備份和時間點復原

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/操作方式-動態模式

AW-每個人-物聯網

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_apigateway_iot`
打字稿	`@aws-solutions-constructs/aws-apigateway-iot`
Java	`software.amazon.awsconstructs.services.apigatewayiot`

Overview

此 AWS 解決方案建構實作連接到 AWS IoT 模式的 Amazon API Gateway REST API。

此建構會在 API Gateway 和 AWS IoT 之間建立可擴充的 HTTPS 代理。當您想要允許不支援 MQTT 或 MQT/WebSocket 通訊協定的舊式裝置與 AWS IoT 平台進行互動時，此功能非常有用。

此實作可讓您在指定的 MQTT 主題上發佈唯寫訊息，並支援 HTTPS 裝置的陰影更新，以允許裝置登錄中的項目。它不涉及 Lambda 函數來代理訊息，而是仰賴 AWS IoT 整合的直接 API Gateway，該閘道同時支援 JSON 訊息和二進位訊息。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案ApiGatewayToIotProps

模式建立道具

名稱	類型	描述
定位點	string	用於將 API Gateway 與其整合的 AWS IoT 端點子網域 (例如：A1234567890123)。
我的創建方式鑰匙？	boolean	如果設為true，則會建立 API 金鑰，並與使用方案相關聯。用戶應該在訪問 RestApi 時指定「X-api 鍵」頭。預設值設定為false。
應用程式執行角色？	iam.Role	API Gateway 用來存取 AWS IoT 的 IAM 角色。如果未指定，則會以萬用字元 ('*') 存取所有主題和項目建立預設角色。
阿比格特威道具？	api.restApiProps	選用的使用者提供的道具，可覆寫 API Gateway REST API 的預設道具。
記錄群組道具？	logs.LogGroupProps	使用者提供的選用道具，以覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
ApeGateway	api.RestApi	返回由模式創建的 API Gateway REST API 的實例。
雲端觀察角色	iam.Role	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。
應用路徑記錄群組	logs.LogGroup	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。
角色	iam.Role	傳回由 API Gateway REST API 模式所建立的 IAM 角色執行個體。

預設設設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon API Gateway

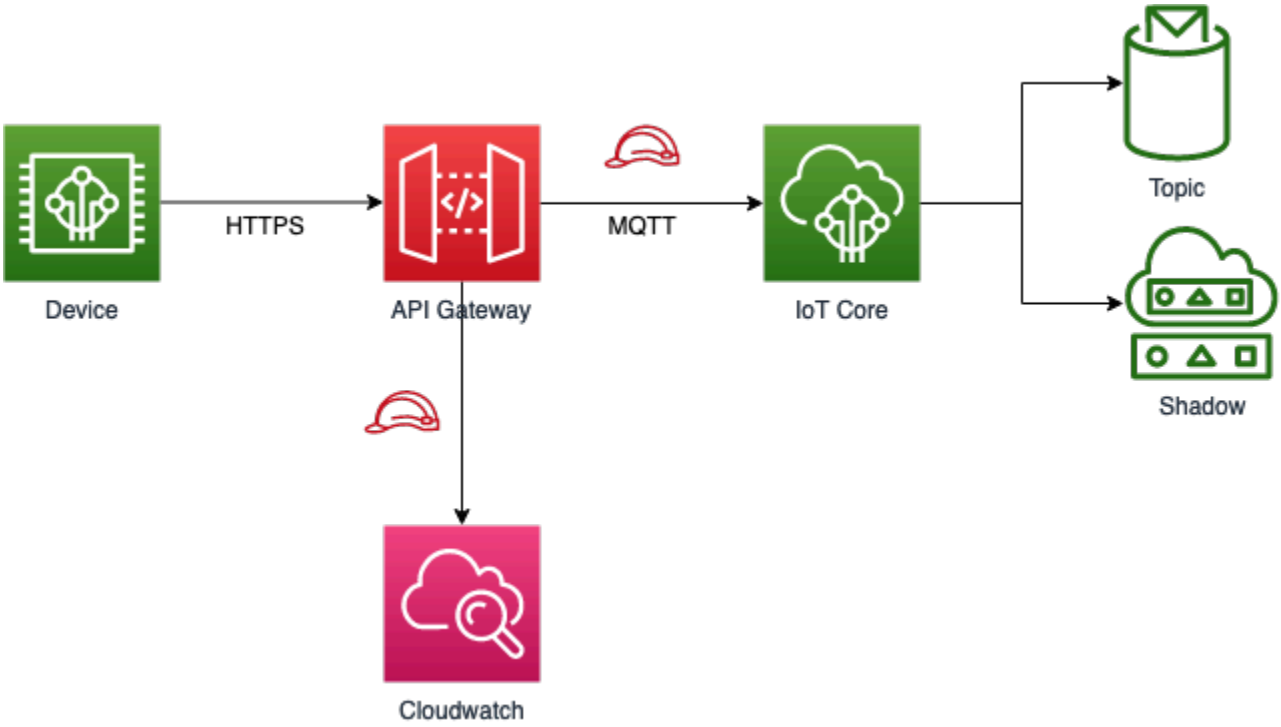
- 部署邊緣最佳化的 API 端點
- 建立 API 資源POST發佈訊息至 IoT 主題的方法
- 建立 API 資源POST發佈訊息至的方法ThingShadow和NamedShadows
- 啟用 API Gateway 的 CloudWatch 日誌
- 針對 API Gateway 設定 IAM 角色，可存取所有主題和內容
- 將所有 API 方法的預設授權類型設定為 IAM
- 啟用 X-Ray 追蹤
- 創建一個 UsagePlan，並關聯到prodstage

以下是部署構造後 API Gateway 公開的不同資源和方法的描述。請參閱範例部分，以取得如何使用輕鬆測試這些端點curl。

方法	資源	查詢參數	傳回代碼	描述
POST	/message/<topics>	qos	200/403/500	通過調用這個端點，你需要傳遞你想發布的主題（例如/message/device/foo`）。
POST	/shadow/<thingName>	無	200/403/500	這條路線允許更新一個事物的陰影文檔，考慮到其thingName 使用未命名（經典）影子類型。身體應符合標準的影子結構，包括state節點和關聯desired和reported節點。請參閱更新裝置影子區段以取得範例。
POST	/shadow/<thingName>/<shadowName>	無	200/403/500	這條路線允許更新一個事物的命名陰影文檔，因為它的thingName 與shadowName 使用命名陰影類型。身體應符合標準的影子結構，包括state節點和關

方法	資源	查詢參數	傳回代碼	描述
				聯desired和reported節點。請參閱更新影子區段以取得範例。

Architecture



Examples

以下範例僅適用於API_KEY身份驗證類型，因為 IAM 授權也需要指定 Sigv4 令牌，請確保apiGatewayCreateApiKey屬性設置為true，否則下面的示例將不起作用。

發佈訊息

您可以使用curl，使用 HTTPS API 在不同的 MQTT 主題上發佈訊息。下面的例子將發布一條消息device/foo主題。

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

請注意：取代stage-id、region，以及api-key參數與您的部署值。

您可以鏈結 URL 中的主題名稱，API 最多可接受 7 個您可以在其上發佈的子主題。例如，下列範例發佈訊息至主題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"}'
```

更新裝置影子

若要更新與指定事物相關聯的陰影文件，您可以使用事物名稱發出陰影狀態請求。請參閱如何更新物件影子下列範例。

```
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" }}}'
```

更新影子

若要更新與指定事物的命名陰影相關聯的陰影文件，您可以使用事物名稱和陰影名稱發出陰影狀態要求。請參閱下面的例子關於如何更新命名的陰影。

```
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" }}}'
```

傳遞二進位承載

可以將二進位承載傳送至代理 API，向下傳送至 AWS IoT 服務。在下列範例中，我們發送的內容`README.md`檔案 (視為二進位資料)`device/foo`主題使用`application/octet-stream`內容類型。

```
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
```

請注意：在此項目的目錄中執行此命令。然後，您可以測試從您的文件系統發送其他類型的二進制文件。

GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/AW-應用-物聯網

AW-每個人的運動流

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
	`aws_solutions_constructs.aws_apigateway_kinesisstreams`

語言	套件
Python	
打字稿	`@aws-solutions-constructs/aws-apigateway-kinesisstreams`
Java	`software.amazon.awsconstructs.services.apigatewaykinesisstreams`

Overview

此模式實作連接到 Amazon Kinesis 資料流的 Amazon API 閘道 REST API。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scope`Construct`
- id`string`
- 提案`ApiGatewayToKinesisStreamsProps`

模式建構道具

名稱	類型	描述
阿比格特威道具？	api.RestApiProps	選用的使用者提供的道具，可覆寫 API Gateway REST API 的預設道具。
記錄請求模板？	string	傳 PutRecord 動作的 API Gateway 要求範本。如果沒有提供，將使用默認的。
記錄請求模型？	api.ModelOptions	傳 PutRecord 動作的 API Gateway 要求模型。如果沒有提供，則會建立預設的。
輸入記錄請求模板？	string	傳 PutRecords 動作的 API Gateway 要求範本。如果沒有提供，將使用默認的。
記錄請求模型？	api.ModelOptions	傳 PutRecords 動作的 API Gateway 要求模型。如果沒有提供，則會建立預設的。
現在的斯特拉莫比？	kinesis.Stream	Kinesis 流的現有實例，提供這個和kinesisStreamProps 將導致錯誤。
運動流道具？	kinesis.StreamProps	選用的使用者提供的道具，可覆寫 Kinesis 串流的預設道具。
記錄群組道具？	logs.LogGroupProps	使用者提供的選用道具，可覆寫 CloudWatch Logs Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
AigGateway	api.RestApi	返回由模式創建的 API Gateway REST API 的實例。
角色	iam.Role	傳回由 API Gateway REST API 模式所建立的 IAM 角色執行個體。
雲端觀察角色	iam.Role	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。
應用路徑記錄群組	logs.LogGroup	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。
KinesisStream	kinesis.Stream	返回由模式創建的 Kinesis 流的實例。

API API 用量範例

方法	請求路徑。	請求內文	佇列動作	描述
POST	/record	{ "data": "Hello World!", "partitionKey": "pk001" }	kinesis:PutRecord	將單一資料記錄寫入串流。

方法	請求路徑。	請求內文	佇列動作	描述
POST	/records	{ "records": [{ "data": "abc", "partitionKey": "pk001" }, { "data": "xyz", "partitionKey": "pk001" }] }	kinesis:PutRecords	在單一呼叫中將多個資料記錄寫入串流。

預設設設設設

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

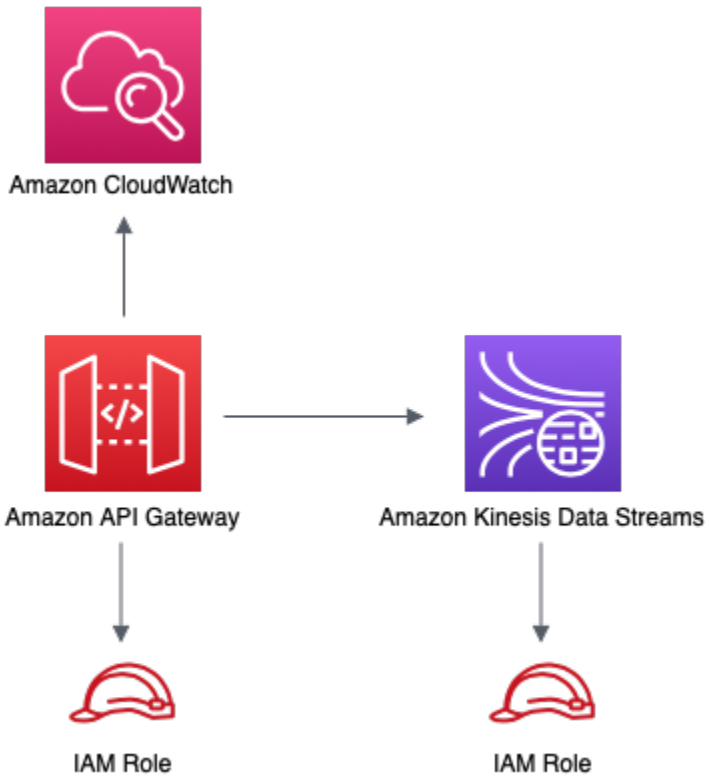
Amazon API Gateway

- 部署邊緣最佳化的 API 端點。
- 啟用 API Gateway 的 CloudWatch 記錄。
- 設定 API Gateway 的最低權限存取 IAM 角色。
- 將所有 API 方法的預設授權類型設定為 IAM。
- 啟用 X-Ray 追蹤。
- 在將資料傳遞給 Kinesis 之前驗證要求主體。

Amazon Kinesis Data Stream

- 設定 Kinesis 串流的最低權限存取 IAM 角色。
- 使用 AWS 受管 KMS 金鑰啟用 Kinesis 串流的伺服器端加密。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/操作方式-運動流

AW-每個人的拉姆達

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_apigateway_lambda`
 打字稿	`@aws-solutions-constructs/aws-apigateway-lambda`
 Java	`software.amazon.awsconstructs.services.apigatewaylambda`

Overview

此 AWS 解決方案建構實作連接到 AWS Lambda 函數的 Amazon API Gateway REST API。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案ApiGatewayToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會造成錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性，可覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
阿比格特威道具？	api.LambdaRestApiProps	可選的使用者提供的道具來覆寫 API 的預設道具。
記錄群組道具？	logs.LogGroupProps	可選用的使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
雲端觀察角色	iam.Role	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。
應用路徑記錄群組	logs.LogGroup	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。
LambdaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。
AgeGateway	api.LambdaRestApi	返回由模式創建的 API Gateway REST API 的實例。

預設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon API Gateway

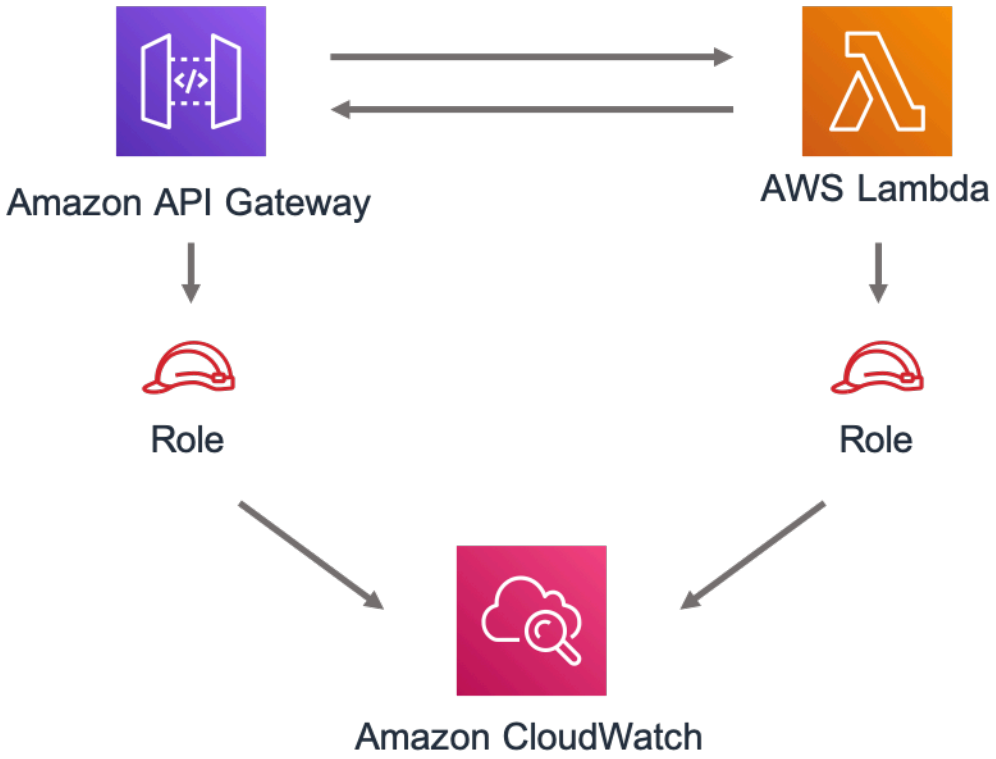
- 部署邊緣最佳化的 API 端點
- 啟用適用於 API Gateway 的 CloudWatch 日誌
- 設定 API Gateway 的最低權限存取 IAM 角色
- 將所有 API 方法的預設授權類型設定為 IAM
- 啟用 X-Ray 追蹤
- 設定環境變數：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（對於節點 10.x 和更高版本的函數）

AWS Lambda 功能

- 針對 Lambda 函數設定有限的權限存取 IAM 角色

- 針對 NodeJS Lambda 函數啟用具有持續作用的重複使用連線
- 啟用 X-Ray 追蹤

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/AW-每個人-lambda

AW-每個地方-邏輯點

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_apigateway_sagemakerendpoint`
 TypeScript	`@aws-solutions-constructs/aws-apigateway-sagemakerendpoint`
 Java	`software.amazon.awsconstructs.services.apigatewaysagemakerendpoint`

Overview

這個 AWS 解決方案建構實作了一個連接到亞馬遜 SagMaker 終端節點的 Amazon API Gateway REST API。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案ApiGatewayToSageMakerEndpointProps

模式建立道具

名稱	類型	描述
阿比格特威道具？	api.RestApiProps	選用的使用者提供的道具，可覆寫 API Gateway REST API 的預設道具。
應用程式執行角色？	iam.Role	API Gateway 使用的 IAM 角色叫用 SageMaker 端點。如果未指定，則會建立預設角色，並存取endpointName 。
Endpoint Name	string	已部署的 SageMaker 推論端點的名稱。

名稱	類型	描述
ResourcenName？	`string`	可選的資源名稱，其中 GET 方法將可用。
resourcePath	`string`	GET 方法的資源路徑。這裡定義的變量可以在`requestMappingTemplate` 。
請求對映範本	`string`	對應範本，將 REST API 上接收的 GET 要求轉換為 SageMaker 端點所預期的 POST 要求。
ResponseMappingTemplate？	`string`	選用的對應範本，可轉換從 SageMaker 端點接收到的回應。
記錄群組道具？	`logs.LogGroupProps`	可選的使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
ApeGateway	`api.LambdaRestApi`	返回由模式創建的 API Gateway REST API 的實例。
應用方式角色	`iam.Role`	傳回由 API Gateway REST API 模式所建立的 IAM 角色執行個體。
雲端觀察角色	`iam.Role`	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。

名稱	類型	描述
應用路徑記錄群組	logs.LogGroup	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。

範例 API 使用

請注意： 每個 SageMaker 端點都是唯一的，API 的回應將取決於部署的模型。下面給出的例子假定從樣本這篇博客文章。有關如何實施的參考，請參閱智能方式分配點覆蓋 .ts。

方法	請求路徑	查詢字串	SageMaker 動作	描述
GET	/321	items=101,131,162	sagemaker:InvokeEndpoint	擷取特定使用者和項目的預測。

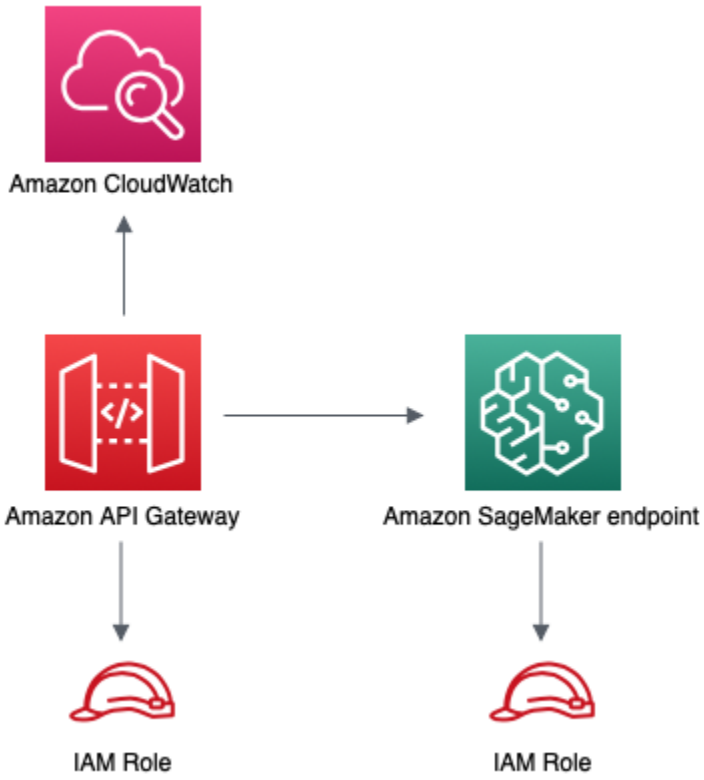
預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon API Gateway

- 部署邊緣最佳化的 API 端點
- 啟用適用於 API Gateway 的 CloudWatch 日誌
- 設定 API Gateway 的最低權限存取 IAM 角色
- 將所有 API 方法的預設授權類型設定為 IAM
- 啟用 X-Ray 追蹤
- 在將數據傳遞給 SageMaker 之前驗證請求參數

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/AW-應用-邏輯點

AW-每個人的-數據

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_apigateway_sqs`
打字稿	`@aws-solutions-constructs/aws-apigateway-sqs`
Java	`software.amazon.awsconstructs.services.apigatewaysqs`

Overview

此 AWS 解決方案建構實作連接到 Amazon SQS 佇列的 Amazon API 閘道 REST API。

下面是 TypeScript 中的最小可部署模式定義：

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

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

Initializer

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

參數

- scope`Construct`

- idstring
- 提案ApiGatewayToSqsProps

模式建立道具

名稱	類型	描述
阿比格特威道具？	api.RestApiProps	可選的使用者提供的道具來覆寫 API Gateway 的預設道具。
佇列道具？	sqs.QueueProps	可選的使用者提供的道具來覆寫佇列的預設道具。
部署死亡佇列？	boolean	是否部署輔助佇列做為無效字母佇列。預設為 true。
maxReceiveCount	number	訊息移到無效字母佇列之前，需交付到無效字母佇列的次數。
是否允許作業？	boolean	是否部署在隊列上創建操作的 API Gateway 方法（即 SQL：發送消息）。
建立請求範本？	string	覆寫建立方法的預設 API Gateway 要求範本，如果allowCreateOperation 已設定為true。
允許操作？	boolean	是否部署在隊列上讀取操作的 API Gateway 方法（即 SQL：接收消息）。
讀取要求樣板？	string	覆寫讀取方法的預設 API Gateway 要求範本，如果allowReadOperation 已設定為true。

名稱	類型	描述
是否允許刪除作業？	`boolean`	是否部署在隊列上刪除操作的 API Gateway 方法（即 SQL：刪除）。
刪除請求範本？	`string`	覆寫刪除方法的預設 API Gateway 要求範本，如果`allowDeleteOperation` 已設定為`true`。
記錄群組道具？	`logs.LogGroupProps`	可選的使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
AigGateway	`api.RestApi`	返回由模式創建的 API Gateway REST API 的實例。
雲端觀察角色	`iam.Role`	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。
應用路徑記錄群組	`logs.LogGroup`	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。
應用方式角色	`iam.Role`	傳回由 API Gateway REST API 模式所建立的 IAM 角色執行個體。
死機隊列？	`sqs.Queue`	返回由模式創建的死信隊列的實例，如果一個被部署。

名稱	類型	描述
平方	sqs.Queue	返回由模式創建的 SQS 隊列的實例。

範例 API 使用

方法	請求路徑。	請求內文	佇列動作	描述
GET	/		sqs::ReceiveMessage	從佇列擷取訊息。
POST	/	{ "data": "Hello World!" }	sqs::SendMessage	將訊息傳遞到佇列。
DELETE	/message?receiptHandle=[value]		sqs::DeleteMessage	從佇列刪除指定的訊息

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon API Gateway

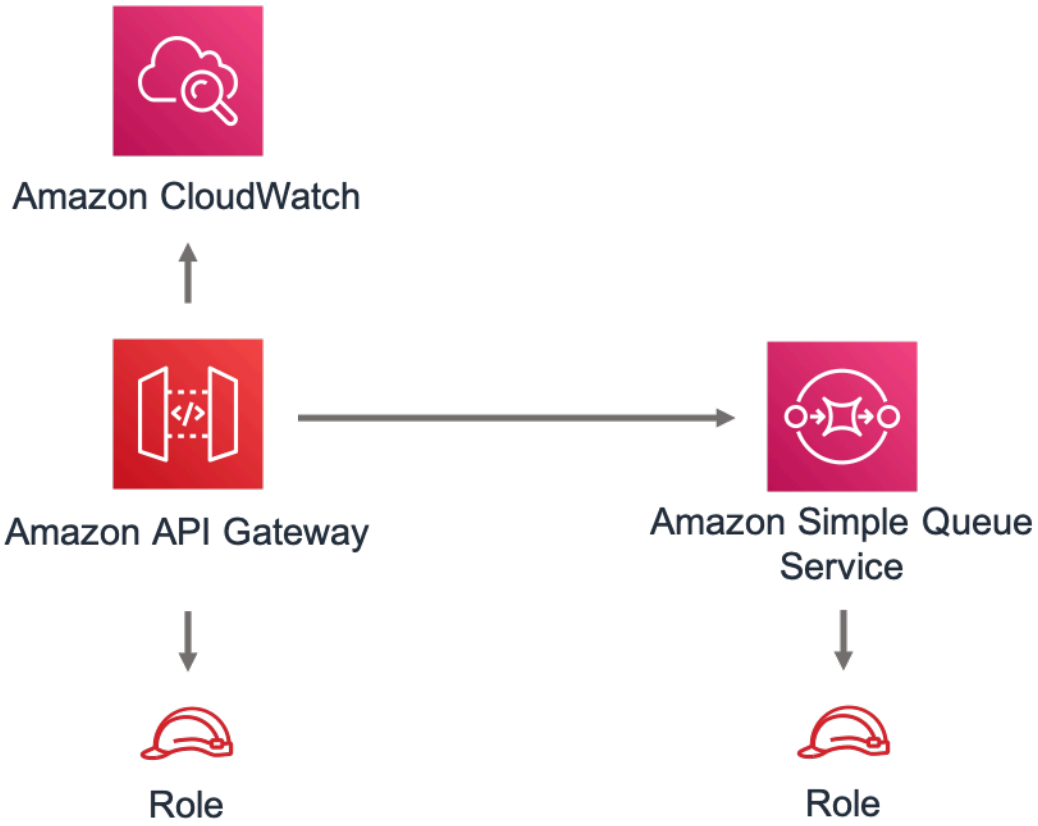
- 部署邊緣最佳化的 API 端點
- 為 API Gateway 啟用 CloudWatch 日誌
- 設定 API Gateway 的最低權限存取 IAM 角色
- 將所有 API 方法的預設授權類型設定為 IAM
- 啟用 X-Ray 追蹤

Amazon SQS 佇列

- 為來源 SQS 佇列建立 SQS 無效字母佇列

- 使用 AWS 受管 KMS Key 為來源 SQS 佇列啟用伺服器端加密
- 強制加密傳輸中的資料

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/操作-應用程序

雲端前面的每個角落

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_cloudfront_apigateway`
 打字稿	`@aws-solutions-constructs/aws-cloudfront-apigateway`
 Java	`software.amazon.awsconstructs.services.cloudfrontapigateway`

Overview

這個 AWS 解決方案構建在 Amazon API Gateway REST API 前面實作了一個 Amazon CloudFront 分發。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案CloudFrontToApiGatewayProps

模式建立道具

名稱	類型	描述
現有的阿比格特瓦尤比	api.RestApi	將使用 CloudFront 前端的區域 API Gateway

名稱	類型	描述
雲端發佈道具？	cloudfront.DistributionProps	可選的使用者提供的道具，以覆寫 CloudFront 分發的預設道具。
插入安全性標頭？	boolean	可選的用戶提供的道具，用於在 CloudFront 的所有響應中打開/關閉最佳實踐 HTTP 安全標頭的自動注入

模式性質

名稱	類型	描述
ApeGateway	api.RestApi	返回由模式創建的 API Gateway REST API 的實例。
雲端記錄桶？	s3.Bucket	傳回 CloudFront 網頁分發模式所建立之記錄儲存貯體的執行個體。
雲端網絡發佈	cloudfront.CloudFrontWebDistribution	返回由模式創建的 CloudFront 網絡分發的實例。
埃德格拉姆針灸版本？	lambda.Version	返回由模式創建的 Lambda 邊緣函數版本的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

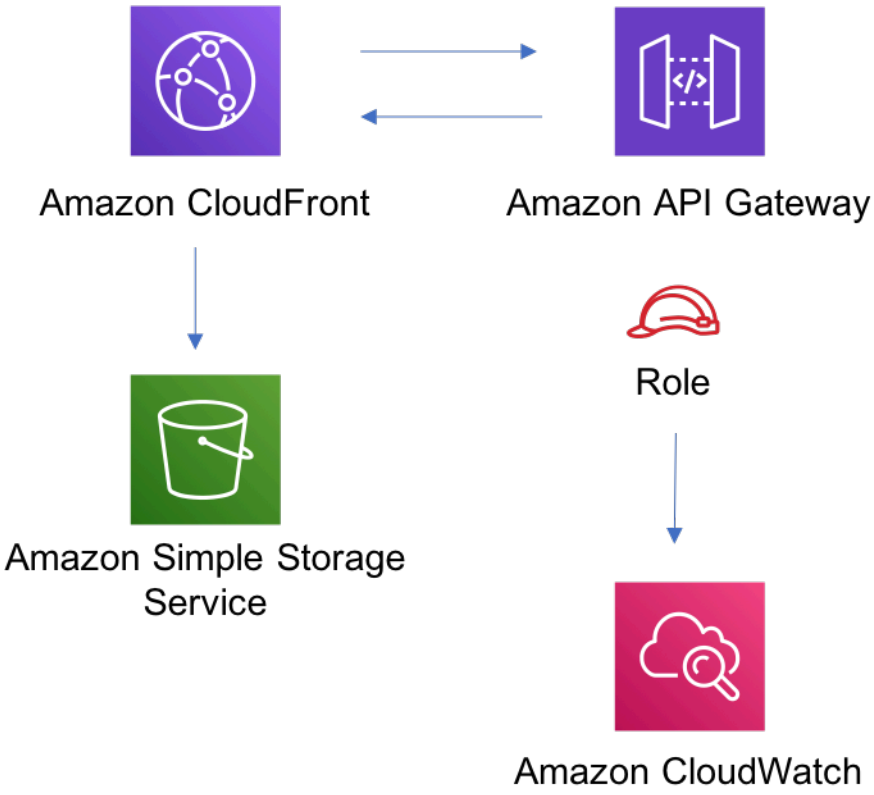
Amazon CloudFront

- 設定 CloudFront 網路發佈的存取記錄
- 在 CloudFront 網路發佈的所有回應中啟用自動插入最佳實務 HTTP 安全性標頭

Amazon API Gateway

- 用戶提供的 API Gateway 對象按原樣使用
- 啟用 X-Ray 追蹤

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-結構/aws-雲前面的方式

aws-云前面的每個角度-lambda

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_cloudfront_apigateway_lambda`
 打字稿	`@aws-solutions-constructs/aws-cloudfront-apigateway-lambda`
 Java	`software.amazon.awsconstructs.services.cloudfrontapigatewaylambda`

Overview

這個 AWS 解決方案建構在 Amazon API 閘道支援 LambDA 的 REST API 前面實作了一個 Amazon 雲端分發。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案CloudFrontToApiGatewayToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性，可覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
阿比格特威道具？	api.LambdaRestApiProps	可選用的使用者提供的道具來覆寫 API Gateway 的預設道具
雲端發佈道具？	cloudfront.DistributionProps	可選的使用者提供的道具，以覆寫 CloudFront 分發的預設道具。

名稱	類型	描述
插入安全性標頭？	boolean	可選的用戶提供的道具，用於在 CloudFront 的所有響應中打開/關閉最佳實踐 HTTP 安全標頭的自動注入
記錄群組道具？	logs.LogGroupProps	使用者提供的可選道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
ApeGateway Gateway	api.RestApi	返回由模式創建的 API Gateway REST API 的實例。
雲端觀察角色	iam.Role	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。
應用路徑記錄群組	logs.LogGroup	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。
雲端記錄桶？	s3.Bucket	傳回 CloudFront 網頁分發模式所建立之記錄儲存貯體的執行個體。
雲端網絡發佈	cloudfront.CloudFrontWebDistribution	返回由模式創建的 CloudFront 網絡分發的實例。
埃德格拉姆針灸版本	lambda.Version	返回由模式創建的 Lambda 邊緣函數版本的實例。

名稱	類型	描述
LambdaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。

預設定定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon CloudFront

- 設定 CloudFront 網路發佈的存取記錄
- 在 CloudFront 網路發佈的所有回應中啟用自動插入最佳實務 HTTP 安全性標頭

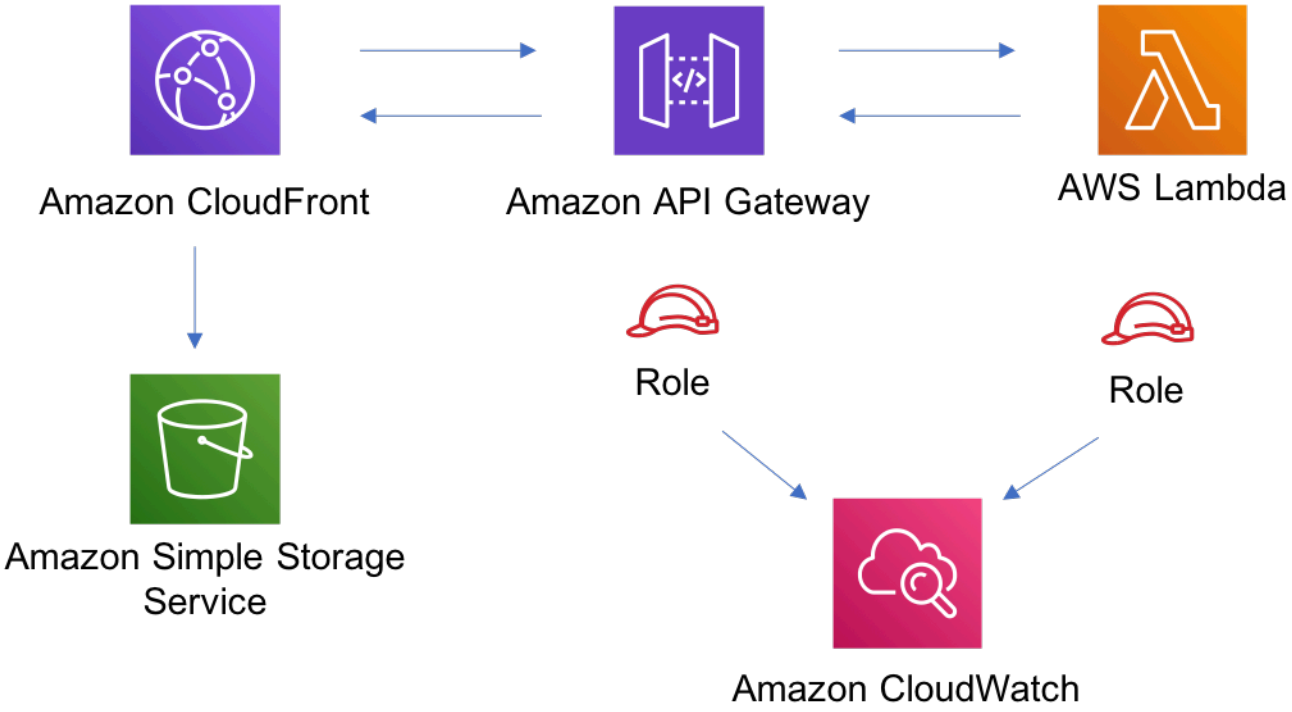
Amazon API Gateway

- 部署區域性 API 端點
- 啟用適用於 API Gateway 的 CloudWatch 日誌
- 設定 API Gateway 的最低權限存取 IAM 角色
- 將所有 API 方法的預設授權類型設定為 IAM
- 啟用 X-Ray 追蹤

AWS Lambda 功能

- 針對 Lambda 函數配置有限的許可存取 IAM 角色
- 針對 NodeJS Lambda 函數啟用具有持續作用的重複使用連線
- 啟用 X-Ray 追蹤
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/AW-雲-應用程序-lambda

雲端前端媒體存放區

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_cloudfront_mediastore`
打字稿	`@aws-solutions-constructs/aws-cloudfront-mediastore`
Java	`software.amazon.awsconstructs.services.cloudfrontmediastore`

Overview

這個 AWS 解決方案建構實作連接到 AWS Elemental MediaStore 容器的 Amazon CloudFront 分發。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scope`Construct`
- id`string`

- 提案CloudFrontToMediaStoreProps

模式建立道具

名稱	類型	描述
存在媒體商店集裝箱？	mediastore.CfnContainer	可選的使用者提供的 MediaStore 容器來覆寫預設的媒體存放區容器。
媒體存儲容器道具？	mediastore.CfnContainerProps	可選的使用者提供的道具來覆寫 MediaStore 容器的預設道具。
雲端發佈道具？	cloudfront.DistributionProps | any	可選的使用者提供的道具，用於覆寫 CloudFront 發佈的預設道具。
插入安全性標頭？	boolean	可選的使用者提供的道具，可在 CloudFront 的所有回應中開啟/關閉最佳做法 HTTP 安全標頭的自動注入。

模式性質

名稱	類型	描述
雲端網絡發佈	cloudfront.CloudFrontWebDistribution	返回由模式創建的 CloudFront 網絡分發的實例。
媒體儲存區容器	mediastore.CfnContainer	返回由模式創建的 MediaStore 容器的實例。
雲端記錄桶	s3.Bucket	傳回 CloudFront 網頁分發模式所建立之記錄儲存貯體的執行個體。

名稱	類型	描述
雲端前端要求原則	cloudfront.OriginRequestPolicy	傳回由 CloudFront 網路分發的模式所建立的 CloudFront 原始碼請求原則的執行個體。
CloudFrontOrigin?	cloudfront.OriginAccessIdentity	傳回由 CloudFront 網路分發的模式所建立的 CloudFront 原始存取身分識別的執行個體。
埃德格拉姆針灸版	lambda.Version	返回由模式創建的 Lambda 邊緣函數版本的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

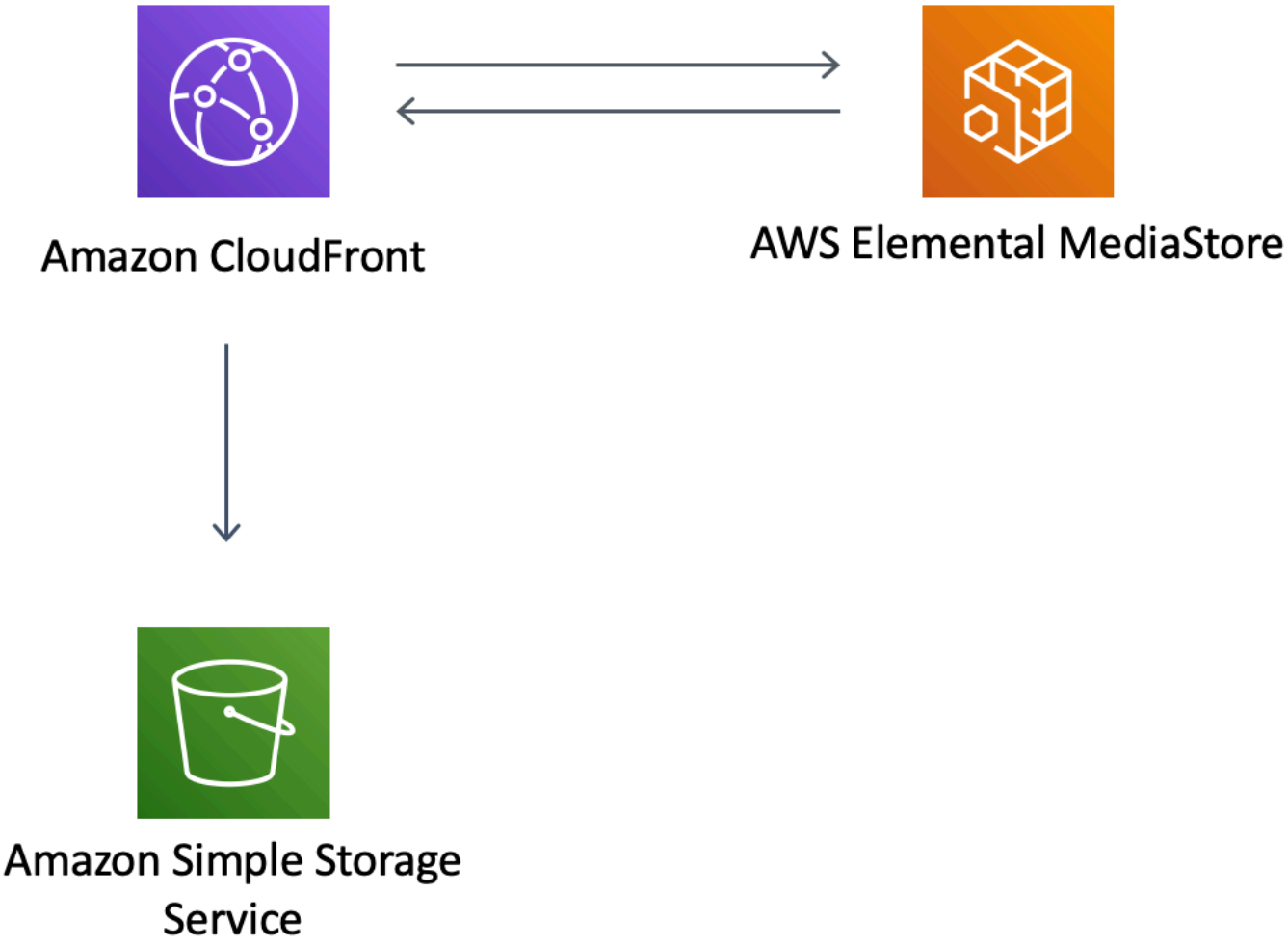
Amazon CloudFront

- 設定 CloudFront 網路分發的存取記錄
- 啟用 AWS Elemental MediaStore 容器的 CloudFront 原始請求政策
- 設定`User-Agent`具有 CloudFront 原始伺服器存取身份的自訂
- 在 CloudFront 網路分發的所有回應中，啟用自動插入最佳實務 HTTP 安全性標頭

AWS Elemental MediaStore

- 設定刪除原則以保留資源
- 使用 CloudFormation 堆疊名稱設定容器名稱
- 設定預設容器跨來源資源分享 (CORS) 政策
- 設定預設物件生命週期政策
- 設定預設容器政策只允許`aws:UserAgent`具有 CloudFrontOrigin 原始伺服器
- 設定預設指標政策
- 啟用存取日誌

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案結構/aws-雲前媒體存儲

雲端前端 3

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模式。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_cloudfront_s3`
TypeScript	`@aws-solutions-constructs/aws-cloudfront-s3`
Java	`software.amazon.awsconstructs.services.cloudfronts3`

Overview

AWS 解決方案建構會在 Amazon S3 儲存貯體前實作 Amazon CloudFront 散發。

下面是 TypeScript 中的最小可部署模式定義：

```
import { CloudFrontToS3 } from '@aws-solutions-constructs/aws-cloudfront-s3';

new CloudFrontToS3(this, 'test-cloudfront-s3', {});
```

Initializer

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

參數

- scopeConstruct
- idstring
- 提案CloudFrontToS3Props

模式建立道具

名稱	類型	描述
現在的巴克托比？	s3.Bucket	S3 桶對象的現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。
水桶道具？	s3.BucketProps	可選的使用者提供的屬性來覆寫儲存貯體的預設屬性。忽略existingBucketObj 提供。
雲端發佈道具？	cloudfront.DistributionProps	可選的使用者提供的道具，以覆寫 CloudFront 分發的預設道具。
插入安全性標頭？	boolean	可選的用戶提供的道具，用於在 CloudFront 的所有響應中打開/關閉最佳實踐 HTTP 安全標頭的自動注入

模式性質

名稱	類型	描述
雲端網絡發佈	cloudfront.CloudFrontWebDistribution	返回由模式創建的 CloudFront 網絡分發的實例。

名稱	類型	描述
S3 儲存貯體？	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。
埃德格拉姆針灸版本？	lambda.Version	返回由模式創建的 Lambda 邊緣函數版本的實例。
雲端記錄桶？	s3.Bucket	傳回 CloudFront 網頁分發模式所建立之記錄儲存貯體的執行個體。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

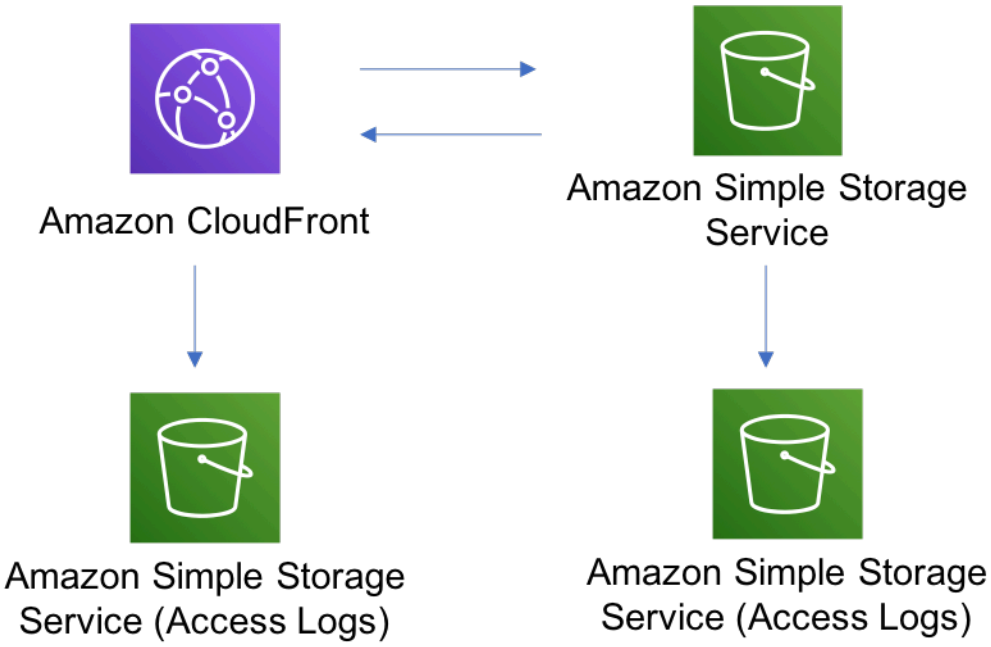
Amazon CloudFront

- 設定 CloudFront 網路發佈的存取記錄
- 在 CloudFront 網路發佈的所有回應中啟用自動插入最佳實務 HTTP 安全性標頭

Amazon S3 儲存貯體

- 設定 S3 儲存貯體的存取記錄
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密
- 打開 S3 存儲桶的版本控制
- 不允許公開存取 S3 儲存貯體
- 刪除 CloudFormation 堆疊時保留 S3 桶
- 強制加密傳輸中的資料
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存區

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案結構/aws-雲前面 3

aw-認識-意大利-拉姆達

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	aws_solutions_constructs.aws_cognito_apigateway_lambda
打字稿	@aws-solutions-constructs/aws-cognito-apigateway-lambda
Java	software.amazon.awsconstructs.services.cognitoapigatewaylambda

Overview

這個 AWS 解決方案建構實作了 Amazon Cognito 保護 Amazon API Gateway 支援 LambDA 的 REST API。

下面是 TypeScript 中的最小可部署模式定義：

```
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'
    }
});
```

如果您在 API 上定義資源和方法（例如`proxy = false`），您必須調用`addAuthorizers()`方法完全定義 API 之後。這可確保 API 中的每個方法都受到保護。

請見此處 TypeScript：

```
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);
```

參數

- scope[Construct]
- id`string`
- 提案[CognitoToApiGatewayToLambdaProps]

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯	lambda.Function	Lambda 函數對象的現有實例，同時提供這個和lambdaFunctionProps 會造成錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
阿比格特威道具？	api.LambdaRestApiProps	可選用的使用者提供的道具來覆寫 API Gateway 的預設道具
干邑吐司道具？	cognito.UserPoolProps	選用的使用者提供的道具，以覆寫 Cognito 使用者集區的預設道具
認識吐司游泳池客戶道具？	cognito.UserPoolClientProps	選擇性的使用者提供的道具，可覆寫 Cognito 使用者集區用戶端的預設道具
記錄群組道具？	logs.LogGroupProps	使用者提供的選用道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
ApeGateway	api.RestApi	返回由模式創建的 API Gateway REST API 的實例。
LambdaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。

名稱	類型	描述
userPool	cognito.UserPool	傳回由模式建立的 Cognito 使用者集區的執行個體。
UserPoolClient	cognito.UserPoolClient	傳回由模式建立的 Cognito 使用者集區用戶端的執行個體。
雲端觀察角色	iam.Role	傳回模式所建立的 IAM 角色執行個體，該模式可啟用從 API Gateway REST API 存取記錄至 CloudWatch。
應用路徑記錄群組	logs.LogGroup	傳回傳送 API Gateway REST API 存取記錄檔的模式所建立之記錄群組的執行個體。
應用程式授權器	api.CfnAuthorizer	返回由模式創建的 API Gateway 授權的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon Cognito

- 設定使用者集區的密碼策略
- 強制執行「使用者集區」的進階安全模式

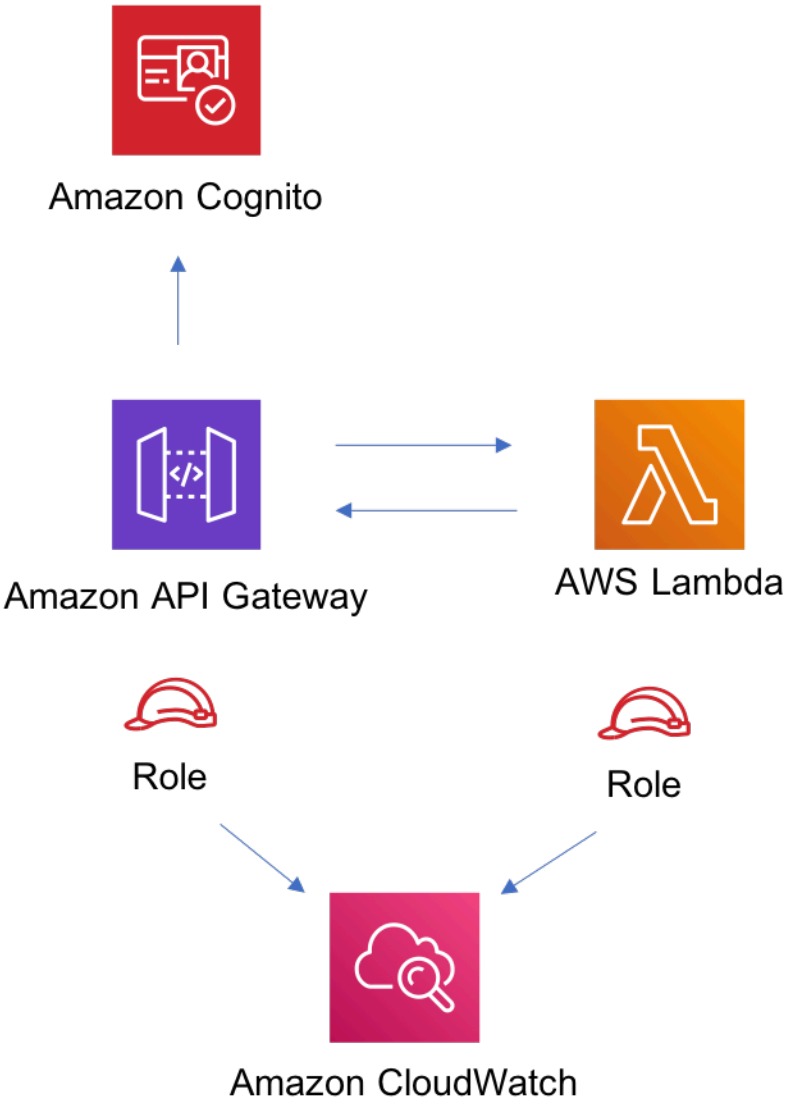
Amazon API Gateway

- 部署邊緣最佳化的 API 端點
- 啟用適用於 API Gateway 的 CloudWatch 日誌
- 設定 API Gateway 的最低權限存取 IAM 角色
- 將所有 API 方法的預設授權類型設定為 IAM
- 啟用 X-Ray 追蹤

AWS Lambda 功能

- 針對 Lambda 函數設定有限的權限存取 IAM 角色
- 針對 NodeJS Lambda 函數啟用具有持續作用的重複使用連線
- 啟用 X-Ray 追蹤
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/AW-認識-應用-lambda

aws-dynamodb-流-lambda

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_dynamodb_stream_lambda`
 打字稿	`@aws-solutions-constructs/aws-dynamodb-stream-lambda`
 Java	`software.amazon.awsconstructs.services.dynamodbstreamlambda`

Overview

此 AWS 解決方案建構實作含串流的模式 Amazon DynamoDB 表格，以使用最低權限叫用 AWS Lambda 函數。

這是一個最小的可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案DynamoDBStreamToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會造成錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預

名稱	類型	描述
		設屬性。忽略existingLambdaObj 提供。
DynamoTable 道具？	dynamodb.TableProps	選擇性的使用者提供的道具，用於覆寫 DynamoDB 表格的預設道具
現有的表格？	dynamodb.Table	DynamoDB 表格物件的現有實體，同時提供這個和dynamoTableProps 會造成錯誤。
華康活動推薦道具？	aws-lambda-event-sources.DynamoEventSourceProps	選擇性的使用者提供的道具，用於覆寫 DynamoDB 事件來源的預設道具

模式性質

名稱	類型	描述
DynaMotion 表格	dynamodb.Table	傳回由樣式建立之 DynamoDB 表格的實體。
LambdaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。

Lambda 功能

此模式需要 Lambda 函數，該函數可以從 DynamoDB 串流將資料張貼至 Elasticsearch 服務。提供範例函數這裡。

預設設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

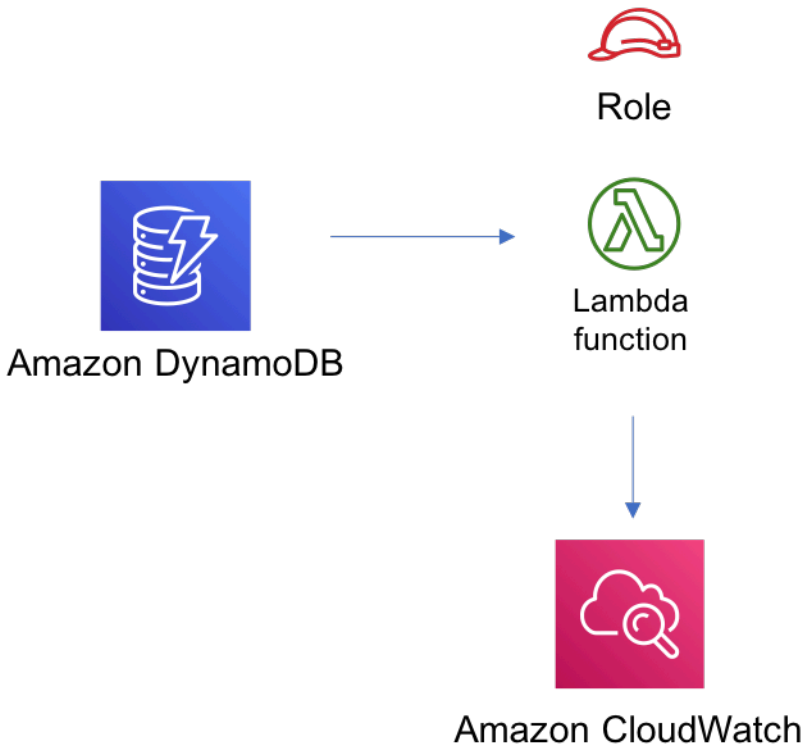
Amazon DynamoDB 資料表

- 將 DynamoDB 表格的計費模式設定為隨選 (按請求付費)
- 使用 AWS 受管的 KMS 金鑰啟用 DynamoDB 表格的伺服器端加密
- 為 DynamoDB 表建立名為「id」的分割區索引鍵
- 刪除 CloudFormation 堆棧時保留表
- 啟用持續備份和時間點復原

AWS Lambda 功能

- 針對 Lambda 函數設定有限權限存取 IAM 角色
- 針對 NodeJS Lambda 函數啟用具有持續作用的重複使用連線
- 啟用 X-Ray 追蹤
- 啟用失敗處理功能：在功能錯誤時啟用二分點；設定預設記錄保留天數上限 (24 小時)；設定預設重試嘗試次數上限 (500)；以及在失敗時部署 SQS 無效字母佇列做為目的地
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aw-DynamoDB 流-lambda

aws-Dynamodb 流-羊肉-彈性搜索-基班納

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_dynamodb_stream_lambda_elasticsearch_kibana`
 打字稿	`@aws-solutions-constructs/aws-dynamodb-stream-lambda-elasticsearch-kibana`
 Java	`software.amazon.awsconstructs.services.dynamodbstreamlambdaelasticsearchkibana`

Overview

此 AWS 解決方案建構實作具有串流、AWS Lambda 函數以及具有最低特權權限的 Amazon DynamoDB 表格。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- id`string`
- 提案DynamoDBStreamToLambdaToElasticSearchAndKibanaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，同時提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
DynamoTable 道具？	dynamodb.TableProps	選擇性的使用者提供的道具，用於覆寫 DynamoDB 表格的預設道具
現有的表格？	dynamodb.Table	DynamoDB 表格物件的現有實體，同時提供這個和dynamoTableProps 會導致錯誤。
華康活動推薦道具？	aws-lambda-event-sources.DynamoEventSourceProps	選擇性的使用者提供的道具，用於覆寫 DynamoDB 事件來源的預設道具
電子域名道具？	elasticsearch.CfnDomainProps	可選用的使用者提供的道具，以覆寫 Amazon Elasticsearch Service 的預設道具
domainName	string	Cognito 和 Amazon Elasticsearch Service 的域名
創建雲端觀察器	boolean	是否要建立建議的警報。

模式性質

名稱	類型	描述
雲觀察屋？	cloudwatch.Alarm[]	傳回模式建立的一或多個 CloudWatch 警示的清單。
DynaMotion 表格	dynamodb.Table	傳回由樣式建立之 DynamoDB 表格的實體。
彈性搜尋網域	elasticsearch.CfnDomain	返回由圖案創建的 Elasticsearch 域的實例。
IdentityPool	cognito.CfnIdentityPool	傳回由模式建立的 Cognito 身分識別集區的執行個體。
LambdaFAULT	lambda.Function	返回由模式創建的 Lambda 函數的實例。
userPool	cognito.UserPool	傳回由模式建立的 Cognito 使用者集區的執行個體。
UserPoolClient	cognito.UserPoolClient	傳回由模式建立的 Cognito 使用者集區用戶端的執行個體。

Lambda 功能

此模式需要 Lambda 函數，該函數可以從 DynamoDB 串流將資料張貼至 Elasticsearch 服務。提供範例函數這裡。

預設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon DynamoDB 表

- 將 DynamoDB 表格的計費模式設定為隨選 (按請求付費)
- 使用 AWS 受管的 KMS 金鑰啟用 DynamoDB 表格的伺服器端加密

- 為 DynamoDB 表格建立名為「id」的分割區索引鍵
- 刪除 CloudFormation 堆棧時保留表
- 啟用連續備份和時間點復原

AWS Lambda 功能

- 針對 Lambda 函數設定有限權限存取 IAM 角色
- 針對 NodeJS Lambda 函數啟用具有持續作用的重複使用連線
- 啟用 X-Ray 追蹤
- 啟用失敗處理功能：在功能錯誤時啟用二分點；設定預設記錄保留天數上限 (24 小時)；設定預設重試嘗試次數上限 (500)；以及在失敗時部署 SQS 無效字母佇列做為目的地
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

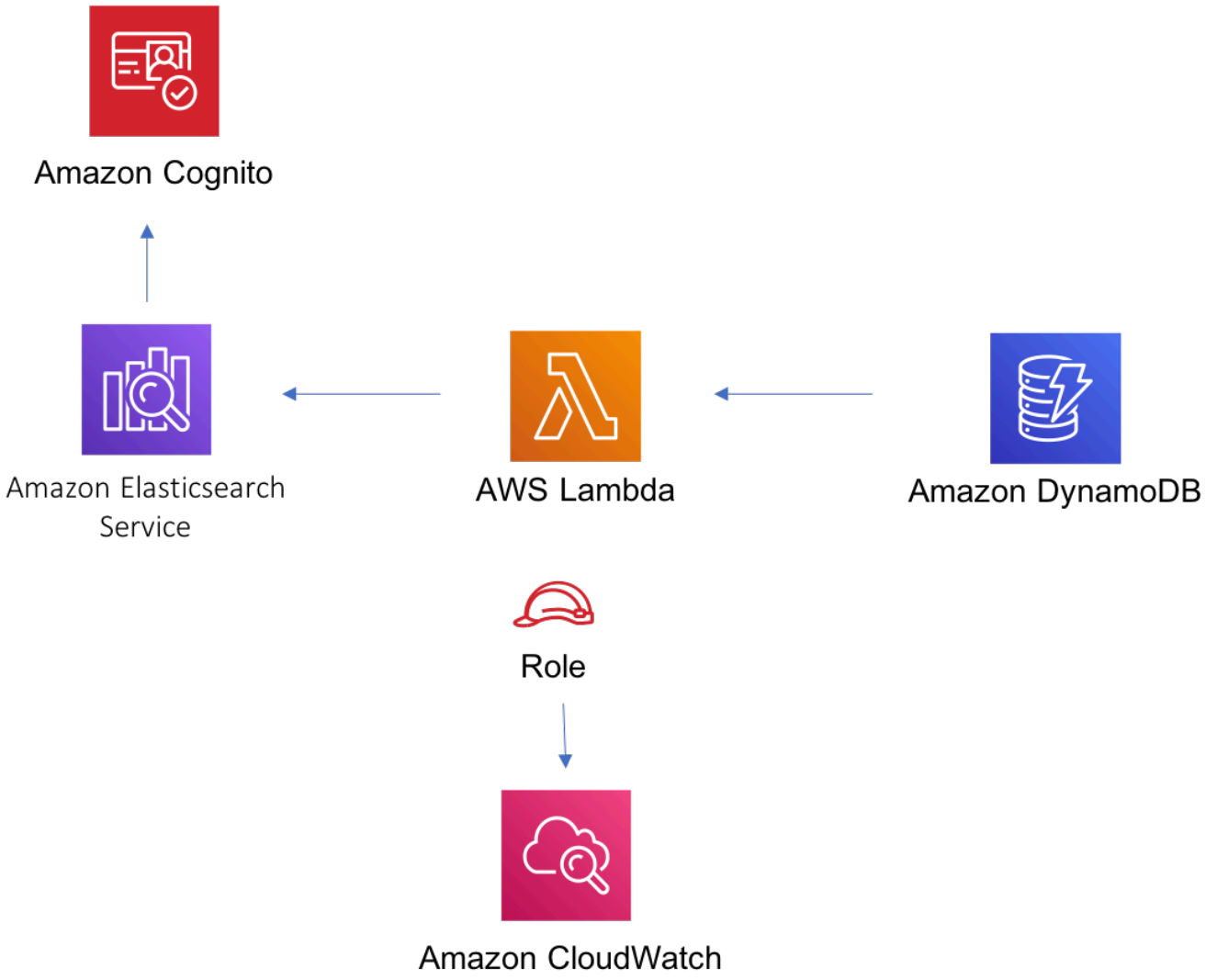
Amazon Cognito

- 設定使用者集區的密碼策略
- 強制執行使用者集區的進階安全模式

Amazon Elasticsearch Service

- 為彈性搜索域部署最佳實踐 CloudWatch 警報
- 使用 Cognito 使用者集區保護 Kibana 儀表板存取
- 使用 AWS 受管的 KMS 金鑰啟用彈性搜尋網域的伺服器端加密
- 啟用彈性搜尋網域的節點對節點加密
- 為 Amazon ES 網域配置叢集

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-Dynamodb 流-羊肉-彈性搜索-kibana

AWS-事件規則-運動防火軟管-3

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模式。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_events_rule_kinesisfirehose_s3`
 打字稿	`@aws-solutions-constructs/aws-events-rule-kinesisfirehose-s3`
 Java	`software.amazon.awsconstructs.services.eventsrulekinesisfirehoses3`

Overview

此 AWS 解決方案建構實作了 Amazon CloudWatch Events 規則，將資料傳送到連接到 Amazon S3 儲存貯體的 Amazon Kinesis 資料火軟管交付串流。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案EventsRuleToKinesisFirehoseToS3Props

模式建立道具

名稱	類型	描述
事件規則	events.RuleProps	使用者提供的屬性，以覆寫 CloudWatch 事件規則的預設屬性。
金斯火焰炮道具？	aws-kinesisfirehose.CfnDeliveryStreamProps	選用的使用者提供的道具，可覆寫 Kinesis 火軟管傳遞串流的預設道具。
現在的巴克托比？	s3.IBucket	S3 存儲桶對象的現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。
水桶道具？	s3.BucketProps	可選的使用者提供的道具來覆寫 S3 儲存貯體的預設道具。

名稱	類型	描述
記錄群組道具？	logs.LogGroupProps	使用者提供的選用道具，用於覆寫 CloudWatch Logs 日誌群組的預設定。

模式性質

名稱	類型	描述
事件規則	events.Rule	傳回模式所建立之事件規則的實體。
動力煙管	kinesisfirehose.CfnDeliveryStream	傳回模式所建立之 Kinesis Firehose 傳遞串流的執行個體。
S3 儲存貯體	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。
事件角色？	iam.Role	傳回由 CloudWatch 事件規則建構所建立角色的執行個體。
啟動式火爐	iam.Role	傳回由 Kinesis 火軟管交付串流模式所建立的 IAM 角色執行個體。
啟動火源群組	logs.LogGroup	傳回由 Kinesis Firehose 存取記錄傳送至此模式所建立的記錄群組執行個體。

預設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon CloudWatch Events

- 為事件規則設定最低權限存取 IAM 角色，以發佈至 Kinesis Firehose 傳遞串流。

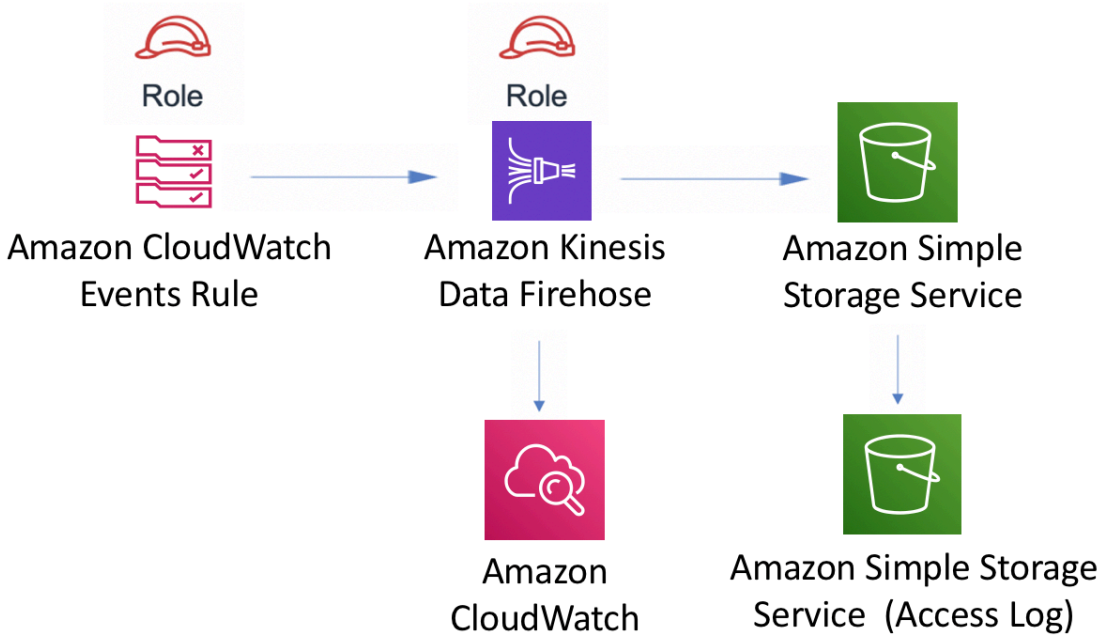
Amazon Kinesis Firehose

- 啟用 Kinesis 防火軟管的 CloudWatch 控記錄功能。
- 為 Amazon Kinesis Firehose 設定最低權限存取 IAM 角色。

Amazon S3 儲存貯體

- 設定儲存貯體的存取記錄。
- 使用 AWS 受管 KMS 金鑰啟用儲存貯體的伺服器端加密。
- 打開存儲桶的版本控制。
- 不允許存儲桶的公開訪問。
- 刪除 CloudFormation 堆疊時保留儲存貯體。
- 套用生命週期規則，可在 90 天後將非目前物件版本移至 Glacier 儲存區。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/AW-事件-規則-運動防火軟管-3

aws--事件-規則-運動流

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	aws_solutions_constructs.aws_events_rule_kinesisstream
TypeScript	@aws-solutions-constructs/aws-events-rule-kinesisstreams
Java	software.amazon.awsconstructs.services.eventsrulekinesisstream

Overview

此 AWS 解決方案建構實作了 Amazon CloudWatch Events 規則，將資料傳送到 Amazon Kinesis 資料流。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案EventsRuleToKinesisStreamsProps

模式建立道具

名稱	類型	描述
事件規則	events.RuleProps	使用者提供的屬性，以覆寫 CloudWatch 事件規則的預設屬性。
現在的斯特拉莫比	kinesis.Stream	Kinesis 流的現有實例，提供這個和kinesisStreamProps 會導致錯誤。
運動流道具？	kinesis.StreamProps	選用的使用者提供的道具，可覆寫 Kinesis 串流的預設道具。
創建雲端觀察	boolean	是否要建立建議的警報。

模式性質

名稱	類型	描述
事件規則	events.Rule	傳回模式所建立之事件規則的實體。
KinesisStream	kinesis.Stream	傳回由模式建立之 Kinesis 串流的實體。

名稱	類型	描述
事件角色？	iam.Role	傳回由 CloudWatch 事件規則建構所建立角色的執行個體。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon CloudWatch Events

- 為事件規則設定最低權限存取 IAM 角色，以發佈至 Kinesis 資料流。

Amazon Kinesis 串流

- 使用 AWS 受管 KMS 金鑰為 Kinesis 資料串流啟用伺服器端加密。

Architecture

GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aw-事件-規則-運動流

AW-事件規則-拉姆達

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_events_rule_lambda`
 TypeScript	`@aws-solutions-constructs/aws-events-rule-lambda`
 Java	`software.amazon.awsconstructs.services.eventsrulelambda`

Overview

此 AWS 解決方案建構實作 AWS 事件規則和 AWS Lambda 函數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案EventsRuleToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會造成錯誤。

名稱	類型	描述
拉姆針灸道具	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
事件規則道具	events.RuleProps	使用者提供的事件規則來覆寫預設值

模式性質

名稱	類型	描述
事件規則	events.Rule	傳回模式所建立之事件規則的實體。
LambdaFAULT	lambda.Function	返回由模式創建的 Lambda 函數的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

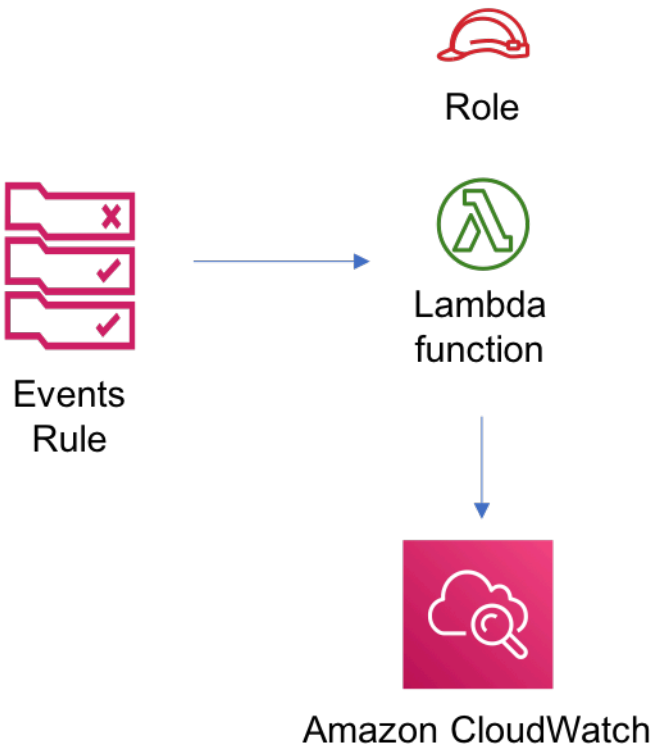
Amazon CloudWatch Events 規則

- 授與最低權限權限給 CloudWatch 事件以觸發 Lambda 函數

AWS Lambda 功能

- 針對 Lambda 函數設定有限權限存取 IAM 角色
- 針對 NodeJS Lambda 函數啟用具有持續作用的重複使用連線
- 啟用 X-Ray 追蹤
- SET DEFAULT
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-事件-規則-lambda

AWS-事件-規則-SN

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	aws_solutions_constructs.aws_events_rule_sns
TypeScript	@aws-solutions-constructs/aws-events-rule-sns
Java	software.amazon.awsconstructs.services.eventsrulesns

Overview

此模式實作連接至 Amazon SNS 主題的 Amazon CloudWatch Events 規則。

這是一個最小的可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案EventsRuleToSnsProps

模式建立道具

名稱	類型	描述
事件規則	events.RuleProps	使用者提供的屬性，以覆寫 CloudWatch 事件規則的預設屬性。
現在的托比科比？	sns.Topic	SNS 主題對象的現有實例，提供這個和topicProps 會導致錯誤。
主題道具？	sns.TopicProps	選用的使用者提供的屬性，可覆寫 SNS 主題的預設屬性。忽略existingTopicObj 提供。
使用客戶管理的金鑰啟用加密？	boolean	是否使用由此 CDK 應用程式管理或匯入的客戶管理的加密金鑰。如果匯入加密金鑰，則必

名稱	類型	描述
		須在encryptionKey 屬性為此建構。
encryptionKey？	kms.Key	選用的現有加密金鑰，而非預設加密金鑰。
加密密鑰道具？	kms.KeyProps	選用的使用者提供的屬性，可覆寫加密金鑰的預設屬性。

模式性質

名稱	類型	描述
事件規則	events.Rule	傳回模式所建立之事件規則的實體。
snsTopic	sns.Topic	傳回由模式建立的 SNS 主題的實例。
encryptionKey	kms.Key	返回由模式創建的加密密鑰的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

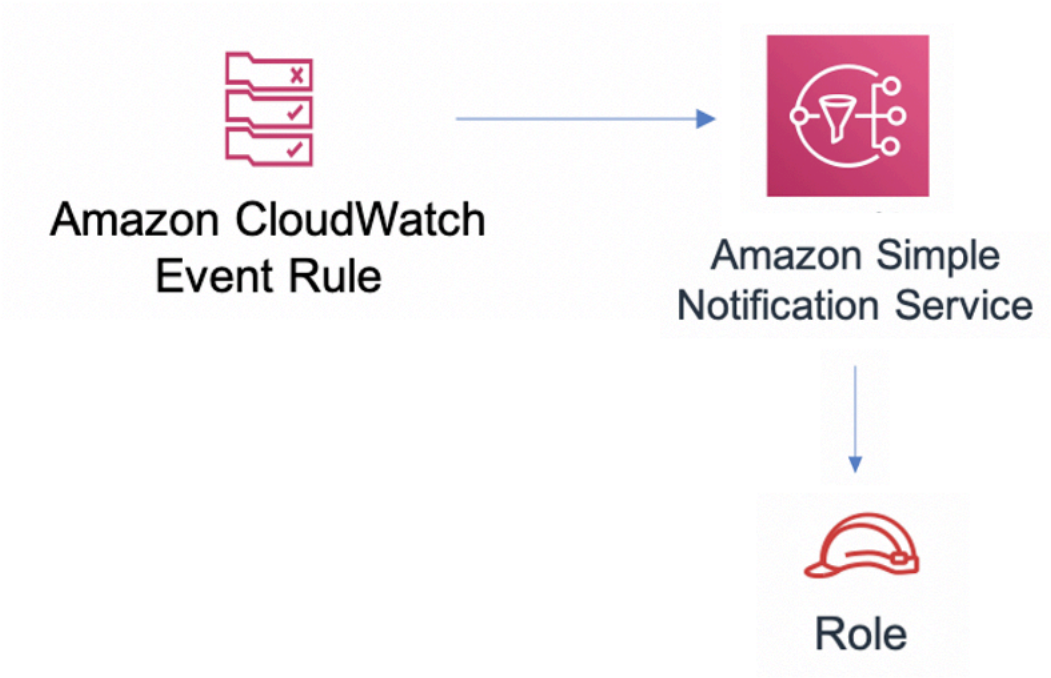
Amazon CloudWatch Events 規則

- 授與最低權限權限給 CloudWatch 事件以發佈至 SNS 主題。

Amazon SNS 主題

- 設定 SNS 主題的最低權限存取權限。
- 使用客戶受管 AWS KMS 金鑰對 SNS 主題啟用伺服器端加密。
- 強制加密傳輸中資料。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-事件-規則-s

aws--事件-規則-

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_events_rule_sqs`
 TypeScript	`@aws-solutions-constructs/aws-events-rule-sqs`
 Java	`software.amazon.awsconstructs.services.eventsrulesqs`

Overview

此模式實作連接至 Amazon SQS 佇列的 Amazon CloudWatch Events 規則。

這是一個最小的可部署模式定義：

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

Initializer

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

參數

- scopeConstruct
- idstring
- 提案EventsRuleToSqsProps

模式建立道具

名稱	類型	描述
事件規則道具	events.RuleProps	使用者提供的屬性，以覆寫 CloudWatch 事件規則的預設屬性。
是否存在佇列中？	sqs.Queue	選擇性的現有 SQS 佇列，而不是預設佇列。同時提供這個和 queueProps 會導致錯誤。
佇列道具？	sqs.QueueProps	選擇性的使用者提供的特性來覆寫 SQS 佇列的預設特性。忽略existingQueueObj 提供。

名稱	類型	描述
是否啟用佇列清除？	boolean	是否授與其他權限給 Lambda 函數，使其能夠清除 SQS 佇列。預設為 false。
部署死亡佇列？	boolean	無論是建立用作無效字母佇列的次要佇列。預設為 true。
死亡排隊道具？	sqs.QueueProps	可選的使用者提供的道具來覆寫死信佇列的預設道具。只有在deployDeadLetterQueue 屬性設為 true。
maxReceiveCount？	number	訊息移到無效字母佇列之前，需交付佇列的次數。預設為 15。
使用客戶管理的金鑰啟用加密？	boolean	是否使用由此 CDK 應用程式管理或匯入的客戶管理的加密金鑰。如果匯入加密金鑰，則必須在encryptionKey 屬性為此建構。
encryptionKey？	kms.Key	選用的現有加密金鑰，而非預設加密金鑰。
加密密鑰道具？	kms.KeyProps	選用的使用者提供的屬性，可覆寫加密金鑰的預設屬性。

模式性質

名稱	類型	描述
事件規則	events.Rule	傳回模式所建立之事件規則的實體。

名稱	類型	描述
平方	sqs.Queue	返回由模式創建的 SQS 隊列的實例。
encryptionKey	kms.Key	返回由模式創建的加密密鑰的實例。
死機隊列？	sqs.Queue	返回由模式創建的死信隊列的實例，如果一個被部署。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

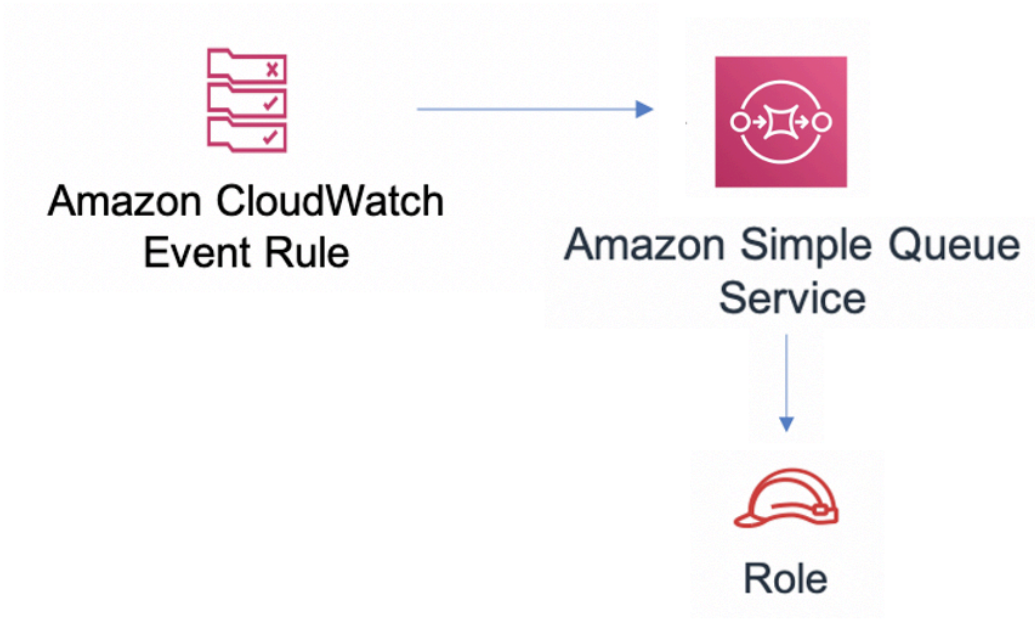
Amazon CloudWatch Events 規則

- 授與最低權限權限給 CloudWatch 事件以發佈至 SQS 佇列。

Amazon SQS 佇列

- 為來源佇列部署無效字母佇列。
- 使用客戶受管 AWS KMS 金鑰，為來源佇列啟用伺服器端加密。
- 強制加密傳輸中的資料。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-事件-規則-

aws 事件規則步驟功能

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_events_rule_step_function`
 打字稿	`@aws-solutions-constructs/aws-events-rule-step-function`
 Java	`software.amazon.awsconstructs.services.eventsrulestepfunction`

Overview

此 AWS 解決方案建構實作 AWS 事件規則和 AWS 步驟函數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案EventsRuleToStepFunctionProps

模式建立道具

名稱	類型	描述
斯塔特阿奇內道具	sfn.StateMachinePr ops	可選的用戶提供的道具來覆蓋 Sfn.Statemachine 的默認道具
事件規則	events.RuleProps	使用者提供的事件規則來覆寫預設值
創建雲端觀察器	boolean	是否要建立建議的警報。
記錄群組道具？	logs.LogGroupProps	可選用的使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
CloudWatch?	cloudwatch.Alarm[]	傳回模式建立的一或多個 CloudWatch 警示的清單。

名稱	類型	描述
事件規則	events.Rule	傳回模式所建立之事件規則的實體。
StateMachine	sfn.StateMachine	返回由模式創建的狀態機的實例。
台北市內洛集團	logs.LogGroup	傳回狀態機器模式所建立之日誌群組的執行個體。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

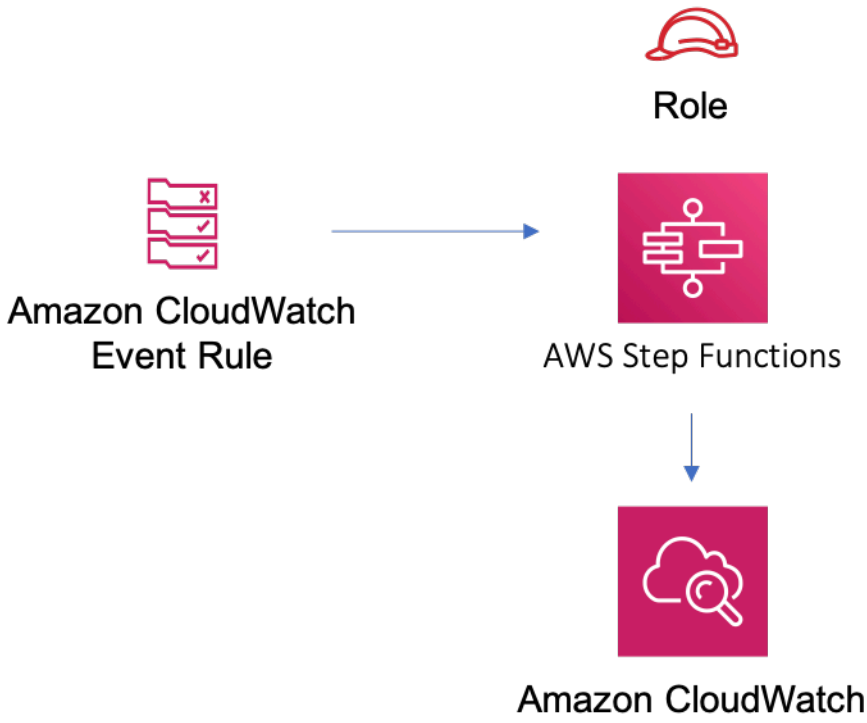
Amazon CloudWatch Events

- 授與最低權限權限給 CloudWatch 事件以觸發 Lambda 函數

AWS Step Function

- 啟用 API Gateway 的 CloudWatch 日誌
- 針對步驟功能部署最佳實務 CloudWatch 警示

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-事件-規則-步驟函數

Aws-io-運動防火軟管-3

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_iot_kinesisfirehose_s3`
TypeScript	`@aws-solutions-constructs/aws-iot-kinesisfirehose-s3`
Java	`software.amazon.awsconstructs.services.iotkinesisfirehoses3`

Overview

此 AWS 解決方案建構實作 AWS IoT MQTT 主題規則，將資料傳送到連接到 Amazon S3 儲存貯體的 Amazon Kinesis 資料火軟管交付串流。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案IotToKinesisFirehoseToS3Props

模式建立道具

名稱	類型	描述
IOTPIC 道具	iot.CfnTopicRuleProps	使用者提供的 CFN 主題規則提示來覆寫預設值
金斯火焰炮道具？	kinesisfirehose.CfnDeliveryStreamProps	選用使用者提供的道具，可覆寫 Kinesis 火軟管傳遞串流的預設道具
現在的巴克托比？	s3.Bucket	S3 存儲桶對象的現有實例，提供這個和bucketProps 會導致錯誤。
水桶道具？	s3.BucketProps	使用者提供的道具可覆寫 S3 儲存貯體的預設道具。如果這是提供的，那麼還提供bucketProps 是錯誤。
記錄群組道具？	logs.LogGroupProps	可選的使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
IOT 動作角色	iam.Role	傳回 IoT 規則模式所建立的 IAM 角色執行個體。
物件規則	iot.CfnTopicRule	傳回模式所建立之 IoT 主題規則的執行個體。
動力煙管	kinesisfirehose.CfnDeliveryStream	傳回模式所建立之 Kinesis Firehose 傳遞串流的執行個體。
啟動火源群組	logs.LogGroup	傳回由 Kinesis Firehose 存取記錄傳送至此模式所建立的記錄群組執行個體。
啟動式火爐	iam.Role	傳回由 Kinesis 火軟管交付串流模式所建立的 IAM 角色執行個體。
S3 儲存貯體？	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon IoT 規則

- 為 Amazon IoT 設定最低權限存取 IAM 角色

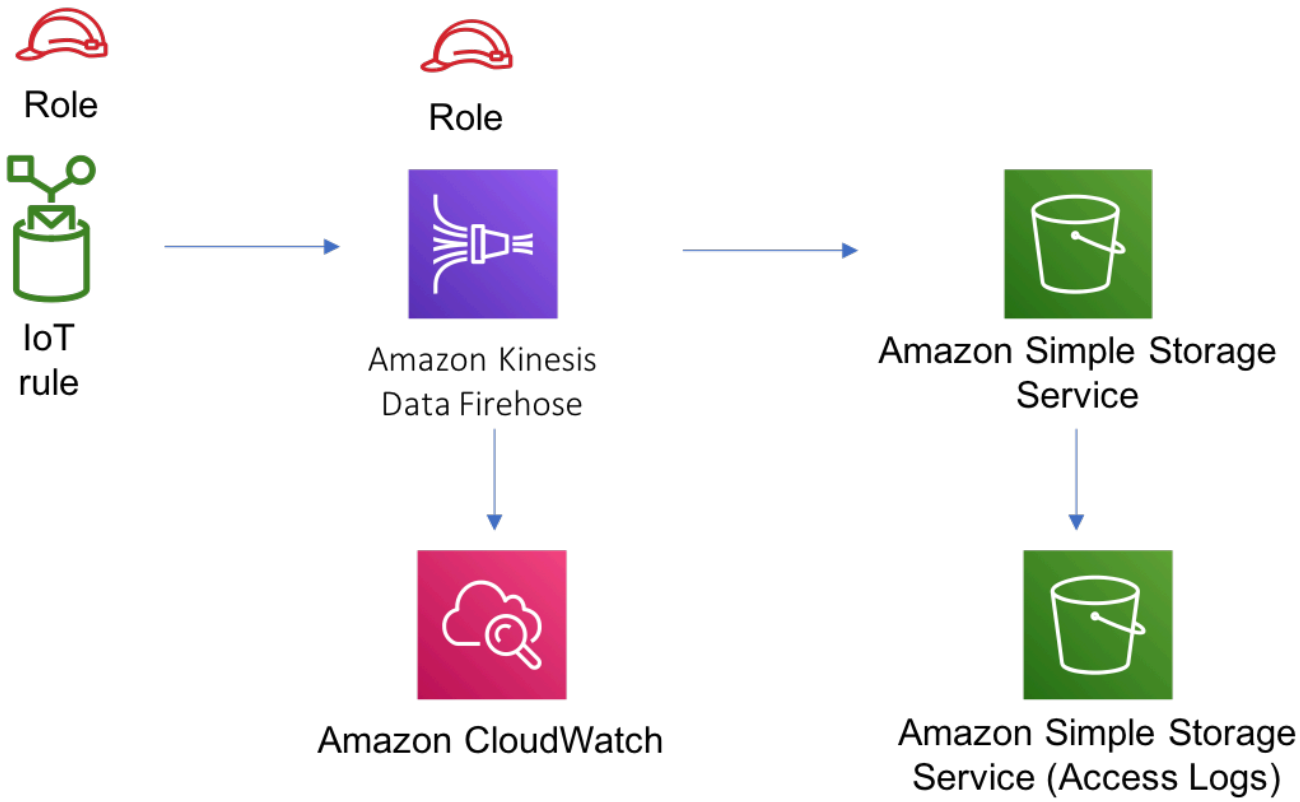
Amazon Kinesis Firehose

- 啟用 CloudWatch 控記錄功能
- 為 Amazon Kinesis Firehose 設定最低權限存取 IAM 角色

Amazon S3 儲存貯體

- 設定 S3 儲存貯體的存取記錄
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密
- 打開 S3 存儲桶的版本控制
- 不允許公開存取 S3 儲存貯體
- 刪除 CloudFormation 堆疊時保留 S3 桶
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存區

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/AW-動態-防火軟體-3

大概-蘭姆達

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_iot_lambda`
 TypeScript	`@aws-solutions-constructs/aws-iot-lambda`
 Java	`software.amazon.awsconstructs.services.iotlambda`

Overview

此 AWS 解決方案建構模式實作 AWS IoT MQTT 主題規則和 AWS Lambda 函數模式。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scope[Construct]
- id`string`
- 提案[IotToLambdaProps]

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
IOTPICE 道具？	iot.CfnTopicRuleProps	使用者提供的 CFN 主題規則提示來覆寫預設值

模式性質

名稱	類型	描述
物件規則	iot.CfnTopicRule	傳回模式所建立之 IoT 主題規則的執行個體。
LambDAFAULT	lambda.Function	返回由模式創建的 Lambda 函數的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

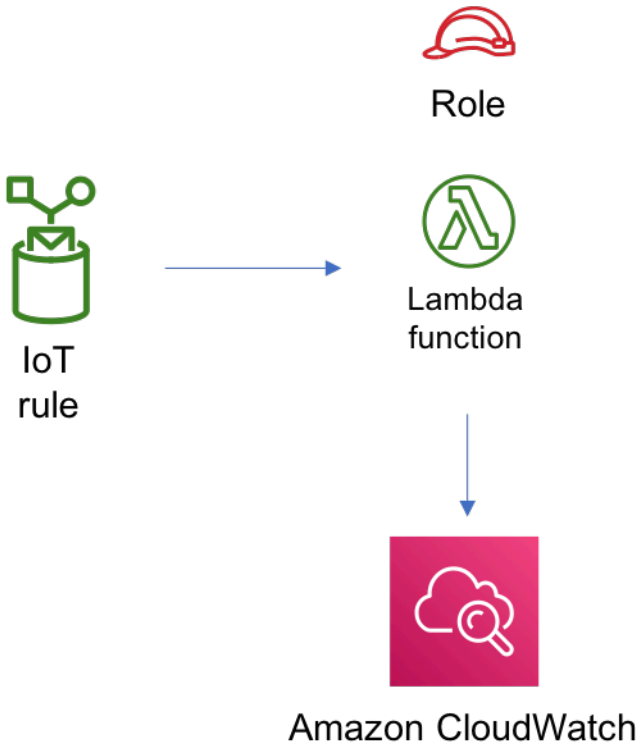
亞馬遜 IoT 規則

- 為 Amazon IoT 設定最低權限存取 IAM 角色。

AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-io-lambda

Aw-io-羊肉-動態學調節模式

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_iot_lambda_dynamodb`
打字稿	`@aws-solutions-constructs/aws-iot-lambda-dynamodb`
Java	`software.amazon.awsconstructs.services.iotlambdadynamodb`

Overview

此 AWS 解決方案建構模式實作 AWS IoT 主題規則、AWS Lambda 函數以及具有最低特權權限的 Amazon DynamoDB 表格。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案IotToLambdaToDynamoDBProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預

名稱	類型	描述
		設屬性。忽略existingLambdaObj 提供。
IOTPIC 道具	iot.CfnTopicRuleProps	用戶提供的道具覆寫預設道具
DynamoTable 道具？	dynamodb.TableProps	選擇性的使用者提供的道具，用於覆寫 DynamoDB 表格的預設道具
表格權限？	string	要授與 Lambda 函數的選擇性資料表權限。可能指定下列其中一個選項：All、Read、ReadWrite，或Write。

模式性質

名稱	類型	描述
DynaMotion 表格	dynamodb.Table	傳回由樣式建立之 DynamoDB 表格的實體。
物件規則	iot.CfnTopicRule	傳回模式所建立之 IoT 主題規則的執行個體。
LambDaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon IoT 規則

- 為 Amazon IoT 設定最低權限存取 IAM 角色。

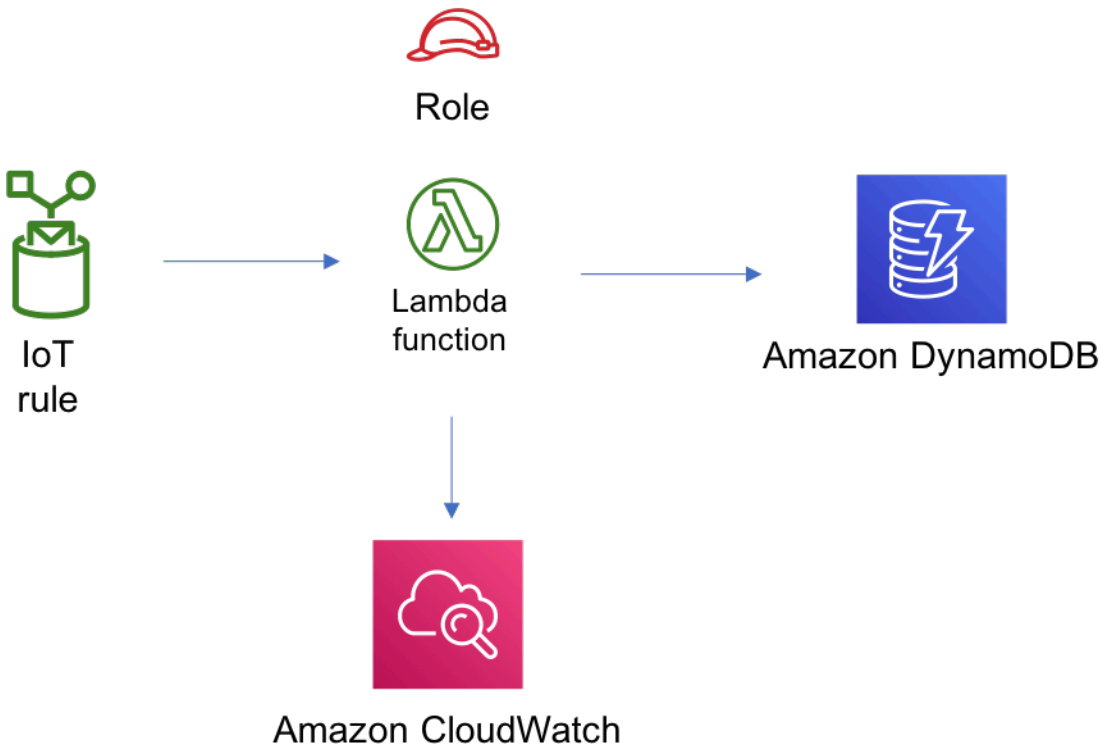
AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Amazon DynamoDB 表

- 將 DynamoDB 表格的計費模式設定為隨選 (按請求付費)。
- 使用 AWS 受管的 KMS 金鑰啟用 DynamoDB 表格的伺服器端加密。
- 為 DynamoDB 表格建立名為「id」的分割區索引鍵。
- 刪除 CloudFormation 堆棧時保留表。
- 啟用連續備份和時間點復原。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-io-lambda-動態模式

AW-運動防火軟管-3

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws-kinesis-firehose-s3`
TypeScript	`@aws-solutions-constructs/aws-kinesisfirehose-s3`
Java	`software.amazon.awsconstructs.services.kinesisfirehoses3`

Overview

此 AWS 解決方案建構實作 Amazon S3 儲存貯體的 Amazon Kinesis Data Firehose 交付串流。

下面是 TypeScript 中的最小可部署模式定義：

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

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

Initializer

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

參數

- scope`Construct`
- id`string`
- 提案`KinesisFirehoseToS3Props`

模式建立道具

名稱	類型	描述
水桶道具？	s3.BucketProps	可選的使用者提供的道具來覆寫 S3 儲存貯體的預設道具。
現在的巴克托比？	s3.IBucket	S3 儲存貯體的可選現有執行個體。如果這是提供的，那麼還提供bucketProps 是錯誤。
現在正在記錄巴克托比？	s3.IBucket	由模式建立之 S3 儲存貯體的可選現有記錄 S3 儲存貯體執行個體。
金斯火焰炮道具？	kinesisfirehose.CfnDeliveryStreamProps | any	選用使用者提供的道具，可覆寫 Kinesis 火軟管傳遞串流的預設道具。
記錄群組道具？	logs.LogGroupProps	可選的使用者提供的道具，以覆寫 CloudWatchLogs 群組的預設道具。

模式屬性

名稱	類型	描述
動力煙管	kinesisfirehose.CfnDeliveryStream	返回由構造創建的一個實例。
啟動火源群組	logs.LogGroup	傳回由 Kinesis Data Firehose 傳遞串流建構所建立之 Logs.logGroup 實體。
啟動式火爐	iam.Role	傳回由 Kinesis 資料火軟管傳送串流的建構所建立的 IAM.Role 執行個體。

名稱	類型	描述
S3 儲存貯體？	s3.Bucket	返回由構造創建的 S3.Bucket 的實例。
S3 記錄桶？	s3.Bucket	返回由構造創建的 S3.Bucket 的實例作為主存儲桶的日誌存儲桶。

預設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

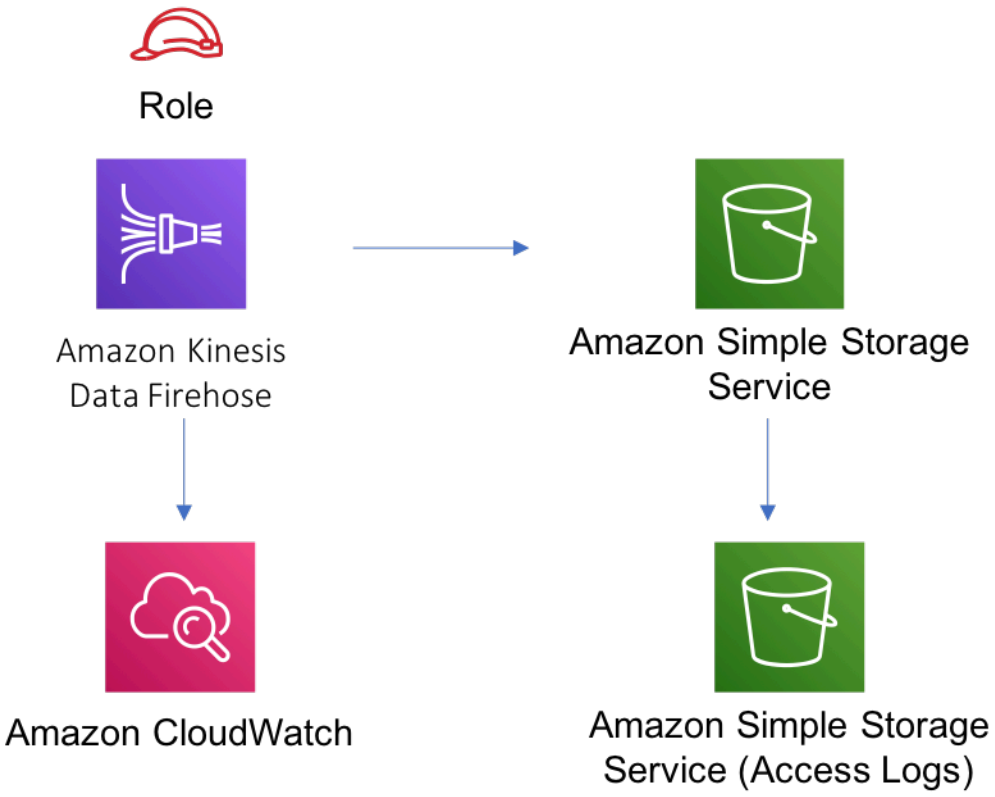
Amazon Kinesis Firehose

- 啟用 CloudWatch 控記錄功能
- 為 Amazon Kinesis Firehose 設定最低權限存取 IAM 角色

Amazon S3 儲存貯體

- 設定 S3 儲存貯體的存取記錄
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密
- 打開 S3 存儲桶的版本控制
- 不允許公開存取 S3 儲存貯體
- 刪除 CloudFormation 堆疊時保留 S3 桶
- 強制加密傳輸中的資料
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存空間

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-建構/aw-運動防火軟管-3

運動防火軟管-3 和運動分析

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_kinesisfirehose_s3_and_kinesisanalytics`
TypeScript	`@aws-solutions-constructs/aws-kinesisfirehose-s3-and-kinesisanalytics`
Java	`software.amazon.awsconstructs.services.kinesisfirehoses3kinesisanalytics`

Overview

此 AWS 解決方案建構實作連接到 Amazon S3 儲存貯體和 Amazon Kinesis 分析應用程式的 Amazon Kinesis 火軟管交付串流。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案KinesisFirehoseToAnalyticsAndS3Props

模式建立道具

名稱	類型	描述
金斯火焰炮道具？	kinesisFirehose.CfnDeliveryStreamProps	選用的使用者提供的道具，可覆寫 Kinesis Firehose 傳遞串流的預設道具。

名稱	類型	描述
運動分析道具？	kinesisAnalytics.CfnApplicationProps	選用的使用者提供的道具，可覆寫 Kinesis 分析應用程式的預設道具。
現在的巴克托比？	s3.IBucket	S3 存儲桶對象的現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。 |
水桶道具？	s3.BucketProps	可選的使用者提供的屬性來覆寫儲存貯體的預設屬性。忽略existingBucketObj 提供。
記錄群組道具？	logs.LogGroupProps	選用使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
運動分析	kinesisAnalytics.CfnApplication	傳回模式所建立之 Kinesis 分析應用程式的執行個體。
動力煙管	kinesisfirehose.CfnDeliveryStream	傳回模式所建立之 Kinesis Firehose 傳遞串流的執行個體。
啟動火源群組	logs.LogGroup	傳回由 Kinesis Firehose 存取記錄傳送至此模式所建立的記錄群組執行個體。
啟動式火爐	iam.Role	傳回由 Kinesis 火軟管交付串流模式所建立的 IAM 角色執行個體。

名稱	類型	描述
S3 儲存貯體？	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon Kinesis Firehose

- 啟用 CloudWatch 控記錄功能
- 為 Amazon Kinesis Firehose 設定最低權限存取 IAM 角色

Amazon S3 儲存貯體

- 設定 S3 儲存貯體的存取記錄
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密
- 打開 S3 存儲桶的版本控制
- 不允許公開存取 S3 儲存貯體
- 刪除 CloudFormation 堆疊時保留 S3 桶
- 強制加密傳輸中的資料
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存空間

Amazon Kinesis Data Analytics

- 為 Amazon Kinesis Analytics 設定最低權限存取 IAM 角色

Architecture

GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

	@aws-解決方案-建構/aw-運動-運動-三-和運動分析

運動流-糖果工作

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模式。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_kinesis_streams_gluejob`
TypeScript	`@aws-solutions-constructs/aws-kinesisstreams-gluejob`
Java	`software.amazon.awsconstructs.services.kinesisstreamsgluejob`

Overview

此 AWS 解決方案建構部署 Amazon Kinesis 資料流，並設定 AWS Glue Job 務以執行自訂 ETL 轉換，其中包含互動和安全的適當資源/屬性。它還創建一個 Amazon S3 存儲桶，其中可以上傳 AWS Glue Job 務的 Python 腳本。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- id`string`
- 提案KinesisstreamsToGluejobProps

模式建立道具

名稱	類型	描述
運動流道具？	kinesis.StreamProps	可選的使用者提供的道具，用於覆寫 Amazon Kinesis 資料流的預設道具。
現在的斯特拉莫比	kinesis.Stream	Kinesis 流的現有實例，提供這個和kinesisStreamProps 會造成錯誤。
葛樂珠的道具？	cfnJob.CfnJobProps	使用者提供的道具，用於覆寫 AWS Glue 工作的預設道具。
現在存在膠工作？	cfnJob.CfnJob	AWS Glue 任務的現有執行個體，提供這個和glueJobProps 會造成錯誤。
是否存在資料庫？	CfnDatabase	要與此建構搭配使用的現有 AWS Glue 資料庫。如果設置了這個，那麼databaseProps 忽略。
數據庫道具？	CfnDatabaseProps	使用者提供的道具，用於覆寫用於建立 AWS Glue 資料庫的預設道具。
現有表格？	CfnTable	AWS Glue 表的現有執行個體。如果設置了這個，那麼tableProps 和fieldSchema 會遭到忽略。
桌面道具？	CfnTableProps	用戶提供的道具，用於覆蓋用於創建 AWS Glue 表的默認道具。

名稱	類型	描述
欄位架構？	CfnTable.ColumnProperty[]	使用者提供的結構描述結構來建立 AWS Glue 表。
輸出資料存放區？	SinkDataStoreProps	使用者為 Amazon S3 儲存貯體提供的道具，可儲存 AWS Glue 作業輸出。目前僅支援 Amazon S3 做為輸出資料存放區類型。

SinkDataStoreProps

名稱	類型	描述
存在 3 輸出桶？	Bucket	S3 儲存貯體的現有執行個體，資料應該寫入。同時提供這個和outputBucketProps 會造成錯誤。
輸出路徑道具	BucketProps	使用者提供的儲存貯體屬性，用於建立用於儲存 AWS Glue 作業輸出的 Amazon S3 儲存貯體。
資料存放區類型	SinkStoreType	接收資料倉庫類型。

SinkStoreType

列舉可能包括 S3、DynamoDB、DocumentDB、RDS 或 Redshift 的資料存放區類型。當前構造實現只支持 S3，但未來可能添加其他輸出類型。

名稱	類型	描述
S3	string	S3 儲存類型

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon Kinesis Streams

- 為 Amazon Kinesis 資料流設定最低權限存取 IAM 角色。
- 使用 AWS 受管 KMS 金鑰為 Amazon Kinesis 串流啟用伺服器端加密。
- 為 Amazon CloudWatch Amazon Kinesis 串流部署最佳實務。

Glue Job

- 建立 AWS Glue 安全組態，以設定 CloudWatch、Job 書籤和 S3 的加密。CloudWatch 和 Job 書籤使用為 AWS 膠水服務建立的 AWS 受管 KMS 金鑰進行加密。S3 儲存貯體是以 SSE-S3 加密模式配置的。
- 設定允許 AWS Glue 從 Amazon Kinesis Data Streams 讀取的服務角色政策。

Glue 資料庫

- 建立 AWS Glue 資料庫。AWS Glue 表將新增至資料庫。此表格定義 Amazon Kinesis 資料串流中緩衝記錄的結構描述。

Glue 表

- 建立 AWS Glue 表格。表格結構描述定義是以 Amazon Kinesis 資料串流中緩衝記錄的 JSON 結構為基礎。

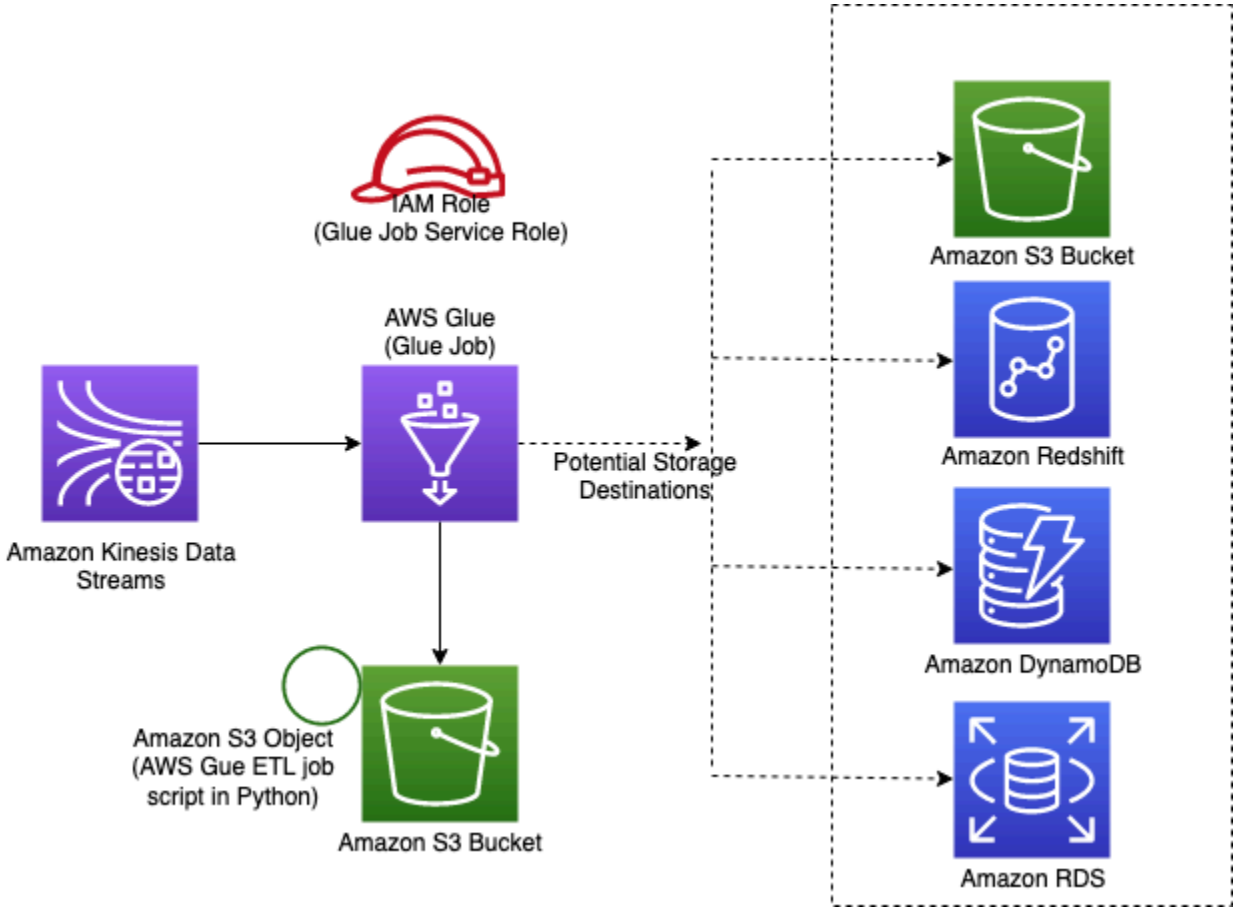
IAM 角色

- 具有權限的工作執行角色 1) 從 Amazon S3 儲存貯體位置讀取 ETL 指令碼、2) 從 Amazon Kinesis 資料流讀取記錄，以及 3) 執行 Amazon Glue 工作。

S3 儲存貯體

- 存放 ETL 轉換輸出的 Amazon S3 儲存貯體。此存儲桶將作為參數傳遞給創建的 AWS Glue 作業，以便在 ETL 腳本中使用它來將數據寫入其中。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-結構/aw-運動流-糖果

AW-運動流-運動防火軟管-3

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_kinesisstreams_kinesisfirehose_s3`
TypeScript	`@aws-solutions-constructs/aws-kinesis-streams-kinesis-firehose-s3`
Java	`software.amazon.awsconstructs.services.kinesisstreamskinesisfirehoses3`

Overview

此 AWS 解決方案建構實作連接到 Amazon S3 儲存貯體的 Amazon Kinesis 資料火管 (KDF) 交付串流的 Amazon Kinesis 資料流 (KDS)。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案KinesisStreams...ToS3Props

模式建立道具

名稱	類型	描述
水桶道具？	s3.BucketProps	可選的使用者提供的道具來覆寫 S3 儲存貯體的預設道具。
創造雲端觀察武器？	boolean	選擇性是否建立建議的 CloudWatch 報。
現在的巴克托比？	s3.IBucket	S3 存儲桶對象的可選現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。
現在正在記錄巴克托比？	s3.IBucket	針對模式建立的 S3 儲存貯體的可選現有記錄 S3 儲存貯體物件的執行個體。
現在的斯特拉莫比？	kinesis.Stream	Kinesis 流的現有實例，提供這個和kinesisStreamProps 會導致錯誤。
金斯火焰炮道具？	aws-kinesisfirehose.CfnDeliveryStreamProps | any	選用使用者提供的道具，可覆寫 Kinesis 火軟管傳遞串流的預設道具。
運動流道具？	kinesis.StreamProps	選用的使用者提供的道具可覆寫 Kinesis 串流的預設道具。
記錄群組道具？	logs.LogGroupProps	可選的使用者提供的道具可覆寫 CloudWatchlog 記錄群組的預設道具。

模式性質

名稱	類型	描述
CloudwatchAlidwatchAlims	cloudwatch.Alarm[]	返回由構造創建的雲 WATCH 實例列表。
動力煙管	kinesisfirehose.CfnDeliveryStream	返回由構造創建的一個實例。
啟動火源群組	logs.LogGroup	傳回由 Kinesis Data Firehose 傳遞串流建構所建立之 Logs.logGroup 實體。
啟動式火爐	iam.Role	傳回由 Kinesis 資料火軟管傳送串流的建構所建立的 IAM.Role 執行個體。
運動串流角色	iam.Role	傳回由 Kinesis 資料流建構所建立的 IAM.Role 的實體。
S3 儲存貯體？	s3.Bucket	返回由構造創建的 S3.Bucket 的實例。
S3 記錄桶？	s3.Bucket	返回由構造創建的 S3.Bucket 的實例作為主存儲桶的日誌存儲桶。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon Kinesis 串流

- 設定 Kinesis 串流的最低權限存取 IAM 角色
- 使用 AWS 受管 KMS 金鑰啟用 Kinesis 串流的伺服器端加密
- 為 Kinesis 串流部署最佳實務 CloudWatch 警示

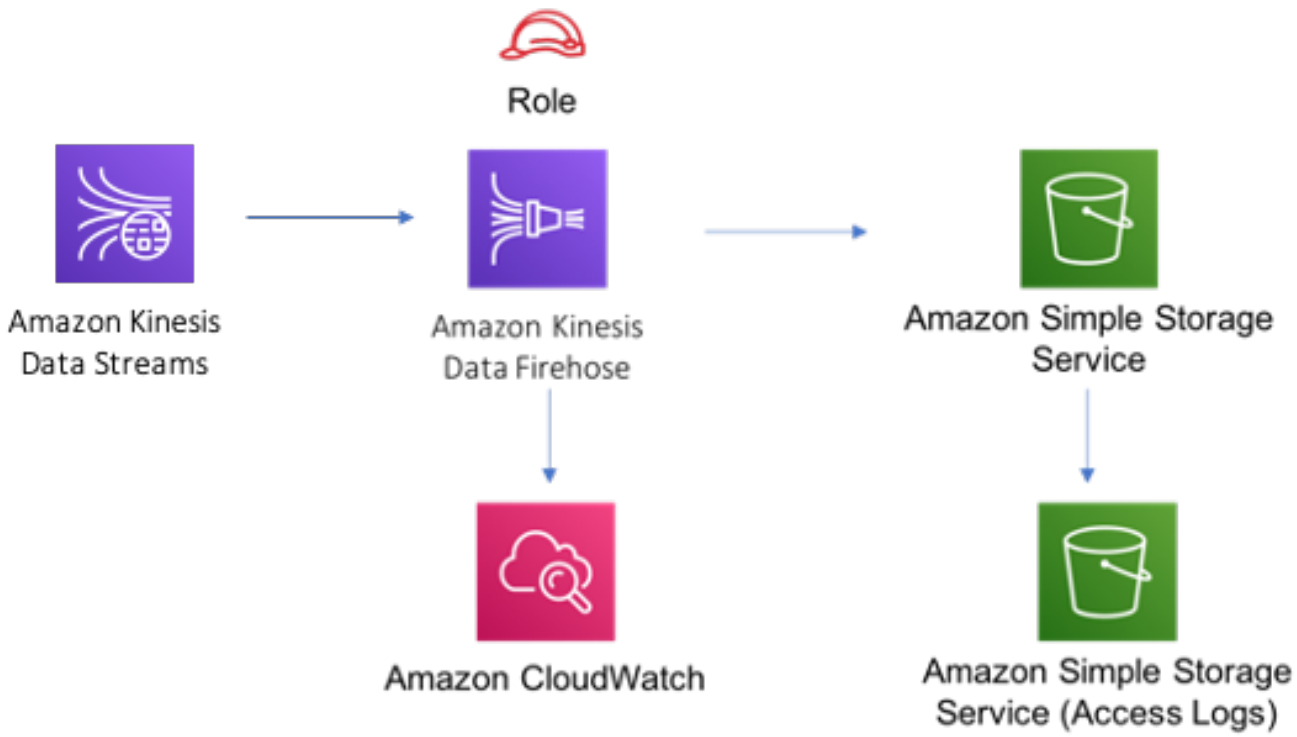
Amazon Kinesis Firehose

- 啟用 CloudWatch 控記錄功能
- 為 Amazon Kinesis Firehose 設定最低權限存取 IAM 角色

Amazon S3 儲存貯體

- S3 儲存貯體設定存取記錄
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密
- 強制加密傳輸中的資料
- 啟用儲存貯體版本設定
- 不允許公開存取 S3 儲存貯體
- 刪除 CloudFormation 堆疊時保留 S3 桶
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存空間

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-建構/aw-運動流-運動火腿-3

aw-運動流-拉姆達

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws-kinesis-streams-lambda`
 打字稿	`@aws-solutions-constructs/aws-kinesisstreams-lambda`
 Java	`software.amazon.awsconstructs.services.kinesisstreamslambda`

Overview

此 AWS 解決方案建構部署 Kinesis 串流和 Lambda 函數，其中包含適當的資源/屬性，以提供互動和安全性。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案KinesisStreamsToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，同時提供這個和lambdaFunctionProps 會導致錯誤。

名稱	類型	描述
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
運動流道具？	kinesis.StreamProps	選用的使用者提供的道具，可覆寫 Kinesis 串流的預設道具。
現在的斯特拉莫比？	kinesis.Stream	Kinesis 流的現有實例，提供這個和kinesisStreamProps 會導致錯誤。
活動活動道具？	aws-lambda-event-sources.KinesisEventSourceProps	選用的使用者提供的道具，可覆寫 Lambda 事件來源對應的預設道具。
創建雲端觀察器	boolean	是否要建立建議的警報。

模式性質

名稱	類型	描述
Kinesis串流	kinesis.Stream	傳回由模式建立之 Kinesis 串流的實體。
Lambda FAULT	lambda.Function	返回由模式創建的 Lambda 函數的實例。
運動串流角色	iam.Role	傳回 Kinesis 串流模式所建立的 IAM 角色執行個體。
CloudwatchAloudwatchAlims？	cloudwatch.Alarm[]	傳回模式建立的一或多個 CloudWatch 警示的清單。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

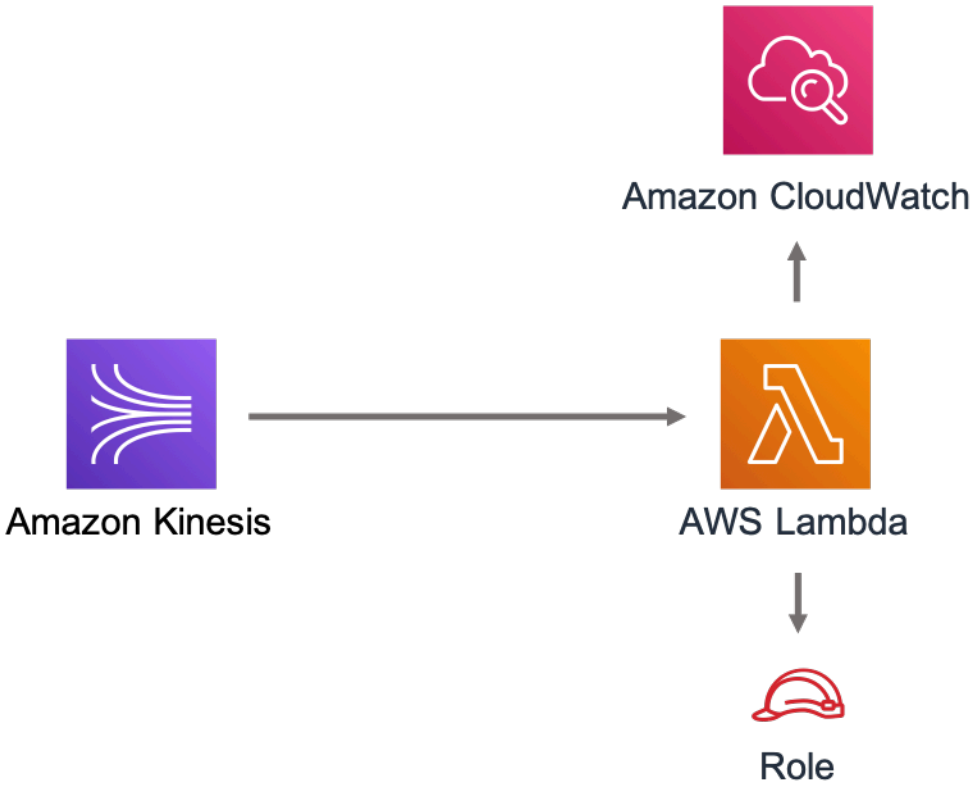
Amazon Kinesis 串流

- 設定 Kinesis 串流的最低權限存取 IAM 角色。
- 使用 AWS 受管的 KMS 金鑰啟用 Kinesis 串流的伺服器端加密。
- 為 Kinesis 串流部署最佳作法 CloudWatch 警示。

AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 針對 NodeJS Lambda 函數啟用重複使用連線的功能。
- 啟用 X-Ray 追蹤。
- 啟用失敗處理功能：在功能錯誤時啟用二分點；設定預設的記錄保留時間上限 (24 小時)；設定預設的重試嘗試次數上限 (500)；以及在失敗時將 SQS 無效字母佇列部署為目的地。
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/AW-運動流-lambda

aws-lambda-dynamodb

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	aws_solutions_constructs.aws_lambda_dynamodb
打字稿	@aws-solutions-constructs/aws-lambda-dynamodb
Java	software.amazon.awsconstructs.services.lambdadynamodb

Overview

此 AWS 解決方案建構實作具有最低權限的 AWS Lambda 函數和 Amazon DynamoDB 表格。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToDynamoDBProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會造成錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
DynamoTable 道具？	dynamodb.TableProps	選擇性的使用者提供的道具，用於覆寫 DynamoDB 表格的預設道具
現有的表格？	dynamodb.Table	DynamoDB 表格物件的現有實體，同時提供這個和dynamoTableProps 會造成錯誤。
表格權限？	string	要授與 Lambda 函數的可選表格權限。您可以指定下列其中一個選項：All、Read、ReadWrite，或Write。

名稱	類型	描述
桌面環境變量名稱？	`string`	為 Lambda 函數設定之 DynamoDB 表格環境變數的選擇性名稱。
是否存在 VPC？	`ec2.IVpc`	應該部署此模式的選用現有 VPC。在 VPC 中部署時，Lambda 函數將使用 VPC 中的 ENI 來存取網路資源，並且將在 Amazon DynamoDB 的 VPC 中建立閘道端點。如果提供了現有的 VPC，則`deployVpc` 屬性不能`true`。這使用`ec2.IVpc`，以允許用戶端使用`ec2.Vpc.fromLookup()` 方法。
VPCProps？	`ec2.VpcProps`	可選的使用者提供的屬性，用於覆寫新 VPC 的預設屬性。`enableDnsHostnames` 、`enableDnsSupport` 、`natGateways` ，以及`subnetConfiguration` 是由模式設置的，因此此處提供的屬性的任何值都將被覆蓋。如果`deployVpc` 不是`true`那麼這個屬性將被忽略。

名稱	類型	描述
部署 vPC？	boolean	是否建立新的 VPC 基於vpcProps來部署此模式。將其設置為 true 將部署最小，最私有的 VPC 來運行該模式： • CDK 程式使用的每個可用區域內有一個隔離子網路 • enableDnsHostnames 和enableDns Support 都將設置為true 如果此屬性為true，然後existingVpc 無法指定。預設為 false。

模式性質

名稱	類型	描述
DynaMotion 表格	dynamodb.Table	傳回由樣式建立之 DynamoDB 表格的實體。
LambdaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。
vpc？	ec2.IVpc	返回模式使用的 VPC 上的接口（如果有的話）。這可能是由模式或提供給模式構造函數的 VPC 創建的 VPC。

預設設設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

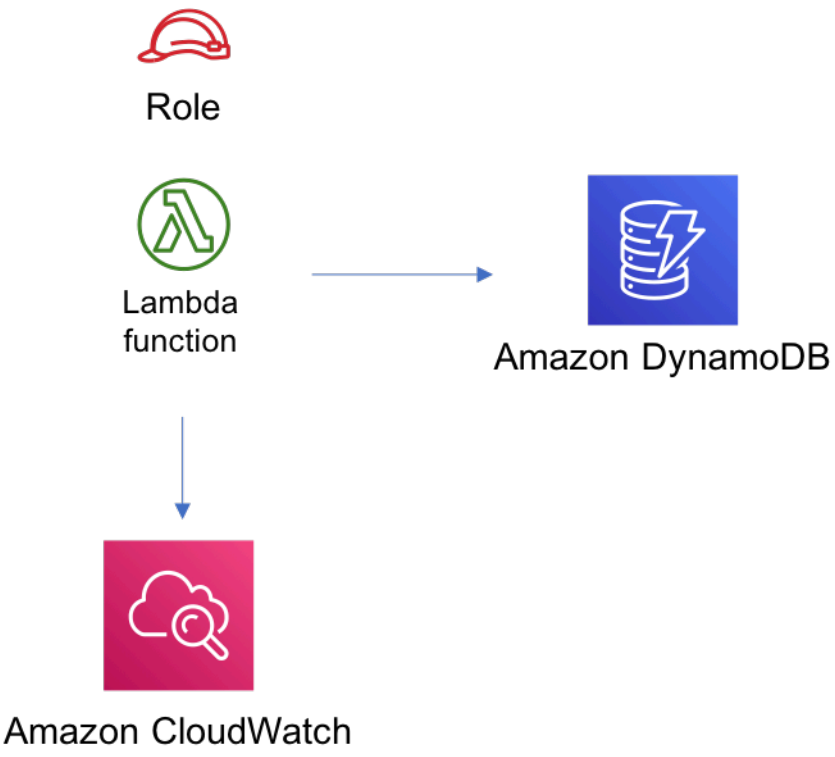
AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `DDB_TABLE_NAME` (default)
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Amazon DynamoDB 表

- 將 DynamoDB 表格的計費模式設定為隨選 (按請求付費)。
- 使用 AWS 受管的 KMS 金鑰啟用 DynamoDB 表格的伺服器端加密。
- 為 DynamoDB 表格建立名為「id」的分割區索引鍵。
- 刪除 CloudFormation 堆棧時保留表。
- 啟用連續備份和時間點復原。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aw-lambda-動態模式

aws-lambda-彈性搜索-基班納

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_lambda_elasticsearch_kibana`
 打字稿	`@aws-solutions-constructs/aws-lambda-elasticsearch-kibana`
 Java	`software.amazon.awsconstructs.services.lambdaelasticsearchkibana`

Overview

此 AWS 解決方案建構實作 AWS Lambda 函數和具有最低權限的 Amazon Elasticsearch Service 網域。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- id`string`
- 提案LambdaToElasticSearchAndKibanaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，同時提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
電子域名道具？	elasticsearch.CfnDomainProps	可選用的使用者提供的道具，以覆寫 Amazon Elasticsearch Service 的預設道具
domainName	string	Cognito 和 Amazon Elasticsearch Service 的域名
科尼托網域名稱？	string	選擇性 Cognito 網域名稱。如果有提供，它將用於 Cognito 網域，而domainName 將用於彈性搜索域。
創建雲端觀察器	boolean	是否要建立建議的警報。
網域端點虛擬名稱變數名稱？	string	為 Lambda 函數設定的 ElasticSearch 網域端點環境變數選擇性名稱。

模式性質

名稱	類型	描述
CloudwatchAlims？	cloudwatch.Alarm[]	傳回模式建立的一或多個 CloudWatch 警示的清單。

名稱	類型	描述
彈性搜尋網域	elasticsearch.CfnDomain	返回由圖案創建的 Elasticsearch 域的實例。
彈性搜尋網域角色	iam.Role	傳回 Elasticsearch 網域模式所建立的 IAM 角色執行個體。
IdentityPool	cognito.CfnIdentityPool	傳回由模式建立的 Cognito 身分識別集區的執行個體。
Lambda FAULT	lambda.Function	返回由模式創建的 Lambda 函數的實例。
userPool	cognito.UserPool	傳回由模式建立的 Cognito 使用者集區的執行個體。
UserPoolClient	cognito.UserPoolClient	傳回由模式建立的 Cognito 使用者集區用戶端的執行個體。

Lambda 功能

此模式需要 Lambda 函數，該函數可以從 DynamoDB 串流將資料張貼至 Elasticsearch 服務。提供範例函數這裡。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 針對 NodeJS Lambda 函數啟用重複使用連線的功能。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `DOMAIN_ENDPOINT` (default)
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

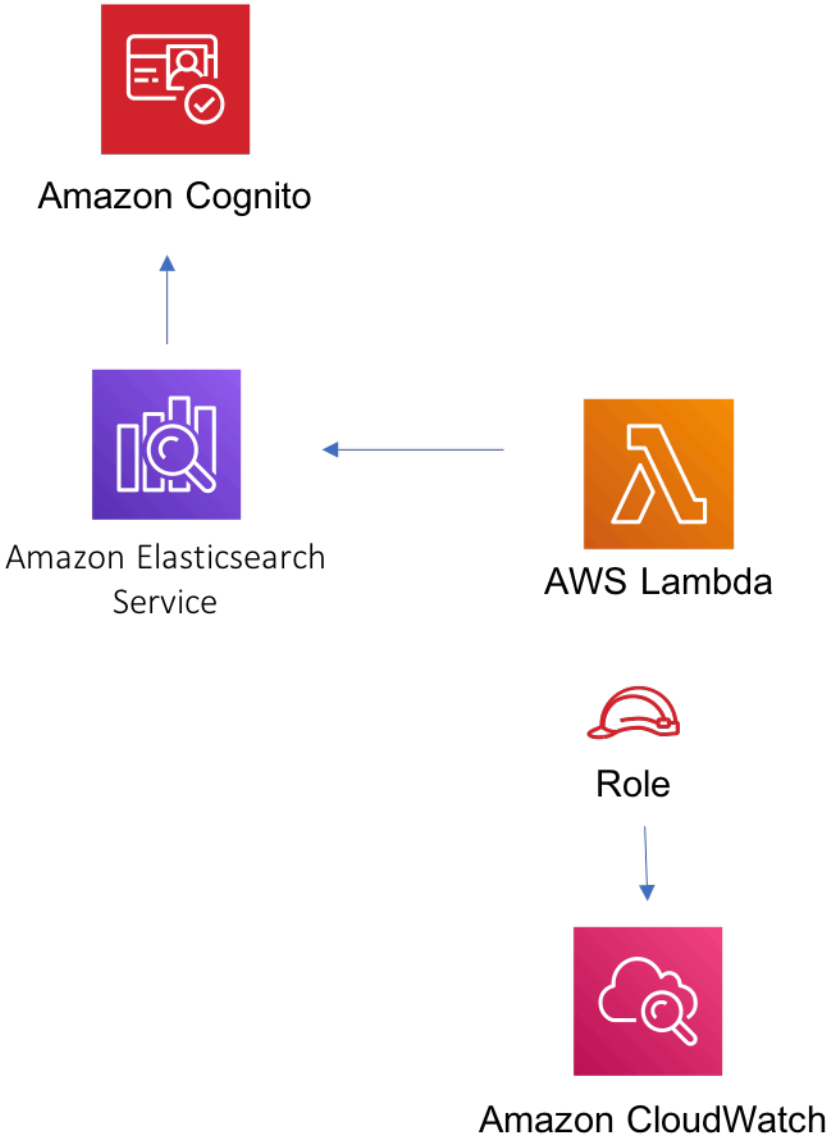
Amazon Cognito

- 設定「使用者集區」的密碼策略。
- 強制執行「使用者集區」的進階安全模式。

Amazon Elasticsearch Service

- 針對彈性搜尋網域部署最佳實務 CloudWatch 警示。
- 使用 Cognito 使用者集區保護 Kibana 儀表板存取權。
- 使用 AWS 受管的 KMS 金鑰為彈性搜尋網域啟用伺服器端加密。
- 啟用 Elastisearch 網域的節點對節點加密。
- 為 Amazon ES 網域設定叢集。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aw-lambda-彈性搜索-kibana

aws-lambda-3

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_lambda_s3`
TypeScript	`@aws-solutions-constructs/aws-lambda-s3`
Java	`software.amazon.awsconstructs.services.lambdas3`

Overview

此 AWS 解決方案建構實作連接到 Amazon S3 儲存貯體的 AWS Lambda函數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToS3Props

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，同時提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
現在的巴克托比？	s3.IBucket	S3 存儲桶對象的現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。
水桶道具？	s3.BucketProps	可選的使用者提供的屬性來覆寫儲存貯體的預設屬性。忽略existingBucketObj 提供。

名稱	類型	描述
儲存區權限？	`string[]`	可選的儲存貯體權限授與 Lambda 函數。可指定下列一或多個：`Delete`、`Put`、`Read`、`ReadWrite`、`Write`。
是否存在 VPC？	`ec2.IVpc`	應該部署此模式的選用現有 VPC。在 VPC 中部署時，Lambda 函數將使用 VPC 中的 ENI 來存取網路資源，並且會在 Amazon SQS 的 VPC 中建立一個介面端點。如果提供了現有的 VPC，`deployVpc` 屬性不能為`true`。這使用`ec2.IVpc`，以允許用戶端使用`ec2.Vpc.fromLookup()` 方法。
部署 vPC？	`boolean`	是否建立新的 VPC`vpcProps`來部署此模式。將此設為`true`將部署最小的，最私有的 VPC 來運行該模式： • CDK 程式使用的每個可用區域內有一個隔離子網路。 • `enableDnsHostnames` 和`enableDnsSupport` 都將設置為`true`。 如果此屬性為`true`，然後`existingVpc` 無法指定。預設為 `false`。

名稱	類型	描述
VPCProps？	ec2.VpcProps	可選的使用者提供的屬性，用於覆寫新 VPC 的預設屬性。enableDnsHostnames 、enableDnsSupport 、natGateways 和subnetConfiguration 是由模式設置的，因此此處提供的屬性的任何值都將被覆蓋。如果deployVpc 不是true那麼這個屬性將被忽略。
時段變數名稱？	string	為 Lambda 函數設定之 S3 儲存貯體環境變數的選擇性名稱。

模式性質

名稱	類型	描述
LambdaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。
S3 儲存貯體？	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。
vpc？	ec2.IVpc	返回模式使用的 VPC 的實例（如果有的話）。這可能是由模式或提供給模式構造函數的 VPC 創建的 VPC。

預設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 針對 NodeJS Lambda 函數啟用重複使用連線的功能。
- 啟用 X-Ray 追蹤
- 設定環境變數：
  - `S3_BUCKET_NAME` (default)
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Amazon S3 儲存貯體

- 設定 S3 儲存貯體的存取記錄。
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密。
- 打開 S3 存儲桶的版本控制。
- 不允許公開存取 S3 儲存貯體。
- 刪除 CloudFormation 堆疊時保留 S3 桶。
- 強制加密傳輸中的資料。
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存區。

Architecture

GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aw-lambda-3

aws-lambda 字符串參數

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_lambda_ssm_string_parameter`
 TypeScript	`@aws-solutions-constructs/aws-lambda-ssmstringparameter`
 Java	`software.amazon.awsconstructs.services.lambdassmstringparameter`

Overview

此 AWS 解決方案建構以最低權限實作 AWS Lambda 函數和 AWS Systems Manager Parameter Store 字串參數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- id`string`
- 提案LambdaToSsmstringparameterProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，同時提供這個和lambdaFunctionProps 會造成錯誤。

名稱	類型	描述
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 已提供。
存在字符串參數？	ssm.StringParameter	SSM 字串參數物件的現有執行個體，同時提供這個和stringParameterProps 會造成錯誤。
字符串參數道具？	ssm.StringParameterProps	可選的使用者提供的道具來覆寫 SSM 字串參數的預設道具。如果existingStringParameterObj 未設定，stringParameterProps 是必要的。唯一支援ssm.StringParameterProps.type 是STRING如果提供了不同的值，它將被覆蓋。
字符串參數變量名稱？	string	為 Lambda 函數設定的 SSM 字串參數環境變數的選擇性名稱。

名稱	類型	描述
是否存在 VPC？	ec2.IVpc	應該部署此模式的選用現有 VPC。在 VPC 中部署時，Lambda 函數將使用 VPC 中的 ENI 來存取網路資源，並且會在 VPC 中為 AWS Systems Manager 參數建立一個介面端點。如果提供了現有的 VPC，deployVpc 屬性不能為true。這會使用ec2.IVpc，以允許用戶端使用ec2.Vpc.fromLookup() 方法。
VPCProps？	ec2.VpcProps	可選的使用者提供的屬性，用於覆寫新 VPC 的預設屬性。enableDnsHostnames 、enableDnsSupport 、natGateways 和subnetConfiguration 是由模式設置的，因此此處提供的屬性的任何值都將被覆蓋。如果deployVpc 不是true那麼這個屬性將忽略。

名稱	類型	描述
部署 vPC？	boolean	是否建立新的 VPC 基於vpcProps來部署此模式。將此設為true將部署最小的，最私有的 VPC 來運行該模式： • CDK 程式使用的每個可用區域內有一個隔離子網路。 • enableDnsHostnames 和enableDns Support 都將設置為true。 如果此屬性已設為true，然後existingVpc 無法指定。預設為 false。
字串參數權限？	string	要授與 Lambda 函數的選用 SSM 字串參數權限。可指定下列任一項：Read、ReadWrite 。

模式屬性

名稱	類型	描述
Lambda Function	lambda.Function	傳回lambda.Function 由建構建立。
StringParameter	ssm.StringParameter	傳回ssm.StringParameter 由建構建立。
vpc？	ec2.IVpc	返回模式使用的 VPC 上的接口（如果有的話）。這可能是由

名稱	類型	描述
		模式或提供給模式構造函數的 VPC 創建的 VPC。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

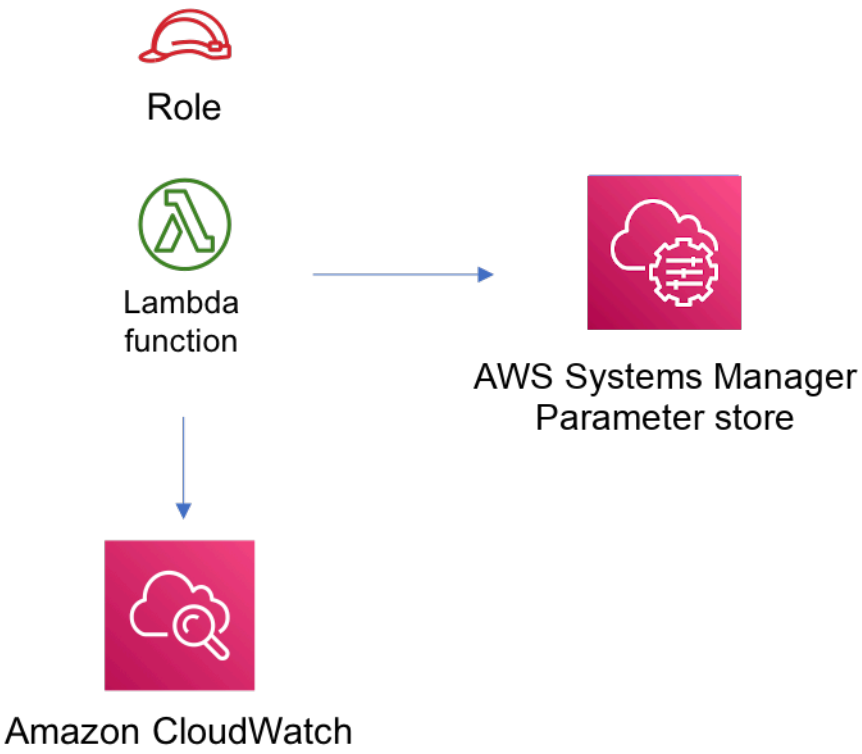
AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 針對 NodeJS Lambda 函數啟用重複使用連線的功能。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `SSM_STRING_PARAMETER_NAME` (default)
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Amazon AWS AWS Systems Manager Parameter Store String

- 啟用相關聯的 AWS Lambda 函數的唯讀存取權。
- 使用提供的值建立新的 SSM 字串參數。
- 刪除 CloudFormation 堆疊時，請保留 SSM 字串參數。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-lambda-字符串參數

aws-lambda-區域分析點

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	aws_solutions_constructs.aws_lambda_sagemakerendpoint
打字稿	@aws-solutions-constructs/aws-lambda-sagemakerendpoint
Java	software.amazon.awsconstructs.services.lambdasagemakerendpoint

Overview

此 AWS 解決方案建構實作連接到 Amazon Sagemaker 終端節點的 AWS Lambda 函數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToSagemakerEndpointProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會造成錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。
現在薩格馬克倫波因托比？	sagemaker.CfnEndpoint	要使用的選擇性現有 Sagemaker 登錄。同時提供這個和endpointProps 會造成錯誤。
模型道具？	sagemaker.CfnModelProps | any	使用者提供的性質，以取代「草繪機模型」的預設

名稱	類型	描述
		性質。至少要modelProps.primaryContainer 必須提供來創建一個模型。依預設，模式會建立具有最低所需權限的角色，但用戶端可以使用modelProps.executionRoleArn 。
端點配置道具？	sagemaker.CfnEndpointConfigProps	可選的使用者提供的性質，用於覆寫 Sagemaker 端點組態的預設性質。
端點道具？	sagemaker.CfnEndpointProps	可選的使用者提供的性質，用於取代「草繪器端點」的預設性質。
是否存在 VPC？	ec2.IVpc	應該部署此建構的選用現有 VPC。在 VPC 中部署時，Lambda 函數和 Sagemaker 端點將使用 VPC 中的 ENI 來存取網路資源。將在亞馬遜 Sagemaker 運行時和 Amazon S3 VPC 端點的 VPC 中創建一個接口端點。如果提供了現有的 VPC，則deployVpc 屬性不能為true。

名稱	類型	描述
VPCProps？	ec2.VpcProps	可選的使用者提供的屬性，用於覆寫新 VPC 的預設屬性。enableDnsHostnames 、enableDnsSupport 、natGateways 和subnetConfiguration 是由構造設置的，所以這裡提供的屬性的任何值都將被覆蓋。如果deployVpc 不是true，則此屬性會忽略。
部署 vPC？	boolean	是否建立新的 VPC 基於vpcProps來部署此模式。將此設為true將部署最小的，最私有的 VPC 來運行該模式： • CDK 程式使用的每個可用區域內有一個隔離子網路。 • enableDnsHostnames 和enableDnsSupport 都將設置為true。 如果此屬性已設為true，然後existingVpc 無法指定。預設為 false。
薩格馬克倫聲音變數名稱？	string	為 Lambda 函數設定的 SageMaker 端點環境變數的可選名稱。

模式屬性

名稱	類型	描述
LambdFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。
相模繪圖連接點	sagemaker.CfnEndpoint	傳回由樣式建立的「切割器端點」的例證。
薩格馬克點配置？	sagemaker.CfnEndpointConfig	返回由模式創建的 SageMaker EndpointConfig 的實例，如果existingSagemakerEndpointObj 未提供。
射影模型？	sagemaker.CfnModel	返回由圖案創建的 Sagemaker 模型的實例，如果existingSagemakerEndpointObj 未提供。
vpc？	ec2.IVpc	返回由模式創建的 VPC 的實例，如果deployVpc 是true，或者如果existingVpc 已提供。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

AWS Lambda 功能

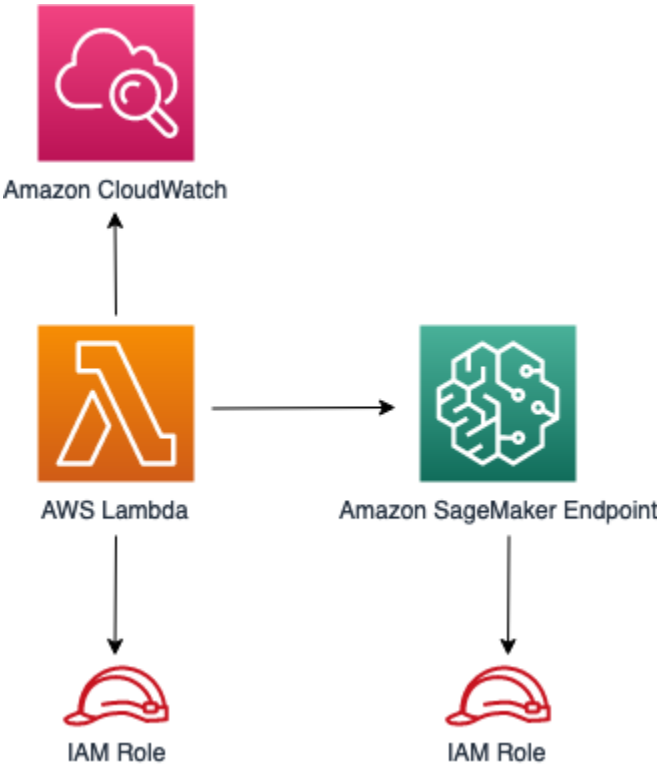
- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 允許函數為「推論」呼叫「草繪器」端點。
- 配置函數以存取 VPC 中部署 Sagemaker 端點的資源。
- 啟用 X-Ray 追蹤。
- 設定環境變數：

- SAGEMAKER_ENDPOINT_NAME (default)
- AWS_NODEJS_CONNECTION_REUSE_ENABLED（對於節點 10.x 和更高版本的函數）

Amazon SagMaker 端點

- 配置有限的權限以創建 Sagemaker 資源。
- 部署「切割器」模型、端點設定和端點。
- 設定要在 VPC 中部署的 Sagemaker 端點。
- 部署 S3 VPC 端點和模具執行階段 VPC 介面。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案結構/aws-lambda-模擬點

aws-lambda 秘密管理器

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_lambda_secretsmanager`
 打字稿	`@aws-solutions-constructs/aws-lambda-secretsmanager`
 Java	`software.amazon.awsconstructs.services.lambdasecretsmanager`

Overview

此 AWS 解決方案建構實作 AWS Lambda 函數和 AWS Secrets Manager 密碼，並具有最低特權權限。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToSecretsmanagerProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	用戶提供的道具來覆蓋 Lambda 函數的默認道具。
秘密道具	secretsmanager.SecretProps	可選的使用者提供的道具來覆寫 Secrets Manager 的預設道具。

名稱	類型	描述
現在的秘密?	secretsmanager.Secret	Secrets Manager 對象的現有實例，如果這被設置，則secretProps 會被忽略。
格蘭特存取權？	boolean	Lambda 函數之密碼的選擇性寫入存取權限 (預設為唯讀)。
秘密變數名稱？	string	為 Lambda 函數設定之 Secrets Manager 秘密環境變數的選擇性名稱。
是否存在 VPC？	ec2.IVpc	應該部署此模式的選用現有 VPC。在 VPC 中部署時，Lambda 函數將使用 VPC 中的 ENI 來存取網路資源，並且會在 AWS Secrets Manager 的 VPC 中建立一個介面端點。如果提供了現有的 VPC，deployVpc 屬性不能為true。這使用ec2.IVpc以允許用戶端提供存在於堆疊外部的 VPC，使用ec2.Vpc.fromLookup() 方法。
虛擬電腦產品？	ec2.VpcProps	可選的使用者提供的屬性，用於覆寫新 VPC 的預設屬性。enableDnsHostnames 、enableDnsSupport 、natGateways ，以及subnetConfiguration 是由模式設置的，因此此處提供的屬性的任何值都將被覆蓋。如果deployVpc 不是true那麼這個屬性將被忽略。

名稱	類型	描述
部署 vPC？	boolean	是否建立新的 VPC 基於vpcProps來部署這種模式。將此設為true將部署最小的，最私有的 VPC 來運行該模式： • CDK 程式使用的每個可用區域內有一個隔離的子網路 • enableDnsHostnames 和enableDns Support 都將被設置為true 如果此屬性為true，然後existingVpc 無法指定。預設為 false。

模式性質

名稱	類型	描述
Lambda	lambda.Function	傳回lambda.Function 由建構建立。
秘密	secretsmanager.Secret	傳回secretsmanager.Secret 由建構建立。
vpc？	ec2.IVpc	返回模式使用的 VPC 上的接口（如果有的話）。這可能是由模式或提供給模式構造函數的 VPC 創建的 VPC。

預設設設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

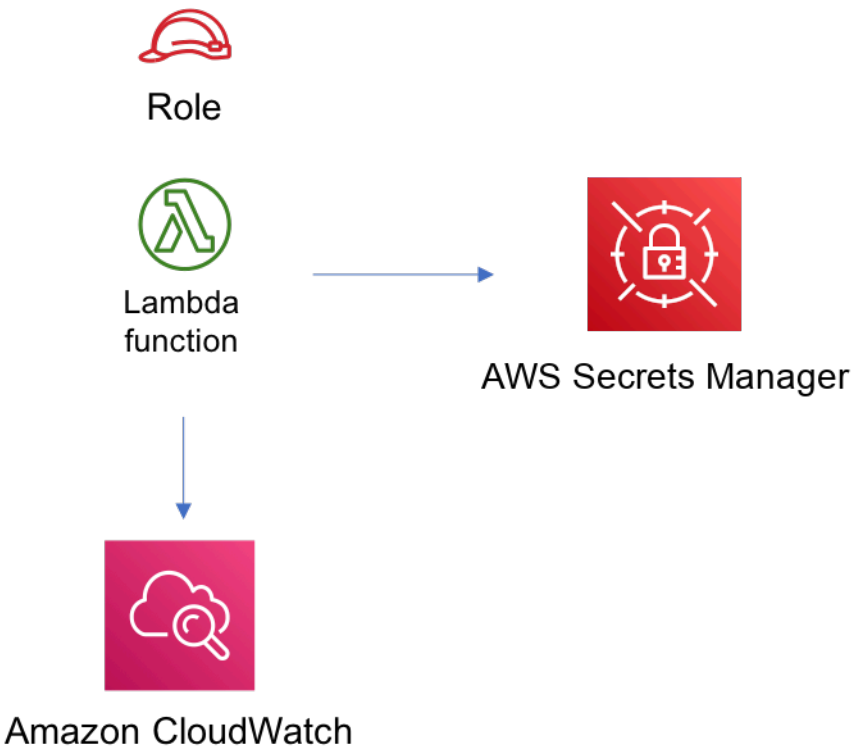
AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - （默認）秘密 _ARN 包含由 CDK 返回的秘密 ARN秘書屬性
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Amazon Secrets Manager 秘密

- 為相關聯的 AWS Lambda 函數啟用唯讀存取
- 使用帳戶和區域的預設 KMS 金鑰啟用伺服器端加密
- 建立新密碼：
  - (預設) 隨機名稱
  - （默認）隨機值
- 刪除 CloudFormation 堆疊時保留秘密

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案構造/aws-lambda-秘密管理器

aws-lambda-

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_lambda_sns`
打字稿	`@aws-solutions-constructs/aws-lambda-sns`
Java	`software.amazon.awsconstructs.services.lambdasns`

Overview

此 AWS 解決方案建構實作連接到 Amazon SNS 主題的 AWS Lambda 函數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToSnsProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性，可覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 已提供。
現在的托比科比？	sns.Topic	SNS 主題對象的現有實例，提供這個和topicProps 會導致錯誤。
主題道具？	sns.TopicProps	選用的使用者提供的屬性，可覆寫 SNS 主題的預設屬性。
是否存在 VPC？	ec2.IVpc	應該部署此模式的選用現有 VPC。在 VPC 中部署時，Lambda 函數將使用 VPC 中的 ENI 來存取網路資源，並且會在 Amazon SQS 的 VPC 中建立一個介面端點。如果提供了現有的 VPC，deployVpc 屬性不能為true。這使用ec2.IVpc以允許用戶端提供存在於堆疊外

名稱	類型	描述
		部的 VPC，使用ec2.Vpc.fromLookup() 方法。
部署 vPC？	boolean	是否建立新的 VPCvpcProps來部署這種模式。將此設為true將部署最小的，最私有的 VPC 來運行該模式： • CDK 程式使用的每個可用區域中有一個隔離子網路。 • enableDnsHostnames 和enableDnsSupport 都將被設置為true。 如果此屬性為true，然後existingVpc 無法指定。預設為 false。
虛擬電腦產品？	ec2.VpcProps	可選的使用者提供的屬性，用於覆寫新 VPC 的預設屬性。enableDnsHostnames 、enableDnsSupport 、natGateways 和subnetConfiguration 是由模式設置的，因此此處提供的屬性的任何值都將被覆蓋。如果deployVpc 不是true那麼這個屬性將被忽略。
主題環境變數名稱？	string	為 Lambda 函數設定的 SNS 主題 ARN 環境變數的選擇性名稱。

名稱	類型	描述
主題名稱環境變數名稱？	`string`	為 Lambda 函數設定的 SNS 主題名稱環境變數的選擇性名稱。

模式性質

名稱	類型	描述
Lambda	`lambda.Function`	返回由模式創建的 Lambda 函數的實例。
snsTopic	`sns.Topic`	傳回由模式建立的 SNS 主題的實例。
vpc？	`ec2.IVpc`	返回模式使用的 VPC 的實例（如果有的話）。這可能是由模式或提供給模式構造函數的 VPC 創建的 VPC。

預設設定

開箱即用的構造實現沒有任何覆蓋將設置以下默認值：

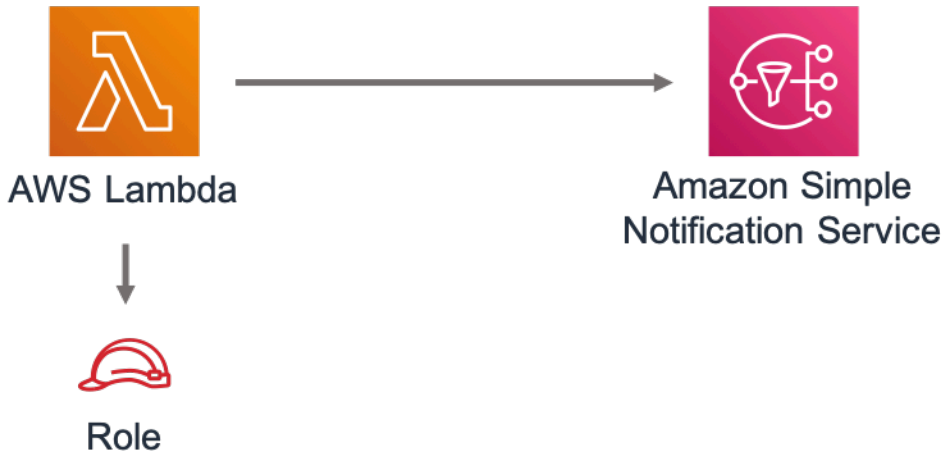
AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `SNS_TOPIC_NAME` (default)
  - `SNS_TOPIC_ARN` (default)
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Amazon SNS 主題

- 設定 SNS 主題的最低權限存取權限。
- 使用 AWS 受管 KMS 金鑰啟用伺服器端加密。
- 強制加密傳輸中的資料。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-lambda-s

aws-lambda-資料

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_lambda_sqs`
TypeScript	`@aws-solutions-constructs/aws-lambda-sqs`
Java	`software.amazon.awsconstructs.services.lambdasqs`

Overview

此 AWS 解決方案建構實作連接到 Amazon SQS 佇列的 AWS Lambda 函數。

下面是 TypeScript 中的最小可部署模式定義：

```
 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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToSqsProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯	lambda.Function	要使用的選用現有 Lambda 函數，而不是預設函數。同時提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性，可覆寫 Lambda 函數的預設屬性。
是否存在佇列中？	sqs.Queue	選擇性的現有 SQS 佇列，而不是預設佇列。同時提供這個和queueProps 會導致錯誤。
佇列道具？	sqs.QueueProps	選擇性的使用者提供的特性，用來覆寫 SQS 佇列的預設特性。
是否啟用佇列清除？	boolean	是否授與其他權限給 Lambda 函數，使其能夠清除 SQS 佇列。預設為 false。
部署死亡佇列？	boolean	無論是建立輔助佇列做為無效字母佇列。預設為 true。
死亡排隊道具？	sqs.QueueProps	可選的使用者提供的道具來覆寫死信佇列的預設道具。只有在deployDeadLetterQueue 屬性設為 true。

名稱	類型	描述
maxReceiveCount	number	訊息移到無效字母佇列之前，需交付佇列的次數。預設為 15。
是否存在 VPC？	ec2.IVpc	應該部署此模式的選用現有 VPC。部署到 VPC 時，Lambda 函數將使用 VPC 中的 ENI 來存取網路資源，並且會在 Amazon SQS 的 VPC 中建立一個介面端點。如果提供了現有的 VPC，deployVpc 屬性不能為true。一個ec2.IVpc用來允許用戶端提供存在於堆疊外部的 VPC，使用ec2.Vpc.fromLookup() 方法。
部署 vPC？	boolean	是否建立新的 VPC 基於vpcProps來部署這種模式。將此設為true將部署最小的，最私有的 VPC 來運行該模式： • CDK 程式使用的每個可用區域內有一個隔離子網路 • enableDnsHostnames 和enableDns Support 都將被設置為true 如果此屬性為true，然後existingVpc 無法指定。預設為 false。

名稱	類型	描述
虛擬電腦產品？	ec2.VpcProps	可選的使用者提供的屬性，用於覆寫新 VPC 的預設屬性。enableDnsHostnames 、enableDnsSupport 、natGateways ，以及subnetConfiguration 是由模式設置的，因此此處提供的屬性的任何值都將被覆蓋。如果deployVpc 不true，那麼此屬性將被忽略。
佇列環境變數名稱？	string	為 Lambda 函數設定的 SQS 佇列 URL 環境變數的選擇性名稱。

模式屬性

名稱	類型	描述
死機隊列？	sqs.Queue	返回由模式創建的死信隊列的實例，如果一個被部署。
Lambda Function	lambda.Function	返回由模式創建的 Lambda 函數的實例。
平方	sqs.Queue	返回由模式創建的 SQS 隊列的實例。
vpc？	ec2.IVpc	返回由模式創建或使用的 VPC 的實例（如果有的話）。這可能是由模式或提供給模式構造函數的 VPC 創建的 VPC。

預設設定

開箱即用的構造實現沒有任何覆蓋將設置以下默認值：

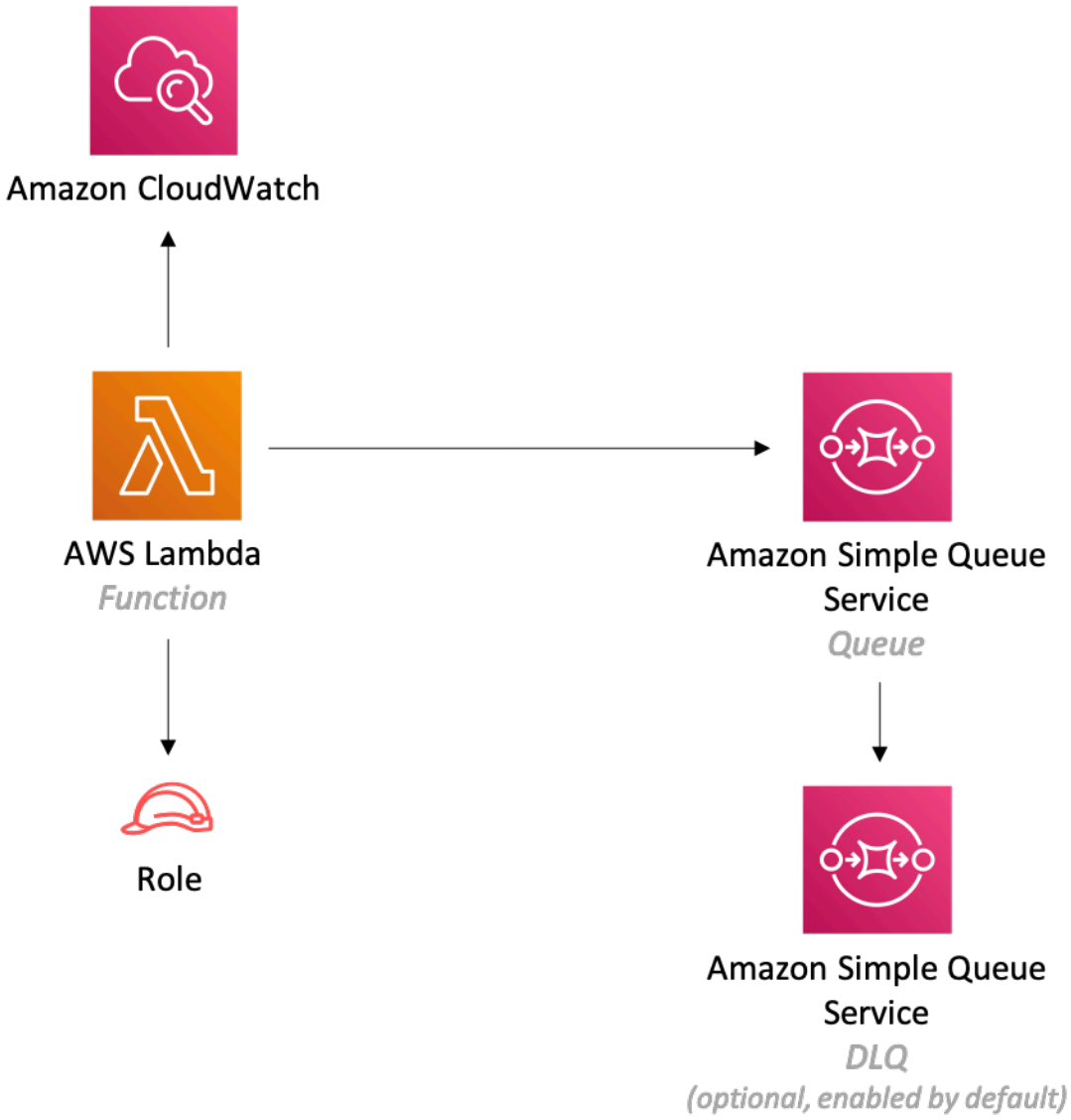
AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 允許函數只將訊息傳送至佇列 (可以使用`enableQueuePurge`屬性）。
- 啟用 X-Ray 追蹤
- 設定環境變數：
  - `SQS_QUEUE_URL`
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Amazon SQS 佇列

- 為來源 SQS 佇列建立 SQS 無效字母佇列。
- 使用 AWS 受管 KMS 金鑰為來源 SQS 佇列啟用伺服器端加密。
- 強制加密傳輸中的資料。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aw-lambda-Q

aw-拉姆達-Q-拉姆達

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_lambda_sqs_lambda`
 TypeScript	`@aws-solutions-constructs/aws-lambda-sqs-lambda`
 Java	`software.amazon.awsconstructs.services.lambdasqslambda`

Overview

此 AWS 解決方案建構模式實作 (1) 設定為將訊息傳送到佇列的 AWS Lambda 函數；(2) Amazon SQS 佇列；以及 (3) 設定為使用佇列中訊息的 AWS Lambda 函數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToSqsToLambdaProps

模式建立道具

名稱	類型	描述
現在的生產商是蘭姆達布公司？	lambda.Function	選擇性的現有 Lambda 函數，用來取代傳送訊息至佇列的預設函數。同時提供這個和 producerLambdaFunctionProps 會導致錯誤。
生產者拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性，可覆寫產生者 Lambda 函數的預設屬性。
是否存在佇列中？	sqs.Queue	選擇性的現有 SQS 佇列，而不是預設佇列。同時提供這個和 queueProps 會導致錯誤。

名稱	類型	描述
佇列道具？	sqs.QueueProps	選擇性的使用者提供的特性來覆寫 SQS 佇列的預設特性。同時提供這個和existingQueueObj 會導致錯誤。
部署死亡佇列？	boolean	無論建立輔助佇列做為無效字母佇列。預設為 true。
死亡排隊道具？	sqs.QueueProps	可選的使用者提供的道具來覆寫死信佇列的預設道具。只有在deployDeadLetterQueue 屬性已設為true。
maxReceiveCount？	number	訊息移到無效字母佇列之前，需交付佇列的次數。預設為 15。
現在還有消費者拉姆達寶？	lambda.Function	選用的現有 Lambda 函數，而不是用來從佇列中擷取/取用訊息的預設函數。同時提供這個和consumerLambdaFunctionProps 會導致錯誤。
消費者拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性，可覆寫取用者 Lambda 函數的預設屬性。
佇列環境變數名稱？	string	為產生者 Lambda 函數設定的 SQS 佇列 URL 環境變數的選擇性名稱。

模式屬性

名稱	類型	描述
消費者拉姆達	lambda.Function	返回由模式創建的消費者 Lambda 函數的實例。
死機隊列？	sqs.Queue	返回由模式創建的死信隊列的實例，如果一個被部署。
生產商 LambDafer	lambda.Function	返回由模式創建的生產者 Lambda 函數的實例。
平方	sqs.Queue	返回由模式創建的 SQS 隊列的實例。

預設定

此構造的開箱即用實現（沒有任何覆蓋的屬性）將堅持以下默認值：

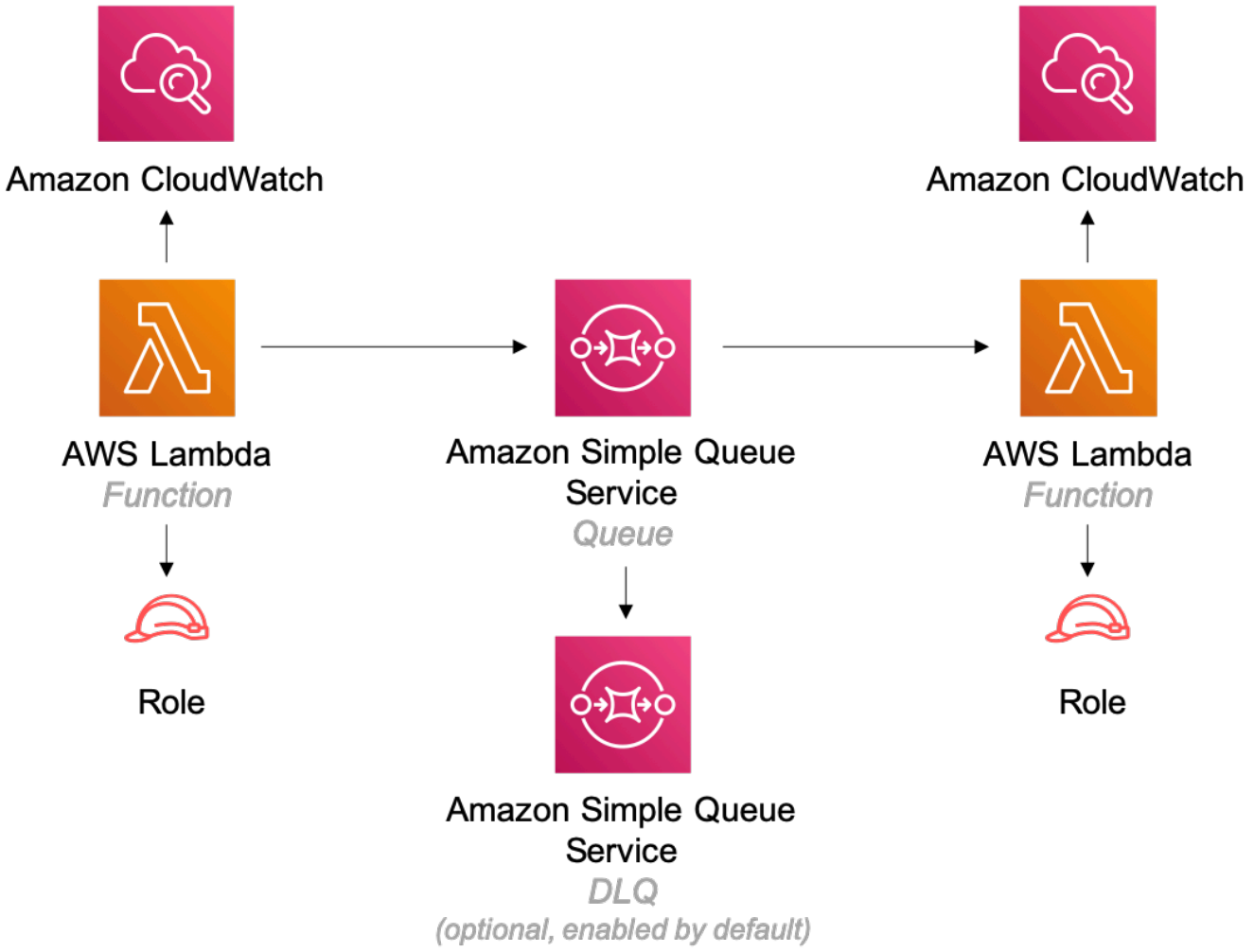
AWS Lambda 函數

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 啟用 NodeJS Lambda 函數的持續作用中重複使用連線。
- 啟用 X-Ray 追蹤
- 設定環境變數：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（對於節點 10.x 和更高版本的函數）

Amazon SQS 佇列

- 為主佇列部署無效字母佇列。
- 使用 AWS 受管 KMS 金鑰為主佇列啟用伺服器端加密。
- 強制加密傳輸中的資料

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aw-lambda-q-lambda

aws-lambda 步驟函數

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_lambda_step_function`
 打字稿	`@aws-solutions-constructs/aws-lambda-step-function`
 Java	`software.amazon.awsconstructs.services.lambdastepfunction`

Overview

此 AWS 解決方案建構實作連接到 AWS 步驟函數的 AWS Lambda 函數。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案LambdaToStepFunctionProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
斯塔特阿奇內道具	sfn.StateMachineProps	用戶為 SFN.Statemachine 提供的道具。
創建雲端觀察器	boolean	是否要建立建議的警報。

名稱	類型	描述
記錄群組道具？	logs.LogGroupProps	可選的使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。
系統環境變數名稱	string	為產生者 Lambda 函數設定的 Step Functions 狀態機環境變數的選擇性名稱。

模式性質

名稱	類型	描述
雲端觀察	cloudwatch.Alarm[]	傳回模式建立的一或多個 CloudWatch Logs 警示的清單。
LambDaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。
StateMachine	sfn.StateMachine	返回由模式創建的狀態機的實例。
台北市內洛集團	logs.LogGroup	傳回狀態機器模式所建立之日誌群組的執行個體。

預設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

AWS Lambda 功能

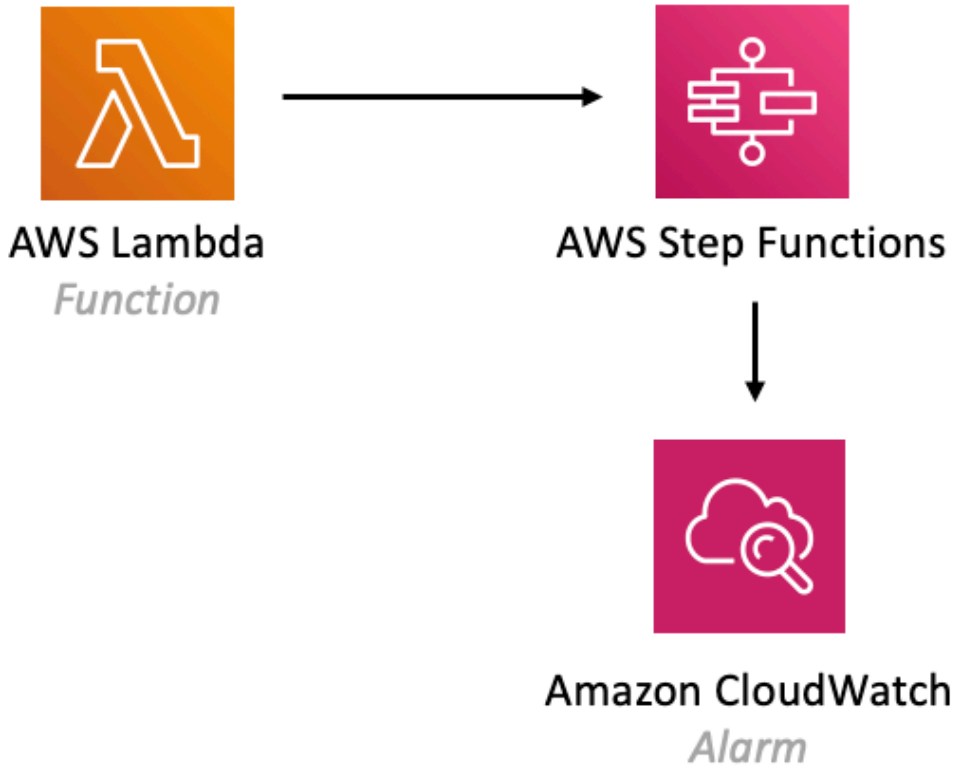
- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 啟用 NodeJS Lambda 函數的持續作用中重複使用連線。
- 啟用 X-Ray 追蹤。
- 設定環境變數：

- `STATE_MACHINE_ARN` (default)
- `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

AWS Step Functions 狀態機

- 為 AWS Step Functions 狀態機器部署最佳實務 CloudWatch 警示。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws 解決方案構造/aws-lambda 步驟函數

差異-3 蘭姆達

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_s3_lambda`
 TypeScript	`@aws-solutions-constructs/aws-s3-lambda`
 Java	`software.amazon.awsconstructs.services.s3lambda`

Overview

此 AWS 解決方案建構實作了連接到 AWS Lambda 函數的 Amazon S3 儲存貯體。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案S3ToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，提供這個和lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。

名稱	類型	描述
現在的巴克托比？	s3.Bucket	S3 桶對象的現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。
水桶道具？	s3.BucketProps	可選的使用者提供的屬性來覆寫儲存貯體的預設屬性。忽略existingBucketObj 提供。
S3 活動推廣道具？	S3EventSourceProps	可選的使用者提供的道具，以覆寫 S3EventProp 的預設道具

模式性質

名稱	類型	描述
Lambda Function	lambda.Function	返回由模式創建的 Lambda 函數的實例。
S3 儲存貯體？	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon S3 儲存貯體

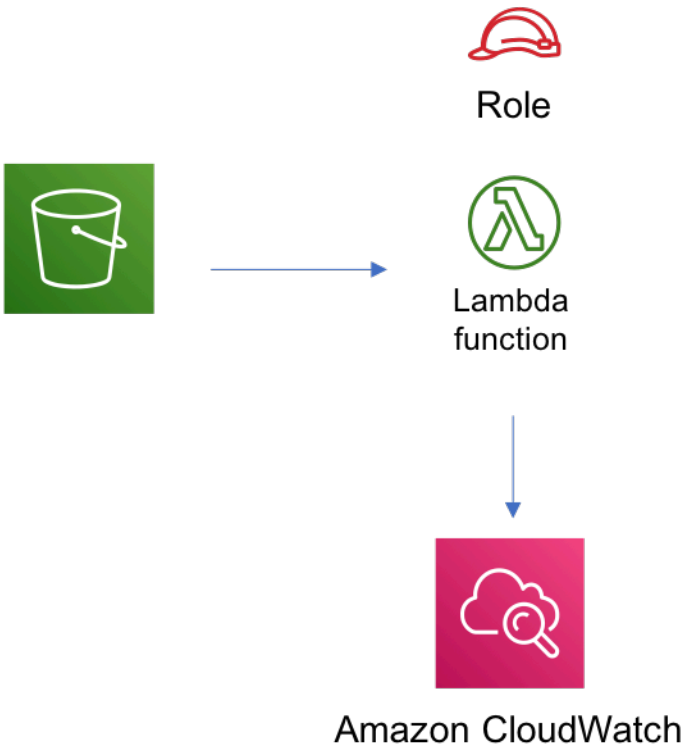
- 設定 S3 儲存貯體的存取記錄。
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密。
- 打開 S3 存儲桶的版本控制。
- 不允許公開存取 S3 儲存貯體。

- 刪除 CloudFormation 堆疊時保留 S3 桶。
- 強制加密傳輸中的資料。
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存區。

AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aws-3 lambda

AWS-3-數據庫

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_s3_sqs`
 TypeScript	`@aws-solutions-constructs/aws-s3-sqs`
 Java	`software.amazon.awsconstructs.services.s3sqs`

Overview

此 AWS 解決方案建構實作了一個 Amazon S3 儲存貯體，該儲存貯體配置為將通知傳送到 Amazon SQS 佇列。

下面是 TypeScript 中的最小可部署模式定義：

```
import { S3ToSqs } from "@aws-solutions-constructs/aws-s3-sqs";

new S3ToSqs(stack, 'S3ToSQSPattern', {});
```

Initializer

```
new S3ToSqs(scope: Construct, id: string, props: S3ToSqsProps);
```

參數

- scopeConstruct
- idstring
- 提案S3ToSqsProps

模式建立道具

名稱	類型	描述
現在的巴克托比？	s3.Bucket	S3 存儲桶對象的現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。
水桶道具？	s3.BucketProps	可選的使用者提供的道具來覆寫 S3 儲存貯體的預設道具。
三個活動類型？	s3.EventType[]	將觸發通知的 S3 事件類型。預設為 s3.EventType.OBJECT_CREATED 。
S3 事件過濾器？	s3.NotificationKeyFilter[]	S3 物件索引鍵篩選規則，以判斷哪些物件會觸發此事件。如果未指定，則不會套用任何篩選規則。

名稱	類型	描述
是否存在佇列中？	sqs.Queue	選擇性的現有 SQS 佇列，而不是預設佇列。同時提供這個和 queueProps 會導致錯誤。如果 SQS 佇列已加密，則用於加密的 KMS 金鑰必須是客戶管理的 CMK。
佇列道具？	sqs.QueueProps	選擇性的使用者提供的特性，用來覆寫 SQS 佇列的預設特性。忽略existingQueueObj 提供。
死亡排隊道具？	sqs.QueueProps	可選的使用者提供的道具來覆寫死信佇列的預設道具。只有在deployDeadLetterQueue 屬性設為 true。
部署死亡佇列？	boolean	無論建立次要佇列做為無效字母佇列。預設為 true。
maxReceiveCount	number	訊息移到無效字母佇列之前，需交付佇列的次數。預設為 15。
使用客戶管理的金鑰啟用加密？	boolean	是否使用 KMS 金鑰，由此 CDK 應用程式管理或匯入。如果匯入加密金鑰，則必須在encryptionKey 屬性為此建構。
encryptionKey？	kms.Key	選用的現有加密金鑰，而非預設加密金鑰。
加密密鑰道具？	kms.KeyProps	選用的使用者提供的屬性，可覆寫加密金鑰的預設屬性。

模式性質

名稱	類型	描述
平方	sqs.Queue	返回由模式創建的 SQS 隊列的實例。
死機隊列？	sqs.Queue	返回由模式創建的死信隊列的實例，如果一個被部署。
encryptionKey	kms.IKey	返回由模式創建的加密密鑰的實例。
S3 儲存貯體？	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon S3 儲存貯體

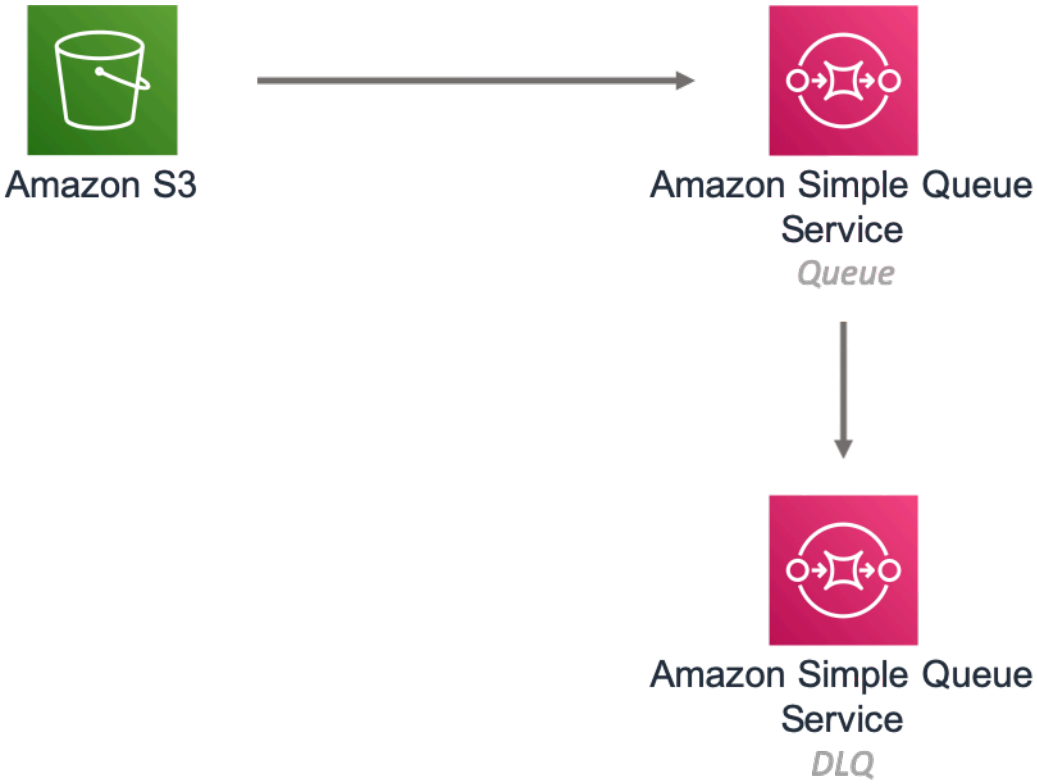
- 設定 S3 儲存貯體的存取記錄
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密
- 打開 S3 存儲桶的版本控制
- 不允許公開存取 S3 儲存貯體
- 刪除 CloudFormation 堆疊時保留 S3 桶
- 強制加密傳輸中的資料
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存空間

Amazon SQS 佇列

- 設定 SQS 佇列的最低權限存取權限
- 為來源 SQS 佇列建立 SQS 佇列

- 使用客戶管理 KMS 金鑰為 SQS 佇列啟用伺服器端加密
- 強制加密傳輸中的資料

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-結構/aws-3-

AWS-3 步驟函數

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_s3_step_function`
 打字稿	`@aws-solutions-constructs/aws-s3-step-function`
 Java	`software.amazon.awsconstructs.services.s3stepfunction`

Overview

此 AWS 解決方案建構實作了連接到 AWS 步驟函數的 Amazon S3 儲存貯體。

> **Note**
>
> 此構造使用亞馬遜 EventBridge（Amazon CloudWatch Events）來觸發 AWS Step Functions。EventBridge 更具彈性，但使用 S3 事件通知觸發 Step Functions 的延遲較少，並且更具成本效益。如果成本和/或延遲是一個問題，您應該考慮部署`aws-s3-lambda`和`aws-lambda-stepfunctions`來代替這個構造。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案S3ToStepFunctionProps

模式建立道具

名稱	類型	描述
現在的巴克托比？	s3.IBucket	S3 桶對象的現有實例。如果這是提供的，那麼還提供bucketProps 是錯誤。
水桶道具？	s3.BucketProps	可選的使用者提供的屬性來覆寫儲存貯體的預設屬性。忽略existingBucketObj 提供。
斯塔特阿奇內道具	sfn.StateMachinePr ops	可選的用戶提供的道具來覆蓋 SFN.Statemachine 的默認道具。
事件道具？	events.RuleProps	可選的使用者提供的 Eventrule Prop 來覆寫預設值。

名稱	類型	描述
部署雲端追蹤？	boolean	是否在 AWS CloudTrail 中部署追蹤以在 Amazon S3 中記錄 API 事件。預設為 true。
創建雲端觀察器	boolean	是否要建立建議的警報。
記錄群組道具？	logs.LogGroupProps	選用使用者提供的道具，用於覆寫 CloudWatch Logs 日誌群組的預設道具。

模式性質

名稱	類型	描述
CloudTrail	cloudtrail.Trail	返回由模式創建的 Cloudrail 軌跡的實例。
雲軌道？	s3.Bucket	返回由用於存儲 Cloudrail 跟踪數據的模式創建的存儲桶的實例。
雲端記錄桶？	s3.Bucket	返回由 Cloudrail 跟踪使用的主存儲桶模式創建的日誌存儲桶的實例。
CloudwatchAlims	cloudwatch.Alarm[]	傳回模式建立的一或多個 CloudWatch Events 警示的清單。
S3 儲存貯體？	s3.Bucket	返回由模式創建的 S3 存儲桶的實例。
S3 記錄桶？	s3.Bucket	返回由 S3 存儲桶模式創建的日誌存儲桶的實例。

名稱	類型	描述
StateMachine	sfn.StateMachine	返回由模式創建的狀態機的實例。
斯塔蒂內洛集團	logs.LogGroup	傳回狀態機器模式所建立之日誌群組的執行個體。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon S3 儲存貯體

- 設定 S3 儲存貯體的存取記錄。
- 使用 AWS 受管 KMS 金鑰啟用 S3 儲存貯體的伺服器端加密。
- 打開 S3 存儲桶的版本控制。
- 不允許公開存取 S3 儲存貯體。
- 刪除 CloudFormation 堆疊時保留 S3 桶。
- 強制加密傳輸中的資料。
- 套用生命週期規則，在 90 天後將非目前物件版本移至 Glacier 儲存區。

AWS CloudTrail

- 在 AWS CloudTrail 中配置追蹤，以在 Amazon S3 中記錄與建構建立的儲存貯體相關的 API 事件。

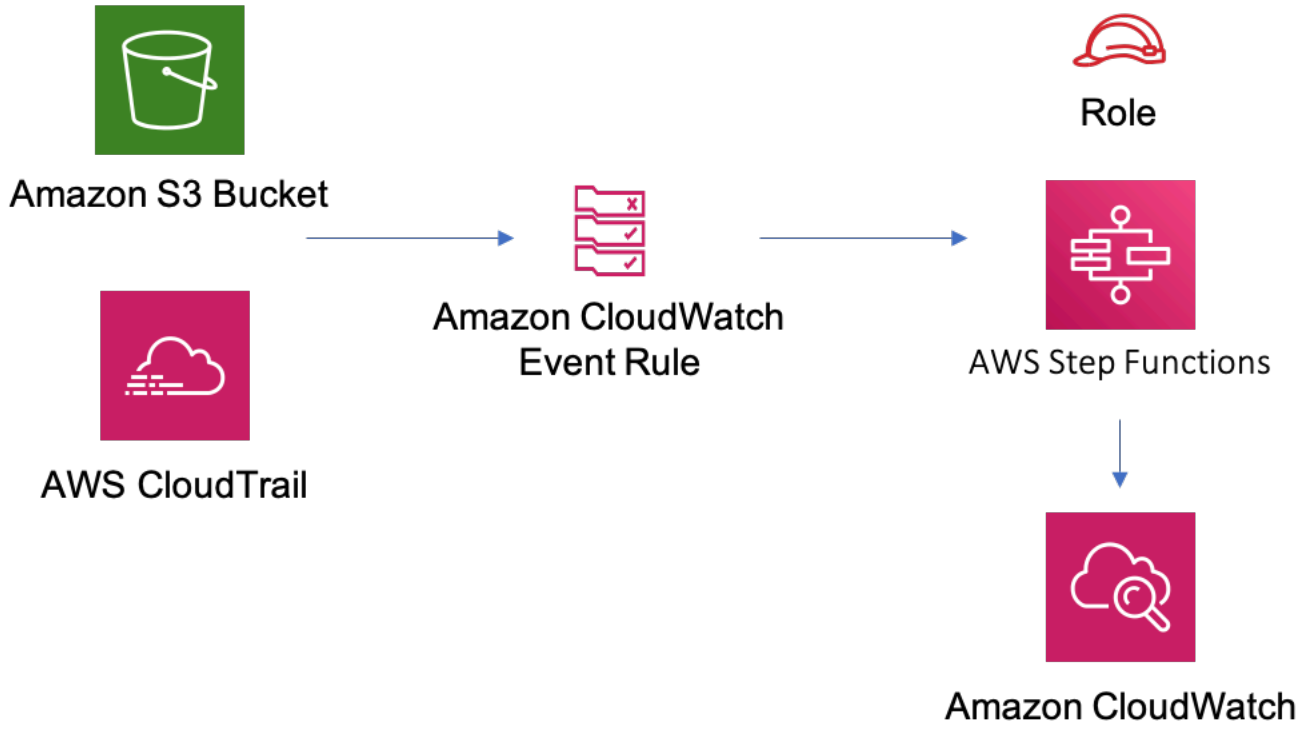
Amazon CloudWatch Events 規則

- 授與最低權限權限給 CloudWatch 事件以觸發 Lambda 函數。

AWS Step Function

- 啟用 API Gateway 的 CloudWatch 記錄。
- 為步驟功能部署最佳實務 CloudWatch 警示。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aws-3 步驟函數

差不多-蘭姆達

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_sns_lambda`
TypeScript	`@aws-solutions-constructs/aws-sns-lambda`
Java	`software.amazon.awsconstructs.services.snslambda`

Overview

此 AWS 解決方案建構實作連接到 AWS Lambda 函數的 Amazon SNS。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案SnsToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯	lambda.Function	Lambda 函數對象的現有實例，同時提供了這個和 lambdaFunctionProps 會導致錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性，可覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
現在的托比科比？	sns.Topic	SNS 主題對象的現有實例，同時提供這個和topicProps 會導致錯誤。
主題道具？	sns.TopicProps	選用的使用者提供的屬性，可覆寫 SNS 主題的預設屬性。

模式性質

名稱	類型	描述
Lambda FaFunction	lambda.Function	返回由模式創建的 Lambda 函數的實例。
snsTopic	sns.Topic	傳回由模式建立的 SNS 主題的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

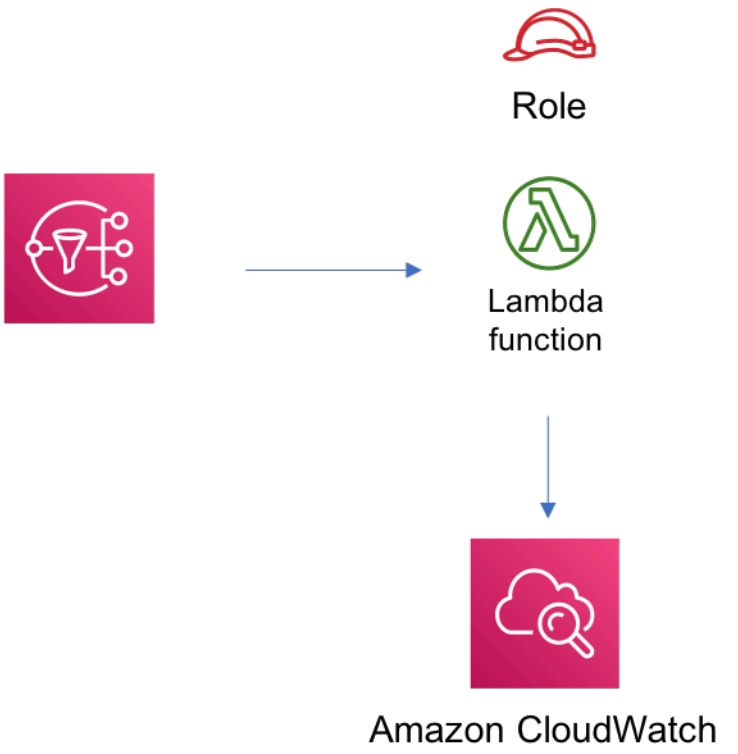
Amazon SNS 主題

- 設定 SNS 主題的最低權限存取權限。
- 使用 AWS 受管 KMS 啟用伺服器端加密。
- 強制加密傳輸中的資料。

AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 為 NodeJS Lambda 函數啟用具有持續作用的連線重複使用。
- 啟用 X-Ray 追蹤。
- 設定環境變數：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：	
	@aws-解決方案-構造/aws-南-蘭姆達

反射-反射-反射

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
 Python	`aws_solutions_constructs.aws_sns_sqs`
 打字稿	`@aws-solutions-constructs/aws-sns-sqs`
 Java	`software.amazon.awsconstructs.services.snssqs`

Overview

此 AWS 解決方案建構實作連接到 Amazon SQS 佇列的 Amazon SNS 主題。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案SnsToSqsProps

模式建立道具

名稱	類型	描述
現在的托比科比？	sns.Topic	SNS 主題對象的現有實例，提供這個和topicProps 會導致錯誤。
主題道具？	sns.TopicProps	選用的使用者提供的屬性，可覆寫 SNS 主題的預設屬性。

名稱	類型	描述
		忽略existingTopicObj 提供。
是否存在佇列中？	sqs.Queue	選擇性的現有 SQS 佇列，而不是預設佇列。同時提供這個和 queueProps 會導致錯誤。
佇列道具？	sqs.QueueProps	選擇性的使用者提供的特性來覆寫 SQS 佇列的預設特性。忽略existingQueueObj 提供。
部署死亡佇列？	boolean	無論建立次要佇列做為無效字母佇列。預設為 true。
死亡排隊道具？	sqs.QueueProps	可選的使用者提供的道具來覆寫死信佇列的預設道具。只有在deployDeadLetterQueue 屬性會設為 true。
maxReceiveCount？	number	訊息移到無效字母佇列之前，需交付佇列的次數。預設為 15。
使用客戶管理的金鑰啟用加密？	boolean	是否使用由此 CDK 應用程式管理或匯入的客戶管理的加密金鑰。如果匯入加密金鑰，則必須在encryptionKey 屬性為此建構。
encryptionKey	kms.Key	選用的現有加密金鑰，而非預設加密金鑰。
加密密鑰道具？	kms.KeyProps	選用的使用者提供的屬性，可覆寫加密金鑰的預設屬性。

模式性質

名稱	類型	描述
snsTopic	sns.Topic	傳回由模式建立的 SNS 主題的實例。
encryptionKey	kms.Key	返回由模式創建的加密密鑰的實例。
平方	sqs.Queue	返回由模式創建的 SQS 隊列的實例。
死機隊列？	sqs.Queue	返回由模式創建的死信隊列的實例，如果一個被部署。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

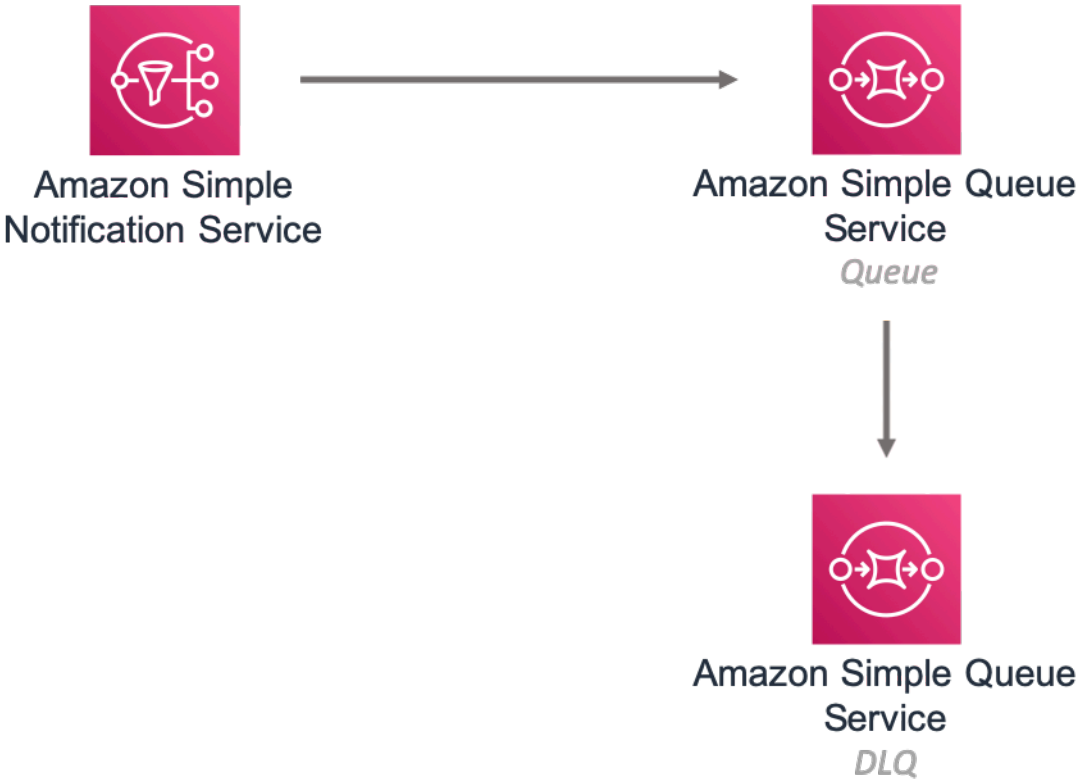
Amazon SNS 主題

- 設定 SNS 主題的最低權限存取權限。
- 使用 AWS 受管 KMS 金鑰啟用伺服器端加密。
- 強制加密傳輸中的資料。

Amazon SQS 佇列

- 設定 SQS 佇列的最低權限存取權限。
- 為來源 SQS 佇列建立無效字母佇列。
- 使用客戶管理的 KMS 金鑰啟用 SQS 佇列的伺服器端加密。
- 強制加密傳輸中的資料。

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-結構/awssssss-

Aws-秒-蘭姆達

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本控制模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

請注意： 為了確保功能正確，專案中的 AWS 解決方案建構套件和 AWS CDK 套件必須是相同的版本。

語言	套件
Python	`aws_solutions_constructs.aws_sqs_lambda`
TypeScript	`@aws-solutions-constructs/aws-sqs-lambda`
Java	`software.amazon.awsconstructs.services.sqslambda`

Overview

此 AWS 解決方案建構實作連接到 AWS Lambda 函數的 Amazon SQS 佇列。

下面是 TypeScript 中的最小可部署模式定義：

```
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);
```

參數

- scopeConstruct
- idstring
- 提案SqsToLambdaProps

模式建立道具

名稱	類型	描述
現在還有蘭姆達伯？	lambda.Function	Lambda 函數對象的現有實例，同時提供這個和lambdaFunctionProps 會造成錯誤。
拉姆針灸道具？	lambda.FunctionProps	選用的使用者提供的屬性來覆寫 Lambda 函數的預設屬性。忽略existingLambdaObj 提供。
是否存在佇列中？	sqs.Queue	選擇性的現有 SQS 佇列，而不是預設佇列。同時提供這個和queueProps 會造成錯誤。
佇列道具？	sqs.QueueProps	選擇性的使用者提供的特性來覆寫 SQS 佇列的預設特性。忽略existingQueueObj 提供。
部署死亡佇列？	boolean	無論建立要做為無效字母佇列使用的輔助佇列。預設為true。
死亡排隊道具？	sqs.QueueProps	可選的使用者提供的道具來覆寫死信佇列的預設道具。只有在deployDeadLetterQueue 屬性設為 true。

名稱	類型	描述
maxReceiveCount？	number	訊息移到無效字母佇列之前，需交付佇列的次數。預設為 15。

模式性質

名稱	類型	描述
死機隊列？	sqs.Queue	返回由模式創建的死信隊列的實例，如果一個被部署。
LambDAFAULT	lambda.Function	返回由模式創建的 Lambda 函數的實例。
平方	sqs.Queue	返回由模式創建的 SQS 隊列的實例。

預設設定

此模式的開箱即用實現沒有任何覆蓋將設置以下默認值：

Amazon SQS 佇列

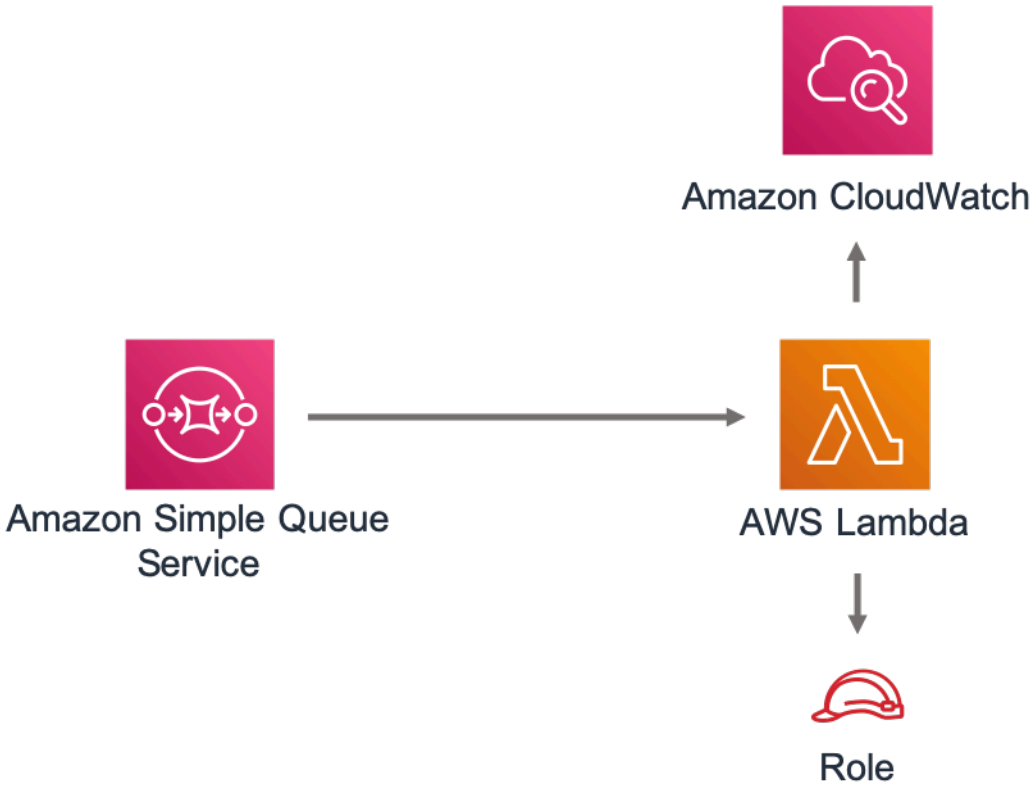
- 為來源 SQS 佇列建立 SQS 佇列。
- 使用 AWS 受管 KMS 金鑰為來源 SQS 佇列啟用伺服器端加密。
- 強制加密傳輸中的資料。

AWS Lambda 功能

- 設定 Lambda 函數的有限權限存取 IAM 角色。
- 針對 NodeJS Lambda 函數啟用重複使用連線的功能。
- 啟用 X-Ray 追蹤。
- 設定環境變數：

- AWS_NODEJS_CONNECTION_REUSE_ENABLED（對於節點 10.x 和更高版本的函數）

Architecture



GitHub

要查看此模式的代碼，創建/查看問題和提取請求，以及更多：

@aws-解決方案-構造/aws-q-lambda

core

STABILITY EXPERIMENTAL

所有類都在積極開發中，並且在任何未來版本中都會受到非向後兼容的更改或刪除。這些不受語義版本設定模型。這意味著，雖然您可以使用它們，但在升級到此軟件包的較新版本時，您可能需要更新源代碼。

核心程式庫包含 AWS 解決方案建立區塊。它定義了其他 AWS 解決方案建構中使用的核心類別。

AWS CDK 建構的預設屬性

核心程式庫會為 AWS 解決方案建構所使用的 AWS CDK 建構設定預設屬性。

例如，以下是 AWS 解決方案建構建立的 S3 儲存貯體建構的預設屬性片段。默認情況下，它將打開服務器端加密，存儲桶版本控制，阻止所有公共訪問並設置 S3 訪問日誌。

```
{
  encryption: s3.BucketEncryption.S3_MANAGED,
  versioned: true,
  blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
  removalPolicy: RemovalPolicy.RETAIN,
  serverAccessLogsBucket: loggingBucket
}
```

覆寫預設屬性

由 Core 庫設置的默認屬性可以被用戶提供的屬性覆蓋。例如，使用者可以覆寫 Amazon S3 區塊公用存取屬性以符合特定要求。

```
  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
    },
  });
```

屬性覆寫警告

當從核心庫的默認屬性被用戶提供的屬性覆蓋，構造將發出一個或多個警告消息到控制台突出顯示更改（S）。這些訊息旨在為使用者提供情境感知，並防止意外覆寫可能造成安全性風險。這些消息將在執行部署/構建相關命令時出現，包括cdk deploy、cdk synth、npm test、等

範例訊息：AWS_CONSTRUCTS_WARNING: An override has been provided for the property: BillingMode. Default value: 'PAY_PER_REQUEST'. You provided: 'PROVISIONED'.

切換覆寫警告

覆寫警告訊息預設為啟用，但可以使用overrideWarningsEnabled殼層變數。

- 若要明確關閉覆寫警告，執行export overrideWarningsEnabled=false。
- 若要明確開啟覆寫警告，執行export overrideWarningsEnabled=true。
- 若要還原為預設值，請執行unset overrideWarningsEnabled。

文件修訂

若要收到 AWS 解決方案建構更新的通知，請訂閱 RSS 摘要。

update-history-change	update-history-description	update-history-date
已更新內容	添加了 aws-lambda 字符串參數模式。其他次要內容更新。	2021 年 5 月 27 日
已更新內容	添加了 aws-lambda 秘密管理器模式。其他次要內容更新。	2021 年 5 月 12 日
已更新內容	屬性會更新以選取 *-lambda 模式。其他次要內容更新。	2021 年 4 月 17 日
已更新內容	已修正 Python 使用者的逐步解說中的問題，以及包含 Lambda 函數之建構的更新屬性範例。	2021 年 3 月 30 日
已更新內容	樣式道具的次要修復/更新，以及選取樣式的預設設定。	2021 年 3 月 8 日
已更新內容	穿越內容的次要修復/更新。	2021 年 3 月 4 日
已更新內容	已新增`aws-lambda-sagemakerendpoint` 模式和更新選取的 Kinesis 火軟管模式的屬性。	2021 年 2 月 24 日
已更新內容	已新增`aws-kinesisstreams-gluejob` 模式和更新的逐步解說步驟。	2021 年 2 月 17 日
已更新內容	已更新的屬性`aws-cloudfront-*` 模式。	2021 年 2 月 9 日
已更新內容	為每個模式添加到 GitHub 的鏈接。	2021 年 2 月 5 日

已更新內容	已更新選取樣式的性質。	2021 年 2 月 1 日
已更新內容	更新了選取樣式的性質和預設設定的文件。	2021 年 1 月 4 日
已更新內容	新增了新的模式：aws-雲端前端媒體商店和 aws-s3-s。	2020 年 12 月 20 日
已更新內容	刪除了 aws-lambda 區域模式。	2020 年 11 月 17 日
已更新內容	新增了新的模式：aws-事件規則運動流，aws--事件規則運動火柄-3，以及 aws-lambda 處理器。	2020 年 10 月 27 日
已更新內容	更新以反映 aws-事件規則-ns 和 aws-事件規則-qs 模式中的突破性更改：類和接口名稱更改為帕斯卡案例。	2020 年 10 月 22 日
已更新內容	添加了 aws-apigateway 模式和 aws-運動流動火-3 模式; 對現有內容的其他次要更新。	2020 年 10 月 20 日
已更新內容	添加了 aws-apigateway 物聯網模式; 對現有內容的其他小更新。	2020 年 10 月 7 日
已更新內容	更新了最小可部署的病毒碼程式碼片段，以及所有病毒碼的最佳作法預設值。	2020 年 10 月 5 日
已更新內容	更新了 aws-運動流-lambda 模式的屬性，以反映突破性更改。	2020 年 9 月 14 日
已更新內容	對穿越的第二部分進行次要修復。	2020 年 9 月 10 日

已更新內容	添加了 aws-apigateway 運動流，aws--事件-規則-規則-和 aws--事件-規則-s 模式。	2020 年 9 月 10 日
已更新內容	新增 aws-sns-sqs 模式；更新所有 SNS 模式；更新次要的印刷修正。	2020 年 9 月 2 日
已更新內容	修復了 aws-sqs-lambda 模式的模塊名稱。	2020 年 8 月 31 日
已更新內容	修復了 Python 模塊名稱的 aws-Dynamodb 流-羊肉-彈性搜索-基班納模式。	2020 年 8 月 31 日
已更新內容	已更新 Lambda 模式的預設值；其他次要更新。	2020 年 8 月 27 日
已更新內容	更新了 S3 模式的公用屬性；更新了 DynamoDB 模式的預設值。	2020 年 8 月 10 日
已更新內容	更新了多種模式，突出顯示傳輸中加密的預設強制執行。	2020 年 8 月 4 日
已更新內容	新增 aws-lambda-qs-lambda 模式；入門指南中改進的組態指示；更新所有模式，以便透過公用屬性取得額外的資源。	2020 年 7 月 27 日
已更新內容	添加了 aws-lambda-qs 模式; 其他次要更新。	2020 年 7 月 20 日
已更新內容	已從相關病毒碼移除部署 LambDA 和部署儲存貯體屬性；其他次要更新。	2020 年 7 月 9 日
已更新內容	新增 aws-lambda 步驟函數模式，並修正輕微的印刷錯誤。	2020 年 7 月 7 日

已更新內容	已新增目前表格 J？ 屬性來選取 DynamoDB 模式。	2020 年 6 月 25 日
已更新內容	數個文字修正和修正中斷的連結。	2020 年 6 月 23 日
初始版本	公開提供的 AWS 解決方案建構。	2020 年 6 月 22 日

Notices

客戶有責任自行對本文件中的資訊進行獨立評估。本文件：(a) 僅供參考之用，(b) 代表目前的 AWS 產品供應項目和實務，如有變更，恕不另行通知，且 (c) 不訂立 AWS 及其關係企業、供應商或授權人的任何承諾或保證。AWS 產品或服務係依「現狀」提供，不提供任何形式的保證、陳述或條件 (無論是明示或暗示)。AWS 對其客戶的責任與義務應由 AWS 協議管轄，本文並非 AWS 與其客戶之間的任何協議的一部分，也並非上述協議的修改。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。