AWS 解决方案

AWS 解决方案构造

AWS 解决方案构造: AWS 解决方案

Table of Contents

AWS 解决方案构造

发布日期：2021 年 5 月(文档修订）

什么是 AWS 解决方案构造？

AWS 解决方案构造（构造）是AWS Cloud Development Kit (AWS CDK)，提供多服务、结构良好的模式，用于在代码中快速定义解决方案，从而创建可预测和可重复的基础设施。目标是加快开发人员使用基于模式的定义为其架构构建任何规模的解决方案的体验。

使用 AWS 解决方案构造以熟悉的编程语言定义您的解决方案。AWS 解决方案构造目前支持 TypeScript、JavaScript、Python 和 Java。

要浏览 AWS 解决方案构造模式的完整目录，Click LE。

为什么要使用 AWS 解决方案构造？

随着云提供商的创新速度，了解和理解最佳实践并确保在整个解决方案中正确实施这些实践可能令人生畏。构造允许您结合预构建、结构良好的模式和用例，以便以可扩展和安全的方式使用云服务执行常见操作。由于 Constructs 为现代编程语言提供了一个库，因此您可以将现有的开发技能和熟悉的工具应用于为您的解决方案构建架构良好的云基础架构的任务。

AWS 解决方案构造的其他优势包括：

- 它构造在 AWS Cloud Development Kit (AWS CDK) 开源软件开发框架之上。
- 在定义解决方案基础架构时使用逻辑（if 语句，for 循环等）。
- 使用面向对象的技术创建系统的模型。
- 定义高级抽象，共享它们，并将其发布到您的团队、公司或社区。
- 将您的解决方案组织成逻辑模块。
- 将您的解决方案作为库共享和重复使用。
- 使用行业标准协议测试您的基础架构代码。
- 使用您现有的代码审查工作流。

AWS 解决方案构造的目的是降低集成常见架构良好的模式时所需的复杂性和粘合逻辑，以便在 AWS 上实现您的解决方案目标。

AWS 解决方案构造入门

本主题介绍如何安装和配置 AWS Cloud Development Kit (AWS CDK)、AWS 解决方案构造，以及如何使用 AWS 解决方案构造模式创建您的第一个 AWS CDK 应用程序。

> **Note**
>
> AWS CDK 版本 ≥ 1.46.0 的 AWS CDK 支持 AWS 解决方案构造。

> **Tip**
>
> 想要深入挖掘？ 试试试试看CDK 研讨会了解真实世界项目的更深入介绍。

> **Tip**
>
> 有关 AWS Cloud Development Kit (AWS CDK) 入门的更多信息，请参阅AWS CDK 开发人员指南。

Prerequisites

AWS 解决方案构造是基于 AWS CDK 构建的，因此您需要安装 Node.js (>= 10.3.0)，即使是那些使用 TypeScript 或 JavaScript 以外的语言工作的解决方案也是如此。这是因为AWS CDK和 AWS 解决方案构造是在 TypeScript 中开发的，并在 Node.js 上运行。其他支持语言的绑定使用此后端和工具集。

您必须提供您的证书和 AWS 区域才能使用 AWS CDK CLI，如指定您的证书和区域中所述。

其他先决条件取决于您的开发语言，如下所示。

语言	先决条件
 P	Python >= 3.6

语言	先决条件
 T t	TypeScript >= 2.7
 J	Java >= 1.8

安装 AWS CDK

要安装和配置 AWS CDK，请参阅 AWS CDK 开发人员指南-安装 AWS CDK。

使用 AWS 解决方案构造

使用 AWS 解决方案构造时创建新应用程序的典型工作流程采用与 AWS CDK 相同的方法。

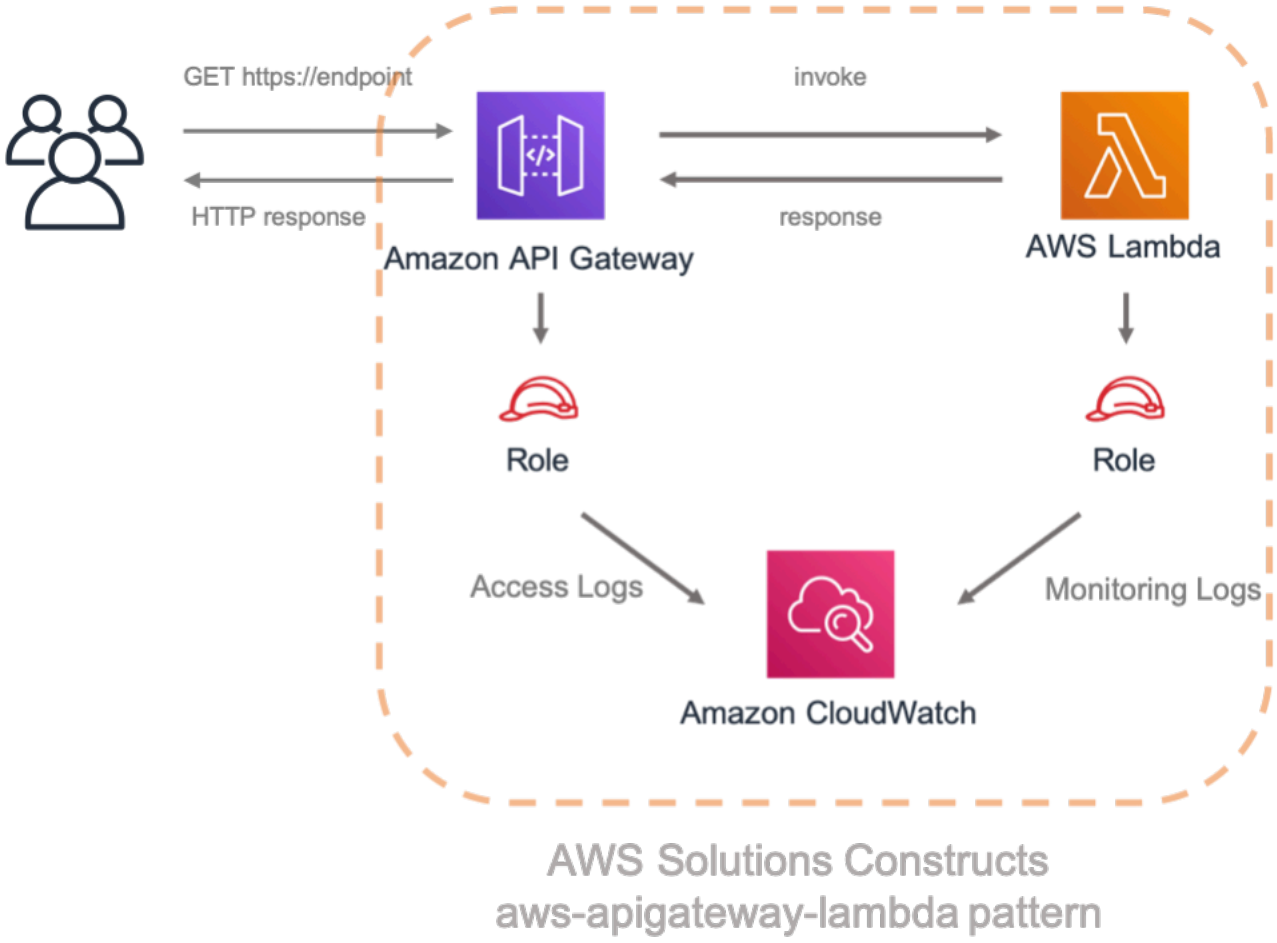
1. 创建应用程序目录。
2. 初始化应用程序。
3. 添加 AWS 解决方案构造模式依赖关系。
4. 将其他代码添加到应用程序。
5. 如有必要，编译应用程序。
6. 部署应用程序中定义的资源。
7. 测试应用程序。

如果存在任何问题，请循环修改、编译（如有必要）、部署和测试。

演练-第 1 部分

> **Note**
>
> AWS CDK 版本等于 1.46.0，支持 AWS 解决方案构造。

本教程将向您介绍如何创建和部署一个简单的 “Hello 构造” AWS CDK 应用程序，该应用程序使用 AWS 解决方案构造中的模式，从初始化项目到部署生成的 AWS CloudFormation 模板。Hello 构造应用程序将创建以下简单的解决方案：



Hello 构造

让我们开始使用基于模式的开发来构建我们的第一个 AWS CDK 应用程序。

> **Note**
>
> 这是一个修改示例`Hello CDK!`来自 的CDK 研讨会。如果这是您第一次使用 AWS CDK，我们建议您从本研讨会开始进行实践演练，以及如何利用 CDK 构建真实项目。

创建应用程序目录并初始化 AWS CDK

为 CDK 应用程序创建目录，然后在该目录中创建 AWS CDK 应用程序。

TypeScript

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

Python

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

> **Tip**
>
> 现在是在您最喜爱的 IDE 中打开项目并进行探索的好时机。要了解有关项目结构的更多信息，请选择相应的链接：
>
> - TypeScript
> - Python

更新项目基础依赖

> **Warning**
>
> 为确保正确的功能，AWS 解决方案构造和 AWS CDK 包必须在您的项目中使用相同的版本号。例如，如果您使用的是 AWS 解决方案构造 v.1.52.0，则还必须使用 AWS CDK v.1.52.0。

> **Tip**
>
> 请注意 AWS 解决方案构造的最新版本，并将该版本号应用于`VERSION_NUMBER`占位符（适用于 AWS 解决方案构造和 AWS CDK 包）。要检查构造库的所有公开版本，Click here。

TypeScript

编辑package.json文件并输入以下信息：

```
  "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

编辑setup.py文件并输入以下信息：

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

安装项目基础依赖关系。

TypeScript

```
npm install
```

Python

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

构建并运行应用程序，并确认它创建了一个空堆栈。

TypeScript

```
npm run build
cdk synth
```

Python

```
cdk synth
```

你应该看到一个类似如下的堆栈，其中CDK-VERSION是 CDK 的版本。（您的输出可能与此处显示的内容略有不同。）

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 处理程序代码

我们将从 AWS Lambda 处理程序代码开始。

创建目录`lambda`，包含在项目树的根目录中。

TypeScript

添加名为的文件`lambda/hello.js`，包含以下内容：

```
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

添加名为的文件`lambda/hello.py`，包含以下内容：

```
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'])
    }
```

这是一个简单的 Lambda 函数，它返回文本 “你好，构造！您已经点击了 [网址路径]”。函数的输出还包括 HTTP 状态代码和 HTTP 标头。API Gateway 使用它们来制定对用户的 HTTP 响应。

这个 Lambda 是在 JavaScript 中提供的。有关用您选择的语言编写 Lambda 函数的详细信息，请参阅AWS Lambda 文档。

安装 AWS CDK 和 AWS 解决方案构建依赖关系

AWS 解决方案构造随附一个丰富的结构库。该库分为模块，每个模块都有一个模块。例如，如果您想为 AWS Lambda 函数定义 Amazon API Gateway 剩余 API，我们将需要使用`aws-apigateway-lambdaClick` 库。

我们还需要从 AWS CDK 添加 AWS Lambda 和 Amazon API Gateway 构造库。

将 AWS Lambda 模块及其所有依赖项安装到我们的项目中：

> **Note**
>
> 请记住，将用于 AWS 解决方案构造和 AWS CDK 的正确匹配版本替换为`VERSION_NUMBER`每个命令的占位符字段。软件包之间的版本不匹配可能会导致错误。

TypeScript

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

Python

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

接下来，将 Amazon API Gateway 模块及其所有依赖项安装到我们的项目中：

TypeScript

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

Python

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

最后，安装 AWS 解决方案构造`aws-apigateway-lambda`模块及其所有依赖关系到我们的项目中：

TypeScript

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

Python

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

将亚马逊API 网关/AWS Lambda 模式添加到堆栈

现在，让我们定义 AWS 解决方案构造模式，用于使用 AWS Lambda 代理实现 Amazon API Gateway。

TypeScript

编辑文件`lib/hello-constructs.ts`, 包含以下内容:

```
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

编辑文件hello_constructs/hello_constructs_stack.py, 包含以下内容:

```
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
                )
            )
        )
```

就是这样 为了定义代理 AWS Lambda 函数的所有请求的 API Gateway，您需要执行这些操作。让我们将我们的新堆栈与原始堆栈进行比较：

TypeScript

```
npm run build
cdk diff
```

Python

```
cdk diff
```

输出应该如下所示：

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

这是不错的。这个简单的示例包含 AWS 解决方案构造中的一个架构良好的模式，为您的堆栈添加了 21 个新资源。

CDK 部署

> **Tip**
>
> 您必须引导 AWS 环境，然后才能部署包含 Lambda 函数的第一个 AWS CDK 应用程序。这将创建一个临时存储桶，AWS CDK 用于部署包含资产的堆栈。如果这是您第一次使用 AWS CDK 部署资产，则需要运行`cdk bootstrap`将 CDK 工具包堆栈部署到您的 AWS 环境中。

好了，已做好部署准备了吗？

```
cdk deploy
```

堆栈输出

部署完成后，您将注意到以下行：

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

这是由 AWS 解决方案构造模式自动添加的堆栈输出，其中包含 API Gateway 终端节点的 URL。

测试应用程序

让我们尝试使用`curl`。复制 URL 并执行（您的前缀和地区可能会有所不同）。

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

输出应该如下所示：

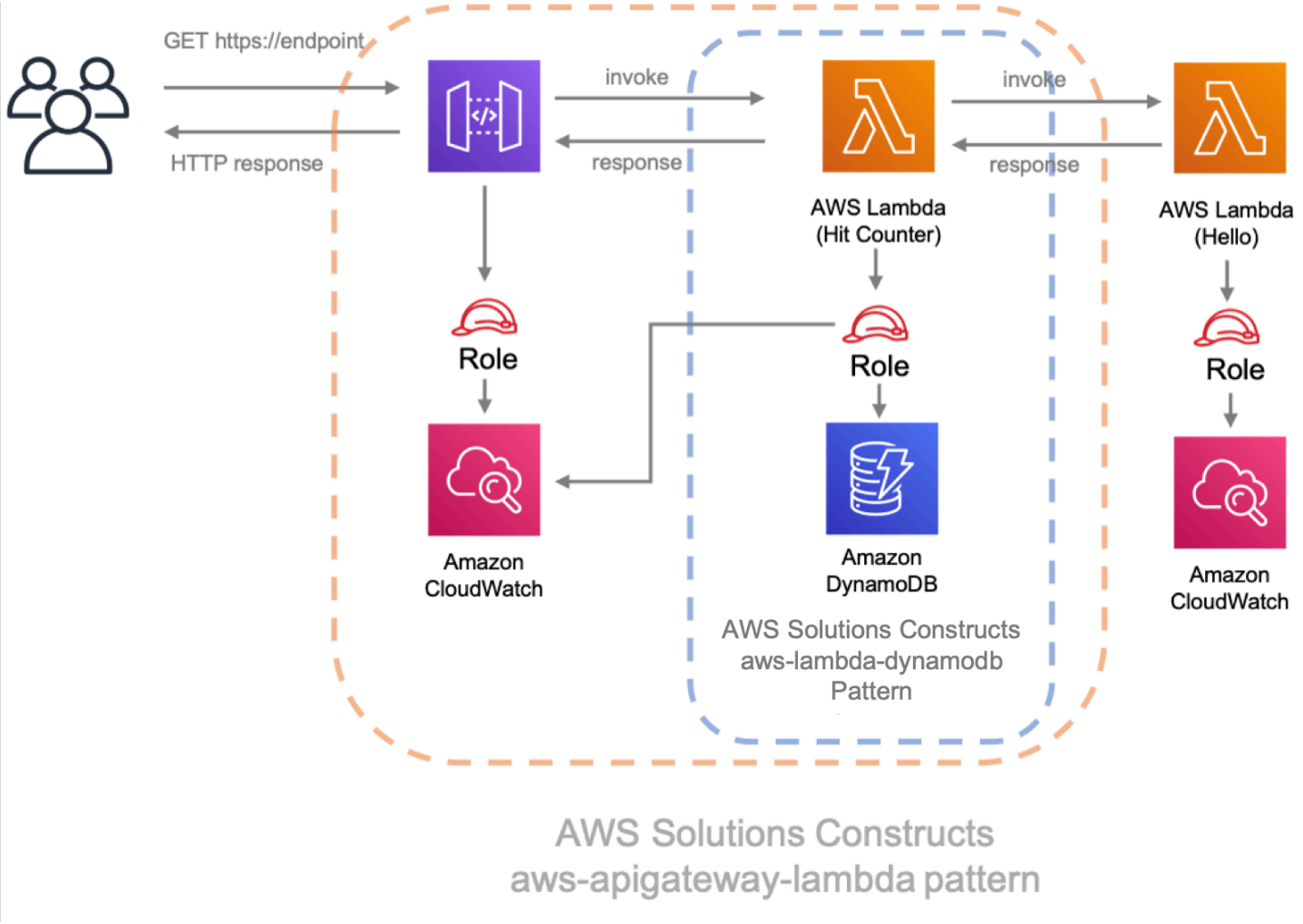
```
Hello, AWS Solutions Constructs! You've hit /
```

如果这是您收到的输出，您的应用程序可以正常工作！

演练-第 2 部分

> **Note**
>
> AWS CDK 版本 ≥ 1.46.0，支持 AWS 解决方案构造。

本教程将向您介绍如何修改第 1 部分。我们的修改将使用 AWS Lambda 向 AWS 解决方案构造中的 DynamoDB 模式添加站点点击计数器。修改 Hello 构造应用程序将导致以下解决方案：



点击计数器 Lambda 代码

让我们先为命中计数器 AWS Lambda 函数编写代码。此函数将：

- 增加一个与 Amazon DynamoDB 表中的 API 路径相关的计数器，
- 调用下游 Hello AWS Lambda 函数，

- 并将响应返回给最终用户。

TypeScript

添加名为的文件lambda/hitcounter.js更改为以下内容：

```
  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

添加名为的文件lambda/hitcounter.py更改为以下内容：

```
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)
```

安装新的依赖关系

> **Note**
>
> 请记住，将用于 AWS 解决方案构造和 AWS CDK 的正确匹配版本替换为VERSION_NUMBER每个命令的占位符字段。这应该与本演练第一部分用于依赖关系的版本号相同。软件包之间的版本不匹配可能会导致错误。

像往常一样，我们首先需要安装我们的解决方案更新所需的依赖关系。首先，我们需要安装 DynamoDB 构造库：

TypeScript

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

Python

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

最后，安装 AWS 解决方案构造`aws-lambda-dynamodb`模块及其所有依赖关系到我们的项目中：

TypeScript

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

Python

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

定义资源

现在，让我们更新我们的堆栈代码以适应我们的新架构。

首先，我们将导入我们的新依赖关系，并将 “Hello” 函数移动到`aws-apigateway-lambda`模式，我们在第 1 部分创建。

TypeScript

编辑文件`lib/hello-constructs.ts`更改为以下内容：

```
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);
  }
}
```

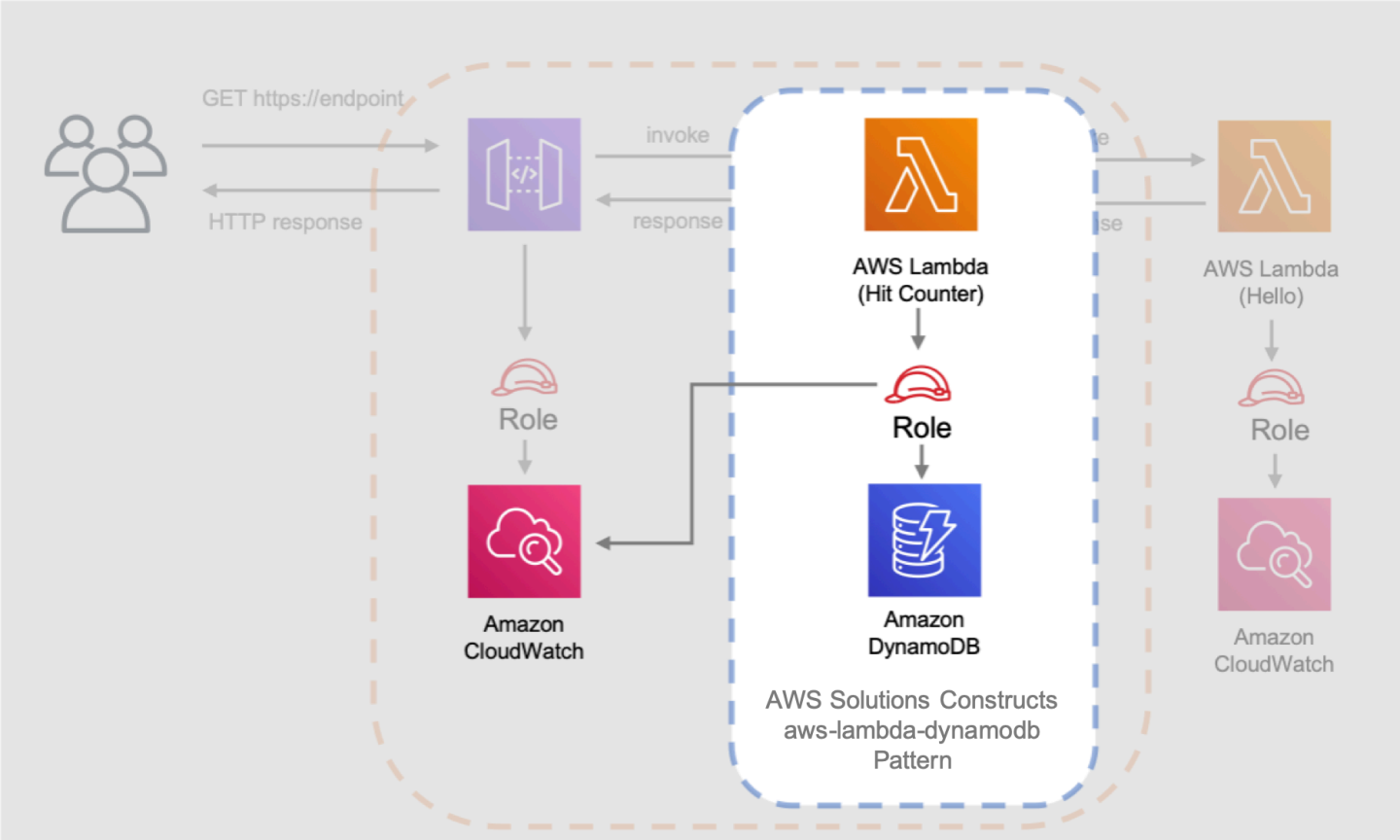
Python

编辑文件hello_constructs/hello_constructs_stack.py更改为以下内容：

```
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
                )
            )
        )
```

接下来，我们将添加aws-lambda-dynamodb模式来构建我们更新的架构的命中计数器服务。



AWS Solutions Constructs
aws-apigateway-lambda pattern

下面的下一个更新定义了`aws-lambda-dynamodb`模式，方法是使用命中计数器处理程序定义 AWS Lambda 函数。此外，Amazon DynamoDB 表的定义名称为`Hits`和一个分区键`path`。

TypeScript

编辑文件`lib/hello-constructs.ts`更改为以下内容：

```
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

编辑文件hello_constructs/hello_constructs_stack.py更改为以下内容：

```
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
                )
            )
        )
```

接下来，我们需要授予从aws-lambda-dynamodb模式来调用我们的 Hello 函数。

TypeScript

编辑文件lib/hello-constructs.ts更改为以下内容：

```
  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

编辑文件hello_constructs/hello_constructs_stack.py更改为以下内容：

```
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
            )
        )
    )
```

最后，我们需要更新原始aws-apigateway-lambda模式来利用我们的新命中计数器函数，该函数是通过aws-lambda-dynamodb模式。

TypeScript

编辑文件lib/hello-constructs.ts更改为以下内容：

```
  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

编辑文件hello_constructs/hello_constructs_stack.py更改为以下内容：

```
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
                )
            )
        )
```

查看更改

让我们构建我们的项目，并回顾我们在部署此项目时将发生的资源变化：

```
npm run build
cdk diff
```

我们的输出应该如下所示：

```
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

好，准备部署了吗？

```
cdk deploy
```

堆栈输出

部署完成后，您将注意到以下内容：

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

测试应用程序

让我们尝试用 curl 击中这个端点。复制 URL 并执行（您的前缀和区域可能会有所不同）。

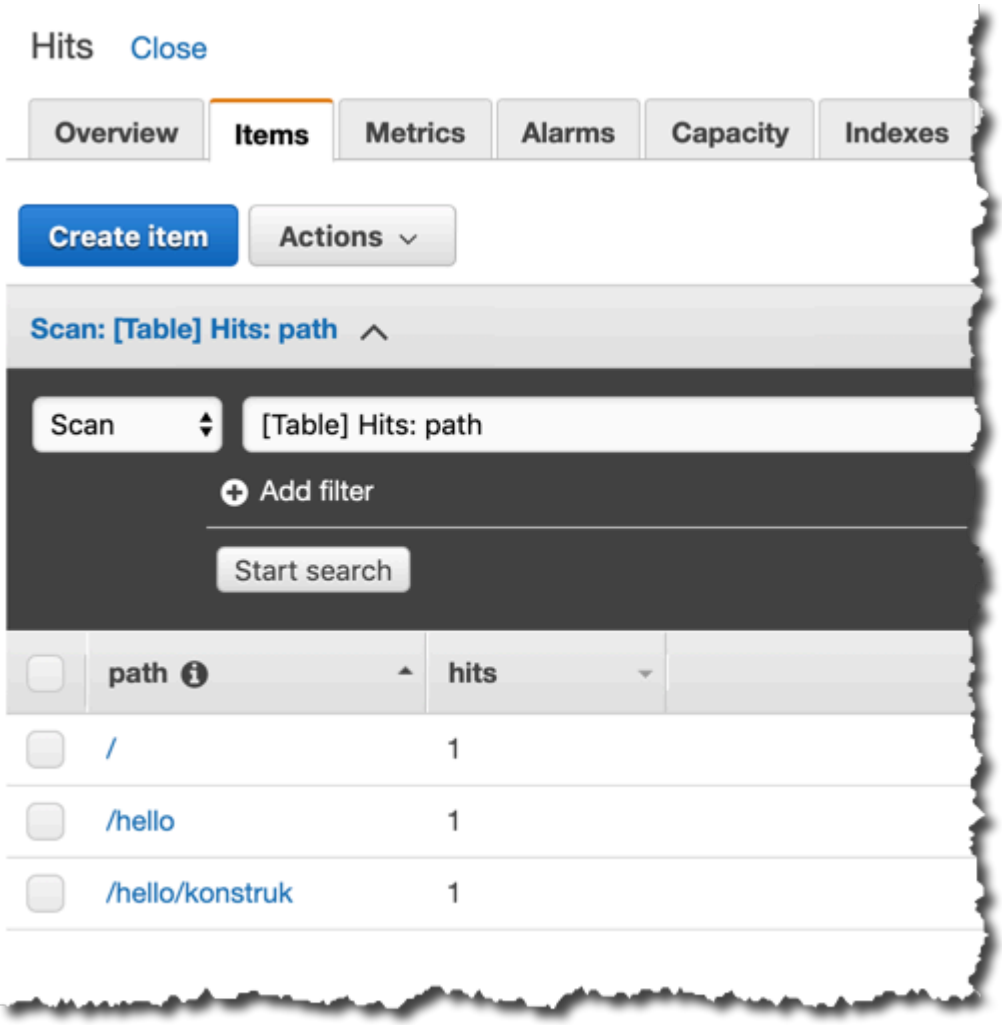
```
curl https://xxxxxxxxxx.execute-api.us-east-1.amazonaws.com/prod/
```

输出应该如下所示：

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

现在，我们来看一下HitsAmazon DynamoDB 表。

1. 转到 DynamoDB 控制台。
2. 确保您位于创建表的区域。
3. Select表，然后选择命中数表。
4. 打开表格并选择 “项目”。
5. 你应该看到你为每个路径获得了多少点击。



6. 尝试点击新路径并刷新 “项目” 视图。您应看到一个新项目，包含`hits`计数为 1。

如果这是您收到的输出，您的应用程序可以正常工作！

示例使用案例

此库包括一系列功能用例实现，用于演示构造架构模式的用法。它们可以用与架构模式相同的方式使用，并且可以被概念化为这些模式的额外 “更高级别” 抽象。以下用例作为功能示例提供：

AWS 静态 S3 网站

这个用例模式（`aws-s3-static-website`）实现了一个 Amazon CloudFront 分发、Amazon S3 存储桶和基于 AWS Lambda 的自定义资源，以复制 Wild Rydes 演示网站的静态网站内容（`aws-serverless-web-app`实现。

源代码（aws-s3-静态网站）

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

AWS 简单无服务器映像处理程序

这个用例模式（`aws-serverless-image-handler`）实现了 Amazon CloudFront 分发、Amazon API Gateway REST API、AWS Lambda 函数和必要的权限/逻辑，以配置功能图像处理程序 API，用于从部署账户中的一个或多个 Amazon S3 存储桶提供图像内容。

源代码（AWS 无服务器图像处理程序）

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

AWS 无服务器 Web 应用程序

这个用例模式（`aws-serverless-web-app`）实现了一个简单的无服务器 Web 应用程序，允许用户从 Wild Rydes 舰队请求独角兽骑行。该应用程序将向用户提供一个基于 HTML 的用户界面，用于指示他们想要被拾取的位置，并将在后端与 RESTful Web 服务进行接口，以提交请求并派遣附近的独角兽。该应用程序还将为用户提供服务注册和登录的设施，然后再申请乘车服务。

源代码（AWS 无服务器网络应用程序）

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

API 参考

AWS 解决方案构造（构造）是 AWS Cloud Development Kit (AWS CDK) 的开源扩展，它提供了多服务、结构良好的模式，用于在代码中快速定义解决方案，以创建可预测和可重复的基础设施。构造的目标是加速开发人员使用基于模式的定义为其架构构建任何规模的解决方案的体验。

构造中定义的模式是 AWS CDK 结构的高级多服务抽象，这些结构具有基于结构良好的最佳实践的默认配置。该库使用面向对象的技术将其组织为逻辑模块，以创建每个架构模式模型。

CDK 可用于以下语言：

- JavaScript，TypeScript Node.js
- 蟒蛇 (蟒蛇 ≥ 3.6)
- Java（Java ≥ 1.8）

Modules

AWS 解决方案构造分为几个模块。它们的命名是这样的：

- aw-xxx：为指定的服务设计精良的模式包。此软件包将包含多个 AWS CDK 服务模块的结构，用于配置给定模式。
- xxx：无法启动的软件包”aw-” 是用于为模式库中使用的服务配置最佳实践默认值的构造核心模块。

模块内容

模块包含以下类型：

- 模式-此库中的所有更高级别的多服务结构。
- 其他类型-所有支持模式的非构造类，接口，结构和枚举。

模式在其构造函数中采用一组（输入）属性；可以在模式的文档页面上看到一组属性（以及哪些属性是必需的）。

模式的文档页面还列出了可用的调用方法以及可用于检索模式实例化后模式信息的属性。

AWS-阿比网关-动态 b

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_apigateway_dynamodb`
 TypeScript	`@aws-solutions-constructs/aws-apigateway-dynamodb`
 Java	`software.amazon.awsconstructs.services.apigatewaydynamodb`

Overview

此 AWS 解决方案构造实现了与 Amazon DynamoDB 表连接的 Amazon API Gateway REST API。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsApiGatewayToDynamoDBProps

模式构建道具

名称	类型	描述
可发电道具	dynamodb.TableProps	用户提供的可选道具来覆盖 DynamoDB 表的默认道具
养蜂网关道具？	api.RestApiProps	用户提供的可选道具，用于覆盖 API Gateway 的默认道具。
允许创建操作	boolean	是否在 DynamoDB 表上部署用于创建操作的 API Gateway 方法。
创建请求模板	string	创建方法的 API Gateway 请求模板，如果允许创建操作设置为 true，则必须填写
允许操作	boolean	是否在 DynamoDB 表上部署用于读取操作的 API Gateway 方法。
允许更新操作	boolean	是否在 DynamoDB 表上部署用于更新操作的 API Gateway 方法。

名称	类型	描述
更新请求模板	`string`	用于更新方法的 API Gateway 请求模板，如果允许更新操作设置为 true，则需要
允许删除操作	`boolean`	是否在 DynamoDB 表上部署用于删除操作的 API Gateway 方法。
日志组道具？	`logs.LogGroupProps`	用户提供的可选道具可覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
APIGateway	`api.RestApi`	返回由模式创建的 API Gateway REST API 的实例。
网关云监视角色	`iam.Role`	返回由模式创建的 IAM 角色的实例，该模式允许从 API Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	`logs.LogGroup`	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。
养蜂网关角色	`iam.Role`	返回由 API Gateway REST API 模式创建的 IAM 角色的实例。
动态表	`dynamodb.Table`	返回由模式创建的 DynamoDB 表的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

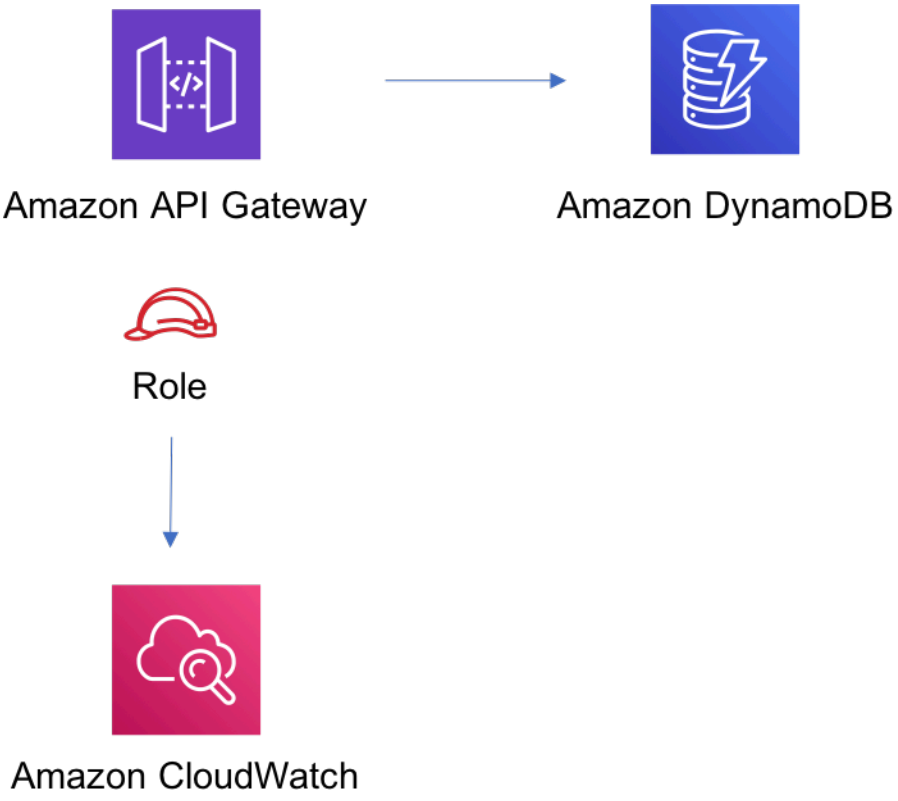
Amazon API Gateway

- 部署边缘优化的 API 终端节点
- 为 API Gateway 启用 CloudWatch 日志
- 为 API Gateway 配置最低权限访问 IAM 角色
- 将所有 API 方法的默认授权类型设置为 IAM
- 启用 X-Ray 跟踪

Amazon DynamoDB 表

- 将 DynamoDB 表的计费模式设置为按需（按请求付费）
- 使用 AWS 托管的 KMS 密钥为 DynamoDB 表启用服务器端加密
- 为 DynamoDB 表创建名为 “id” 的分区键
- 删除 CloudFormation 堆栈时保留表
- 实现连续备份和时间点恢复

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/AWS-网关-动态 b

AWS 网关-物联网

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_apigateway_iot`
TypeScript	`@aws-solutions-constructs/aws-apigateway-iot`
Java	`software.amazon.awsconstructs.services.apigatewayiot`

Overview

此 AWS 解决方案构造实现了连接到 AWS IoT 模式的 Amazon API Gateway REST API。

此构造可在 API Gateway 和 AWS IoT 之间创建可扩展的 HTTPS 代理。当希望允许不支持 MQTT 或 MQT/WebSocket 协议的旧式设备与 AWS IoT 平台进行交互时，这会很方便。

此实现允许在给定的 MQTT 主题上发布只写消息，并且还支持 HTTPS 设备的卷影更新以允许设备注册表中的内容。它不涉及用于代理消息的 Lambda 函数，而是依赖于直接 API Gateway 到 AWS IoT 集成，该网关支持 JSON 消息和二进制消息。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsApiGatewayToIotProps

模式构建道具

名称	类型	描述
物联网点	string	用于集成 API Gateway 的 AWS IoT 终端节点子域（例如 ，
养蜂网关创建密钥？	boolean	如果设置为true时，将创建 API 密钥并与 UsagePlan 相关联。用户在访问 RestApi 时应该指定 “X-api 键” 标头。默认值设置为false。
网关执行角色？	iam.Role	API Gateway 用于访问 AWS IoT 的 IAM 角色。如果未指定，则会创建一个默认角色，并使用通配符 ('*') 访问所有主题和内容。
养蜂网关道具？	api.restApiProps	用户提供的可选道具，用于覆盖 API Gateway REST API 的默认道具。
日志组道具？	logs.LogGroupProps	可选的用户提供的道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
APIGateway	api.RestApi	返回由模式创建的 API Gateway REST API 的实例。
网关云监视角色	iam.Role	返回由模式创建的 IAM 角色的实例，该模式允许从 API Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	logs.LogGroup	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。
养蜂网关角色	iam.Role	返回由 API Gateway REST API 模式创建的 IAM 角色的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon API Gateway

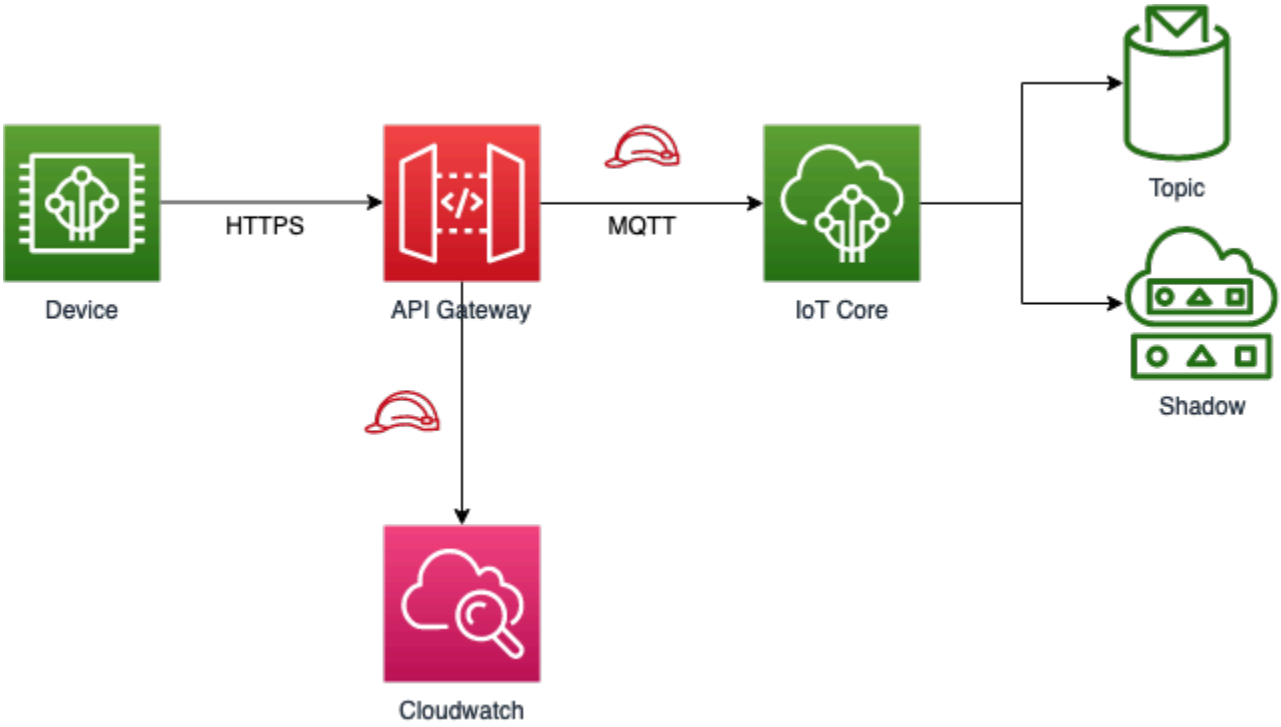
- 部署边缘优化的 API 终端
- 创建 API 资源POST将消息发布到 IoT 主题的方法
- 创建 API 资源POST将消息发布到的方法ThingShadow和NamedShadows
- 为 API Gateway 启用 CloudWatch 日志
- 为 API Gateway 配置 IAM 角色，可访问所有主题和内容
- 将所有 API 方法的默认授权类型设置为 IAM
- 启用 X-Ray 跟踪
- 创建 UsagePlan 并将prodstage

以下是 API Gateway 在部署构造后公开的不同资源和方法的说明。请参阅示例部分，获取有关如何使用轻松测试这些终端的更多信息curl。

方法	资源	查询参数	返回代码	描述
POST	/message/<topics>	qos	200/403/500	通过调用此终端节点，您需要传递要发布的主题（例如/message/device/foo`）。
POST	/shadow/<thingName>	无	200/403/500	这条路由允许更新一个事物的影子文档，因为它thingName 使用未命名的（经典）影子类型。身体必须符合标准的影子结构，包括state节点和关联desired和reported节点。请参阅更新设备影子节中的示例。
POST	/shadow/<thingName>/<shadowName>	无	200/403/500	这条路由允许更新某个事物的命名影子文档，因为它thingName 和shadowName 使用命名阴影类型。身体必须符合标准的影子结构，包括state节点和关

方法	资源	查询参数	返回代码	描述
				联desired和reported节点。请参阅更新命名的影子节中的示例。

Architecture



Examples

以下示例仅适用于API_KEY身份验证类型，因为 IAM 授权也需要指定一个 sigv4 令牌，所以请确保apiGatewayCreateApiKey属性设置为true，否则下面的示例将无法正常工作。

发布消息

您可以使用curl使用 HTTPS API 发布有关不同 MQTT 主题的消息。下面的示例将在device/foo主题。

```
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"}'
```

注意：将替换为stage-id、region, 和api-key参数与您的部署值一起使用。

您可以在 URL 中链接主题名称，API 接受最多 7 个可以发布的子主题。例如，以下示例将在主题上发布消息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"}'
```

更新设备影子

要更新与给定事物关联的影子文档，您可以使用事物名称发出影子状态请求。有关如何更新事物影子的以下示例。

```
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" }}}'
```

更新命名的影子

要更新与给定事物的命名影子关联的影子文档，您可以使用事物名称和影子名称发出影子状态请求。请参阅以下示例，了解如何更新命名阴影。

```
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" }}}'
```

发送二进制负载

可以将二进制负载发送到代理 API，直至 AWS IoT 服务。在以下示例中，我们将向发送README.md文件（被视为二进制数据）到device/foo主题使用application/octet-stream内容类型。

```
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
```

注意：在此项目的目录中执行此命令。然后，您可以测试从文件系统发送其他类型的二进制文件。

GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案结构/AWS-网关-物联网

aws-Api网关/动力流

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	aws_solutions_constructs.aws_apigateway_kinesisstreams
	@aws-solutions-constructs/aws-apigateway-kinesisstreams

语言	程序包
TypeScript	
 Java	`software.amazon.awsconstructs.services.apigatewaykinesisstreams`

Overview

此模式实现了与亚马逊 Kinesis 数据流连接的 Amazon API Gateway REST API。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scope`Construct`
- id`string`
- props`ApiGatewayToKinesisStreamsProps`

模式构建道具

名称	类型	描述
养蜂网关道具？	api.RestApiProps	用户提供的可选道具，用于覆盖 API Gateway REST API 的默认道具。
是否记录请求模板？	string	PutRecord tory 操作的 API Gateway 请求模板。如果未提供，将使用默认值。
是否记录请求模型？	api.ModelOptions	PutRecord tory 操作的 API Gateway 请求模型。如果未提供，则将创建一个默认值。
打破记录请求模板？	string	PutRecords 操作的 API Gateway 请求模板。如果未提供，将使用默认值。
是否记录请求模型？	api.ModelOptions	PutRecords 操作的 API Gateway 请求模型。如果未提供，则将创建一个默认值。
现有的河流 J？	kinesis.Stream	现有的 Kinesis 流实例，提供了这个和kinesisStreamProps 会导致错误。
运动流道具？	kinesis.StreamProps	用户提供的可选道具，用于覆盖 Kinesis 流的默认道具。
日志组道具？	logs.LogGroupProps	用户提供的可选道具来覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
APIGateway Gateway	api.RestApi	返回由模式创建的 API Gateway REST API 的实例。
养蜂网关角色	iam.Role	返回由 API Gateway REST API 模式创建的 IAM 角色的实例。
网关云监视角色	iam.Role	返回由模式创建的 IAM 角色的实例，该模式允许从 API Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	logs.LogGroup	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。
KinesisStream	kinesis.Stream	返回由模式创建的 Kinesis 流的实例。

示例 API 使用量

方法	请求路径	请求正文	队列操作	描述
POST	/record	`{ "data": "Hello World!", "partitionKey": "pk001" }`	kinesis:PutRecord	将单个数据记录写入流。

方法	请求路径	请求正文	队列操作	描述
POST	/records	{ "records": [{ "data": "abc", "partitionKey": "pk001" }, { "data": "xyz", "partitionKey": "pk001" }] }	kinesis:PutRecords	在一次调用中将多个数据记录写入流。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

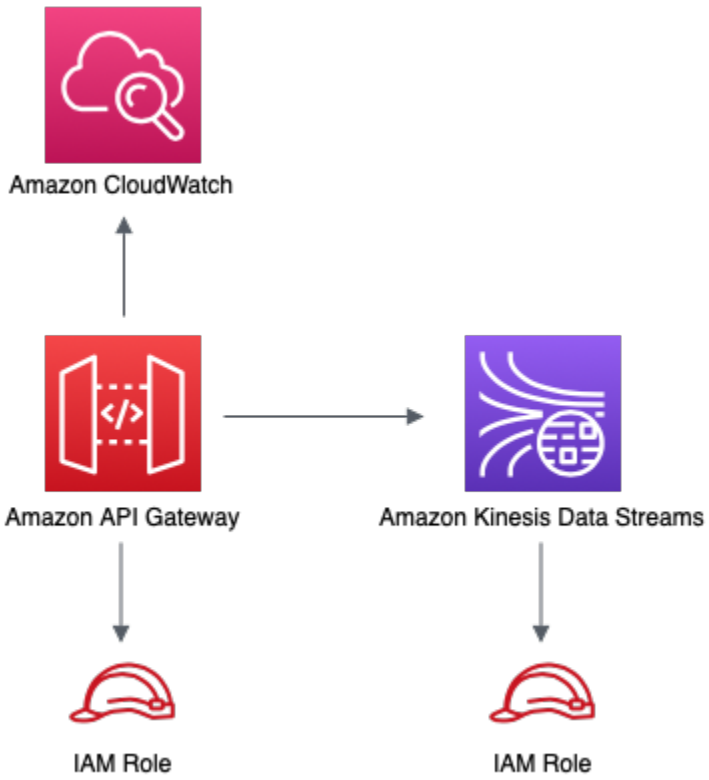
Amazon API Gateway

- 部署边缘优化的 API 终端节点。
- 为 API Gateway 启用 CloudWatch 日志记录。
- 为 API Gateway 配置最低权限访问 IAM 角色。
- 将所有 API 方法的默认授权类型设置为 IAM。
- 启用 X-Ray 跟踪。
- 在将数据传递给 Kinesis 之前验证请求正文。

Amazon Kinesis Data Stream

- 为 Kinesis 流配置最低权限访问 IAM 角色。
- 使用 AWS 托管 KMS 密钥为 Kinesis 流启用服务器端加密。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/AWS-网关-动态流

AWS-阿比网关-拉姆达

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_apigateway_lambda`
 TypeScript	`@aws-solutions-constructs/aws-apigateway-lambda`
 Java	`software.amazon.awsconstructs.services.apigatewaylambda`

Overview

此 AWS 解决方案构造实现了与 AWS Lambda 函数连接的 Amazon API Gateway REST API。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsApiGatewayToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略existingLambdaObj 提供。
养蜂网关道具？	api.LambdaRestApiProps	用户提供的可选道具来覆盖 API 的默认道具。
日志组道具？	logs.LogGroupProps	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
网关云监视角色	iam.Role	返回由模式创建的 IAM 角色的实例，该模式允许从 API Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	logs.LogGroup	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
ApeGateway	api.LambdaRestApi	返回由模式创建的 API Gateway REST API 的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon API Gateway

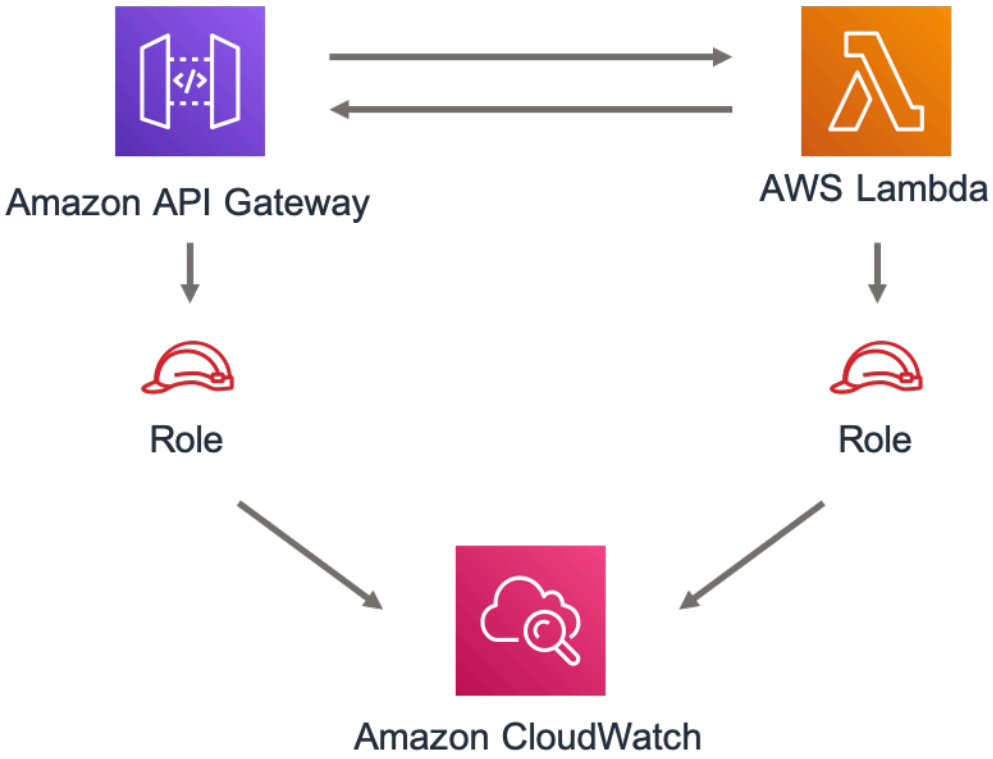
- 部署边缘优化的 API 终端节点
- 为 API Gateway 启用 CloudWatch 日志
- 为 API Gateway 配置最低权限访问 IAM 角色
- 将所有 API 方法的默认授权类型设置为 IAM
- 启用 X-Ray 跟踪
- 设置环境变量：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

AWS Lambda 函数

- 为 Lambda 函数配置受限权限访问 IAM 角色

- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接
- 启用 X-Ray 跟踪

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/AWS-阿比网关-lambda

AWS-阿皮盖特路-萨玛凯伦点

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造实现了与 Amazon SageMaker 终端节点连接的 Amazon API Gateway REST API。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsApiGatewayToSageMakerEndpointProps

模式构建道具

名称	类型	描述
养蜂网关道具？	api.RestApiProps	用户提供的可选道具来覆盖 API Gateway REST API 的默认道具。
网关执行角色？	iam.Role	API Gateway 用于调用 SageMaker 终端节点的 IAM 角色。如果未指定，则会创建一个默认角色，可访问endpointName 。

名称	类型	描述
EndpointName	`string`	已部署的 SageMaker 推理端点的名称。
ResourcenName？	`string`	GET 方法可用的可选资源名称。
resourcePath	`string`	GET 方法的资源路径。这里定义的变量可以在`requestMappingTemplate` 。
RequestMappingTemplate	`string`	将 REST API 上收到的 GET 请求转换为 SageMaker 终端节点预期的 POST 请求的映射模板。
RequestMappingTemplate？	`string`	用于转换从 SageMaker 终端节点收到的响应的可选映射模板。
日志组道具？	`logs.LogGroupProps`	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
APIGateway	`api.LambdaRestApi`	返回由模式创建的 API Gateway REST API 的实例。
养蜂网关角色	`iam.Role`	返回由 API Gateway REST API 模式创建的 IAM 角色的实例。
网关云监视角色	`iam.Role`	返回由模式创建的 IAM 角色的实例，该模式允许从 API

名称	类型	描述
		Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	logs.LogGroup	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。

示例 API 使用量

注意： 每个 SageMaker 终端节点都是唯一的，API 的响应将取决于部署的模型。下面给出的示例假定这篇博客帖子。有关如何实现的参考，请参阅综合性网关-萨马克信息点-覆盖 .ts。

方法	请求路径	查询字符串	SageMaker 行动	描述
GET	/321	items=101,131,162	sagemaker:InvokeEndpoint	检索特定用户和项目的预测。

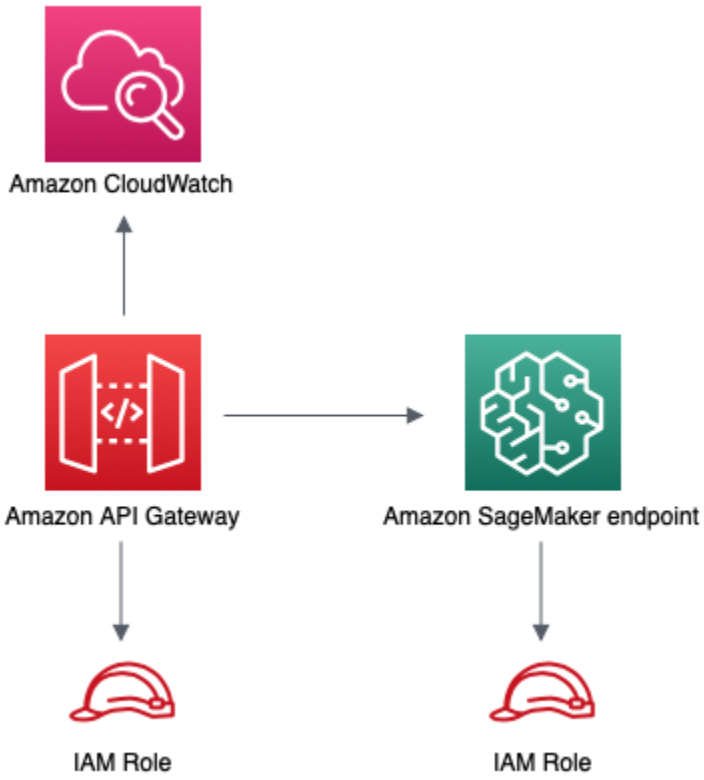
默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon API Gateway

- 部署边缘优化的 API 终端节点
- 为 API Gateway 启用 CloudWatch 日志
- 为 API Gateway 配置最低权限访问 IAM 角色
- 将所有 API 方法的默认授权类型设置为 IAM
- 启用 X-Ray 跟踪
- 在将数据传递给 SageMaker 之前验证请求参数

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/AWS-阿比网关-传奇点

AWS 网关平方米

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_apigateway_sqs`
 TypeScript	`@aws-solutions-constructs/aws-apigateway-sqs`
 Java	`software.amazon.awsconstructs.services.apigatewaysqs`

Overview

此 AWS 解决方案构造实现了连接到 Amazon SQS 队列的 Amazon API Gateway REST API。

以下是 TypeScript 中的最小可部署模式定义：

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

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

Initializer

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

参数

- scope`Construct`

- idstring
- propsApiGatewayToSqsProps

模式构建道具

名称	类型	描述
养蜂网关道具？	api.RestApiProps	用户提供的可选道具，用于覆盖 API Gateway 的默认道具。
队列道具？	sqs.QueueProps	用户提供的可选道具来覆盖队列的默认道具。
部署死信件队列？	boolean	是否要部署要用作死信队列的辅助队列。默认值为 true。
maxReceiveCount	number	消息在发送到死信队列之前移动到队列的次数。
是否允许创建操作？	boolean	是否在队列上部署用于创建操作的 API Gateway 方法（即 SQL：发送消息）。
是否创建请求模板？	string	覆盖创建方法的默认 API Gateway 请求模板（如果allowCreateOperation 设置为true。
是否允许操作？	boolean	是否为队列上的读取操作部署 API Gateway 方法（即 SQL：接收消息）。
是否已读请求模板？	string	覆盖读取方法的默认 API Gateway 请求模板（如果allowReadOperation 设置为true。

名称	类型	描述
是否允许删除操作？	boolean	是否在队列上部署用于删除操作的 API Gateway 方法（即 SQL：删除信息）。
是否删除请求模板？	string	覆盖删除方法的默认 API Gateway 请求模板（如果allowDeleteOperation 设置为true。
日志组道具？	logs.LogGroupProps	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
APIGateway Sway	api.RestApi	返回由模式创建的 API Gateway REST API 的实例。
网关云监视角色	iam.Role	返回由模式创建的 IAM 角色的实例，该模式允许从 API Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	logs.LogGroup	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。
养蜂网关角色	iam.Role	返回由 API Gateway REST API 模式创建的 IAM 角色的实例。
死信队列？	sqs.Queue	返回由模式创建的死信队列的实例（如果已部署）。

名称	类型	描述
SQUUE	sqs.Queue	返回由模式创建的 SQS 队列的实例。

示例 API 使用情况

方法	请求路径	请求正文	队列操作	描述
GET	/		sqs::ReceiveMessage	从队列中检索消息。
POST	/	{ "data": "Hello World!" }	sqs::SendMessage	将消息传送到队列。
DELETE	/message?receiptHandle=[value]		sqs::DeleteMessage	从队列中删除指定的消息

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon API Gateway

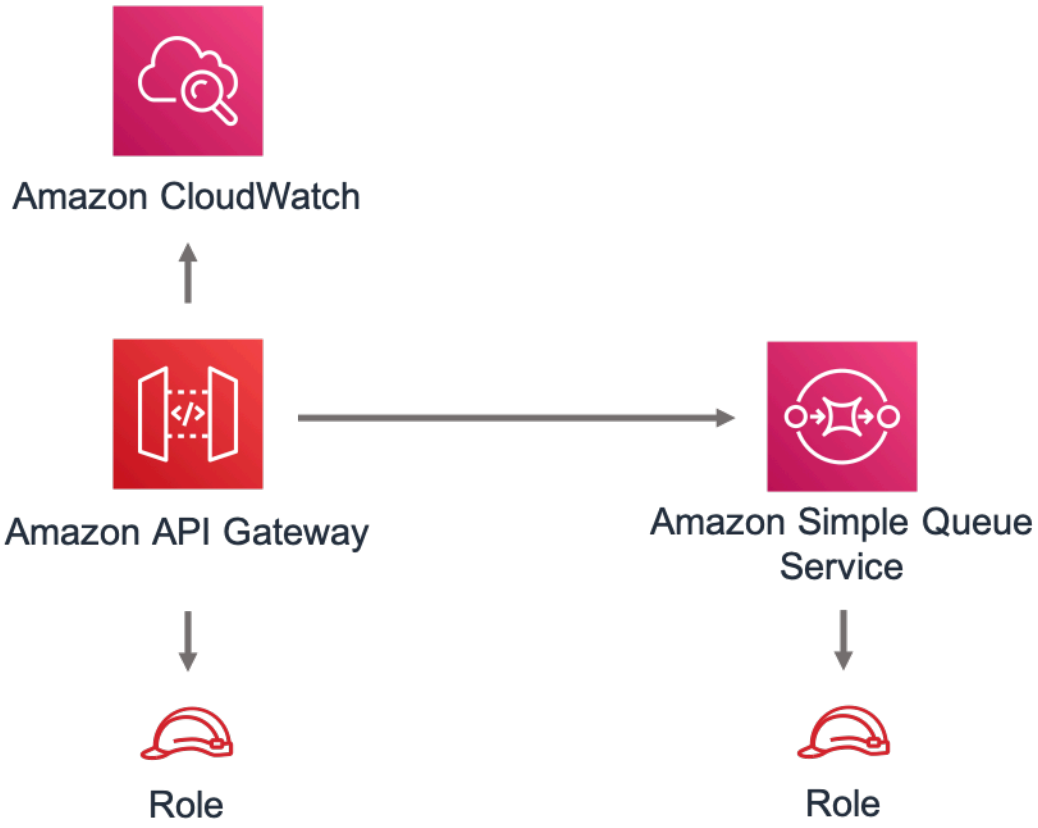
- 部署边缘优化的 API 终端节点
- 启用 API Gateway 的 CloudWatch 日志
- 为 API Gateway 配置最低权限访问 IAM 角色
- 将所有 API 方法的默认授权类型设置为 IAM
- 启用 X-Ray 跟踪

Amazon SQS 队列

- 为源 SQS 队列部署 SQS 死信队列

- 使用 AWS 托管 KMS 密钥为源 SQS 队列启用服务器端加密
- 实施传输中数据加密

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/AWS-网关-平方

AWS 云前端接口网关

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_cloudfront_apigateway`
TypeScript	`@aws-solutions-constructs/aws-cloudfront-apigateway`
Java	`software.amazon.awsconstructs.services.cloudfrontapigateway`

Overview

此 AWS 解决方案构造在 Amazon API Gateway REST API 前面实现了一个 Amazon CloudFront 分发。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsCloudFrontToApiGatewayProps

模式构建道具

名称	类型	描述
现有的大门威 OBJ	api.RestApi	将使用 CloudFront 端的区域 API Gateway

名称	类型	描述
云前端分发道具？	cloudfront.DistributionProps	可选的用户提供的道具来覆盖 CloudFront 分发的默认道具。
是否插入安全标头？	boolean	可选用户提供的道具，用于在 CloudFront 的所有响应中打开/关闭最佳实践 HTTP 安全标头的自动注入

模式属性

名称	类型	描述
APIGateway	api.RestApi	返回由模式创建的 API Gateway REST API 的实例。
云前端记录存储桶？	s3.Bucket	返回由 CloudFront Web 分发模式创建的日志记录存储桶的实例。
云前端网络分发	cloudfront.CloudFrontWebDistribution	返回由模式创建的 CloudFront Web 分发的实例。
埃德格兰姆达功能版本？	lambda.Version	返回由模式创建的 Lambda 边函数版本的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

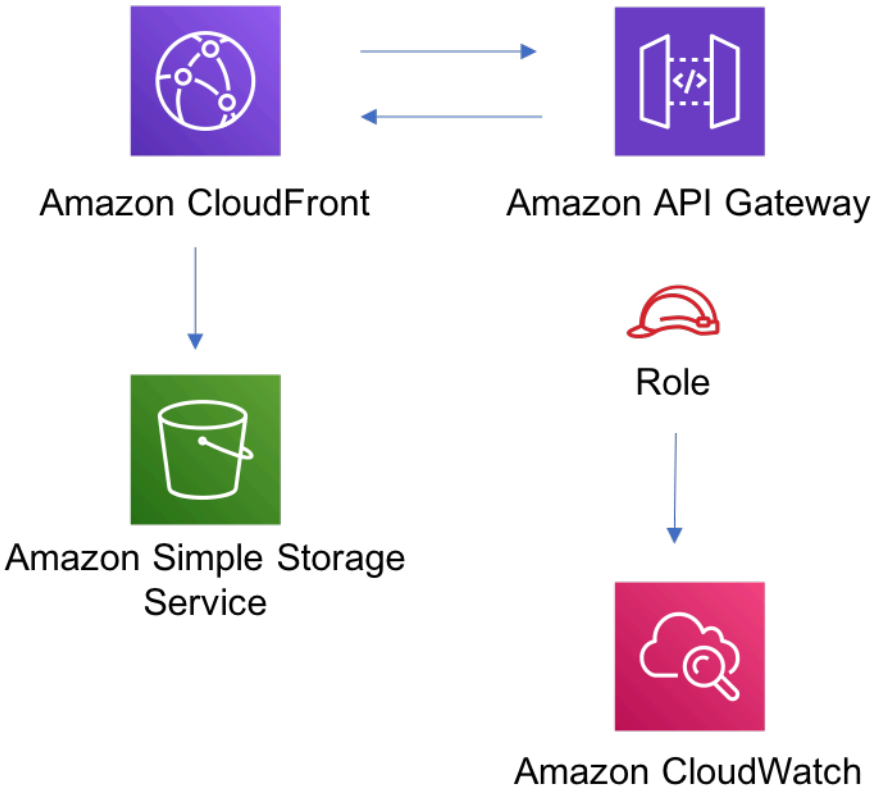
Amazon CloudFront

- 为 CloudFront 网络分发配置访问日志记录
- 支持在 CloudFront 网络分发的所有响应中自动注入最佳实践 HTTP 安全标头

Amazon API Gateway

- 用户提供的 API Gateway 对象按原样使用
- 启用 X-Ray 跟踪

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aw-云前端-接口网关

AWS 云前端-网关-lambda

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_cloudfront_apigateway_lambda`
 TypeScript	`@aws-solutions-constructs/aws-cloudfront-apigateway-lambda`
 Java	`software.amazon.awsconstructs.services.cloudfrontapigatewaylambda`

Overview

此 AWS 解决方案构造在 Amazon API Gateway LambDA 支持的 REST API 前面实现了一个 Amazon CloudFront 分发。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsCloudFrontToApiGatewayToLambdaProps

模式构造道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略existingLambdaObj 提供。
养蜂网关道具？	api.LambdaRestApiProps	用户提供的可选道具来覆盖 API Gateway 的默认道具
云前端分发道具？	cloudfront.DistributionProps	可选的用户提供的道具来覆盖 CloudFront 分发的默认道具。

名称	类型	描述
是否插入安全标头？	boolean	可选用户提供的道具，用于在 CloudFront 的所有响应中打开/关闭最佳实践 HTTP 安全标头的自动注入
日志组道具？	logs.LogGroupProps	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
APIGateway	api.RestApi	返回由模式创建的 API Gateway REST API 的实例。
网关云监视角色	iam.Role	返回由模式创建的 IAM 角色的实例，该模式允许从 API Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	logs.LogGroup	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。
云前端记录存储桶？	s3.Bucket	返回由 CloudFront Web 分发模式创建的日志记录存储桶的实例。
云前端网络分发	cloudfront.CloudFrontWebDistribution	返回由模式创建的 CloudFront Web 分发的实例。
埃德格兰姆达功能版本？	lambda.Version	返回由模式创建的 Lambda 边函数版本的实例。

名称	类型	描述
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon CloudFront

- 为 CloudFront 网络分发配置访问日志记录
- 支持在 CloudFront 网络分发的所有响应中自动注入最佳实践 HTTP 安全标头

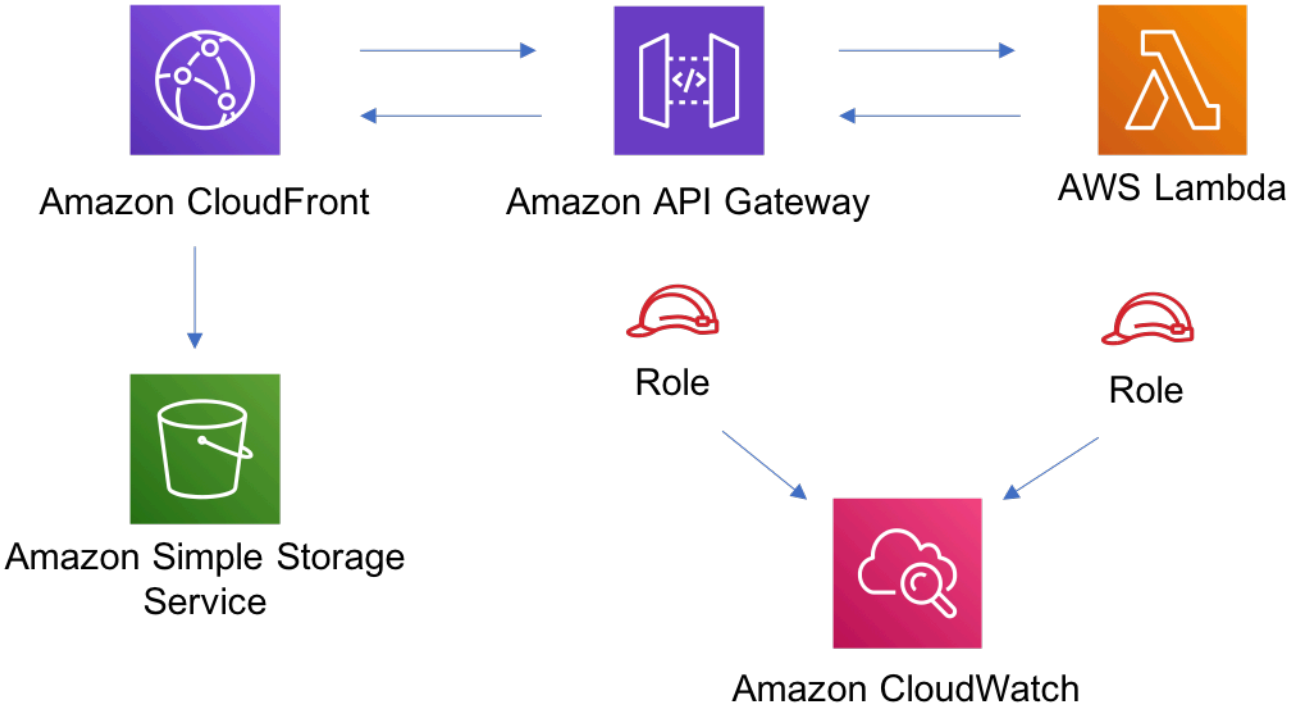
Amazon API Gateway

- 部署区域 API 终端节点
- 为 API Gateway 启用 CloudWatch 日志
- 为 API Gateway 配置最低权限访问 IAM 角色
- 将所有 API 方法的默认授权类型设置为 IAM
- 启用 X-Ray 跟踪

AWS Lambda 函数

- 为 Lambda 配置受限权限访问 IAM 角色
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接
- 启用 X-Ray 跟踪
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aw-云前端-网关-lambda

AWS 云前端媒体存储

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_cloudfront_mediastore`
TypeScript	`@aws-solutions-constructs/aws-cloudfront-mediastore`
Java	`software.amazon.awsconstructs.services.cloudfrontmediastore`

Overview

此 AWS 解决方案构造实现了连接到 AWS Elemental MediaStore 容器的 Amazon CloudFront 分配。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scope`Construct`
- id`string`
- props`CloudFrontToMediaStoreProps`

模式构建道具

名称	类型	描述
现有的媒体存储集装箱外壳？	mediastore.CfnContainer	可选的用户提供的 MediaStore 容器，用于覆盖默认的媒体存储容器。
媒体商店集装箱道具？	mediastore.CfnContainerProps	用户提供的可选道具来覆盖 MediaStore 容器的默认道具。
云前端分发道具？	cloudfront.DistributionProps | any	用户提供的可选道具来覆盖 CloudFront 分发的默认道具。
是否插入安全标头？	boolean	可选的用户提供的道具，用于在 CloudFront 的所有响应中打开/关闭最佳实践 HTTP 安全标头的自动注入。

模式属性

名称	类型	描述
云端网络分发	cloudfront.CloudFrontWebDistribution	返回由模式创建的 CloudFront Web 分发的实例。
媒体存储容器	mediastore.CfnContainer	返回由模式创建的 MediaStore 容器的实例。
云前端记录存储桶	s3.Bucket	返回由 CloudFront Web 分发模式创建的日志记录存储桶的实例。
云前端来源请求策略	cloudfront.OriginRequestPolicy	返回由 CloudFront Web 分发模式创建的 CloudFront 源请求策略的实例。

名称	类型	描述
CloudFront 源访问性？	cloudfront.OriginAccessIdentity	返回由 CloudFront Web 分发模式创建的 CloudFront 源访问身份的实例。
封边功能版本	lambda.Version	返回由模式创建的 Lambda 边函数版本的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

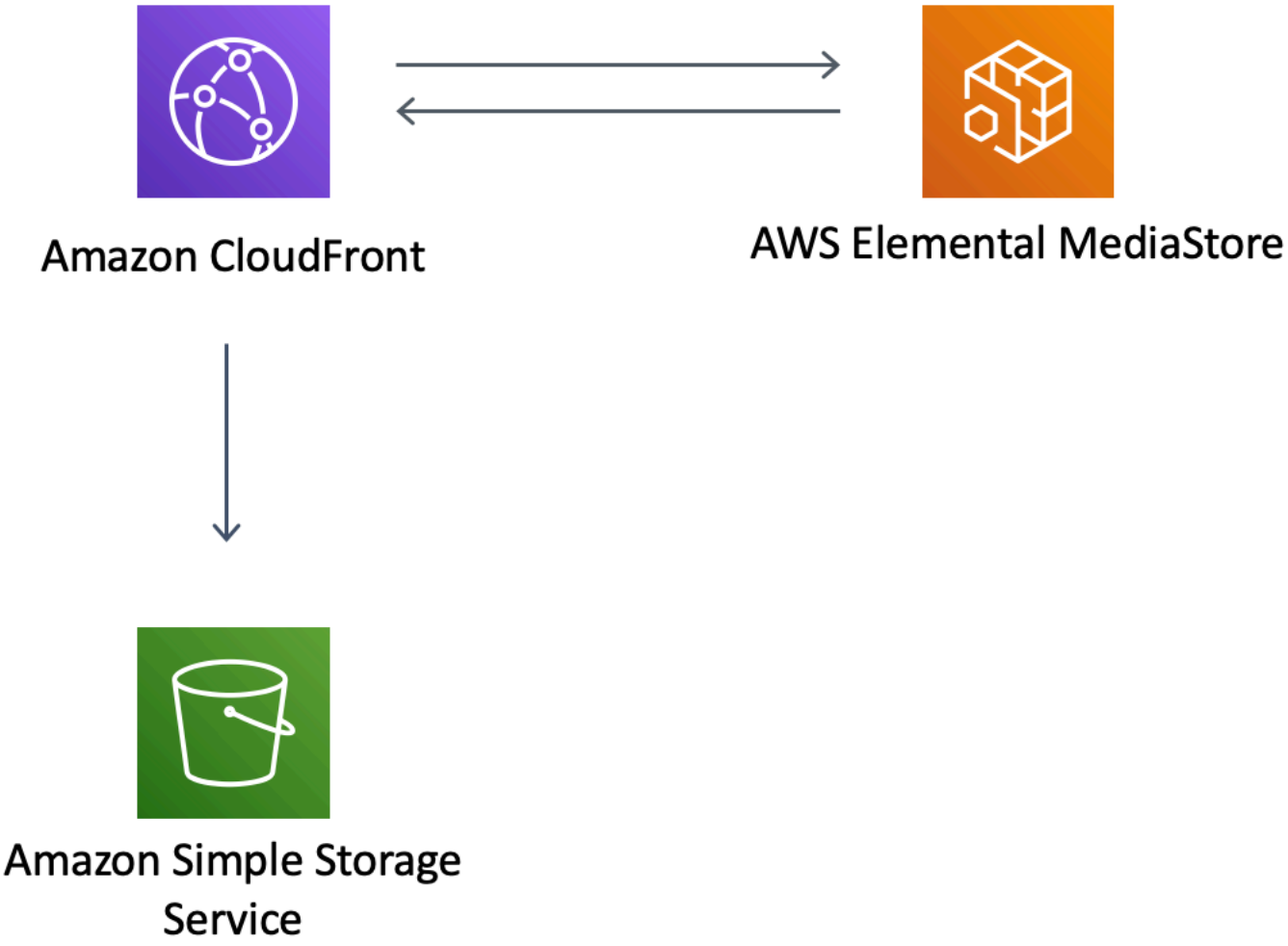
Amazon CloudFront

- 为 CloudFront 网络分发配置访问日志记录
- 为 AWS Elemental MediaStore 容器启用 CloudFront 源请求策略
- Set`User-Agent`CloudFront 源访问身份的自定义标头
- 支持在来自 CloudFront Web 分发的所有响应中自动注入最佳实践 HTTP 安全标头

AWS Elemental MediaStore

- 设置删除策略以保留资源
- 使用 CloudFormation 堆栈名称设置容器名称
- 设置默认容器跨源资源共享 (CORS) 策略
- 设置默认对象生命周期策略
- 设置默认容器策略仅允许`aws:UserAgent`CloudFront 源访问身份
- 设置默认指标策略
- 启用访问日志记录

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aw-云前媒体存储

AWS 云前端-S3

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_cloudfront_s3`
SET DEFAULT	`@aws-solutions-constructs/aws-cloudfront-s3`
Java	`software.amazon.awsconstructs.services.cloudfronts3`

Overview

此 AWS 解决方案构建在 Amazon S3 存储桶之前实现 Amazon CloudFront 分配。

以下是 TypeScript 中的最小可部署模式定义：

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

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

Initializer

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

参数

- scopeConstruct
- idstring
- propsCloudFrontToS3Props

模式构建道具

名称	类型	描述
现有的存储桶吗？	s3.Bucket	S3 存储桶对象的现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。
桶道具？	s3.BucketProps	用户提供的可选属性，用于覆盖存储桶的默认属性。忽略，如果existingBucketObj 提供。
云前端分发道具？	cloudfront.DistributionProps	可选的用户提供的道具来覆盖 CloudFront 分发的默认道具。
是否插入安全标头？	boolean	可选用户提供的道具，用于在 CloudFront 的所有响应中打开/关闭最佳实践 HTTP 安全标头的自动注入

模式属性

名称	类型	描述
云前端网络分发	cloudfront.CloudFrontWebDistribution	返回由模式创建的 CloudFront Web 分发的实例。

名称	类型	描述
S3Bucket	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。
埃德格兰姆达功能版本？	lambda.Version	返回由模式创建的 Lambda 边函数版本的实例。
云前端记录存储桶？	s3.Bucket	返回由 CloudFront Web 分发模式创建的日志记录存储桶的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

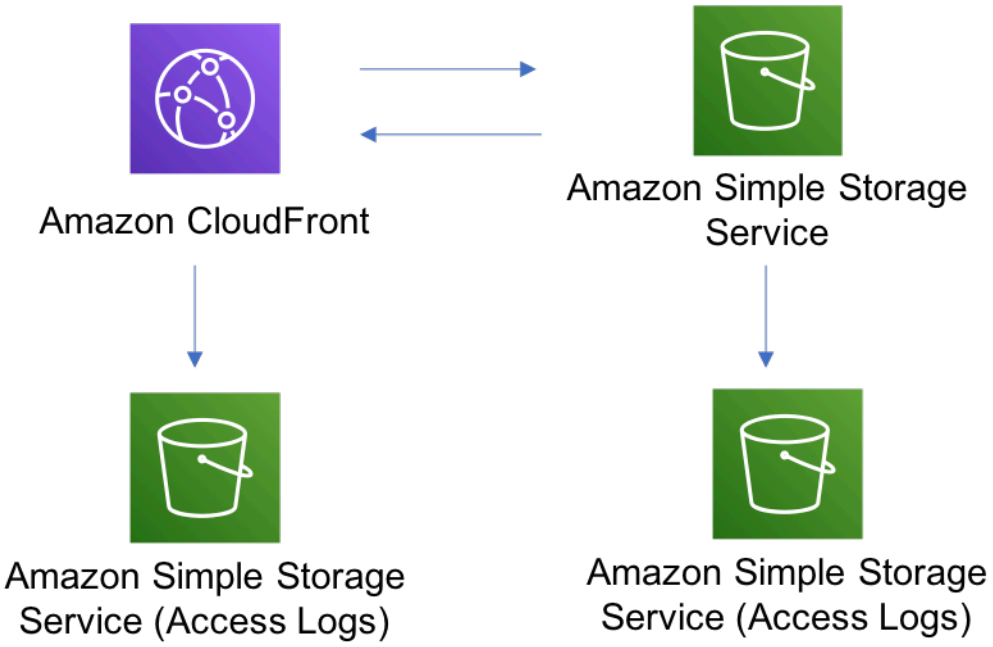
Amazon CloudFront

- 为 CloudFront 网络分发配置访问日志记录
- 支持在 CloudFront 网络分发的所有响应中自动注入最佳实践 HTTP 安全标头

Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密
- 启用 S3 存储桶的版本控制
- 不允许 S3 存储桶的公共访问
- 删除 CloudFormation 堆栈时保留 S3 存储桶
- 实施传输中数据加密
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aw-云前-S3

AWS 认知识-养蜂网关-lambda

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_cognito_apigateway_lambda`
TypeScript	`@aws-solutions-constructs/aws-cognito-apigateway-lambda`
Java	`software.amazon.awsconstructs.services.cognitoapigatewaylambda`

Overview

此 AWS 解决方案构造实施了 Amazon Cognito 保护由 Amazon API Gateway LambDA 支持的 REST API。

以下是 TypeScript 中的最小可部署模式定义：

```
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 上定义资源和方法（例如`proxy = false`），您必须调用`addAuthorizers()`方法在 API 完全定义之后。这可确保 API 中的每个方法都受到保护。

以下是 TypeScript 中的一个示例：

```
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);
```

参数

- scope`Construct`
- id`string`
- props`CognitoToApiGatewayToLambdaProps`

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
养蜂网关道具？	api.LambdaRestApiProps	用户提供的可选道具来覆盖 API Gateway 的默认道具
认知服务器池道具？	cognito.UserPoolProps	用户提供的可选道具来覆盖 Cognito 用户池的默认道具
认知服务器客户端道具？	cognito.UserPoolClientProps	用户提供的可选道具来覆盖 Cognito 用户池客户端的默认道具
日志组道具？	logs.LogGroupProps	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
APIGateway	api.RestApi	返回由模式创建的 API Gateway REST API 的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。

名称	类型	描述
userPool	cognito.UserPool	返回由模式创建的 Cognito 用户池的实例。
UserPoolClient	cognito.UserPoolClient	返回由模式创建的 Cognito 用户池客户端的实例。
网关云监视角色	iam.Role	返回由模式创建的 IAM 角色的实例，该模式允许从 API Gateway REST API 访问日志记录到 CloudWatch。
网关日志组	logs.LogGroup	返回由 API Gateway REST API 访问日志发送到的模式创建的日志组的实例。
网关授权器	api.CfnAuthorizer	返回由模式创建的 API Gateway 授权程序的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon Cognito

- 为用户池设置密码策略
- 强制执行用户池的高级安全模式

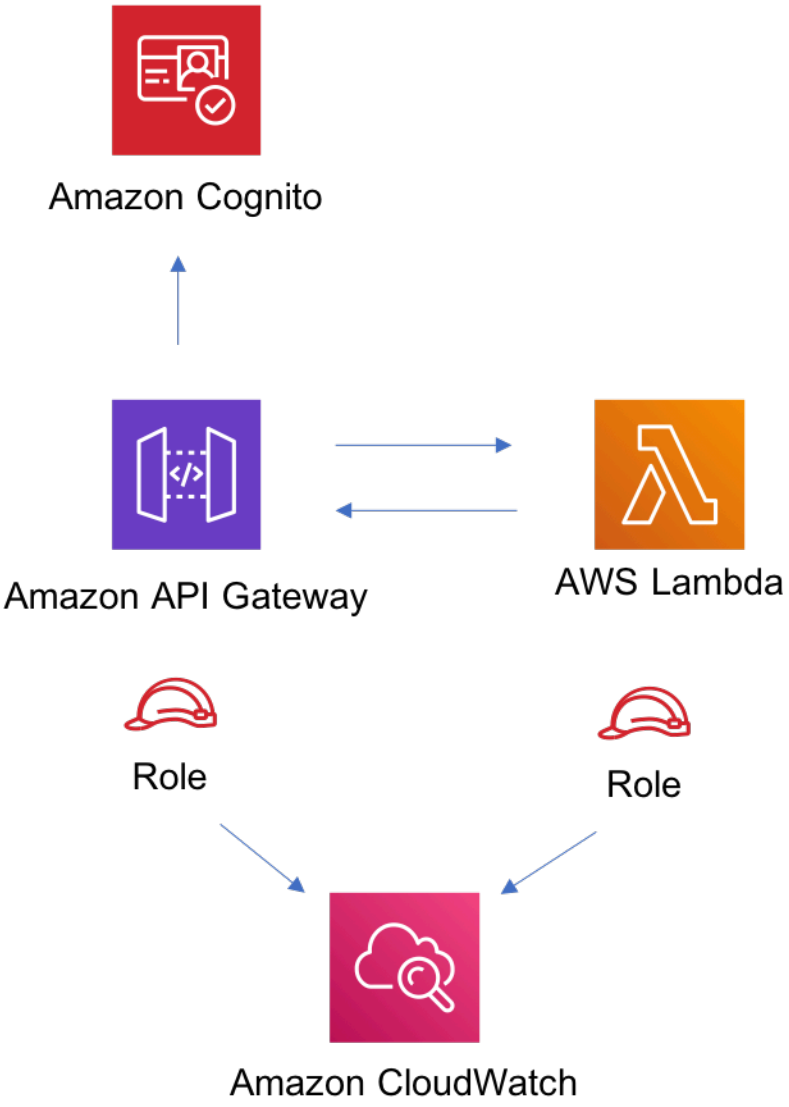
Amazon API Gateway

- 部署边缘优化的 API 终端节点
- 为 API Gateway 启用 CloudWatch 日志记录
- 为 API Gateway 配置最低权限访问 IAM 角色
- 将所有 API 方法的默认授权类型设置为 IAM
- 启用 X-Ray 跟踪

AWS Lambda 函数

- 为 Lambda 函数配置受限权限访问 IAM 角色
- 使用节点 JS Lambda 函数启用重复使用连接
- 启用 X-Ray 跟踪
- 设置环境变量：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案结构/AWS-认知-养蜂网关-lambda

aws-dynamodb-stream-lambda

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_dynamodb_stream_lambda`
 TypeScript	`@aws-solutions-constructs/aws-dynamodb-stream-lambda`
 Java	`software.amazon.awsconstructs.services.dynamodbstreamlambda`

Overview

此 AWS 解决方案构造实现了具有流的模式 Amazon DynamoDB 表，以调用具有最低特权权限的 AWS Lambda 函数。

这是一个最小的可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsDynamoDBStreamToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默

名称	类型	描述
		认属性。忽略existingLambdaObj 提供。
可发电道具？	dynamodb.TableProps	用户提供的可选道具来覆盖 DynamoDB 表的默认道具
是否存在表格？	dynamodb.Table	DynamoDB 表对象的现有实例，提供了这个和dynamoTablePropsProps 会导致错误。
发电机事件源道具？	aws-lambda-event-sources.DynamoEventSourceProps	用户提供的可选道具来覆盖 DynamoDB 事件源的默认道具

模式属性

名称	类型	描述
动态表	dynamodb.Table	返回由模式创建的 DynamoDB 表的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。

Lambda 函数

此模式需要一个 Lambda 函数，该函数可以从 DynamoDB 流将数据发布到 Elasticsearch 服务。提供示例函数此处。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon DynamoDB 表

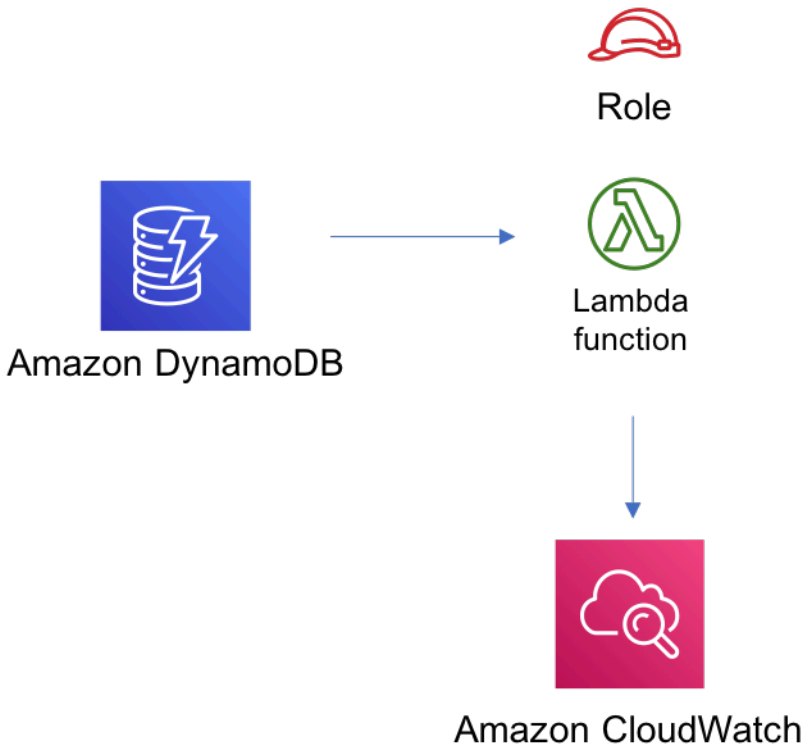
- 将 DynamoDB 表的计费模式设置为按需（按请求付费）

- 使用 AWS 托管的 KMS 密钥为 DynamoDB 表启用服务器端加密
- 为 DynamoDB 表创建名为 “id” 的分区键
- 删除 CloudFormation 堆栈时保留表
- 实现连续备份和时间点恢复

AWS Lambda 函数

- 为 Lambda 函数配置受限权限访问 IAM 角色
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接
- 启用 X-Ray 跟踪
- 启用故障处理功能：启用等分功能错误；设置默认的最长记录时间 (24 小时)；设置默认的最大重试次数 (500)；以及在出现故障时将 SQS 死信队列部署为目标
- 设置环境变量：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案结构/aw-动态-流-lambda

aws--发动机-流-拉姆达-弹性搜索-基巴纳

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 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 解决方案构造实现了具有流、AWS Lambda 函数和具有最低特权权限的 Amazon Elasticsearch 服务的 Amazon DynamoDB 表。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsDynamoDBStreamToLambdaToElasticSearchAndKibanaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会引发错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
可发电道具？	dynamodb.TableProps	用户提供的可选道具来覆盖 DynamoDB 表的默认道具
是否存在表格？	dynamodb.Table	DynamoDB 表对象的现有实例，提供了这个和dynamoTableProps 会引发错误。
发电机事件源道具？	aws-lambda-event-sources.DynamoEventSourceProps	用户提供的可选道具来覆盖 DynamoDB 事件源的默认道具
埃斯领土道具？	elasticsearch.CfnDomainProps	用户提供的可选道具来覆盖 Amazon Elasticsearch Service 的默认道具
domainName	string	Cognito 和 Amazon Elasticsearch Service 的域名
创造云监视图	boolean	是否创建推荐的 CloudWatch 警报。

模式属性

名称	类型	描述
CloudwatchArms	cloudwatch.Alarm[]	返回模式创建的一个或多个 CloudWatch 警报的列表。
动态表	dynamodb.Table	返回由模式创建的 DynamoDB 表的实例。
弹性搜索域	elasticsearch.CfnDomain	返回由模式创建的 Elasticsearch 域的实例。
IdentityPool	cognito.CfnIdentityPool	返回由模式创建的 Cognito 身份池的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
userPool	cognito.UserPool	返回由模式创建的 Cognito 用户池的实例。
UserPoolClient	cognito.UserPoolClient	返回由模式创建的 Cognito 用户池客户端的实例。

Lambda 函数

此模式需要一个 Lambda 函数，该函数可以从 DynamoDB 流将数据发布到 Elasticsearch 服务。提供示例函数此处。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon DynamoDB 表

- 将 DynamoDB 表的计费模式设置为按需（按请求付费）
- 使用 AWS 托管的 KMS 密钥为 DynamoDB 表启用服务器端加密

- 为 DynamoDB 表创建名为 “id” 的分区键
- 删除 CloudFormation 堆栈时保留表
- 实现连续备份和时间点恢复

AWS Lambda 函数

- 为 Lambda 函数配置受限权限访问 IAM 角色
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接
- 启用 X-Ray 跟踪
- 启用故障处理功能：启用等分功能错误；设置默认的最长记录时间 (24 小时)；设置默认的最大重试次数 (500)；以及在出现故障时将 SQS 死信队列部署为目标
- 设置环境变量：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

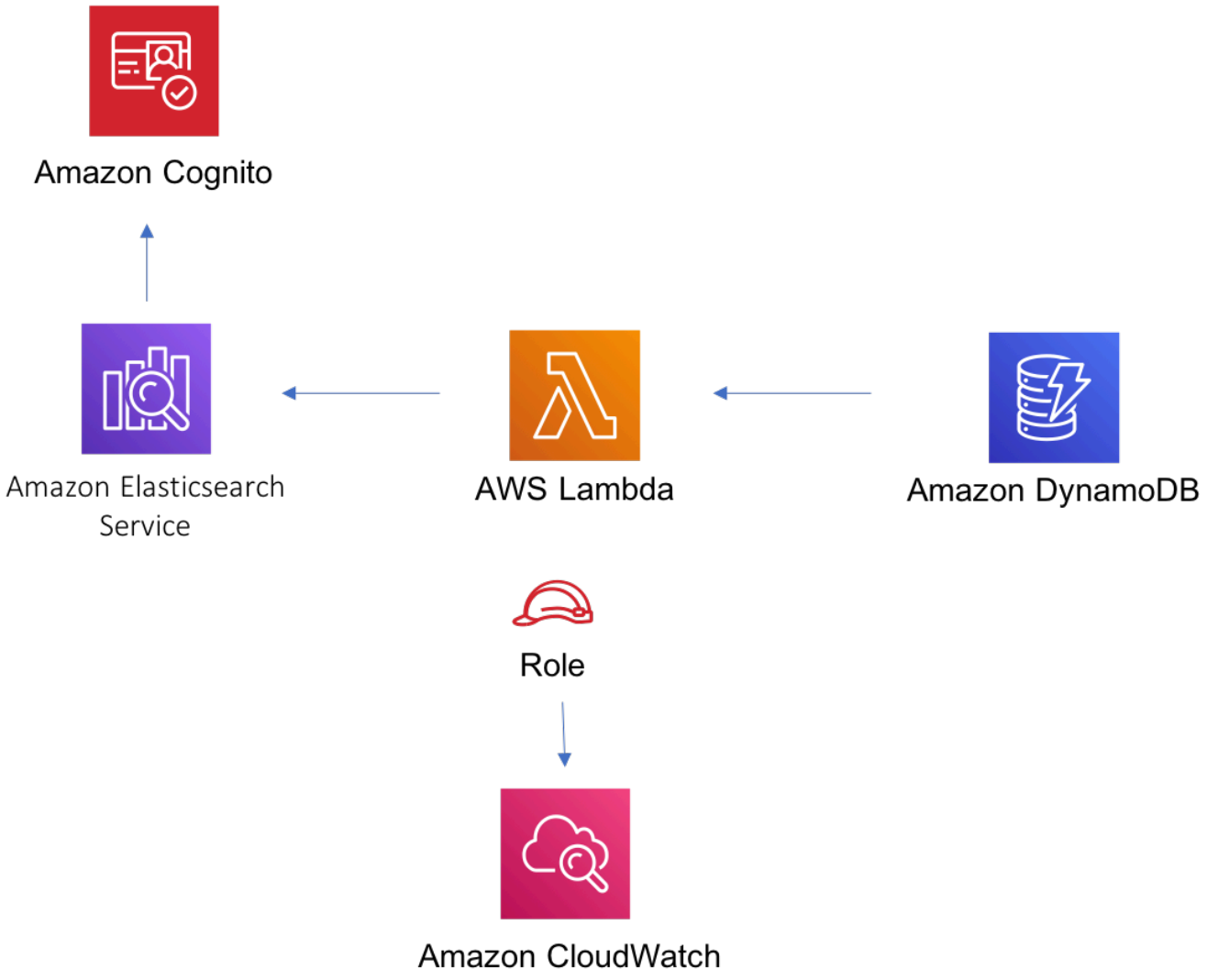
Amazon Cognito

- 为 UserPool 设置密码策略
- 强制执行用户池的高级安全模式

Amazon Elasticsearch Service

- 为弹性搜索域部署最佳实践 CloudWatch 警报
- 使用 Cognito 用户池保护 Kibana 仪表板访问
- 使用 AWS 托管的 KMS 密钥为 Elasticsearch 域启用服务器端加密
- 启用 Elasticsearch 域的节点到节点加密
- 为 Amazon ES 域配置集群

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aw-动态-流-lambda-弹性搜索-基巴纳

AWS 事件-规则-运动火星-3

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_events_rule_kinesisfirehose_s3`
TypeScript	`@aws-solutions-constructs/aws-events-rule-kinesisfirehose-s3`
Java	`software.amazon.awsconstructs.services.eventsrulekinesisfirehoses3`

Overview

此 AWS 解决方案构造实施了 Amazon CloudWatch Events 规则，将数据发送到连接到 Amazon S3 存储桶的 Amazon Amazon Kinesis Data Firehose 传输流。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsEventsRuleToKinesisFirehoseToS3Props

模式构建道具

名称	类型	描述
事件道具	events.RuleProps	用户提供的属性，用于覆盖 CloudWatch 事件规则的默认属性。
火焰管道具？	aws-kinesisfirehose.CfnDeliveryStreamProps	可选用户提供的道具来覆盖 Kinesis 消防管交付流的默认道具。
现有的存储桶吗？	s3.IBucket	S3 存储桶对象的现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。
桶道具？	s3.BucketProps	用户提供的可选道具，用于覆盖 S3 存储桶的默认道具。

名称	类型	描述
日志组道具？	logs.LogGroupProps	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
事件规则	events.Rule	返回由模式创建的事件规则的实例。
运动消防管	kinesisfirehose.CfnDeliveryStream	返回由模式创建的 Kinesis 消防管传递流的实例。
S3 存储桶	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。
事件角色？	iam.Role	返回由构造为 CloudWatch 事件规则创建的角色实例。
火焰丝	iam.Role	返回由 Kinesis 防火管交付流模式创建的 IAM 角色的实例。
运动消防管理集团	logs.LogGroup	返回由 Kinesis Firehose 访问日志发送到的模式创建的日志组的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon CloudWatch Events

- 为事件规则配置最低权限访问 IAM 角色，以发布到 Kinesis 防火管传递流。

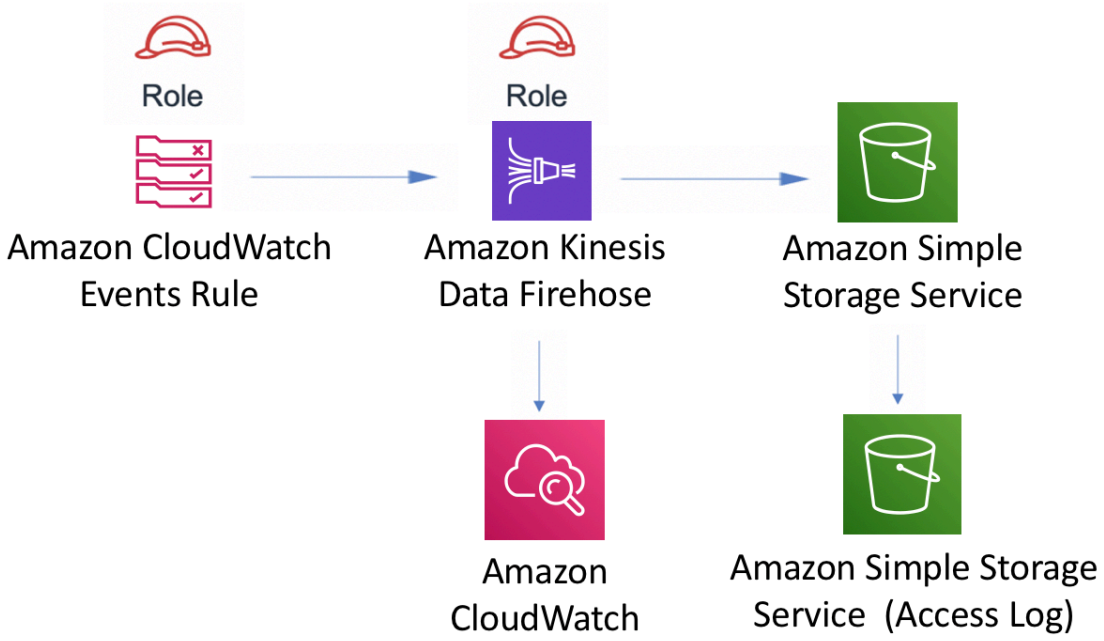
Amazon Kinesis Firehose

- 为 Kinesis 消防管启用 CloudWatch 日志记录。
- 配置 Amazon Kinesis Firehose 的 Amazon Firehose 的 Amazon 的最低权限访问 IAM 角色。

Amazon S3 存储桶

- 配置存储桶的访问日志记录。
- 使用 AWS 托管 KMS 密钥为存储桶启用服务器端加密。
- 开启对存储桶的版本控制。
- 不允许存储桶的公共访问。
- 删除 CloudFormation 堆栈时保留存储桶。
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-结构/aw-事件-规则-运动防火管-S3

aws-事件-规则-动态流

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
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 解决方案构造实施了 Amazon CloudWatch Events 规则，将数据发送到 Amazon Kinesis 数据流。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsEventsRuleToKinesisStreamsProps

模式构建道具

名称	类型	描述
事件道具	events.RuleProps	用户提供的属性，用于覆盖 CloudWatch 事件规则的默认属性。
现有的河流 J？	kinesis.Stream	现有的 Kinesis 流实例，提供了这个和kinesisStreamProps 会导致错误。
运动流道具？	kinesis.StreamProps	用户提供的可选道具来覆盖 Kinesis 流的默认道具。
创造云监视图	boolean	是否创建推荐的 CloudWatch 警报。

模式属性

名称	类型	描述
事件规则	events.Rule	返回由模式创建的事件规则的实例。
KinesisStream	kinesis.Stream	返回由模式创建的 Kinesis 流的实例。

名称	类型	描述
事件角色？	iam.Role	返回由构造为 CloudWatch 事件规则创建的角色实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon CloudWatch Events 规则

- 为事件规则配置最低权限访问 IAM 角色，以发布到 Kinesis 数据流。

Amazon Kinesis Stream

- 使用 AWS 托管 KMS 密钥为 Kinesis 数据流启用服务器端加密。

Architecture

GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案结构/awS-事件-规则-动态流

AWS 事件-规则-lambda

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 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 解决方案构造实现 AWS Events 规则和 AWS Lambda 函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsEventsRuleToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。

名称	类型	描述
Lambda 功能道具	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
事件道具	events.RuleProps	用户提供的事件道具来覆盖默认值

模式属性

名称	类型	描述
事件规则	events.Rule	返回由模式创建的事件规则的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

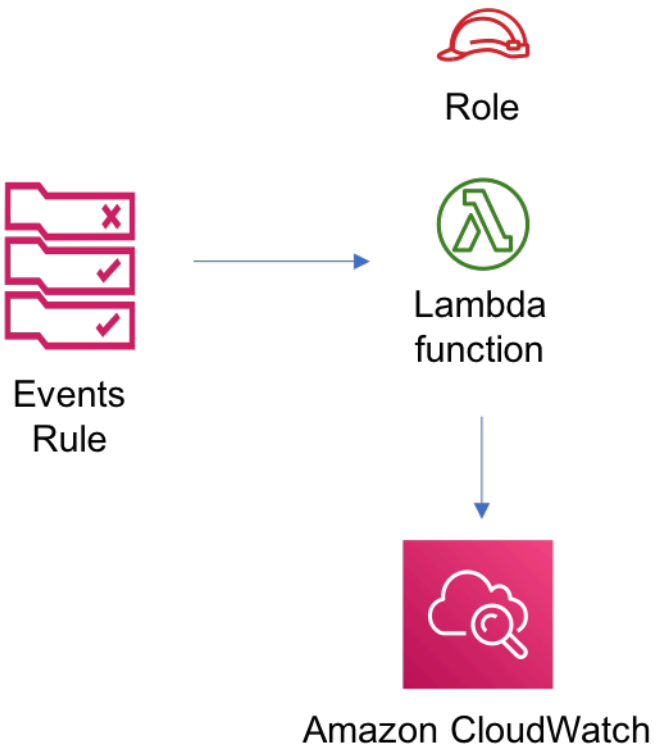
Amazon CloudWatch Events 规则

- 向 CloudWatch 事件授予最低权限以触发 Lambda 函数

AWS Lambda 函数

- 为 Lambda 函数配置受限权限访问 IAM 角色
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接
- 启用 X-Ray 跟踪
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-构造/AWS-事件-规则-lambda

AWS 事件-规则-sns

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_events_rule_sns`
TypeScript	`@aws-solutions-constructs/aws-events-rule-sns`
Java	`software.amazon.awsconstructs.services.eventsrulesns`

Overview

此模式实施与 Amazon SNS 主题相连的 Amazon CloudWatch Events 规则。

这是一个最小的可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsEventsRuleToSnsProps

模式构建道具

名称	类型	描述
事件道具	events.RuleProps	用户提供的属性，用于覆盖 CloudWatch 事件规则的默认属性。
现有的托比克吗？	sns.Topic	SNS 主题对象的现有实例，同时提供此和topicProps 会导致错误。
主题道具？	sns.TopicProps	用户提供的可选属性，用于覆盖 SNS 主题的默认属性。忽略，如果existingTopicObj 提供。
是否启用用客户管理的密钥加密？	boolean	是否使用由此 CDK 应用程序管理或导入的客户管理的加密密钥。如果导入加密密钥，则必须在encryptionKey 属性。

名称	类型	描述
encryptionKey	kms.Key	要使用的可选现有加密密钥，而不是默认加密密钥。
加密钥匙道具？	kms.KeyProps	用户提供的可选属性，用于覆盖加密密钥的默认属性。

模式属性

名称	类型	描述
事件规则	events.Rule	返回由模式创建的事件规则的实例。
snsTopic	sns.Topic	返回由模式创建的 SNS 主题的实例。
encryptionKey	kms.Key	返回由模式创建的加密密钥的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

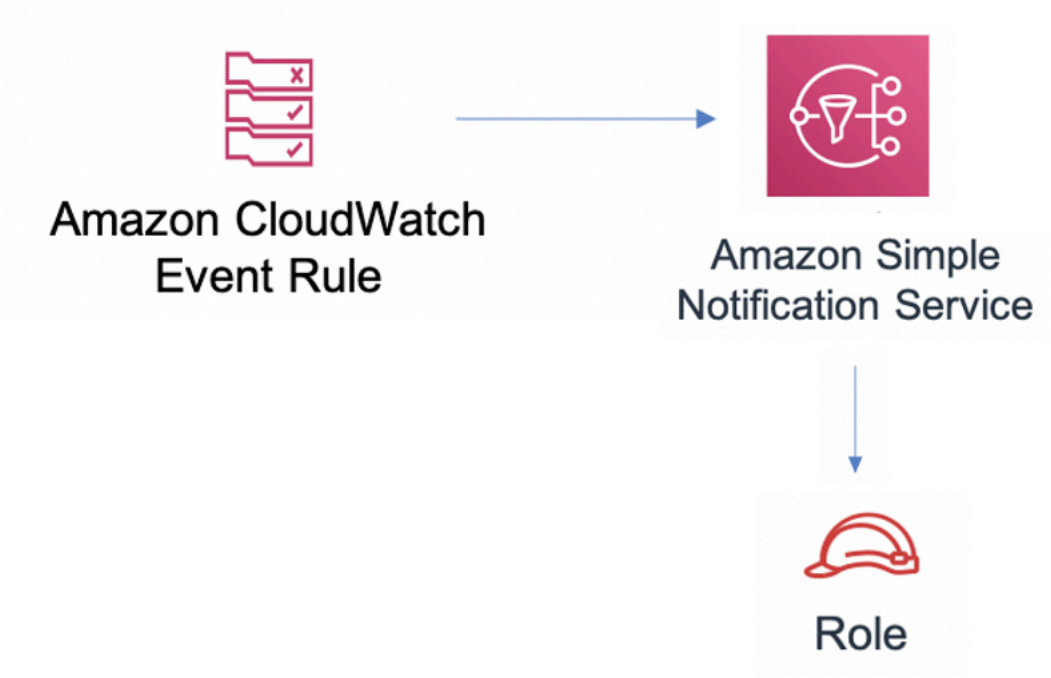
Amazon CloudWatch Events

- 向 CloudWatch 事件授予最低权限，以发布到 SNS 主题。

Amazon SNS 主题

- 为 SNS 主题配置最小权限访问权限。
- 使用客户托管的 AWS KMS 密钥为 SNS 主题启用服务器端加密。
- 强制传输中数据加密。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-构造/AWS-事件-规则-sns

AWS 事件规则平方米

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此模式实现与 Amazon SQS 队列相连的 Amazon CloudWatch Events 规则。

这是一个最小的可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsEventsRuleToSqsProps

模式构建道具

名称	类型	描述
事件道具	events.RuleProps	用户提供的属性，用于覆盖 CloudWatch 事件规则的默认属性。
现有队列 OBJ？	sqs.Queue	要使用的可选现有 SQS 队列，而不是默认队列。同时提供此和queueProps 会导致错误。
队列道具？	sqs.QueueProps	用户提供的可选属性，用于覆盖 SQS 队列的默认属性。忽略，如果existingQueueObj 提供。

名称	类型	描述
是否启用队列清除？	boolean	是否向 Lambda 函数授予其他权限，使其能够清除 SQS 队列。默认值为 false。
部署死信件队列？	boolean	是否创建要用作死信队列的辅助队列。默认值为 true。
死书队列道具？	sqs.QueueProps	用户提供的可选道具，用于覆盖死信队列的默认道具。仅当deployDeadLetterQueue 属性设置为 true。
maxReceiveCount？	number	消息在发送到死信队列之前可能会失败出队的次数。默认值为 15。
是否启用用客户管理的密钥加密？	boolean	是否使用由此 CDK 应用程序管理或导入的客户管理的加密密钥。如果导入加密密钥，则必须在encryptionKey 属性。
encryptionKey？	kms.Key	要使用的可选现有加密密钥，而不是默认加密密钥。
加密钥匙道具？	kms.KeyProps	用户提供的可选属性，用于覆盖加密密钥的默认属性。

模式属性

名称	类型	描述
事件规则	events.Rule	返回由模式创建的事件规则的实例。

名称	类型	描述
SQUUE	sqs.Queue	返回由模式创建的 SQS 队列的实例。
encryptionKey	kms.Key	返回由模式创建的加密密钥的实例。
死信队列？	sqs.Queue	返回由模式创建的死信队列的实例（如果已部署）。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

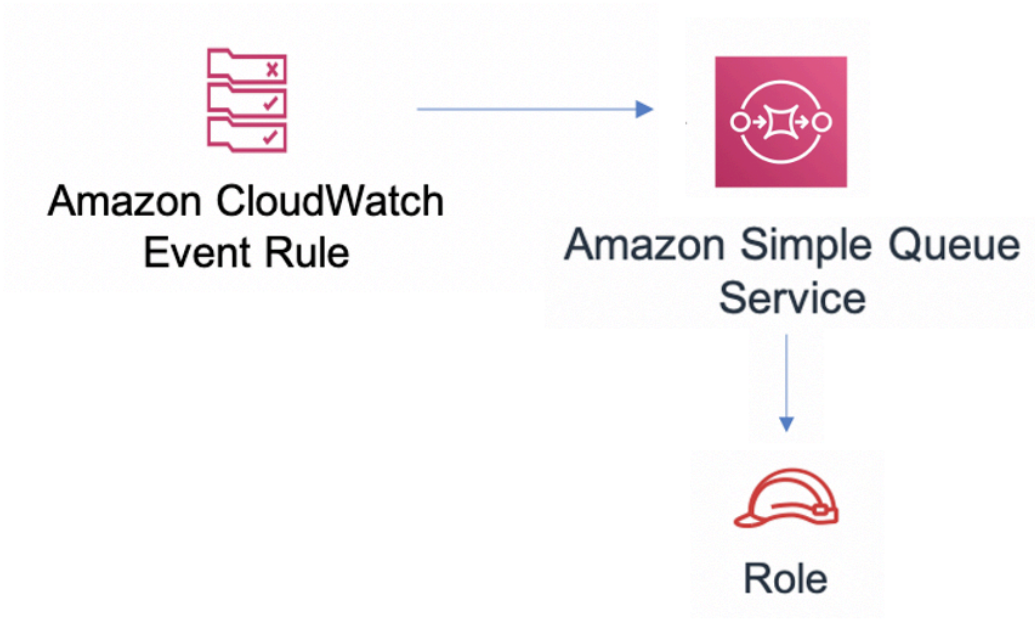
Amazon CloudWatch Events 规则

- 向 CloudWatch 事件授予最低权限以发布到 SQS 队列的权限。

Amazon SQS 队列

- 为源队列部署死信队列。
- 使用客户托管的 AWS KMS 密钥为源队列启用服务器端加密。
- 强制对传输中数据加密。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-构造/AWS-事件-规则平方

aws-事件-规则步进函数

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_events_rule_step_function`
 TypeScript	`@aws-solutions-constructs/aws-events-rule-step-function`
 Java	`software.amazon.awsconstructs.services.eventsrulestepfunction`

Overview

此 AWS 解决方案构造实现了 AWS 事件规则和 AWS 步骤函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsEventsRuleToStepFunctionProps

模式构建道具

名称	类型	描述
国家道具	sfn.StateMachineProps	用户提供的可选道具来覆盖 SFN.Statemachine 的默认道具
事件道具	events.RuleProps	用户提供的事件道具来覆盖默认值
创造云监视图	boolean	是否创建推荐的 CloudWatch 警报。
日志组道具？	logs.LogGroupProps	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
CloudWatch	cloudwatch.Alarm[]	返回模式创建的一个或多个 CloudWatch 警报的列表。

名称	类型	描述
事件规则	events.Rule	返回由模式创建的事件规则的实例。
StatataMachine	sfn.StateMachine	返回由模式创建的状态机的实例。
国家机械学组	logs.LogGroup	返回由状态机模式创建的日志组的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

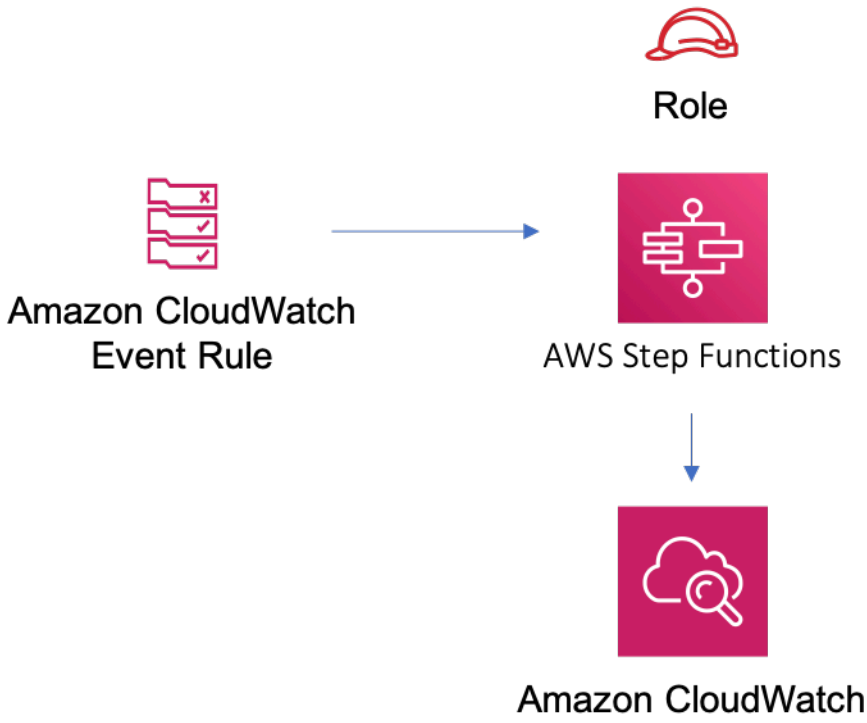
Amazon CloudWatch Events 规则

- 向 CloudWatch 事件授予最低权限以触发 Lambda 函数

AWS Step Function

- 启用 API Gateway 的 CloudWatch 日志
- 针对步进功能部署最佳实践 CloudWatch 警报

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/awS-事件-规则步进函数

aws-IoT 运动火管-3

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 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 解决方案构造实施了 AWS IoT MQTT 主题规则，将数据发送到连接到 Amazon S3 存储桶的 Amazon Amazon Kinesis Data Firehose 传输流。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsIotToKinesisFirehoseToS3Props

模式构建道具

名称	类型	描述
物质规则道具	iot.CfnTopicRuleProps	用户提供的 CFNTopicrup 来覆盖默认值
火焰管道具？	kinesisfirehose.CfnDeliveryStreamProps	可选用户提供的道具来覆盖 Kinesis 消防管交付流的默认道具
现有的存储桶吗？	s3.Bucket	S3 存储桶对象的现有实例，同时提供此和bucketProps 会导致错误。
桶道具？	s3.BucketProps	用户提供的道具来覆盖 S3 存储桶的默认道具。如果提供了这一点，那么还提供bucketProps 是一个错误。
日志组道具？	logs.LogGroupProps	可选的用户提供的道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
IOT 行动角色	iam.Role	返回由 IoT 规则的模式创建的 IAM 角色的实例。
国际主题规则	iot.CfnTopicRule	返回由模式创建的 IoT 主题规则的实例。
运动消防管	kinesisfirehose.CfnDeliveryStream	返回由模式创建的 Kinesis 消防管传递流的实例。
运动消防管理集团	logs.LogGroup	返回由 Kinesis Firehose 访问日志发送到的模式创建的日志组的实例。
火焰丝	iam.Role	返回由 Kinesis 防火管交付流模式创建的 IAM 角色的实例。
S3Bucket？	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon IoT 规则

- 为亚马逊 IoT 配置最低权限访问 IAM 角色

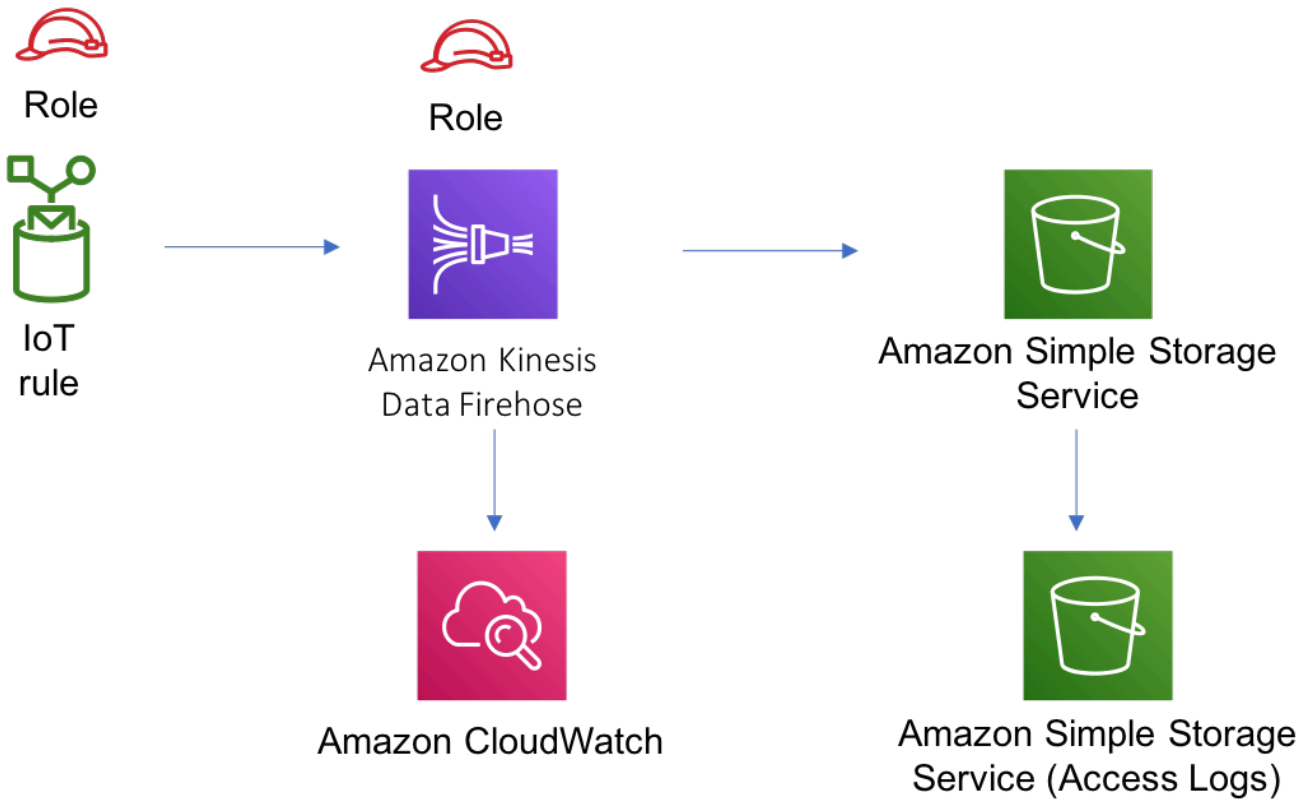
Amazon Kinesis Firehose

- 为 Kinesis 消防管启用 CloudWatch 视日志记录
- 为 Amazon Kinesis Firehose 配置最低权限访问 IAM 角色

Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密
- 启用 S3 存储桶的版本控制
- 不允许 S3 存储桶的公共访问
- 删除 CloudFormation 堆栈时保留 S3 存储桶
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/AWS-物联网-运动防火管-S3

AWS-很少-拉姆达

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造模式实现了 AWS IoT MQTT 主题规则和 AWS Lambda 函数模式。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsIotToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属

名称	类型	描述
		性。忽略，如果existingLambdaObj 提供。
物质规则道具？	iot.CfnTopicRuleProps	用户提供的 CFNTopicrup 来覆盖默认值

模式属性

名称	类型	描述
国际主题规则	iot.CfnTopicRule	返回由模式创建的 IoT 主题规则的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

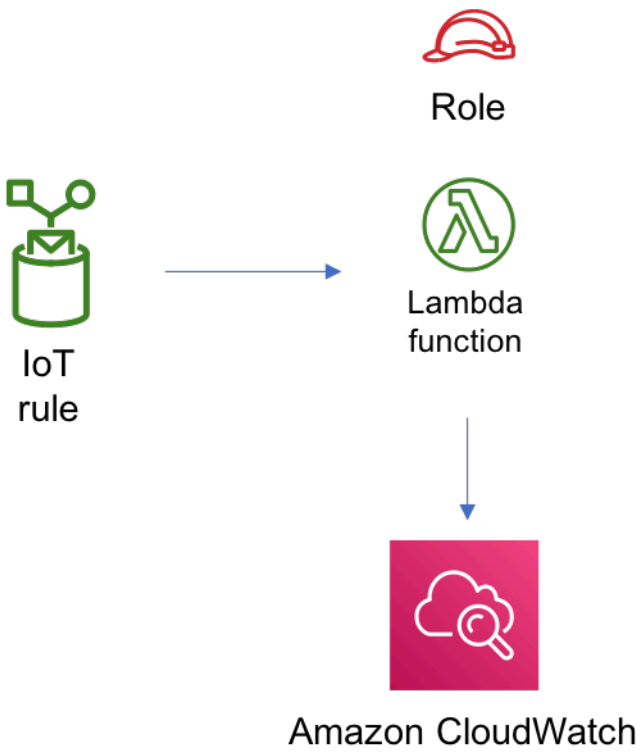
亚马逊 IoT 规则

- 为 Amazon IoT 配置最低权限访问 IAM 角色。

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-结构/AWS-物联网-lambda

aws-IoT-拉姆达-发电机 b

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_iot_lambda_dynamodb`
TypeScript	`@aws-solutions-constructs/aws-iot-lambda-dynamodb`
Java	`software.amazon.awsconstructs.services.iotlambdadynamodb`

Overview

此 AWS 解决方案构造模式实现具有最低特权权限的 AWS IoT 主题规则、AWS Lambda 函数和 Amazon DynamoDB 表。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsIotToLambdaToDynamoDBProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
物质规则道具	iot.CfnTopicRuleProps	用户提供的道具可覆盖默认道具
可发电道具？	dynamodb.TableProps	用户提供的可选道具来覆盖 DynamoDB 表的默认道具
表权限？	string	要授予 Lambda 函数的可选表权限。可能会指定以下任一选

名称	类型	描述
		项：All、Read、ReadWrite，或者Write。

模式属性

名称	类型	描述
动态表	dynamodb.Table	返回由模式创建的 DynamoDB 表的实例。
国际主题规则	iot.CfnTopicRule	返回由模式创建的 IoT 主题规则的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon IoT 规则

- 为 Amazon IoT 配置最低权限访问 IAM 角色。

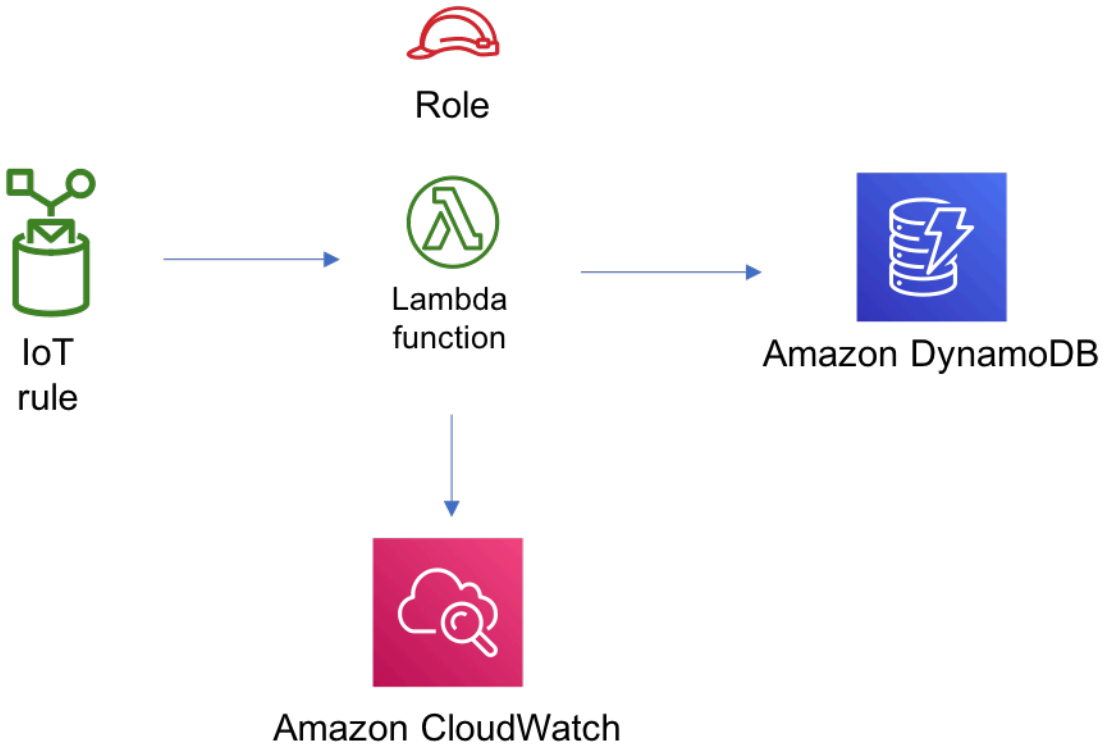
AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Amazon DynamoDB

- 将 DynamoDB 表的计费模式设置为按需（按请求付费）。
- 使用 AWS 托管的 KMS 密钥为 DynamoDB 表启用服务器端加密。
- 为 DynamoDB 表创建一个名为 “id” 的分区键。
- 删除 CloudFormation 堆栈时保留该表。
- 启用连续备份和时间点恢复。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-构造/AWS-物联网-拉姆达-发电机 b

AWS-运动火焰管-3

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

AWS 解决方案构造实现 Amazon Kinesis Data Firehose 传输流。

以下是 TypeScript 中的最小可部署模式定义：

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

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

Initializer

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

参数

- scopeConstruct
- idstring
- propsKinesisFirehoseToS3Props

模式构建道具

名称	类型	描述
桶道具？	s3.BucketProps	用户提供的可选道具来覆盖 S3 存储桶的默认道具。
现有的存储桶吗？	s3.IBucket	S3 存储桶的可选现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。
现有日志存储桶 TobJ？	s3.IBucket	由模式创建的 S3 存储桶日志 S3 存储桶的可选现有实例。
火焰管道具？	kinesisfirehose.CfnDeliveryStreamProps | any	可选用户提供的道具来覆盖 Kinesis 消防管交付流的默认道具。
日志组道具？	logs.LogGroupProps	可选用户提供的道具来覆盖 CloudWatchLogs 组的默认道具。

模式属性

名称	类型	描述
运动消防管	kinesisfirehose.CfnDeliveryStream	返回由构造创建的动力火管。CFN 交付流的实例。
运动消防管理集团	logs.LogGroup	返回由为 Kinesis Data Firehose 传递流构造创建的 logs.Loggroup 的实例。
火焰丝	iam.Role	返回由结构为 Kinesis Data Firehose 传递流创建的 IAM.Role 实例。
S3Bucket	s3.Bucket	返回由构造创建的 S3.bucket 实例。
S3 记录桶？	s3.Bucket	返回由构造创建的 S3.bucket 的实例，作为主存储桶的日志记录桶。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon Kinesis Firehose

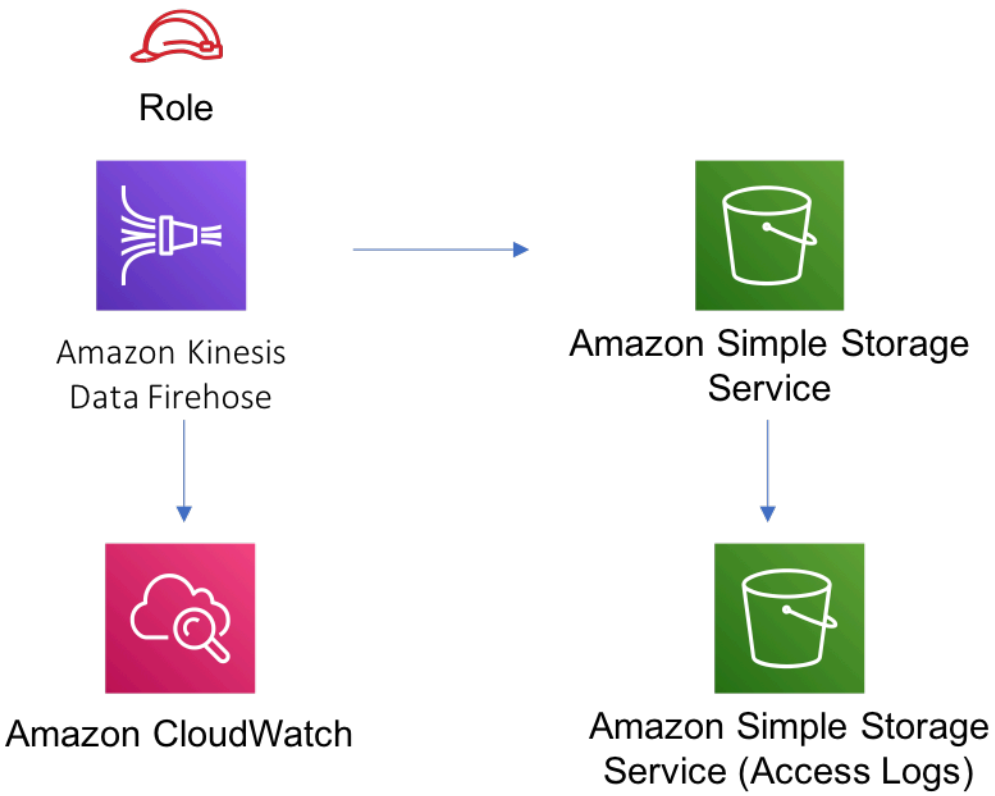
- 为 Kinesis 消防管启用 CloudWatch 视日志记录
- 为 Amazon Kinesis Firehose 配置最低权限访问 IAM 角色

Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密
- 启用 S3 存储桶的版本控制
- 不允许 S3 存储桶的公共访问

- 删除 CloudFormation 堆栈时保留 S3 存储桶
- 实施传输中数据加密
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/awS-运动防火管-S3

aws--动力学防火丝管-3 和动力学分析

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
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 解决方案构造实现了连接到 Amazon S3 存储桶的 Amazon Amazon Kinesis Firehose 传输流和 Amazon Kinesis Analytics 应用程序。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scope`Construct`
- id`string`
- props`KinesisFirehoseToAnalyticsAndS3Props`

模式构建道具

名称	类型	描述
火焰管道具？	kinesisFirehose.CfnDeliveryStreamProps	可选的用户提供的道具来覆盖 Kinesis 消防管交付流的默认道具。
运动分析道具？	kinesisAnalytics.CfnApplicationProps	用户提供的可选道具，用于覆盖 Kinesis 分析应用程序的默认道具。
现有的存储桶吗？	s3.IBucket	S3 存储桶对象的现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。|
桶道具？	s3.BucketProps	用户提供的可选属性，用于覆盖存储桶的默认属性。忽略，如果existingBucketObj 提供。
日志组道具？	logs.LogGroupProps	可选的用户提供的道具覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
运动分析	kinesisAnalytics.CfnApplication	返回由模式创建的 Kinesis 分析应用程序的实例。
运动消防管	kinesisfirehose.CfnDeliveryStream	返回由模式创建的 Kinesis 消防管传递流的实例。

名称	类型	描述
运动消防管理集团	logs.LogGroup	返回由 Kinesis Firehose 访问日志发送到的模式创建的日志组的实例。
火焰丝	iam.Role	返回由 Kinesis 防火管交付流模式创建的 IAM 角色的实例。
S3Bucket	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon Kinesis Firehose

- 为 Kinesis 消防管启用 CloudWatch 视日志记录
- 为 Amazon Kinesis Firehose 配置最低权限访问 IAM 角色

Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密
- 启用 S3 存储桶的版本控制
- 不允许 S3 存储桶的公共访问
- 删除 CloudFormation 堆栈时保留 S3 存储桶
- 实施传输中数据加密
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储

Amazon Kinesis Data Analytics

- 为 Amazon Kinesis Analytics 配置最低权限访问 IAM 角色

Architecture

GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/awS-运动防火丝管-S3 和运动分析

aws-kinesis-强力工作

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造部署 Amazon Kinesis 数据流，并配置 AWS Glue Job，使用适当的资源/属性执行自定义 ETL 转换，以实现交互和安全。它还会创建一个 Amazon S3 存储桶，其中可以上传 AWS Glue 合 Job 的 Python 脚本。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scope`Construct`
- id`string`
- props`KinesisstreamsToGluejobProps`

模式构建道具

名称	类型	描述
运动流道具？	kinesis.StreamProps	用户提供的可选道具，用于覆盖 Amazon Kinesis 数据流的默认道具。
现有的河流 J？	kinesis.Stream	现有的 Kinesis 流实例，提供了这个和kinesisStreamProps 会导致错误。
粘合工作道具？	cfnJob.CfnJobProps	用户提供的道具，用于覆盖 AWS Glue 作业的默认道具。
现有的粘合工作？	cfnJob.CfnJob	AWS Glue Job 的现有实例，同时提供此和glueJobProps 会导致错误。
是否存在数据库？	CfnDatabase	要与此构造一起使用的现有 AWS Glue 数据库。如果设置了此设置，则databaseProps 将被忽略。
数据库道具？	CfnDatabaseProps	用户提供的道具，用于覆盖用于创建 AWS Glue 数据库的默认道具。
现有表？	CfnTable	AWS Glue 表的现有实例。如果设置了此设置，则tableProps 和fieldSchema 将被忽略。
餐具道具？	CfnTableProps	用户提供的道具，用于覆盖用于创建 AWS Glue 表的默认道具。

名称	类型	描述
字段架构？	CfnTable.ColumnProperty[]	用户提供的架构结构，用于创建 AWS Glue 表。
输出数据存储？	SinkDataStoreProps	用户为 Amazon S3 存储桶提供的道具，用于存储 AWS Glue 作业的输出。目前仅支持 Amazon S3 作为输出数据存储类型。

SinkDataStoreProps

名称	类型	描述
存在 3 个输出桶？	Bucket	应在其中写入数据的 S3 存储桶的现有实例。同时提供此和outputBucketProps 会导致错误。
输出桶道具	BucketProps	用户提供的存储桶属性，用于创建用于存储 AWS Glue 任务输出的 Amazon S3 存储桶。
数据存储类型	SinkStoreType	汇数据存储类型。

SinkStoreType

数据存储类型的枚举，其中可能包括 S3、DynamoDB、DocumentDB、RDS 或 Redshift。当前的构造实现仅支持 S3，但将来有可能添加其他输出类型。

名称	类型	描述
S3	string	S3 存储类型

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon Kinesis Streams

- 为 Amazon Kinesis 数据流配置最低权限访问 IAM 角色。
- 使用 AWS 托管 KMS 密钥为 Amazon Kinesis 流启用服务器端加密。
- 为亚马逊 Kinesis 流部署最佳实践亚马逊 CloudWatch 警报。

Glue Job

- 创建 AWS Glue 安全配置，用于为 CloudWatch、Job 书签和 S3 配置加密。CloudWatch 和 Job 书签使用为 AWS 胶水服务创建的 AWS 托管 KMS 密钥进行加密。S3 存储桶配置为 SSE-S3 加密模式。
- 配置允许 AWS Glue 从 Amazon Kinesis Data Streams 读取的服务角色策略。

Glue 数据库

- 创建 AWS Glue 数据库。AWS Glue 表将添加到数据库中。此表定义了 Amazon Kinesis 数据流中缓冲的记录的架构。

Glue 桌子

- 创建 AWS Glue 表。表架构定义基于 Amazon Kinesis 数据流中缓冲记录的 JSON 结构。

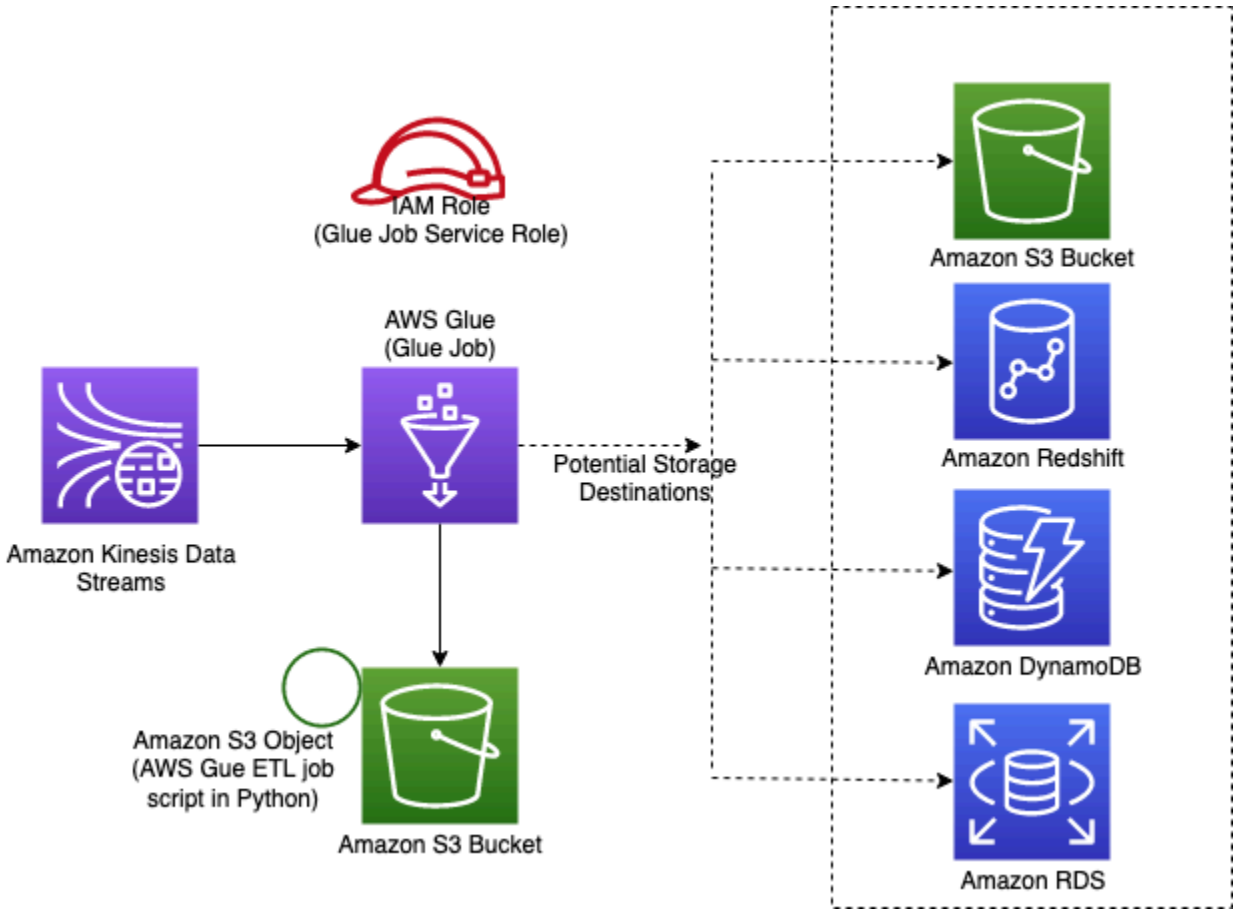
IAM 角色

- 具有以下权限的任务执行角色：1) 从 Amazon S3 存储桶位置读取 ETL 脚本；2) 从 Amazon Kinesis 数据流读取记录；3) 执行 Amazon Glue 作业。

输出 S3 存储桶

- 用于存储 ETL 转换输出的 Amazon S3 存储桶。此存储桶将作为参数传递给已创建的 AWS Glue 任务，以便在 ETL 脚本中使用该存储桶将数据写入其中。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-结构/awS-运动流-葡萄工作

AWS-运动系统-火焰管-3

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 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 解决方案构造实现了连接到 Amazon S3 存储桶的 Amazon Kinesis 数据消防软管 (KDF) 传输流的 Amazon Kinesis 数据流 (KDS)。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsKinesisStreams...ToS3Props

模式构建道具

名称	类型	描述
桶道具？	s3.BucketProps	用户提供的可选道具来覆盖 S3 存储桶的默认道具。
创造云监视器？	boolean	可选是否创建推荐的 CloudWatch 警报。
现有的存储桶吗？	s3.IBucket	S3 存储桶对象的可选现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。
现有日志存储桶 TobJ？	s3.IBucket	由模式创建的 S3 存储桶日志 S3 存储桶对象的可选现有实例。
现有的河流 J？	kinesis.Stream	现有的 Kinesis 流实例，提供了这个和kinesisStreamProps 会导致错误。
火焰管道具？	aws-kinesisfirehose.CfnDeliveryStreamProps | any	可选用户提供的道具来覆盖 Kinesis 消防管交付流的默认道具。
运动流道具？	kinesis.StreamProps	可选用户提供的道具来覆盖 Kinesis 流的默认道具。
日志组道具？	logs.LogGroupProps	可选的用户提供的道具来覆盖 CloudWatchLogs 组的默认道具。

模式属性

名称	类型	描述
CloudwatchAlinam	cloudwatch.Alarm[]	返回由构造创建的云监视。警报实例的列表。
运动消防管	kinesisfirehose.CfnDeliveryStream	返回由构造创建的动力火管。CFN 交付流的实例。
运动消防管理集团	logs.LogGroup	返回由为 Kinesis Data Firehose 传递流构造创建的 logs.Loggroup 的实例。
火焰丝	iam.Role	返回由结构为 Kinesis Data Firehose 传递流创建的 IAM.Role 实例。
运动流角色	iam.Role	返回由结构为 Kinesis 流创建的 IAM.Role 的实例。
S3Bu桶？	s3.Bucket	返回由构造创建的 S3.bucket 实例。
S3 记录桶？	s3.Bucket	返回由构造创建的 S3.bucket 的实例，作为主存储桶的日志记录桶。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon Kinesis Stream

- 为 Kinesis 流配置最低权限访问 IAM 角色
- 使用 AWS 托管 KMS 密钥为 Kinesis 流启用服务器端加密
- 为 Kinesis 流部署最佳实践 CloudWatch 警报

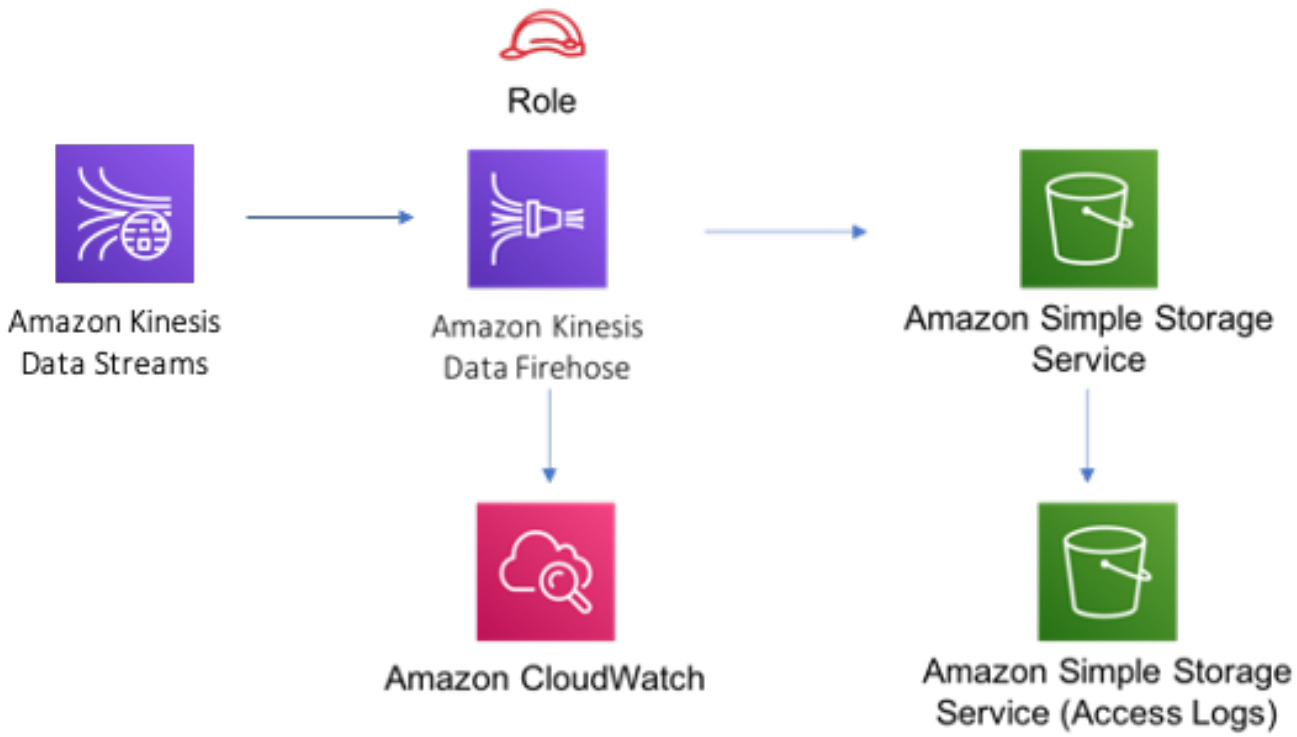
Amazon Kinesis Firehose

- 为 Kinesis 消防管启用 CloudWatch 视日志记录
- 为 Amazon Kinesis Firehose 配置最低权限访问 IAM 角色

Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密
- 实施传输中数据加密
- 启用存储桶版本控制
- 不允许 S3 存储桶的公共访问
- 删除 CloudFormation 堆栈时保留 S3 存储桶
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案结构/AWS-运动流-运动防火管-S3

aws-kinesis-拉姆达

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws-kinesis-streams-lambda`
 TypeScript	`@aws-solutions-constructs/aws-kinesisstreams-lambda`
 Java	`software.amazon.awsconstructs.services.kinesisstreamslambda`

Overview

此 AWS 解决方案构造部署了一个 Kinesis 流和 Lambda 函数，并具有适当的资源/属性，以实现交互和安全。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsKinesisStreamsToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
运动流道具？	kinesis.StreamProps	用户提供的可选道具，用于覆盖 Kinesis 流的默认道具。
现有的河流 J？	kinesis.Stream	现有的 Kinesis 流实例，提供了这个和kinesisStreamProps 会导致错误。
运动七源道具？	aws-lambda-event-sources.KinesisEventSourceProps	用户提供的可选道具，用于覆盖 Lambda 事件源映射的默认道具。
创造云监视图	boolean	是否创建推荐的 CloudWatch 警报。

模式属性

名称	类型	描述
Kinesis Stream	kinesis.Stream	返回由模式创建的 Kinesis 流的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。

名称	类型	描述
运动流角色	iam.Role	返回由 Kinesis 流的模式创建的 IAM 角色的实例。
CloudwatchAltam	cloudwatch.Alarm[]	返回模式创建的一个或多个 CloudWatch 警报的列表。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

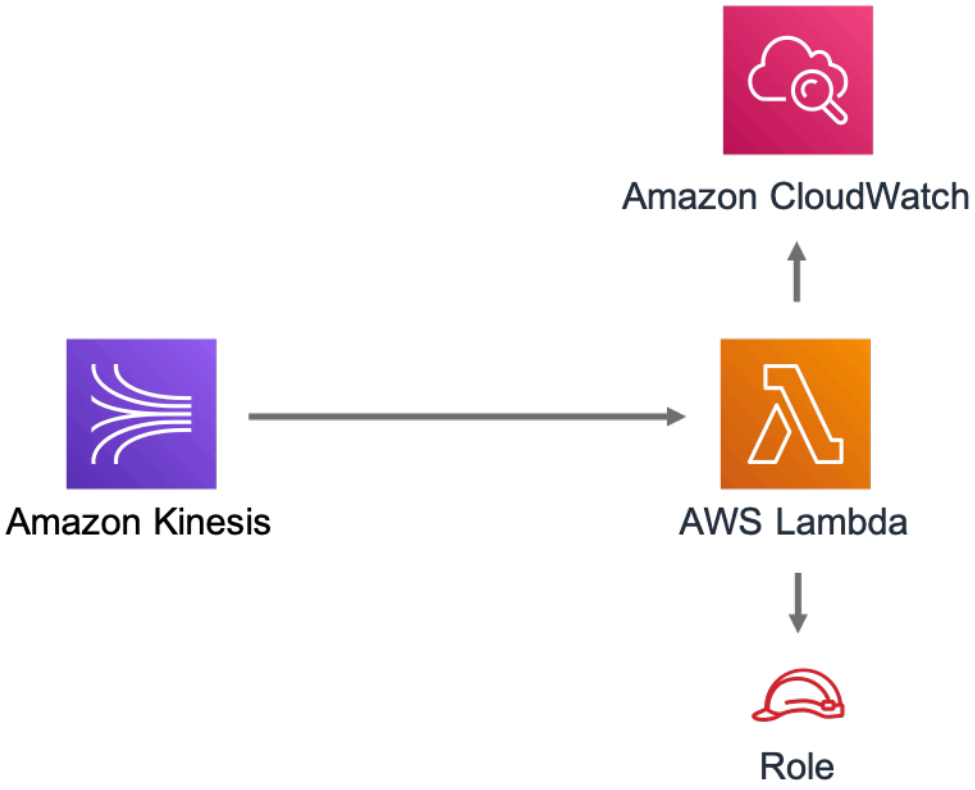
Amazon Kinesis Stream

- 为 Kinesis 流配置最低权限访问 IAM 角色。
- 使用 AWS 托管的 KMS 密钥为 Kinesis 流启用服务器端加密。
- 为 Kinesis 流部署最佳实践 CloudWatch 警报。

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 启用故障处理功能：启用功能 “错误” 等级；设置默认的最长记录时间 (24 小时)；设置默认的最大重试次数 (500)；以及在失败时将 SQS 死信队列部署为目标。
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aw-动态流-lambda

aws-lambda-dynamoDB

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_lambda_dynamodb`
TypeScript	`@aws-solutions-constructs/aws-lambda-dynamodb`
Java	`software.amazon.awsconstructs.services.lambdadynamodb`

Overview

此 AWS 解决方案构造实现具有最低权限权限的 AWS Lambda 函数和 Amazon DynamoDB 表。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToDynamoDBProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 则会提供。
可发电道具？	dynamodb.TableProps	用户提供的可选道具来覆盖 DynamoDB 表的默认道具
现有表格 J？	dynamodb.Table	DynamoDB 表对象的现有实例，提供了这个和dynamoTableProps 会导致错误。
表权限？	string	要授予 Lambda 函数的可选表权限。可指定下列选项之一：All、Read、ReadWrite，或者Write。
表环境变量名？	string	为 Lambda 函数设置的 DynamoDB 表环境变量的可选名称。

名称	类型	描述
现有 VPC？	ec2.IVpc	应在其中部署此模式的可选现有 VPC。在 VPC 中部署时，Lambda 函数将使用 VPC 中的 ENI 访问网络资源，并在 Amazon DynamoDB 的 VPC 中创建网关终端节点。如果提供了现有 VPC，则deployVpc 属性不能true。此操作使用ec2.IVpc允许客户端使用ec2.Vpc.fromLookup() 方法。
VPCPROP？	ec2.VpcProps	用户提供的可选属性，用于覆盖新 VPC 的默认属性。enableDnsHostnames 、enableDnsSupport 、natGateways ，和subnetConfiguration 是由模式设置的，所以这里提供的这些属性的任何值都将被覆盖。如果deployVpc 不是true那么这个属性将被忽略。

名称	类型	描述
部署 VPC？	boolean	是否创 VPC 基于vpcProps来部署这种模式。将此设置为 true 将部署最小的、最私有的 VPC 来运行模式： • CDK 程序使用的每个可用区内有一个隔离子网 • enableDnsHostnames 和enableDns Support 都将被设置为true 如果此属性为true，然后existingVpc 则无法指定 。默认值为 false。

模式属性

名称	类型	描述
动态表	dynamodb.Table	返回由模式创建的 DynamoDB 表的实例。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
VPC？	ec2.IVpc	返回模式使用的 VPC 上的接口（如果有）。这可能是由模式创建的 VPC，也可以是提供给模式构造函数的 VPC。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

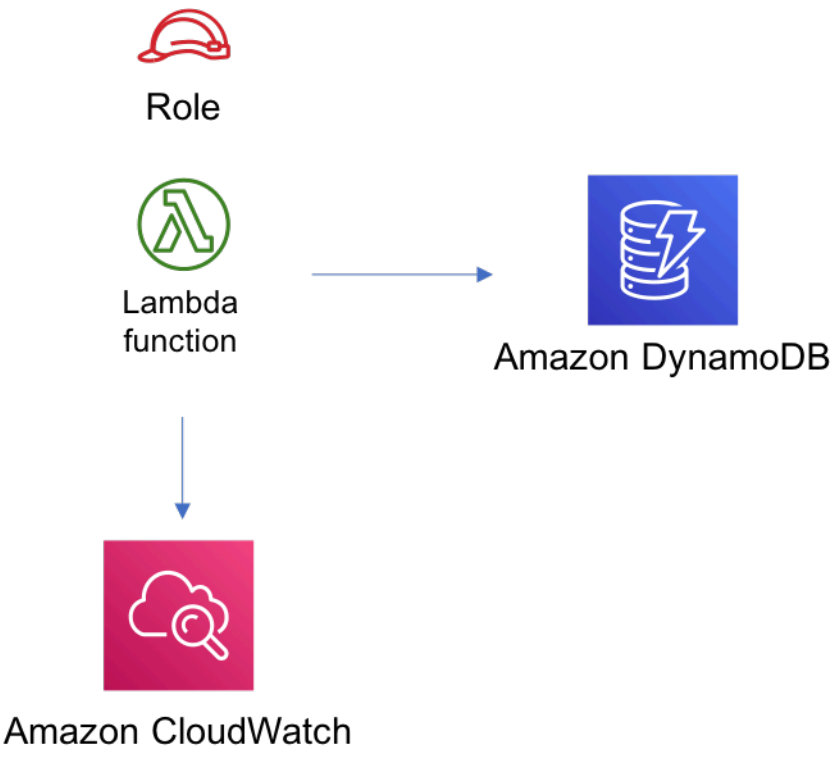
AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - DDB_TABLE_NAME (默认值)
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Amazon DynamoDB 表

- 将 DynamoDB 表的计费模式设置为按需（按请求付费）。
- 使用 AWS 托管的 KMS 密钥为 DynamoDB 表启用服务器端加密。
- 为 DynamoDB 表创建名为 “id” 的分区键。
- 删除 CloudFormation 堆栈时保留表。
- 启用连续备份和时间点恢复。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案结构/aw-拉姆达-发电机 b

-拉姆达-弹性搜索-基巴纳

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_lambda_elasticsearch_kibana`
 TypeScript	`@aws-solutions-constructs/aws-lambda-elasticsearch-kibana`
 Java	`software.amazon.awsconstructs.services.lambdaelasticsearchkibana`

Overview

此 AWS 解决方案构造实现了一个 AWS Lambda 函数和一个具有最低权限的 Amazon Elasticsearch Service 域。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scope`Construct`
- id`string`
- props`LambdaToElasticSearchAndKibanaProps`

模式构造道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略existingLambdaObj 提供。
埃斯领土道具？	elasticsearch.CfnDomainProps	用户提供的可选道具来覆盖 Amazon Elasticsearch Service 的默认道具
domainName	string	Cognito 和 Amazon Elasticsearch Service 的域名
认知域名？	string	可选的 Cognito 域名称。如果提供，它将用于 Cognito 域，并且domainName 将用于弹性搜索域。
创造云监视图	boolean	是否创建推荐的 CloudWatch 警报。
域端点变量名称？	string	为 Lambda 函数设置的 ElasticSearch 域端点环境变量的可选名称。

模式属性

名称	类型	描述
CloudwatchAltms？	cloudwatch.Alarm[]	返回模式创建的一个或多个 CloudWatch 警报的列表。
弹性搜索域	elasticsearch.CfnDomain	返回由模式创建的 Elasticsearch 域的实例。
弹性搜索域角色	iam.Role	返回由模式为 Elasticsearch 域创建的 IAM 角色的实例。
IdentityPool	cognito.CfnIdentityPool	返回由模式创建的 Cognito 身份池的实例。
LambdaFunction	lambda.Function	返回模式创建的 Lambda 函数的实例。
userPool	cognito.UserPool	返回由模式创建的 Cognito 用户池的实例。
UserPoolClient	cognito.UserPoolClient	返回由模式创建的 Cognito 用户池客户端的实例。

Lambda 函数

此模式需要一个 Lambda 函数，该函数可以从 DynamoDB 流将数据发布到 Elasticsearch 服务。提供示例函数HERE。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数启用重复使用连接。

- 启用 X-Ray 跟踪。
- 设置环境变量：
  - DOMAIN_ENDPOINT (默认值)
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

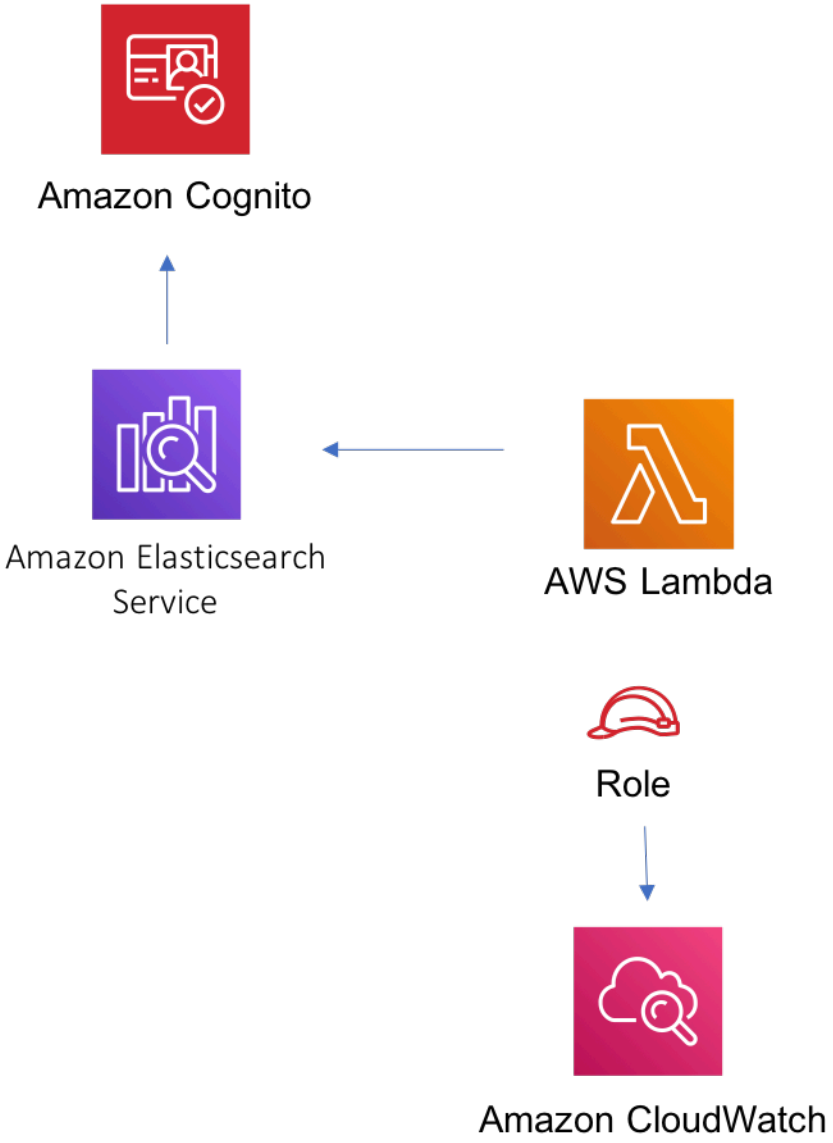
Amazon Cognito

- 设置用户池的密码策略。
- 强制执行用户池的高级安全模式。

Amazon Elasticsearch Service

- 为弹性搜索域部署最佳实践 CloudWatch 警报。
- 使用 Cognito 用户池保护 Kibana 仪表板访问权限。
- 使用 AWS 托管的 KMS 密钥为 Elasticsearch 域启用服务器端加密。
- 启用 Elasticsearch 域的节点到节点加密。
- 配置 Amazon ES 域的集群。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

	@aws-解决方案结构/aws-lambda-弹性搜索-基巴纳

aws-lambda-3

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造实现连接到 Amazon S3 存储桶的 AWS Lambda 函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToS3Props

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
现有的存储桶吗？	s3.IBucket	S3 存储桶对象的现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。
桶道具？	s3.BucketProps	用户提供的可选属性，用于覆盖存储桶的默认属性。忽略，如果existingBucketObj 提供。

名称	类型	描述
存储桶权限？	`string[]`	授予 Lambda 函数的可选存储桶权限。可能会指定下列一个或多个：`Delete`、`Put`、`Read`、`ReadWrit`、`Write`。
是否存在 VPC？	`ec2.IVpc`	应在其中部署此模式的可选现有 VPC。在 VPC 中部署时，Lambda 函数将使用 VPC 中的 ENI 访问网络资源，并在 Amazon SQS 的 VPC 中创建接口终端节点。如果提供了现有 VPC，则`deployVpc` 属性不能`true`。这使用`ec2.IVpc`允许客户端使用`ec2.Vpc.fromLookup()` 方法。
部署 VPC？	`boolean`	是否创 VPC 基于`vpcProps`将此模式部署到其中。将此设置为`true`将部署最小的、最私有的 VPC 来运行模式： • CDK 计划使用的每个可用区中有一个隔离的子网。 • `enableDnsHostnames` 和`enableDnsSupport` 都将被设置为`true`。 如果此属性为`true`，然后`existingVpc` 则无法指定 。默认值为 `false`。

名称	类型	描述
VPCPROP？	ec2.VpcProps	用户提供的可选属性，用于覆盖新 VPC 的默认属性。enableDnsHostnames 、enableDnsSupport 、natGateways 和subnetConfiguration 是由模式设置的，所以这里提供的这些属性的任何值都将被覆盖。如果deployVpc 不是true那么这个属性将被忽略。
存储桶变量名称？	string	为 Lambda 函数设置的 S3 存储桶环境变量的可选名称。

模式属性

名称	类型	描述
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
S3Bucket？	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。
VPC？	ec2.IVpc	返回模式使用的 VPC 实例（如果有）。这可能是由模式创建的 VPC，也可以是提供给模式构造函数的 VPC。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪
- 设置环境变量：
  - `S3_BUCKET_NAME` (默认值)
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录。
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密。
- 启用 S3 存储桶的版本控制。
- 不允许 S3 存储桶的公共访问。
- 删除 CloudFormation 堆栈时保留 S3 存储桶。
- 实施传输中数据加密。
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储。

Architecture

GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案-结构/aw-拉姆达-S3

aws-lambda-管理字符串参数

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 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 解决方案构造使用最低权限实现 AWS Lambda 函数和 AWS Systems Manager Parameter Store 字符串参数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToSsmstringparameterProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。

名称	类型	描述
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略existingLambdaObj 提供。
现有的字符串参数 ROBJ？	ssm.StringParameter	SSM String 参数对象的现有实例，同时提供此和stringParameterProps 会导致错误。
字符串参数道具？	ssm.StringParameterProps	可选的用户提供的道具来覆盖 SSM 字符串参数的默认道具。如果existingStringParameterObj 未设置，stringParameterProps 是必需的。唯一受支持ssm.StringParameterProps.type 是STRING如果提供了不同的值，它将被覆盖。
字符串参数变量名称？	string	为 Lambda 函数设置的 SSM 字符串参数环境变量的可选名称。

名称	类型	描述
是否存在 VPC？	ec2.IVpc	应在其中部署此模式的可选现有 VPC。在 VPC 中部署时，Lambda 函数将使用 VPC 中的 ENI 访问网络资源，并将在适用于 AWS Systems Manager 参数的 VPC 中创建接口终端节点。如果提供了现有 VPC，则deployVpc 属性不能是true。这使用ec2.IVpc，以允许客户端使用ec2.Vpc.fromLookup() 方法。
VPCPROP？	ec2.VpcProps	用户提供的可选属性，用于覆盖新 VPC 的默认属性。enableDns Hostnames 、enableDns Support 、natGatewa ys 和subnetCon figuration 是由模式设置的，所以这里提供的这些属性的任何值都将被覆盖。如果deployVpc 不是true那么这个属性将被忽略。

名称	类型	描述
部署 VPC？	boolean	是否创建新 VPCvpcProps将此模式部署到其中。将此设置为true将部署最小的、最私有的 VPC 来运行模式： • CDK 程序使用的每个可用区中都有一个隔离的子网。 • enableDnsHostnames 和enableDns Support 都将被设置为true。 如果此属性设置为true，然后existingVpc 则无法指定 。默认值为 false。
字符串参数权限？	string	授予 Lambda 函数的可选 SSM 字符串参数权限。可指定以下内容之一：Read、ReadWrite 。

模式属性

名称	类型	描述
LambdaFunction	lambda.Function	返回lambda.Function 由构造创建。
StringParameter	ssm.StringParameter	返回ssm.StringParameter 由构造创建。
VPC？	ec2.IVpc	返回模式使用的 VPC 上的接口（如果有）。这可能是由模式

名称	类型	描述
		创建的 VPC，也可以是提供给模式构造函数的 VPC。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

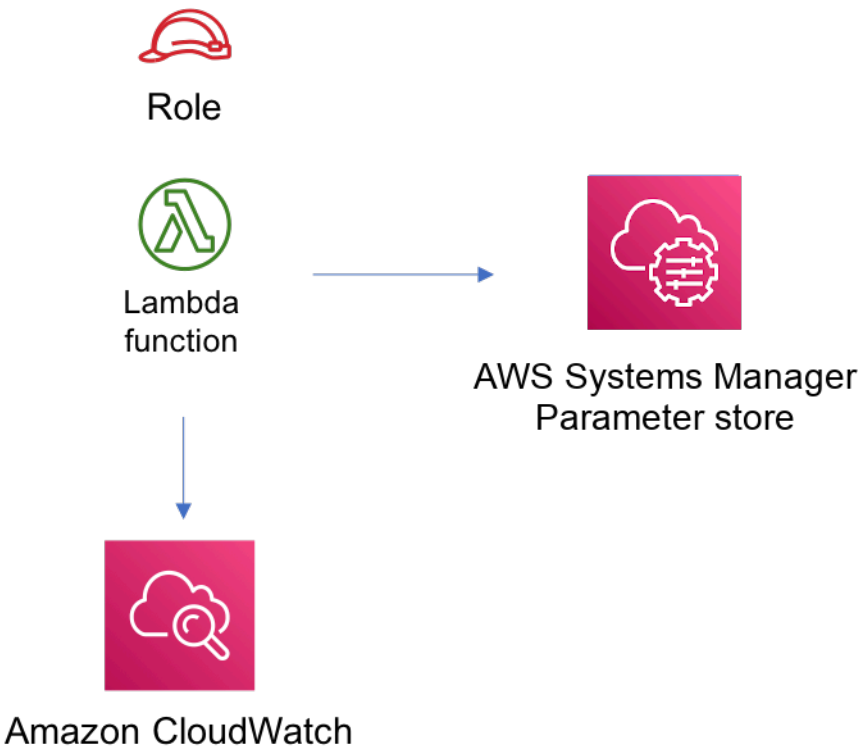
AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - `SSM_STRING_PARAMETER_NAME` (默认值)
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

Amazon AWS Systems Manager Parameter Store 字符串

- 启用关联的 AWS Lambda 函数的只读访问权限。
- 使用提供的值创建一个新的 SSM 字符串参数。
- 删除 CloudFormation 堆栈时，保留 SSM 字符串参数。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-lambda-管理字符串参数

aws-Lambda-圣马可点

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_lambda_sagemakerendpoint`
TypeScript	`@aws-solutions-constructs/aws-lambda-sagemakerendpoint`
Java	`software.amazon.awsconstructs.services.lambdasagemakerendpoint`

Overview

此 AWS 解决方案构造实现了连接到 Amazon Sagemaker 终端节点的 AWS Lambda 函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToSagemakerEndpointProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。
现有的信息传递点 TOBJ？	sagemaker.CfnEndpoint	要使用的可选现有 Sagemaker Enpoint。同时提供此和endpointProps 会导致错误。
模型道具？	sagemaker.CfnModelProps | any	用户提供的属性用于覆盖 Sagemaker 模型的默认属性。

名称	类型	描述
		至少`modelProps.primaryContainer` 才能创建模型。默认情况下，模式将创建具有最低所需权限的角色，但客户端可以使用`modelProps.executionRoleArn` 。
终端配置道具？	`sagemaker.CfnEndpointConfigProps`	用户提供的可选属性，用于覆盖 Sagemaker 端点配置的默认属性。
终端道具？	`sagemaker.CfnEndpointProps`	用户提供的可选属性，用于覆盖 Sagemaker 终端节点的默认属性。
现有 VPC？	`ec2.IVpc`	应在其中部署此构造的可选现有 VPC。在 VPC 中部署时，Lambda 函数和 Sagemaker 终端节点将使用 VPC 中的 ENI 访问网络资源。将在 VPC 中为 Amazon Sagemaker 运行时和 Amazon S3 VPC 终端节点创建一个接口终端节点。如果提供了现有 VPC，则`deployVpc` 属性不能`true`。

名称	类型	描述
VPCPROP？	ec2.VpcProps	用户提供的可选属性，用于覆盖新 VPC 的默认属性。enableDns Hostnames 、enableDns Support 、natGatewa ys 和subnetCon figuration 是由构造设置的，因此此处提供的这些属性的任何值都将被覆盖。如果deployVpc 不是true，则此属性将被忽略。
部署 VPC？	boolean	是否创 VPC 基于vpcProps来部署这种模式。将此设置为true将部署最小的、最私有的 VPC 来运行模式： • CDK 计划使用的每个可用区中的一个隔离子网。 • enableDnsHostnames 和enableDns Support 都将被设置为true。 如果该属性设置为true，然后existingVpc 则无法指定 。默认值为 false。
SagEMAKEN 变量名称？	string	为 Lambda 函数设置的 SageMaker 终端节点环境变量的可选名称。

模式属性

名称	类型	描述
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
萨格玛凯伦点	sagemaker.CfnEndpoint	返回由模式创建的 Sagemaker 端点的实例。
SagEMA 连接点配置？	sagemaker.CfnEndpointConfig	返回由模式创建的 SageMaker EndpointConfig 的实例，如果existingSagemakerEndpointObj 未提供。
萨格·制造模型？	sagemaker.CfnModel	返回由阵列创建的 Sagemaker 模型的实例，如果existingSagemakerEndpointObj 未提供。
VPC？	ec2.IVpc	返回由模式创建的 VPC 的实例，如果deployVpc 是true，或者如果existingVpc 提供。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

AWS Lambda 函数

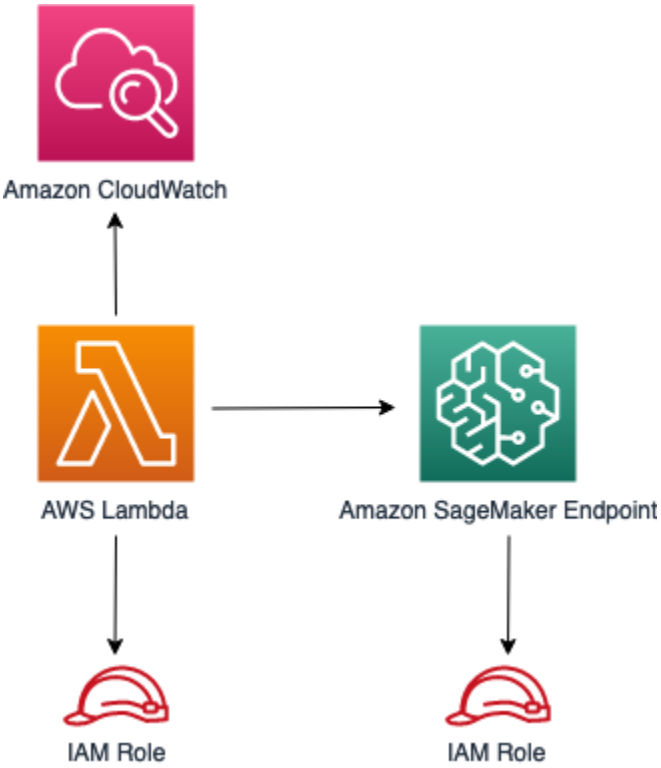
- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 允许函数调用 Sagemaker 端点进行推理。
- 配置函数以访问部署 Sagemaker 终端节点的 VPC 中的资源。
- 启用 X-Ray 跟踪。
- 设置环境变量：

- SAGEMAKER_ENDPOINT_NAME (默认值)
- AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Amazon SageMaker 终端节点

- 配置有限权限以创建 Sagemaker 资源。
- 部署 Sagemaker 模型、端点配置和端点。
- 配置要在 VPC 中部署的 Sagemaker 终端节点。
- 部署 S3 VPC 终端节点和 Sagemaker 运行时 VPC 接口。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-结构/aws-拉姆达-圣马可点

aws-lambda-秘密管理器

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_lambda_secretsmanager`
TypeScript	`@aws-solutions-constructs/aws-lambda-secretsmanager`
Java	`software.amazon.awsconstructs.services.lambdasecretsmanager`

Overview

此 AWS 解决方案构造实现了具有最低特权权限的 AWS Lambda 函数和 AWS Secrets Manager 密钥。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToSecretsmanagerProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的道具来覆盖 Lambda 函数的默认道具。
秘密道具？	secretsmanager.SecretProps	可选的用户提供的道具来覆盖密码管理器的默认道具。
是否存在秘密烟草？	secretsmanager.Secret	Secrets Manager 秘密对象的现有实例，如果设置了此项，

名称	类型	描述
		则secretProps 忽略此限制。
授权访问？	boolean	Lambda 函数对密钥的可选写入访问权限（默认情况下为只读）。
秘密变量名称？	string	为 Lambda 函数设置的密钥管理器密钥环境变量的可选名称。
是否存在 VPC？	ec2.IVpc	应在其中部署此模式的可选现有 VPC。在 VPC 中部署时，Lambda 函数将使用 VPC 中的 ENI 访问网络资源，并且将在适用于 AWS Secrets Manager 的 VPC 中创建接口终端节点。如果提供了现有 VPC，则deployVpc 属性不能true。这使用ec2.IVpc，以允许客户端使用ec2.Vpc.fromLookup() 方法。
VPCPROP？	ec2.VpcProps	用户提供的可选属性，用于覆盖新 VPC 的默认属性。enableDnsHostnames 、enableDnsSupport 、natGateways ，和subnetConfiguration 是由模式设置的，所以这里提供的这些属性的任何值都将被覆盖。如果deployVpc 不是true那么这个属性将被忽略。

名称	类型	描述
部署 VPC？	boolean	是否创 VPC 基于vpcProps将此模式部署到其中。将此设置为true将部署最小的、最私有的 VPC 来运行模式： • CDK 计划使用的每个可用区内都有一个隔离子网 • enableDnsHostnames 和enableDns Support 都将被设置为true 如果此属性为true，然后existingVpc 则无法指定 。默认值为 false。

模式属性

名称	类型	描述
LambdaFunction	lambda.Function	返回lambda.Function 由构造创建。
密钥	secretsmanager.Secret	返回secretsmanager.Secret 由构造创建。
VPC？	ec2.IVpc	返回模式使用的 VPC 上的接口（如果有）。这可能是由模式创建的 VPC，也可以是提供给模式构造函数的 VPC。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

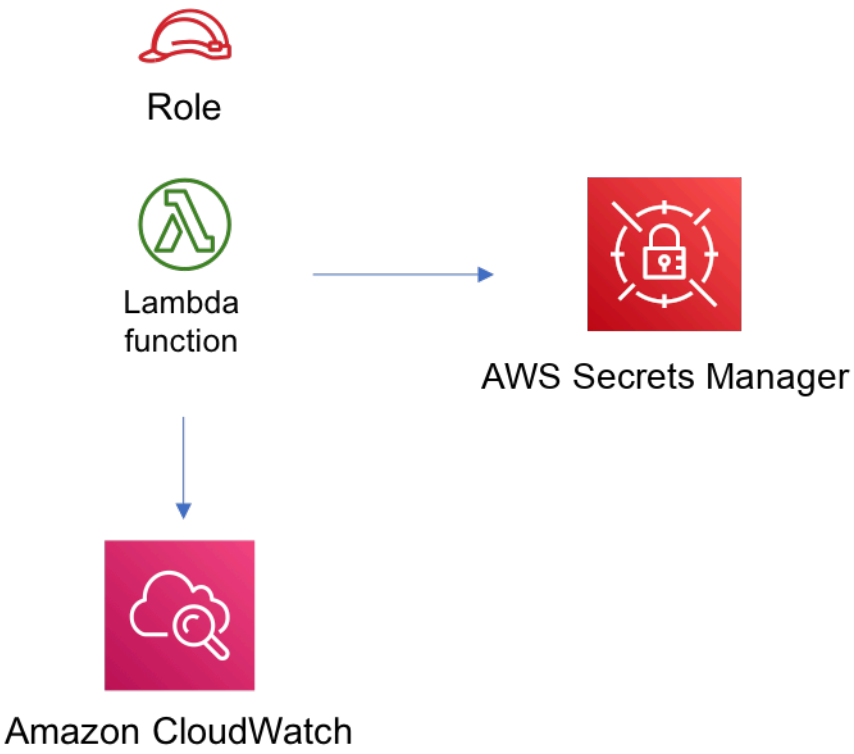
AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - （默认值）SEECT_ARN，其中包含由 CDK 返回的密钥的 ARNSecreate属性
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

Amazon Secrets Manager 密钥

- 启用关联的 AWS Lambda 函数的只读访问
- 使用帐户和区域的默认 KMS 密钥启用服务器端加密
- 创建新密钥：
  - （默认）随机名称
  - （默认值）随机值
- 删除 CloudFormation 堆栈时保留秘密

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-lambda-秘密管理器

aws-lambda-sns

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_lambda_sns`
TypeScript	`@aws-solutions-constructs/aws-lambda-sns`
Java	`software.amazon.awsconstructs.services.lambdasns`

Overview

此 AWS 解决方案构造实现了与 Amazon SNS 主题相连的 AWS Lambda 函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToSnsProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略existingLambdaObj 提供。
现有的托比克吗？	sns.Topic	SNS 主题对象的现有实例，同时提供此和topicProps 会导致错误。
主题道具？	sns.TopicProps	用户提供的可选属性用于覆盖 SNS 主题的默认属性。
是否存在 VPC？	ec2.IVpc	应在其中部署此模式的可选现有 VPC。在 VPC 中部署时，Lambda 函数将使用 VPC 中的 ENI 访问网络资源，并在 Amazon SQS 的 VPC 中创建接口终端节点。如果提供了现有 VPC，则deployVpc 属性不能true。此操作使用ec2.IVpc允许客户端使用ec2.Vpc.fromLookup() 方法。

名称	类型	描述
部署 VPC？	boolean	是否创 VPC 基于vpcProps将此模式部署到其中。将此设置为true将部署最小的、最私有的 VPC 来运行模式： • CDK 计划使用的每个可用区内有一个隔离的子网。 • enableDnsHostnames 和enableDns Support 都将被设置为true。 如果此属性为true，然后existingVpc 则无法指定 。默认值为 false。
VPCPROP？	ec2.VpcProps	用户提供的可选属性，用于覆盖新 VPC 的默认属性。enableDns Hostnames 、enableDns Support 、natGatewa ys 和subnetCon figuration 是由模式设置的，所以这里提供的这些属性的任何值都将被覆盖。如果deployVpc 不是true那么这个属性将被忽略。
主题环境变量名称？	string	为 Lambda 函数设置的 SNS 主题 ARN 环境变量的可选名称。
主题名称环境变量名称？	string	为 Lambda 函数设置的 SNS 主题名称环境变量的可选名称。

模式属性

名称	类型	描述
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
snsTopic	sns.Topic	返回由模式创建的 SNS 主题的实例。
VPC？	ec2.IVpc	返回模式使用的 VPC 实例（如果有）。这可能是由模式创建的 VPC，也可以是提供给模式构造函数的 VPC。

默认设置

没有任何覆盖的构造的开箱即用实现将设置以下默认值：

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - SNS_TOPIC_NAME (默认值)
  - SNS_TOPIC_ARN (默认值)
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Amazon SNS 主题

- 为 SNS 主题配置最小权限访问权限。
- 使用 AWS 托管 KMS 密钥启用服务器端加密。
- 强制对传输中数据加密。

Architecture

GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案-结构/aw-拉姆达-sns

aws-lambda 平方米

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_lambda_sqs`

语言	程序包
TypeScript	`@aws-solutions-constructs/aws-lambda-sqs`
Java	`software.amazon.awsconstructs.services.lambdasqs`

Overview

此 AWS 解决方案构造实现了连接到 Amazon SQS 队列的 AWS Lambda 函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scope`Construct`

- idstring
- propsLambdaToSqsProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	要使用的可选现有 Lambda 函数，而不是默认函数。同时提供此和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。
现有队列 OBJ？	sqs.Queue	要使用的可选现有 SQS 队列，而不是默认队列。同时提供此和queueProps 会导致错误。
队列道具？	sqs.QueueProps	用户提供的可选属性，用于覆盖 SQS 队列的默认属性。
是否启用队列清除？	boolean	是否向 Lambda 函数授予其他权限，使其能够清除 SQS 队列。默认值为 false。
部署死信件队列？	boolean	是否创建要用作死信队列的辅助队列。默认值为 true。
死书队列道具？	sqs.QueueProps	用户提供的可选道具，用于覆盖死信队列的默认道具。仅当deployDeadLetterQueue 属性设置为 true。
maxReceiveCount？	number	消息在发送到死信队列之前移动到消息失败的次数。默认值为 15。

名称	类型	描述
是否存在 VPC？	ec2.IVpc	应在其中部署此模式的可选现有 VPC。当部署到 VPC 中时，Lambda 函数将使用 VPC 中的 ENI 访问网络资源，并在 Amazon SQS 的 VPC 中创建接口终端节点。如果提供了现有 VPC，deployVpc 属性不能为true。一个ec2.IVpc用于允许客户端使用ec2.Vpc.fromLookup() 方法。
部署 VPC？	boolean	是否创 VPC 基于vpcProps将此模式部署到其中。将此设置为true将部署最小的、最私有的 VPC 来运行模式： • CDK 计划使用的每个可用区中有一个隔离子网 • enableDnsHostnames 和enableDnsSupport 都将被设置为true 如果此属性为true，然后existingVpc 则无法指定 。默认值为 false。

名称	类型	描述
VPCPROP？	ec2.VpcProps	用户提供的可选属性，用于覆盖新 VPC 的默认属性。enableDnsHostnames 、enableDnsSupport 、natGateways ，和subnetConfiguration 由模式设置，因此此处提供的这些属性的任何值都将被覆盖。如果deployVpc 不是true，则此属性将被忽略。
队列环境变量名称？	string	为 Lambda 函数设置的 SQS 队列 URL 环境变量的可选名称。

模式属性

名称	类型	描述
死信队列？	sqs.Queue	返回由模式创建的死信队列的实例（如果已部署）。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
SQUUE	sqs.Queue	返回由模式创建的 SQS 队列的实例。
VPC？	ec2.IVpc	返回模式创建或使用的 VPC 的实例（如果有）。这可能是由模式创建的 VPC，也可以是提供给模式构造函数的 VPC。

默认设置

没有任何覆盖的构造的开箱即用实现将设置以下默认值：

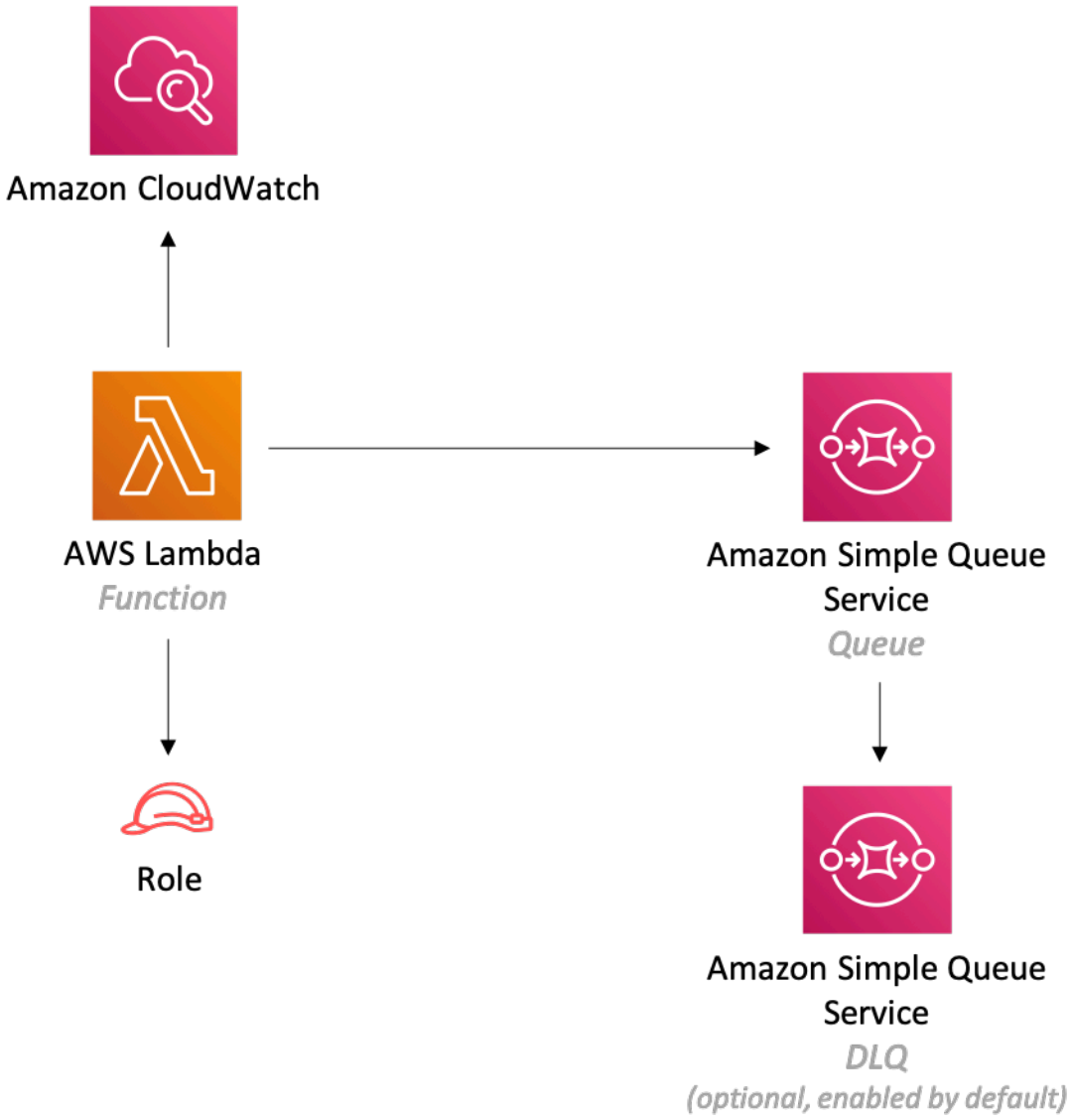
AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 允许函数仅向队列发送消息（可以使用`enableQueuePurge`属性）。
- 启用 X-Ray 跟踪
- 设置环境变量：
  - `SQS_QUEUE_URL`
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

Amazon SQS 队列

- 为源 SQS 队列部署 SQS 死信队列。
- 使用 AWS 托管 KMS 密钥为源 SQS 队列启用服务器端加密。
- 实施传输中数据加密。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-lambda 平方米

aws-lambda 平方米-lambda

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
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 解决方案构造模式实现 (1) 配置为向队列发送消息的 AWS Lambda 函数；(2) Amazon SQS 队列；(3) 配置为使用队列中的消息的 AWS Lambda 函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToSqsToLambdaProps

模式构建道具

名称	类型	描述
现有的制作人兰姆道夫？	lambda.Function	一个可选的现有 Lambda 函数，而不是用于向队列发送消息的默认函数。同时提供此和producerLambdaFunctionProps 会导致错误。
生产者兰姆达功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖生成器 Lambda 函数的默认属性。
现有队列 OBJ？	sqs.Queue	要使用的可选现有 SQS 队列，而不是默认队列。同时提供此和queueProps 会导致错误。

名称	类型	描述
队列道具？	sqs.QueueProps	用户提供的可选属性，用于覆盖 SQS 队列的默认属性。同时提供此和existingQueueObj 会导致错误。
部署死信件队列？	boolean	是否创建要用作死信队列的辅助队列。默认值为 true。
死书队列道具？	sqs.QueueProps	用户提供的可选道具，用于覆盖死信队列的默认道具。仅当deployDeadLetterQueue 属性将设定为true。
maxReceiveCount？	number	消息在发送到死信队列之前移动到源队列的次数。默认值为 15。
现有的消费者兰姆道夫 J？	lambda.Function	用于接收/使用队列消息的可选现有 Lambda 函数，而不是默认函数。同时提供此和consumerLambdaFunctionProps 会导致错误。
消费者拉姆达功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖使用者 Lambda 函数的默认属性。
队列环境变量名称？	string	为生成器 Lambda 函数设置的 SQS 队列 URL 环境变量的可选名称。

模式属性

名称	类型	描述
消费者 Lambda 函数	lambda.Function	返回由模式创建的消费者 Lambda 函数的实例。
死信队列？	sqs.Queue	返回由模式创建的死信队列的实例（如果已部署）。
生产者 Lambda 函数	lambda.Function	返回由模式创建的生成器 Lambda 函数的实例。
SQUUE	sqs.Queue	返回由模式创建的 SQS 队列的实例。

默认设置

此构造的开箱即用实现（没有任何覆盖的属性）将遵循以下默认值：

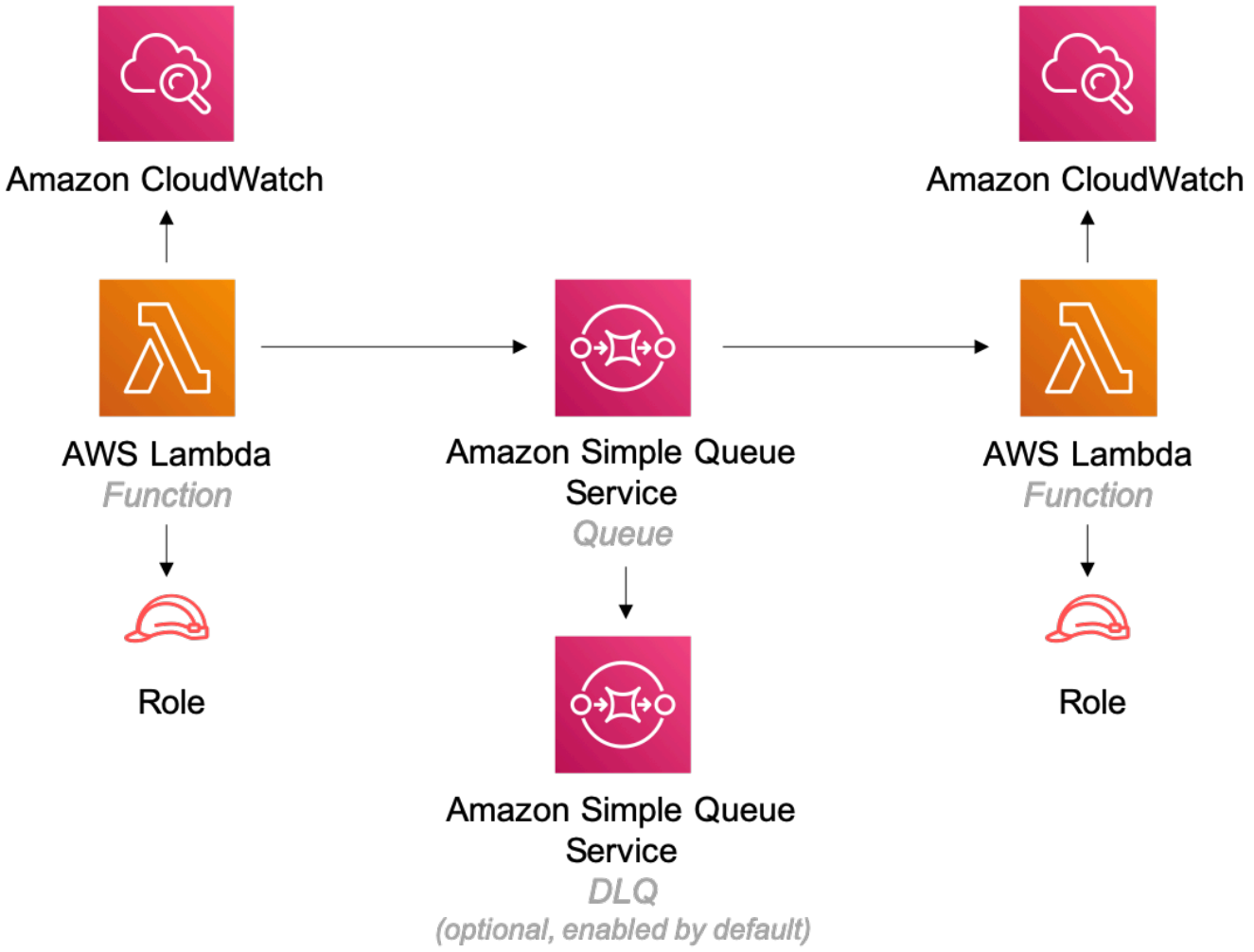
AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 为节点 JS Lambda 函数启用使用保持活动状态的连接重复使用。
- 启用 X-Ray 跟踪
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Amazon SQS 队列

- 部署用于主队列的死信队列。
- 使用 AWS 托管 KMS 密钥为主队列启用服务器端加密。
- 实施传输中数据加密

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-结构/aw-拉姆达-平方-lambda

aws-lambda 步进函数

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_lambda_step_function`
TypeScript	`@aws-solutions-constructs/aws-lambda-step-function`
Java	`software.amazon.awsconstructs.services.lambdastepfunction`

Overview

此 AWS 解决方案构造实现了连接到 AWS 步骤函数的 AWS Lambda 函数。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsLambdaToStepFunctionProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略existingLambdaObj 提供。
国家机械道具	sfn.StateMachineProps	用户为 SFN.Statemachine 提供的道具。
创造云监视图	boolean	是否创建推荐的 CloudWatch 警报。

名称	类型	描述
日志组道具？	logs.LogGroupProps	用户提供的可选道具覆盖 CloudWatch Logs 日志组的默认道具。
状态环境变量名称	string	为生成器 Lambda 函数设置的 Step Functions 状态计算机环境变量的可选名称。

模式属性

名称	类型	描述
CloudwatchAltms	cloudwatch.Alarm[]	返回模式创建的一个或多个 CloudWatch 警报的列表。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
StatadeMachine	sfn.StateMachine	返回由模式创建的状态机的实例。
国家机械学组	logs.LogGroup	返回由状态机模式创建的日志组的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

AWS Lambda 函数

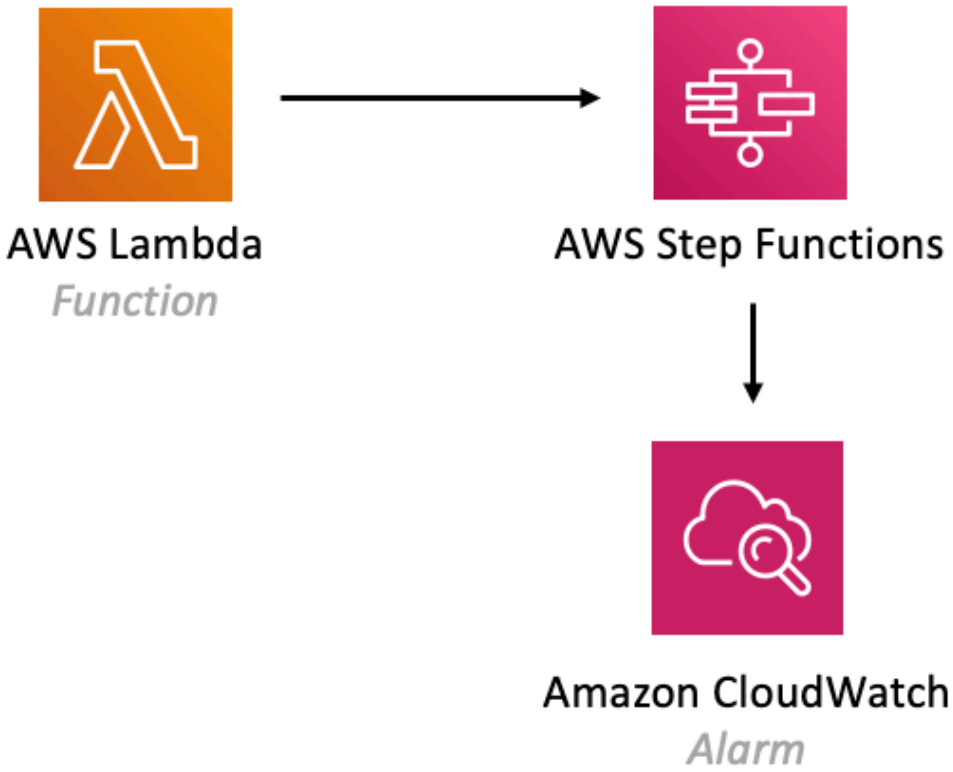
- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 为节点 JS Lambda 函数启用使用保持活动状态的连接重复使用。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - STATE_MACHINE_ARN (默认值)

- AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

AWS Step Functions 状态机

- 为 AWS Step Functions 状态机部署最佳实践 CloudWatch 警报。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-lambda-步进函数

aws-s3 lambda

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造实现了连接到 AWS Lambda 函数的 Amazon S3 存储桶。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsS3ToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会引发错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
现有的存储桶吗？	s3.Bucket	S3 存储桶对象的现有实例。如果提供了这一点，那么还提供bucketProps 是错误。
桶道具？	s3.BucketProps	用户提供的可选属性，用于覆盖存储桶的默认属性。忽略，如果existingBucketObj 提供。

名称	类型	描述
S3 事件源道具？	S3EventSourceProps	可选的用户提供的道具来覆盖 S3 事件源道具的默认道具

模式属性

名称	类型	描述
LambdaFunction	lambda.Function	返回模式创建的 Lambda 函数的实例。
S3Bucket？	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon S3 存储桶

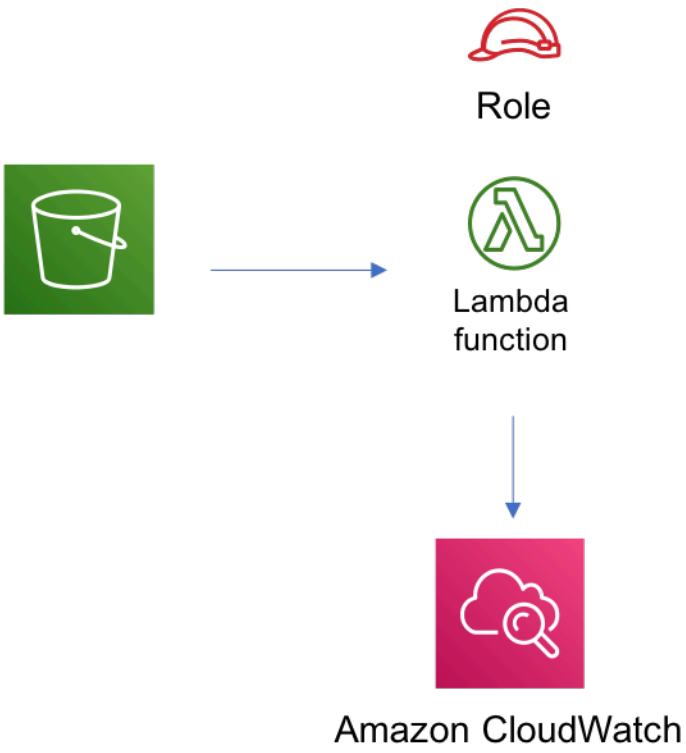
- 配置 S3 存储桶的访问日志记录。
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密。
- 启用 S3 存储桶的版本控制。
- 不允许 S3 存储桶的公共访问。
- 删除 CloudFormation 堆栈时保留 S3 存储桶。
- 强制传输中数据加密。
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储。

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数启用重复使用连接。

- 启用 X-Ray 跟踪。
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案-结构/aws-s3-lambda

aws-s3 平方米

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造实现了配置为向 Amazon SQS 队列发送通知的 Amazon S3 存储桶。

以下是 TypeScript 中的最小可部署模式定义：

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

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

Initializer

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

参数

- scopeConstruct
- idstring
- propsS3ToSqsProps

模式构建道具

名称	类型	描述
现有的存储桶吗？	s3.Bucket	S3 存储桶对象的现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。
桶道具？	s3.BucketProps	用户提供的可选道具，用于覆盖 S3 存储桶的默认道具。
S3 事件类型？	s3.EventType[]	将触发通知的 S3 事件类型。默认值为 s3.EventType.OBJECT_CREATED 。
S3 事件过滤器？	s3.NotificationKeyFilter[]	S3 对象键筛选规则用于确定哪些对象触发此事件。如果未指定，则不会应用过滤规则。
现有队列 OBJ？	sqs.Queue	要使用的可选现有 SQS 队列，而不是默认队列。同时提供此和queueProps 会导致错误。如果 SQS 队列已加密，则用于加密的 KMS 密钥必须是客户管理的 CMK。
队列道具？	sqs.QueueProps	用户提供的可选属性，用于覆盖 SQS 队列的默认属性。忽略，如果existingQueueObj 提供。

名称	类型	描述
死书队列道具？	sqs.QueueProps	用户提供的可选道具，用于覆盖死信队列的默认道具。仅当deployDeadLetterQueue 属性设置为 true。
部署死信件队列？	boolean	是否创建要用作死信队列的辅助队列。默认值为 true。
maxReceiveCount？	number	消息在发送到死信队列之前移动到队列的次数。默认值为 15。
是否启用用客户管理的密钥加密？	boolean	是否使用 KMS 密钥（由此 CDK 应用程序管理或导入）。如果导入加密密钥，则必须在encryptionKey 属性。
encryptionKey	kms.Key	要使用的可选现有加密密钥，而不是默认加密密钥。
加密钥匙道具？	kms.KeyProps	用户提供的可选属性，用于覆盖加密密钥的默认属性。

模式属性

名称	类型	描述
SQUUE	sqs.Queue	返回由模式创建的 SQS 队列的实例。
死信队列？	sqs.Queue	返回由模式创建的死信队列的实例（如果已部署）。
encryptionKey	kms.IKey	返回由模式创建的加密密钥的实例。

名称	类型	描述
S3Bucket？	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

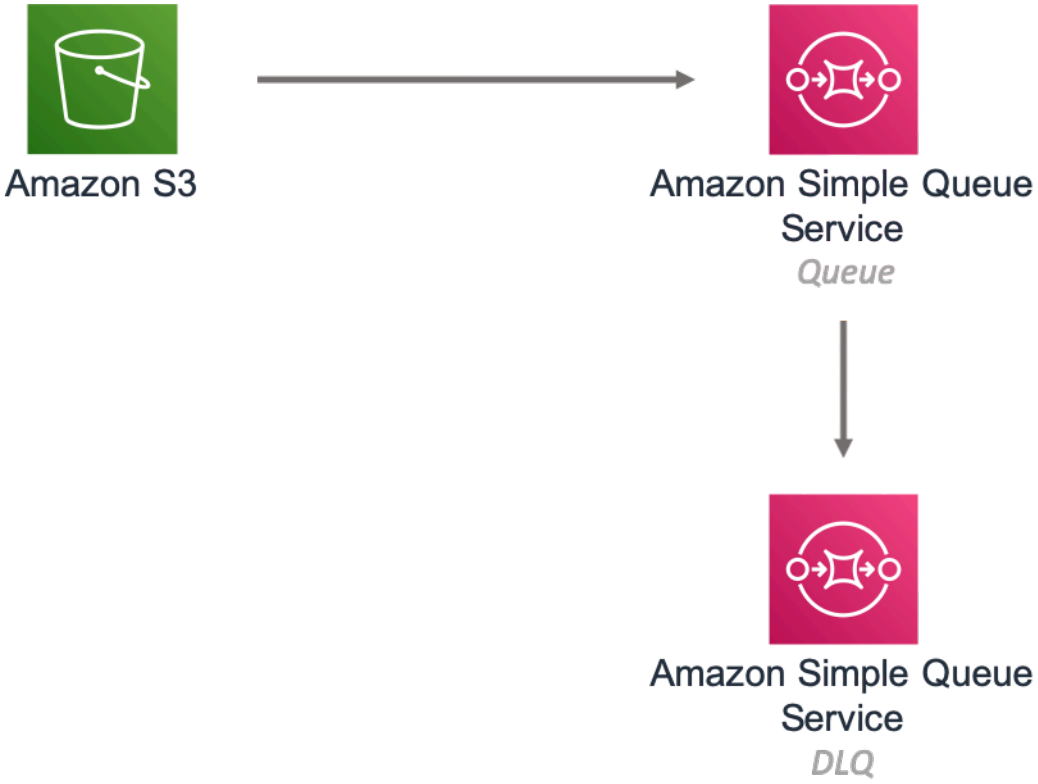
Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密
- 启用 S3 存储桶的版本控制
- 不允许 S3 存储桶的公共访问
- 删除 CloudFormation 堆栈时保留 S3 存储桶
- 实施传输中数据加密
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储

Amazon SQS 队列

- 为 SQS 队列配置最小权限访问权限
- 为源 SQS 队列部署 SQS 死信队列
- 使用客户管理的 KMS 密钥为 SQS 队列启用服务器端加密
- 实施传输中数据加密

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-s3 平方米

aws-s3 步骤函数

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
Python	`aws_solutions_constructs.aws_s3_step_function`
TypeScript	`@aws-solutions-constructs/aws-s3-step-function`
Java	`software.amazon.awsconstructs.services.s3stepfunction`

Overview

此 AWS 解决方案构造实现了连接到 AWS 步骤函数的 Amazon S3 存储桶。

> **Note**
>
> 此构造使用亚马逊 EventBridge（Amazon CloudWatch Events）来触发 AWS Step Functions。EventBridge 更加灵活，但使用 S3 事件通知触发 Step Functions 的延迟更少，而且更具成本效益。如果成本和/或延迟是一个问题，则应考虑部署`aws-s3-lambda`和`aws-lambda-stepfunctions`代替这个结构。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsS3ToStepFunctionProps

模式构建道具

名称	类型	描述
现有的存储桶吗？	s3.IBucket	S3 存储桶对象的现有实例。如果提供了这一点，那么还提供bucketProps 是一个错误。
桶道具？	s3.BucketProps	用户提供的可选属性，用于覆盖存储桶的默认属性。忽略的是existingBucketObj 提供。
国家道具	sfn.StateMachineProps	可选用户提供的道具来覆盖 SFN.Statemachine 的默认道具。
事件道具？	events.RuleProps	可选用户提供的事件 Props 来覆盖默认值。

名称	类型	描述
部署云跟踪？	boolean	是否在 AWS CloudTrail 中部署跟踪以在 Amazon S3 中记录 API 事件。默认值为 true。
创造云监视图	boolean	是否创建推荐的 CloudWatch 警报。
日志组道具？	logs.LogGroupProps	用户提供的可选道具，用于覆盖 CloudWatch Logs 日志组的默认道具。

模式属性

名称	类型	描述
CloudTrail？	cloudtrail.Trail	返回由模式创建的 Cloudrail 跟踪的实例。
云行道桶？	s3.Bucket	返回由模式创建的用于存储 Cloudrail 跟踪数据的存储桶实例。
云跟踪记录存储桶？	s3.Bucket	返回由 Cloudrail 跟踪使用的主存储桶模式创建的日志记录桶实例。
CloudwatchAltms	cloudwatch.Alarm[]	返回模式创建的一个或多个 CloudWatch 警报的列表。
S3Bucket	s3.Bucket	返回模式创建的 S3 存储桶的实例。
S3 记录桶？	s3.Bucket	返回由模式为 S3 存储桶创建的日志记录存储桶的实例。

名称	类型	描述
StadeMachine	sfn.StateMachine	返回由模式创建的状态机的实例。
国家机械学组	logs.LogGroup	返回由状态机模式创建的日志组的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

Amazon S3 存储桶

- 配置 S3 存储桶的访问日志记录。
- 使用 AWS 托管 KMS 密钥为 S3 存储桶启用服务器端加密。
- 启用 S3 存储桶的版本控制。
- 不允许 S3 存储桶的公共访问。
- 删除 CloudFormation 堆栈时保留 S3 存储桶。
- 实施传输中数据加密。
- 应用生命周期规则在 90 天后将非当前对象版本移动到 Glacier 存储。

AWS CloudTrail

- 在 AWS CloudTrail 中配置跟踪，以在 Amazon S3 中记录与构造创建的存储桶相关的 API 事件。

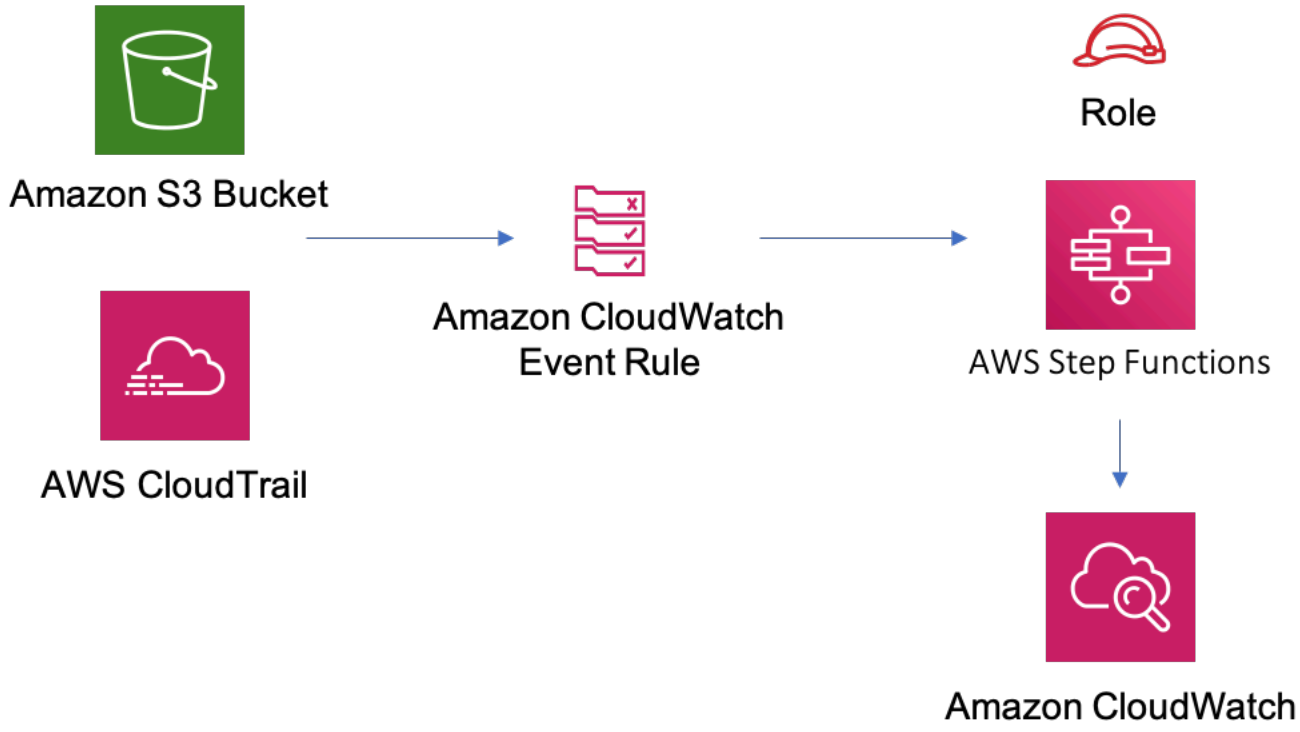
Amazon CloudWatch Events

- 向 CloudWatch 事件授予最低权限以触发 Lambda 函数。

AWS Step Function

- 为 API Gateway 启用 CloudWatch 日志记录。
- 为步骤功能部署最佳实践 CloudWatch 警报。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-S3 步骤函数

-SNS-拉姆达

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造实现了连接到 AWS Lambda 函数的 Amazon SNS。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsSnsToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略existingLambdaObj 提供。
现有的托比克吗？	sns.Topic	SNS 主题对象的现有实例，同时提供此和topicProps 会导致错误。
主题道具？	sns.TopicProps	用户提供的可选属性用于覆盖 SNS 主题的默认属性。

模式属性

名称	类型	描述
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
snsTopic	sns.Topic	返回由模式创建的 SNS 主题的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

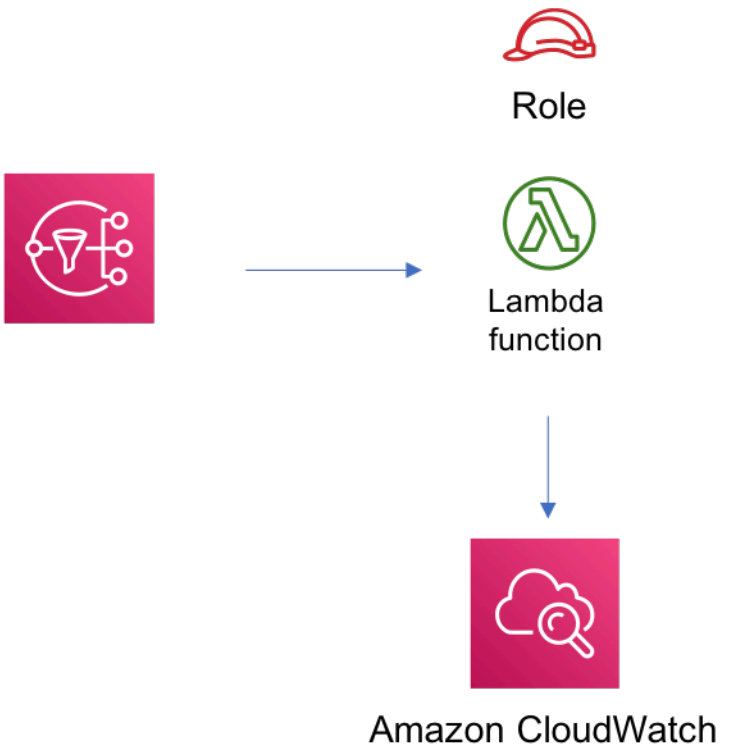
Amazon SNS 主题

- 为 SNS 主题配置最小权限访问权限。
- 使用 AWS 托管 KMS 密钥启用服务器端加密。
- 实施传输中数据加密。

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - `AWS_NODEJS_CONNECTION_REUSE_ENABLED`（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：	
	@aws-解决方案-结构/AWS-SNS-lambda

AWS-SNS 平方米

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

语言	程序包
 Python	`aws_solutions_constructs.aws_sns_sqs`
 TypeScript	`@aws-solutions-constructs/aws-sns-sqs`
 Java	`software.amazon.awsconstructs.services.snssqs`

Overview

此 AWS 解决方案构造实施 Amazon SNS 主题，与 Amazon SQS 队列相连。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsSnsToSqsProps

模式构建道具

名称	类型	描述
现有的托比克吗？	sns.Topic	SNS 主题对象的现有实例，同时提供此和topicProps 会导致错误。
主题道具？	sns.TopicProps	用户提供的可选属性，用于覆盖 SNS 主题的默认属

名称	类型	描述
		性。忽略，如果existingTopicObj 提供。
现有队列 OBJ？	sqs.Queue	要使用的可选现有 SQS 队列，而不是默认队列。同时提供此和queueProps 会导致错误。
队列道具？	sqs.QueueProps	用户提供的可选属性，用于覆盖 SQS 队列的默认属性。忽略，如果existingQueueObj 提供。
部署死信件队列？	boolean	是否创建要用作死信队列的辅助队列。默认值为 true。
死书队列道具？	sqs.QueueProps	用户提供的可选道具，用于覆盖死信队列的默认道具。仅当deployDeadLetterQueue 属性设置为 true。
maxReceiveCount？	number	消息在发送到死信队列之前移动到达队列的次数。默认值为 15。
是否启用用客户管理的密钥加密？	boolean	是否使用由此 CDK 应用程序管理或导入的客户管理的加密密钥。如果导入加密密钥，则必须在encryptionKey 属性。
encryptionKey	kms.Key	要使用的可选现有加密密钥，而不是默认加密密钥。
加密钥匙道具？	kms.KeyProps	用户提供的可选属性，用于覆盖加密密钥的默认属性。

模式属性

名称	类型	描述
snsTopic	sns.Topic	返回由模式创建的 SNS 主题的实例。
encryptionKey	kms.Key	返回由模式创建的加密密钥的实例。
SQUUE	sqs.Queue	返回由模式创建的 SQS 队列的实例。
死信队列？	sqs.Queue	返回由模式创建的死信队列的实例（如果已部署）。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

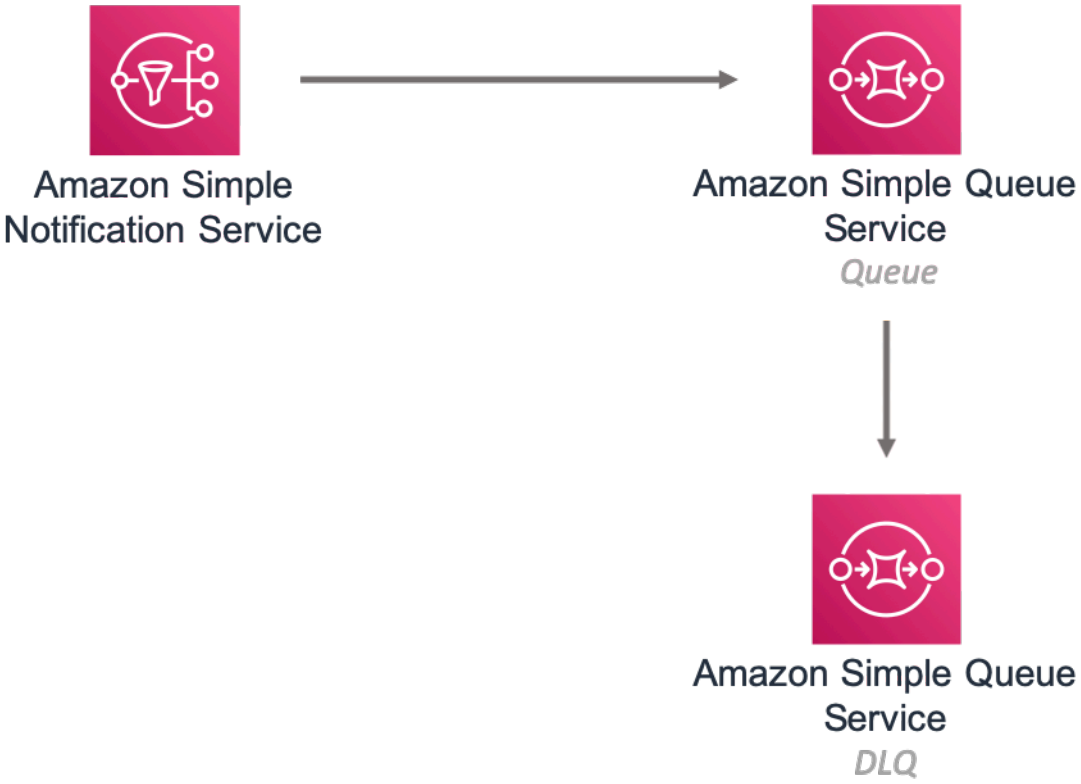
Amazon SNS 主题

- 为 SNS 主题配置最小权限访问权限。
- 使用 AWS 托管 KMS 密钥启用服务器端加密。
- 实施传输中数据加密。

Amazon SQS 队列

- 为 SQS 队列配置最低权限访问权限。
- 部署源 SQS 队列的死信队列。
- 使用客户管理的 KMS 密钥为 SQS 队列启用服务器端加密。
- 实施传输中数据加密。

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-sns-平方米

aws-平方米-lambda

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

注意： 为确保正确的功能，您项目中的 AWS 解决方案构造软件包和 AWS CDK 包必须是相同的版本。

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

Overview

此 AWS 解决方案构造实现了连接到 AWS Lambda 函数的 Amazon SQS 队列。

以下是 TypeScript 中的最小可部署模式定义：

```
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);
```

参数

- scopeConstruct
- idstring
- propsSqsToLambdaProps

模式构建道具

名称	类型	描述
现在的兰姆道夫？	lambda.Function	Lambda 函数对象的现有实例，提供了这个和lambdaFunctionProps 会导致错误。
Lambda 功能道具？	lambda.FunctionProps	用户提供的可选属性，用于覆盖 Lambda 函数的默认属性。忽略，如果existingLambdaObj 提供。
现有队列 OBJ？	sqs.Queue	要使用的可选现有 SQS 队列，而不是默认队列。同时提供此和queueProps 会导致错误。
队列道具？	sqs.QueueProps	用户提供的可选属性，用于覆盖 SQS 队列的默认属性。忽略，如果existingQueueObj 提供。
部署死信件队列？	boolean	是否创建要用作死信队列的辅助队列。默认值为 true。
死书队列道具？	sqs.QueueProps	用户提供的可选道具，用于覆盖死信队列的默认道具。仅当deployDeadLetterQueue 属性设置为 true。
maxReceiveCount？	number	消息在发送到死信队列之前移动到消息失败的次数。默认值为 15。

模式属性

名称	类型	描述
死信队列？	sqs.Queue	返回由模式创建的死信队列的实例（如果已部署）。
LambdaFunction	lambda.Function	返回由模式创建的 Lambda 函数的实例。
SQUUE	sqs.Queue	返回由模式创建的 SQS 队列的实例。

默认设置

没有任何覆盖的此模式的开箱即用实现将设置以下默认值：

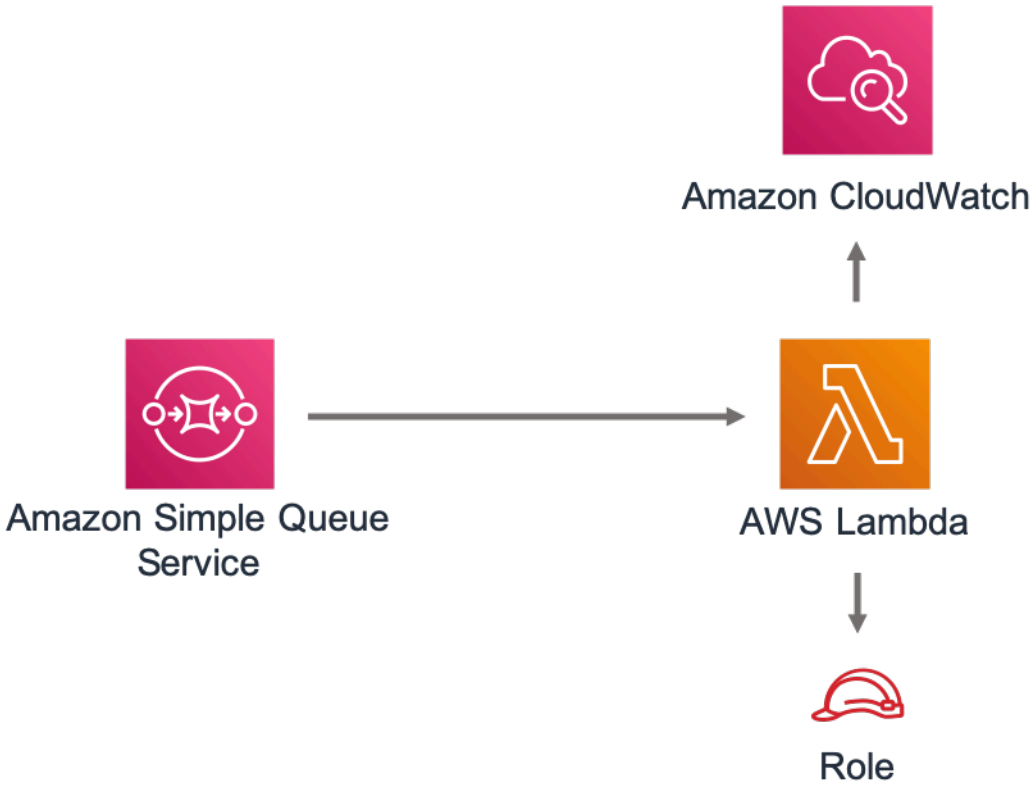
Amazon SQS 队列

- 为源 SQS 队列部署 SQS 死信队列。
- 使用 AWS 托管 KMS 密钥为源 SQS 队列启用服务器端加密。
- 实施传输中数据加密。

AWS Lambda 函数

- 为 Lambda 函数配置有限权限访问 IAM 角色。
- 使用节点 JS Lambda 函数保持活动状态，启用重复使用连接。
- 启用 X-Ray 跟踪。
- 设置环境变量：
  - AWS_NODEJS_CONNECTION_REUSE_ENABLED（用于节点 10.x 和更高版本的功能）

Architecture



GitHub

要查看此模式的代码、创建/查看问题和拉取请求等，请执行以下操作：

@aws-解决方案结构/aws-sq-lambda

core

STABILITY EXPERIMENTAL

所有类都处于积极开发之中，并且在任何未来版本中都会受到非向后兼容的更改或删除。这些不受语义版本控制模型。这意味着，虽然您可以使用它们，但在升级到此软件包的较新版本时，您可能需要更新源代码。

核心库包含 AWS 解决方案构建块的基本构建块。它定义了在其余 AWS 解决方案构造中使用的核心类。

AWS CDK 结构的默认属性

核心库设置 AWS 解决方案构造所使用的 AWS CDK 结构的默认属性。

例如，以下是由 AWS 解决方案构造构造创建的 S3 存储桶构造的默认属性片段。默认情况下，它将打开服务器端加密、存储桶版本控制、阻止所有公共访问以及设置 S3 访问日志记录。

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

覆盖默认属性

Core 库设置的默认属性可以被用户提供的属性覆盖。例如，用户可以覆盖 Amazon S3 阻止公共访问属性以满足特定要求。

```
  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
    },
  });
```

属性覆盖

当 Core 库中的默认属性被用户提供的属性覆盖时，Contuts 将向控制台发出一条或多条警告消息，突出显示更改。这些消息旨在向用户提供情况感知，并防止可能造成安全风险的意外覆盖。每当执行与部署/构建相关的命令时，这些消息都会出现，包括`cdk deploy`、`cdk synth`、`npm test`，等

示例消息：`AWS_CONSTRUCTS_WARNING: An override has been provided for the property: BillingMode. Default value: 'PAY_PER_REQUEST'. You provided: 'PROVISIONED'.`

切换覆盖警告

默认情况下，覆盖警告消息处于启用状态，但可以使用`overrideWarningsEnabled`shell 变量。

- 显式显式提供关闭覆盖警告，运行`export overrideWarningsEnabled=false`。
- 显式显式提供启用覆盖警告，运行`export overrideWarningsEnabled=true`。
- 恢复为默认值，请运行`unset overrideWarningsEnabled`。

文档修订

要获得有关 AWS 解决方案构造更新的通知，请订阅 RSS 源。

更新-历史记录-更改	更新-历史记录-描述	更新-历史记录-日期
更新的内容	添加了 AWS-lambda-管理参数模式。其他次要内容更新。	2020 年 5 月 27 日
更新的内容	添加了 AWS-lambda-密钥管理器模式。其他次要内容更新。	2020 年 5 月 12 日
更新的内容	属性更新以选择 *-lambda 模式。其他次要内容更新。	2019 年 4 月 17 日
更新的内容	修复了 Python 用户演练中的一个问题，并更新了包含 Lambda 函数的构造的属性示例。	2019 年 3 月 30 日
更新的内容	对图案道具的小修复/更新以及选择图案的默认设置。	2021 年 3 月 8 日
更新的内容	演练内容的次要修复/更新。	2019 年 3 月 4 日
更新的内容	添加了`aws-lambda-sagemakerendpoint` 模式和选择 Kinesis 火管模式的更新属性。	2021 年 2 月 24 日
更新的内容	添加了`aws-kinesisstreams-gluejob` 模式和更新的演练步骤。	2021 年 2 月 17 日
更新的内容	更新的属性`aws-cloudfront-*` 模式。	2021 年 2 月9 日
更新的内容	为每个模式添加了 GitHub 的链接。	2021 年 2 月 5 日

更新的内容	更新了选择阵列的属性。	2021 年 2 月 1 日
更新的内容	更新了选择模式的属性和默认设置的文档。	2019 年 1 月 4 日
更新的内容	添加了新的模式：AWS 云前媒体存储和 aws-s3 平方米。	2020 年 12 月 20 日
更新的内容	删除了 aws-lambda-魔术制造商模式。	2020 年 11 月 17 日
更新的内容	增加了新的模式：AWS 事件-规则-动态流，AWS 事件-规则-运动-火焰-S3，以及 aws-lambda-战士制造商。	2020 年 10 月 27 日
更新的内容	更新以反映在 aws-事件规则 sns 和 aws-事件规则 sqs 模式中的突破性变化：类和接口名称更改为 pascal 大小写。	2020 年 10 月 22 日
更新的内容	添加了 AWS-支持网络-传输-运动-火焰-S3 模式; 其他小更新现有内容。	2020 年 10 月 20 日
更新的内容	添加了 AWS 网关-物联网模式；对现有内容进行其他次要更新。	2020 年 10 月 7 日
更新的内容	更新了所有模式的最小可部署特征码段和最佳实践默认值。	2020 年 10 月 5 日
更新的内容	更新了 aws-kinesis流-lambda 模式的属性，以反映突发性变化。	2020 年 9 月 14 日
更新的内容	对演练的第二部分进行了小修复。	2020 年 9 月 10 日

更新的内容	添加了 AWS 网关运动流，AWS 事件-规则-sns 和 aws-事件-规则-平方模式。	2020 年 9 月 10 日
更新的内容	添加了 aws-sns-sqs 模式；对所有 SNS 模式进行更新；轻微的排版修正。	2020 年 9 月 2 日
更新的内容	修复了 aws-sqs-lambda 模式的模块名称。	2020 年 8 月 31 日
更新的内容	修正了 AWS-动态-流-lambda-弹性搜索-基巴纳模式的 Python 模块名称。	2020 年 8 月 31 日
更新的内容	更新了 Lambda 模式的默认值；其他次要更新。	2020 年 8 月 27 日
更新的内容	更新了 S3 模式的公共属性；更新了 DynamoDB 模式的默认值。	2020 年 8 月 10 日
更新的内容	更新了多种模式以突出显示传输过程中加密的默认实施。	2020 年 8 月 4 日
更新的内容	添加了 aws-lambda-sqs-lambda 模式；在入门指南中改进了配置说明；更新了所有模式，以通过公共属性使其他资源可用。	2020 年 7 月 27 日
更新的内容	添加了 aws-lambda-sqs 模式; 其他次要更新。	2020 年 7 月 20 日
更新的内容	从相关模式中删除了部署 Lambda 和部署存储桶属性；其他次要更新。	2020 年 7 月 9 日

更新的内容	添加了 aws-lambda 步进函数模式，并纠正了轻微的印刷错误。	2020 年 7 月 7 日
更新的内容	已将添加到现有的表格？ 属性来选择 DynamoDB 模式。	2020 年 6 月 25 日
更新的内容	针对断开链接的几个文本更正和修复。	2020 年 6 月 23 日
首次发布	AWS 解决方案构造公开发布。	2020 年 6 月 22 日

Notices

客户有责任对本文档中的信息进行单独评估。本文档：(a) 仅供参考；(b) 代表当前提供的 AWS 产品和实践，如有更改，恕不另行通知；并且 (c) AWS 及其附属机构、供应商或许可方不做任何承诺或保证。AWS 产品或服务“按原样”提供，不提供任何形式的保证、陈述或条件，无论是明示还是暗示。AWS 对其客户承担的责任和义务受 AWS 协议制约，本文档不是 AWS 与客户直接的协议的一部分，也不构成对该协议的修改。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。