INTERNSHIP

Service Provider Management Service – ALEXA DATA SERVICES

Summary of the Work Done during the Internship period at Amazon, Inc.

An Internship Report

Submitted in the partial fulfilment of the requirements for

the award of the degree of

Bachelor of Technology in

Department of Computer Science and Engineering

By

S Munichithra (R170631)

Under the esteemed guidance of

Mr. Ketan Deshpande

SDM, Amazon

Department of Computer Science and Engineering

Rajiv Gandhi University of knowledge Technologies

Rajiv Gandhi University of Knowledge Technologies

RK Valley, Kadapa (Dist), Andhra Pradesh, 516330

DECLARATION

I am here by declare that the report of the Btech major project work entitle “Sevice Provider Management Service” (Internship) which is being submitted to Rajiv Gandhi University of Knowledge Technologies, Rk valley, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering, is a bonafide report of the work carried out by me. The material contained in this report has not been submitted to any university or instituition for award of any degree.

Sadhu Munichithra-R170631

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

RGUKT

(AP Government Act 18 of 2008)

RGUKT, RK VALLEY

Department of Computer Science and Engineering

Certificate

This is Certified that the project entitled **"Service Provider Management Service (Internship)"** which is internship work carried out by Sadhu Munichithra (R170631) in partial fulfilment for the award of the degree of **Bachelor of Technology** in Department **Computer Science and Engineering**, during the year **2022-23**. The internship has been approved as it satisfies the academic requirements.

Project Internal Guide **Head of the Department**

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success. I am extremely grateful to our respected Diretor, Prof. K. SANDHYA RANI for fostering an excellent academic climate in our institution. I also express my sincere gratitude to our respected Head of the department Mr. SATYANANDARAM For their encouragement. Overall guidance in viewing this internship a good asset and effort in bringing our this project. I would like to convey thanks to our guide at college Ms. SRAVANI for their guidance, encouragement, co-operation and kindness during the entire duation of the course and academics. My sincere thanks to all the members who helped me directly and indirectly in the completion of the project work. I express my profounf gratittude to all our friends, family, Office collegues for their encouragement.

Table of Contents

Introduction

An AWS CDK app (p. 103) is an application written in TypeScript, JavaScript, Python, Java, C# or Go that uses the AWS CDK to define AWS infrastructure. An app defines one or more stacks (p. 108). Stacks (equivalent to AWS CloudFormation stacks) contain constructs (p. 84). Each construct defines one or more concrete AWS resources, such as Amazon S3 buckets, Lambda functions, or Amazon DynamoDB tables. Constructs (and also stacks and apps) are represented as classes (types) in your programming language of choice. You instantiate constructs within a stack to declare them to AWS, and connect them to each other using well-defined interfaces. The AWS CDK includes the CDK Toolkit (also called the CLI), a command line tool for working with your AWS CDK apps and stacks. Among other functions, the Toolkit provides the ability to do the following:

• Convert one or more AWS CDK stacks to AWS CloudFormation templates and related assets (a process called synthesis)

• Deploy your stacks to an AWS account and Region

The AWS CDK includes a library of AWS constructs called the AWS Construct Library, organized into various modules. The library contains constructs for each AWS service. The main CDK package is called aws-cdk-lib, and it contains the majority of the AWS Construct Library. It also contains base classes like Stack and App that are used in most CDK applications. The actual package name of the main CDK package varies by language.

TypeScript

Install npm install aws-cdk-lib
Import import * as cdk from 'aws-cdk-lib';

AWS CloudFormation-only or L1 (short for "layer 1"). These constructs correspond directly to resource types defined by AWS CloudFormation. In fact, these constructs are automatically generated from the AWS CloudFormation specification. Therefore, when a new AWS service is launched, the AWS CDK supports it a short time after AWS CloudFormation does

AWS CloudFormation resources always have names that begin with Cfn. For example, for the Amazon S3 service, CfnBucket is the L1 construct for an Amazon S3 bucket. All L1 resources are in aws-cdk-lib.

• Curated or L2. These constructs are carefully developed by the AWS CDK team to address specific use cases and simplify infrastructure development. For the most part, they encapsulate L1 resources, providing sensible defaults and best practice security policies. For example, Bucket is the L2 construct for an Amazon S3 bucket. Libraries may also define supporting resources needed by the primary L2 resource. Some services have more than one L2 namespace in the Construct Library for organizational purposes. aws-cdk-lib contains L2 constructs that are designated stable, i.e., ready for production use. If a service's L2 support is still under development, its constructs are designated experimental and provided in a separate module.

• Patterns or L3. Patterns declare multiple resources to create entire AWS architectures for particular use cases. All the plumbing is already hooked up, and configuration is boiled down to a few important parameters.

As with L2 constructs, L3 constructs that are ready for production use (stable) are included in awscdk-lib, while those still under development are in separate modules.

Finally, the constructs package contains the Construct base class. It's in its own package because it's used by other construct-based tools in addition to the AWS CDK, including CDK for Terraform and CDK for Kubernetes.

Numerous third parties have also published constructs compatible with the AWS CDK. Visit Construct Hub to explore the AWS CDK construct partner ecosystem.

TypeScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket',
  versioned: true,
  websiteRedirect: {hostName: 'aws.amazon.com'}});
```

JavaScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket',
  versioned: true,
  websiteRedirect: {hostName: 'aws.amazon.com'}});
```

Python

```
bucket = s3.Bucket("MyBucket", bucket_name="my-bucket", versioned=True,
           website_redirect=s3.RedirectTarget(host_name="aws.amazon.com"))
```

Java

```
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .bucketName("my-bucket")
                      .versioned(true)
                      .websiteRedirect(new RedirectTarget.Builder()
                          .hostName("aws.amazon.com").build())
                      .build();
```

C#

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
                      BucketName = "my-bucket",
                      Versioned  = true,
                      WebsiteRedirect = new RedirectTarget {
                              HostName = "aws.amazon.com"
                      }});
```

Go

```
bucket := awss3.NewBucket(scope, jsii.String("MyBucket"), &awss3.BucketProps {
 BucketName: jsii.String("my-bucket"),
 Versioned: jsii.Bool(true),
 WebsiteRedirect: &awss3.RedirectTarget {
  HostName: jsii.String("aws.amazon.com"),
 },
})
```

TypeScript was the first language supported by the AWS CDK, and much AWS CDK example code is written in TypeScript. This guide includes a topic specifically to show how to adapt TypeScript AWS CDK code for use with the other supported languages.

All AWS CDK developers, even those working in Python, Java, or C#, need Node.js 10.13.0 or later. All supported languages use the same backend, which runs on Node.js. We recommend a version in active long-term support. Your organization may have a different recommendation.

Important Node.js versions 13.0.0 through 13.6.0 are not compatible with the AWS CDK due to compatibility issues with its dependencies.

You must configure your workstation with your credentials and an AWS Region, if you have not already done so. If you have the AWS CLI installed, we recommend running the following command:

```
aws configure
```

Provide your AWS access key ID, secret access key, and default Region when prompted.

You can also manually create or edit the `~/.aws/config` and `~/.aws/credentials` (macOS/Linux) or `%USERPROFILE%\.aws\config` and `%USERPROFILE%\.aws\credentials` (Windows) files to contain credentials and a default Region. Use the following format.

- In `~/.aws/config` or `%USERPROFILE%\.aws\config`

```
[default]
region=us-west-2
```

- In `~/.aws/credentials` or `%USERPROFILE%\.aws\credentials`

```
[default]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

Although the AWS CDK uses credentials from the same configuration files as other AWS tools and SDKs, including the AWS CLI, it might behave somewhat differently from these tools. In particular, if you use a named profile from the credentials file, the config must have a profile of the same name specifying the Region. The AWS CDK does not fall back to reading the Region from the [default] section in config. Also, do not use a profile named "default" (e.g. [profile default]). See Setting credentials for complete details on setting up credentials for the AWS SDK for JavaScript, which the AWS CDK uses under the hood. The AWS CDK natively supports AWS IAM Identity Center (successor to AWS Single Sign-On). To use IAM Identity Center with the CDK, first create a profile using aws configure sso. Then log in using aws sso login. Finally, specify this profile when issuing cdk commands using the –profile option or the AWS_PROFILE environment variable. Alternatively, you can set the environment variables AWS_ACCESS_KEY_ID,AWS_SECRET_ACCESS_KEY,and AWS_DEFAULT_REGION to appropriate values.

Important

Using your AWS root account is the fastest way to get started with the AWS CDK. Once you've worked through a few tutorials, however, we strongly recommend against using your root account for day-to-day tasks. Instead, create a user in IAM and use its credentials with the CDK.

Best practices are to change this account's access key regularly and to use a least-privileges role. The AWS CDK includes roles that your account should have permission to assume, for example using the policy here.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/cdk-*"
            ]
        }
    ]
}
```

Install the AWS CDK Toolkit globally using the following Node Package Manager command.

npm install -g aws-cdk

Run the following command to verify correct installation and print the version number of the AWS CDK.

cdk –version

Your first AWS CDK app:

• The structure of an AWS CDK project

• How to use the AWS Construct Library to define AWS resources using code

• How to synthesize, diff, and deploy collections of resources using the AWS CDK Toolkit command line tool

The standard AWS CDK development workflow is similar to what you're already familiar with as a developer, with only a few extra steps.

1. Create the app from a template provided by the AWS CDK.

2. Add code to the app to create resources within stacks.

3. (Optional) Build the app. (The AWS CDK Toolkit does this for you if you forget.)

4. Synthesize one or more stacks in the app to create an AWS CloudFormation template.

5. Deploy one or more stacks to your AWS account.

The build step catches syntax and type errors. The synthesis step catches logical errors in defining your AWS resources. The deployment may find permission issues. As always, you go back to the code, find the problem, fix it, then build, synthesize, and deploy again.

Create the app

Each AWS CDK app should be in its own directory, with its own local module dependencies. Create a new directory for your app. Starting in your home directory, or another directory if you prefer, issue the following commands.

Important

Be sure to name your project directory hello-cdk, exactly as shown here. The AWS CDK project template uses the directory name to name things in the generated code.

```
mkdir hello-cdk
cd hello-cdk
```

Now initialize the app by using the **cdk init** command. Specify the desired template ("app") and programming language as shown in the following examples:

TypeScript

```
cdk init app --language typescript
```

JavaScript

```
cdk init app --language javascript
```

Python

```
cdk init app --language python
```

Build the app:

In most programming environments, after changing your code, you build (compile) it. This isn't strictly necessary with the AWS CDK—the Toolkit does it for you so that you can't

forget. But you can still build manually whenever you want to catch syntax and type errors. The cdk init command creates a number of files and folders inside the hello-cdk directory to help you organize the source code for your AWS CDK app If you have Git installed, each project you create using cdk init is also initialized as a Git repository. We'll ignore that for now, but it's there when you need it.

TypeScript

```
npm run build
```

JavaScript

No build step is necessary.

Python

No build step is necessary.

Java

```
mvn compile -q
```

Or press Control-B in Eclipse (other Java IDEs may vary)

C#

```
dotnet build src
```

Or press F6 in Visual Studio

Go

```
go build
```

List the stacks in the app

```
cdk ls
```

If you don't see HelloCdkStack, make sure you named your app's directory hello-cdk

Principle

The CDK's Amazon S3 support is part of its main library, aws-cdk-lib, so we don't need to install another library. We can define an Amazon S3 bucket in the stack using the Bucket construct.

TypeScript

In `lib/hello-cdk-stack.ts`:

```
import * as cdk from 'aws-cdk-lib';
import { aws_s3 as s3 } from 'aws-cdk-lib';

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

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}
```

JavaScript

In `lib/hello-cdk-stack.js`:

```
const cdk = require('aws-cdk-lib');
const s3 = require('aws-cdk-lib/aws-s3');

class HelloCdkStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}

module.exports = { HelloCdkStack }
```

Python

In hello_cdk/hello_cdk_stack.py:

```
import aws_cdk as cdk
import aws_cdk.aws_s3 as s3

class HelloCdkStack(cdk.Stack):

    def __init__(self, scope: cdk.App, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)
```

```
        bucket = s3.Bucket(self, "MyFirstBucket", versioned=True)
```

Java

In src/main/java/com/myorg/HelloCdkStack.java:

```
package com.myorg;

import software.amazon.awscdk.*;
import software.amazon.awscdk.services.s3.Bucket;

public class HelloCdkStack extends Stack {
    public HelloCdkStack(final App scope, final String id) {
        this(scope, id, null);
    }

    public HelloCdkStack(final App scope, final String id, final StackProps props) {
        super(scope, id, props);

        Bucket.Builder.create(this, "MyFirstBucket")
            .versioned(true).build();
    }
}
```

C#

In src/HelloCdk/HelloCdkStack.cs:

```
using Amazon.CDK;
using Amazon.CDK.AWS.S3;

namespace HelloCdk
{
    public class HelloCdkStack : Stack
    {
        public HelloCdkStack(App scope, string id, IStackProps props=null) :
 base(scope, id, props)
        {
            new Bucket(this, "MyFirstBucket", new BucketProps
            {
                Versioned = true
            });
        }
    }
}
```

C#

In src/HelloCdk/HelloCdkStack.cs:

```
using Amazon.CDK;
using Amazon.CDK.AWS.S3;

namespace HelloCdk
{
    public class HelloCdkStack : Stack
    {
        public HelloCdkStack(App scope, string id, IStackProps props=null) :
 base(scope, id, props)
        {
            new Bucket(this, "MyFirstBucket", new BucketProps
            {
                Versioned = true
            });
        }
    }
}
```

Go

In hello-cdk.go:

```
package main

import (
 "github.com/aws/aws-cdk-go/awscdk/v2"
 "github.com/aws/aws-cdk-go/awscdk/v2/awss3"
 "github.com/aws/constructs-go/constructs/v10"
 "github.com/aws/jsii-runtime-go"
)

type HelloCdkStackProps struct {
 awscdk.StackProps
```

Bucket is the first construct that we've seen, so let's take a closer look. Like all constructs, the Bucket class takes three parameters.

• scope: Tells the bucket that the stack is its parent: it is defined within the scope of the stack. You can define constructs inside of constructs, creating a hierarchy (tree). Here, and in most cases, the scope is this (self in Python), meaning the construct that contains the bucket: the stack.

• Id: The logical ID of the Bucket within your AWS CDK app. This (plus a hash based on the bucket's location within the stack) uniquely identifies the bucket across deployments. This way, the AWS CDK can update it if you change how it's defined in your app. Here, it's "MyFirstBucket." Buckets can also have a name, which is separate from this ID (it's the bucketName property).

• props: A bundle of values that define properties of the bucket. Here we've defined only one property: versioned, which enables versioning for the files in the bucket. All constructs take these same three arguments, so it's easy to stay oriented as you learn about new ones. And as

you might expect, you can subclass any construct to extend it to suit your needs, or if you want to change its defaults.

Tip

If a construct's props are all optional, you can omit the props parameter entirely. Props are represented differently in the languages supported by the AWS CDK.

• In TypeScript and JavaScript, props is a single argument and you pass in an object containing the desired properties.

• In Python, props are passed as keyword arguments.of development. Advancements in global shutter speeds and a number of companies taking a greater interest in ToF saw a steady stream of advancement.

Synthesize an AWS CloudFormation template

Synthesize an AWS CloudFormation template for the app, as follows.

cdk synth

If your app contained more than one stack, you'd need to specify which stack or stacks to synthesize. But since it only contains one, the CDK Toolkit knows you must mean that one.

Tip

If you received an error like --app is required..., it's probably because you are running the command from a subdirectory. Navigate to the main app directory and try again. The cdk synth command executes your app, which causes the resources defined in it to be translated into an AWS CloudFormation template. The displayed output of cdk synth is a YAML-format template. Following, you can see the beginning of our app's output. The template is also saved in the cdk.out directory in JSON format.

```
Resources:
  MyFirstBucketB8884501:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
    Metadata:...
```

Even if you aren't familiar with AWS CloudFormation, you can find the bucket definition and see how the versioned property was translated.

Note

Every generated template contains a AWS::CDK::Metadata resource by default. (We haven't shown it here.) The AWS CDK team uses this metadata to gain insight into how the AWS CDK is used, so that we can continue to improve it. For details, including how to opt out of version reporting, see Version reporting (p. 305). The cdk synth generates a perfectly valid AWS CloudFormation template. You could take it and deploy it using the AWS CloudFormation console or another tool. But the AWS CDK Toolkit can also do that. avoidance and obstacle detection. These tandem successes have made Time-of-Flight cameras one of the leading contenders for depth sensing in automated guided vehicles.

Deploying the stack:

To deploy the stack using AWS CloudFormation, issue:

cdk deploy

As with cdk synth, you don't need to specify the name of the stack since there's only one in the app. It is optional (though good practice) to synthesize before deploying. The AWS CDK synthesizes your stack before each deployment. If your code has security implications, you'll see a summary of these and need to confirm them before deployment proceeds. This isn't the case in our stack.

cdk deploy displays progress information as your stack is deployed. When it's done, the command prompt reappears. You can go to the AWS CloudFormation console and see that it now lists HelloCdkStack. You'll also find MyFirstBucket in the Amazon S3 console. You've deployed your first stack using the AWS CDK—congratulations! But that's not all there is to

the AWS CDK.

Modifying the app:

The AWS CDK can update your deployed resources after you modify your app. Let's change the bucket so it can be automatically deleted when deleting the stack. This involves changing the bucket's RemovalPolicy. Also, use the autoDeleteObjects property to ask the AWS CDK to delete the objects from the bucket before destroying it. (AWS CloudFormation doesn't delete S3 buckets that contain any objects.)

TypeScript

Update lib/hello-cdk-stack.ts.

```
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  autoDeleteObjects: true
});
```

JavaScript

Update lib/hello-cdk-stack.js.

```
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  autoDeleteObjects: true
});
```

Python

Update hello_cdk/hello_cdk_stack.py.

```
bucket = s3.Bucket(self, "MyFirstBucket",
    versioned=True,
    removal_policy=cdk.RemovalPolicy.DESTROY,
    auto_delete_objects=True)
```

To see these changes, we'll use the cdk diff command.

cdk diff

The AWS CDK Toolkit queries your AWS account for the last-deployed AWS CloudFormation template for the HelloCdkStack. Then, it compares the last-deployed template with the template it just synthesized from your app. The output should look like the following.

```
Stack HelloCdkStack
IAM Statement Changes
###############################################################################################################
#   # Resource                    # Effect # Action                      # Principal
              # Condition #
###############################################################################################################
# + # ${Custom::S3AutoDeleteObject # Allow  # sts:AssumeRole             #
 Service:lambda.amazonaws.com #           #
#   # sCustomResourceProvider/Role #        #                           #
              #           #
#   # .Arn}                       #        #                           #
              #           #
###############################################################################################################
# + # ${MyFirstBucket.Arn}         # Allow  # s3:DeleteObject*           # AWS:
${Custom::S3AutoDeleteOb #           #
#   # ${MyFirstBucket.Arn}/*       #        # s3:GetBucket*             #
 jectsCustomResourceProvider/ #           #
#   #                             #        # s3:GetObject*             # Role.Arn}
              #           #
#   #                             #        # s3:List*                  #
              #           #
###############################################################################################################
IAM Policy Changes
###############################################################################################################
#   # Resource                                                # Managed Policy ARN
                          #
###############################################################################################################
# + # ${Custom::S3AutoDeleteObjectsCustomResourceProvider/Ro # {"Fn::Sub":"arn:
${AWS::Partition}:iam::aws:policy/serv #
#   # le}                                                     # ice-role/
AWSLambdaBasicExecutionRole"}               #
```

This diff has four sections.

• IAM Statement Changes and IAM Policy Changes - These permission changes are there because we set the AutoDeleteObjects property on our Amazon S3 bucket. The auto-delete feature uses a custom resource to delete the objects in the bucket before the bucket itself is deleted. The IAM objects grant the custom resource's code access to the bucket.

• Parameters - The AWS CDK uses these entries to locate the Lambda function asset for the custom resource.

• Resources - The new and changed resources in this stack. We can see the previously mentioned IAM objects, the custom resource, and its associated Lambda function being added.

We can also see that the bucket's DeletionPolicy and UpdateReplacePolicy attributes are being updated. These allow the bucket to be deleted along with the stack, and to be replaced with a new one.

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

Parameters
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/S3Bucket
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3BucketBF7A7F3F:
 {"Type":"String","Description":"S3 bucket for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
S3VersionKey
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3VersionKeyFAF93626:
 {"Type":"String","Description":"S3 key for asset version
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
ArtifactHash
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392ArtifactHashE56CD69A:
 {"Type":"String","Description":"Artifact hash for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}

Resources
[+] AWS::S3::BucketPolicy MyFirstBucket/Policy MyFirstBucketPolicy3243DEFD
[+] Custom::S3AutoDeleteObjects MyFirstBucket/AutoDeleteObjectsCustomResource
 MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E
[+] AWS::IAM::Role Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092
[+] AWS::Lambda::Function Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F
[~] AWS::S3::Bucket MyFirstBucket MyFirstBucketB8884501
 ## [~] DeletionPolicy
 #   ## [-] Retain
 #   ## [+] Delete
 ## [~] UpdateReplacePolicy
     ## [-] Retain
     ## [+] Delete
```

All AWS CDK v2 deployments use dedicated AWS resources to hold data during deployment. Therefore, your AWS account and Region must be bootstrapped (p. 195) to create these resources before you can deploy.

Deploy

cdk deploy:

The AWS CDK warns you about the security policy changes we've already seen in the diff. Enter y to approve the changes and deploy the updated stack. The CDK Toolkit updates the bucket configuration as you requested.

```
HelloCdkStack: deploying...
[0%] start: Publishing
 4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392:current
[100%] success: Published
 4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392:current
HelloCdkStack: creating CloudFormation changeset...
 0/5 | 4:32:31 PM | UPDATE_IN_PROGRESS   | AWS::CloudFormation::Stack  | HelloCdkStack User
 Initiated
 0/5 | 4:32:36 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092)
 1/5 | 4:32:36 PM | UPDATE_COMPLETE      | AWS::S3::Bucket             | MyFirstBucket
 (MyFirstBucketB8884501)
 1/5 | 4:32:36 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092) Resource creation Initiated
 3/5 | 4:32:54 PM | CREATE_COMPLETE      | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::Lambda::Function
     | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::Lambda::Function
     | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F) Resource creation
 Initiated
 3/5 | 4:32:57 PM | CREATE_COMPLETE      | AWS::Lambda::Function
     | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F)
 3/5 | 4:32:57 PM | CREATE_IN_PROGRESS   | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD) Resource creation Initiated
 4/5 | 4:32:57 PM | CREATE_COMPLETE      | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD)
 4/5 | 4:32:59 PM | CREATE_IN_PROGRESS   | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E)
 5/5 | 4:33:06 PM | CREATE_IN_PROGRESS   | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E) Resource creation Initiated
 5/5 | 4:33:06 PM | CREATE_COMPLETE      | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E)
 5/5 | 4:33:08 PM | UPDATE_COMPLETE_CLEA | AWS::CloudFormation::Stack  | HelloCdkStack
 6/5 | 4:33:09 PM | UPDATE_COMPLETE      | AWS::CloudFormation::Stack  | HelloCdkStack
```

```
 #  HelloCdkStack

Stack ARN:
arn:aws:cloudformation:REGION:ACCOUNT:stack/HelloCdkStack/UNIQUE-ID
```

Destroying the app's resources:

Now that you're done with the quick tour, destroy your app's resources to avoid incurring any costs from the bucket you created, as follows.

cdk destroy

Enter y to approve the changes and delete any stack resources.

Note

If we didn't change the bucket's RemovalPolicy, the stack deletion would completesuccessfully, but the bucket would become orphaned (no longer associated with the stack).

Working with the AWS CDK:

The AWS Cloud Development Kit (AWS CDK) lets you define your AWS cloud infrastructure in a generalpurpose programming language. Currently, the AWS CDK supports TypeScript, JavaScript, Python, Java, C#, and Go. It is also possible to use other JVM and .NET languages, though we are unable to provide support for every such language.

We develop the AWS CDK in TypeScript and use JSII to provide a "native" experience in other supported languages. For example, we distribute AWS Construct Library modules using your preferred language's standard repository, and you install them using the language's standard package manager. Methods and properties are even named using your language's recommended naming patterns.

AWS CDK prerequisites:

To use the AWS CDK, you need an AWS account and a corresponding access key. If you don't have an AWS account yet, see Create and Activate an AWS Account. To find out how to obtain an access key ID and secret access key for your AWS account, see Understanding and Getting Your Security Credentials. To find out how to configure your workstation so the AWS CDK uses your credentials, see Setting Credentials in Node.js.

Tip

If you have the AWS CLI installed, the simplest way to set up your workstation with your AWS credentials is to open a command prompt and type:

```
aws configure
```

After installing Node.js, install the AWS CDK Toolkit (the cdk command):

```
npm install -g aws-cdk
```

Note
If you get a permission error, and have administrator access on your system, try `sudo npm install -g aws-cdk`.

Test the installation by issuing `cdk --version`.

AWS Construct Library:

The AWS CDK includes the AWS Construct Library, a collection of constructs organized by AWS service. The library's constructs are mainly in a single module, colloquially called aws-cdk-lib because that's its name in TypeScript. The actual package name of the main CDK package varies by language.

TypeScript

Install	`npm install aws-cdk-lib`
Import	`const cdk = require('aws-cdk-lib');`

JavaScript

Install	`npm install aws-cdk-lib`
Import	`const cdk = require('aws-cdk-lib');`

Python

Install	`python -m pip install aws-cdk-lib`
Import	`import aws_cdk as cdk`

Java

Add to `pom.xml`	Group `software.amazon.awscdk`; artifact `aws-cdk-lib`
Import	`import software.amazon.awscdk.App;` (for example)

Interfaces vs. construct classes:

The AWS CDK uses interfaces in a specific way that might not be obvious even if you are familiar with interfaces as a programming concept. The AWS CDK supports using resources

defined outside CDK applications using methods such as Bucket.fromBucketArn(). External resources cannot be modified and may not have all the functionality available with resources defined in your CDK app using e.g. the Bucket class. Interfaces, then, represent the bare minimum functionality available in the CDK for a given AWS resource type, including external resources. When instantiating resources in your CDK app, then, you should always use concrete classes such as Bucket. When specifying the type of an argument you are accepting in one of your own constructs, use the interface type such as IBucket if you are prepared to deal with external resources (that is, you won't need to change them). If you require a CDK-defined construct, specify the most general type you can use. Some interfaces are minimum versions of properties or options bundles associated with specific classes, rather than constructs. Such interfaces can be useful when subclassing to accept arguments that you'll pass on to your parent class. If you require one or more additional properties, you'll want to implement or derive from this interface, or from a more specific type.

Note

Some programming languages supported by the AWS CDK don't have an interface feature. In these languages, interfaces are just ordinary classes. You can identify them by their names, which follow the pattern of an initial "I" followed by the name of some other construct (e.g. IBucket). The same rules apply.

Amazon API Gateway Construct:

Amazon API Gateway is a fully managed service that makes it easy for developers to publish, maintain, monitor, and secure APIs at any scale. Create an API to access data, business logic,

or functionality from your back-end services, such as applications running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, or any web application.

APIs are defined as a hierarchy of resources and methods. addResource and addMethod can be used to build this hierarchy. The root resource is api.root. For example, the following code defines an API that includes the following HTTP endpoints: ANY /, GET /books, POST /books, GET /books/{book_id}, DELETE /books/{book_id}.

```
const api = new apigateway.RestApi(this, 'books-api');

api.root.addMethod('ANY');

const books = api.root.addResource('books');
books.addMethod('GET');
books.addMethod('POST');

const book = books.addResource('{book_id}');
book.addMethod('GET');
book.addMethod('DELETE');
```

AWS Lambda-backed APIs:

A very common practice is to use Amazon API Gateway with AWS Lambda as the backend integration. The LambdaRestApi construct makes it easy:

```
declare const backend: lambda.Function;
new apigateway.LambdaRestApi(this, 'myapi', {
  handler: backend,
});
```

You can also supply proxy: false, in which case you will have to explicitly define the API model:

```
declare const backend: lambda.Function;
const api = new apigateway.LambdaRestApi(this, 'myapi', {
  handler: backend,
  proxy: false
});

const items = api.root.addResource('items');
items.addMethod('GET');  // GET /items
items.addMethod('POST'); // POST /items

const item = items.addResource('{item}');
item.addMethod('GET');   // GET /items/{item}

// the default integration for methods is "handler", but one can
// customize this behavior per method or even a sub path.
item.addMethod('DELETE', new apigateway.HttpIntegration('http://amazon.com'))
```

AWS StepFunctions backed APIs:

You can use Amazon API Gateway with AWS Step Functions as the backend integration, specifically Synchronous Express Workflows. The StepFunctionsRestApi only supports integration with Synchronous Express state machine. The StepFunctionsRestApi construct makes this easy by setting up input, output and error mapping. The construct sets up an API endpoint and maps the ANY HTTP method and any calls to the API endpoint starts an express workflow execution for the underlying state machine. Invoking the endpoint with any HTTP method (GET, POST, PUT, DELETE, ...) in the example below will send the request to the state machine as a new execution. On success, an HTTP code 200 is returned with the execution output as the Response Body. If the execution fails, an HTTP 500 response is returned with the error and cause from the execution output as the Response Body. If the request is invalid (ex. bad execution input) HTTP code 400 is returned. The response from the invocation contains only the output field from the StartSyncExecution API. In case of failures, the fields error and cause are returned as part of the response. Other metadata such as billing details, AWS account ID and resource ARNs are not returned in the API response. By default, a prod stage is provisioned

In order to reduce the payload size sent to AWS Step Functions, headers are not forwarded to the Step Functions execution input. It is possible to choose whether headers, requestContext, path, querystring, and authorizer are included or not. By default, headers are excluded in all

requests. More details about AWS Step Functions payload limit can be found at https://docs.aws.amazon.com/step-functions/latest/dg/limits-overview.html#service-limits-task-executions. The following code defines a REST API that routes all requests to the specified AWS StepFunctions state machine: The following code defines a REST API that routes all requests to the specified AWS Lambda function:

```
const stateMachineDefinition = new stepfunctions.Pass(this, 'PassState');

const stateMachine: stepfunctions.IStateMachine = new stepfunctions.StateMach
  definition: stateMachineDefinition,
  stateMachineType: stepfunctions.StateMachineType.EXPRESS,
});

new apigateway.StepFunctionsRestApi(this, 'StepFunctionsRestApi', {
  deploy: true,
  stateMachine: stateMachine,
});
```

When the REST API endpoint configuration above is invoked using POST, as follows

```
curl -X POST -d '{ "customerId": 1 }' https://example.com/
```

AWS Step Functions will receive the request body in its input as follows:

```
{
  "body": {
    "customerId": 1
  },
  "path": "/",
  "querystring": {}
}
```

When the endpoint is invoked at path '/users/5' using the HTTP GET method as below:

```
curl -X GET https://example.com/users/5?foo=bar
```

AWS Step Functions will receive the following execution input:

```
{
  "body": {},
  "path": {
     "users": "5"
  },
  "querystring": {
    "foo": "bar"
  }
}
```

Additional information around the request such as the request context, authorizer context, and headers can be included as part of the input forwarded to the state machine. The following example enables headers to be included in the input but not query string.

```
new apigateway.StepFunctionsRestApi(this, 'StepFunctionsRestApi', {
  stateMachine: machine,
  headers: true,
  path: false,
  querystring: false,
  authorizer: false,
  requestContext: {
    caller: true,
    user: true,
  },
});
```

In such a case, when the endpoint is invoked as below:

```
curl -X GET https://example.com/
```

AWS Step Functions will receive the following execution input:

```
{
  "headers": {
    "Accept": "...",
    "CloudFront-Forwarded-Proto": "...",
  },
  "requestContext": {
     "accountId": "...",
     "apiKey": "...",
  },
  "body": {}
}
```

AWS Lambda Construct:

This construct library allows you to define AWS Lambda Functions.

```
const fn = new lambda.Function(this, 'MyFunction', {
  runtime: lambda.Runtime.NODEJS_16_X,
  handler: 'index.handler',
  code: lambda.Code.fromAsset(path.join(__dirname, 'lambda-handler')),
});
```

Handler Code:

The lambda.Code class includes static convenience methods for various types of runtime code. lambda.Code.fromBucket(bucket, key[, objectVersion]) - specify an S3 object that contains the archive of your runtime code. lambda.Code.fromInline(code) - inline the handle code as a string. This is limited to supported runtimes and the code cannot exceed 4KiB. lambda.Code.fromAsset(path) - specify a directory or a .zip file in the local filesystem which will be zipped and uploaded to S3 before deployment. See also bundling asset code. lambda.Code.fromDockerBuild(path, options) - use the result of a Docker build as code. The runtime code is expected to be located at /asset in the image and will be zipped and uploaded to S3 as an asset. The following example shows how to define a Python function and deploy the code from the local directory my-lambda-handler to it:

```
new lambda.Function(this, 'MyLambda', {
  code: lambda.Code.fromAsset(path.join(__dirname, 'my-lambda-handler'
  handler: 'index.main',
  runtime: lambda.Runtime.PYTHON_3_9,
});
```

When deploying a stack that contains this code, the directory will be zip archived and then uploaded to an S3 bucket, then the exact location of the S3 objects will be passed when the stack is deployed.

Execution Role:

Lambda functions assume an IAM role during execution. In CDK by default, Lambda functions will use an autogenerated Role if one is not provided.

The autogenerated Role is automatically given permissions to execute the Lambda function. To reference the autogenerated Role:

```
const fn = new lambda.Function(this, 'MyFunction', {
  runtime: lambda.Runtime.NODEJS_16_X,
  handler: 'index.handler',
  code: lambda.Code.fromAsset(path.join(__dirname, 'lambda-handler')),
});

const role = fn.role; // the Role
```

You can also provide your own IAM role. Provided IAM roles will not automatically be given permissions to execute the Lambda function. To provide a role and grant it appropriate permissions.

Function Timeout:

AWS Lambda functions have a default timeout of 3 seconds, but this can be increased up to 15 minutes. The timeout is available as a property of Function so that you can reference it elsewhere in your stack. For instance, you could use it to create a CloudWatch alarm to report when your function timed out.

```
import * as cdk from '@aws-cdk/core';
import * as cloudwatch from '@aws-cdk/aws-cloudwatch';

const fn = new lambda.Function(this, 'MyFunction', {
   runtime: lambda.Runtime.NODEJS_16_X,
   handler: 'index.handler',
   code: lambda.Code.fromAsset(path.join(__dirname, 'lambda-handler'))
   timeout: cdk.Duration.minutes(5),
});

if (fn.timeout) {
   new cloudwatch.Alarm(this, `MyAlarm`, {
      metric: fn.metricDuration().with({
         statistic: 'Maximum',
      }),
      evaluationPeriods: 1,
      datapointsToAlarm: 1,
      threshold: fn.timeout.toMilliseconds(),
      treatMissingData: cloudwatch.TreatMissingData.IGNORE,
      alarmName: 'My Lambda Timeout',
   });
}
```

```
const myRole = new iam.Role(this, 'My Role', {
  assumedBy: new iam.ServicePrincipal('sns.amazonaws.com'),
});

const fn = new lambda.Function(this, 'MyFunction', {
  runtime: lambda.Runtime.NODEJS_16_X,
  handler: 'index.handler',
  code: lambda.Code.fromAsset(path.join(__dirname, 'lambda-handler')),
  role: myRole, // user-provided role
});

myRole.addManagedPolicy(iam.ManagedPolicy.fromAwsManagedPolicyName("se
myRole.addManagedPolicy(iam.ManagedPolicy.fromAwsManagedPolicyName("se
```

Resource-based Policies:

AWS Lambda supports resource-based policies for controlling access to Lambda functions and layers on a per-resource basis. In particular, this allows you to give permission to AWS services and other AWS accounts to modify and invoke your functions. You can also restrict permissions given to AWS services by providing a source account or ARN (representing the account and identifier of the resource that accesses the function or layer).

```
declare const fn: lambda.Function;
const principal = new iam.ServicePrincipal('my-service');

fn.grantInvoke(principal);

// Equivalent to:
fn.addPermission('my-service Invocation', {
  principal: principal,
});
```

Providing an unowned principal (such as account principals, generic ARN principals, service principals, and principals in other accounts) to a call to fn.grantInvoke will result in a resource-based policy being created. If the principal in question has conditions limiting the source account or ARN of the operation (see above), these conditions will be automatically added to the resource policy.

During synthesis, the CDK expects to find a directory on disk at the asset directory specified. Note that we are referencing the asset directory relatively to our CDK project directory. This is especially important when we want to share this construct through a library. Different programming languages will have different techniques for bundling resources into libraries.

```
declare const fn: lambda.Function;
const servicePrincipal = new iam.ServicePrincipal('my-service');
const sourceArn = 'arn:aws:s3:::my-bucket';
const sourceAccount = '111122223333';
const servicePrincipalWithConditions = servicePrincipal.withConditions
  ArnLike: {
    'aws:SourceArn': sourceArn,
  },
  StringEquals: {
    'aws:SourceAccount': sourceAccount,
  },
});

fn.grantInvoke(servicePrincipalWithConditions);

// Equivalent to:
fn.addPermission('my-service Invocation', {
  principal: servicePrincipal,
  sourceArn: sourceArn,
  sourceAccount: sourceAccount,
});
```

Amazon DynamoDB Construct:

Here is a minimal deployable DynamoDB table definition:

```
const table = new dynamodb.Table(this, 'Table', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
});
```

Importing existing tables:

To import an existing table into your CDK application, use the Table.fromTableName, Table.fromTableArn or Table.fromTableAttributes factory method. This method accepts table name or table ARN which describes the properties of an already existing table: If you intend to use the tableStreamArn (including indirectly, for example by creating an @aws-cdk/aws-lambda-event-source.DynamoEventSource on the imported table), you must use the Table.fromTableAttributes method and the tableStreamArn property must be populated.

```
declare const user: iam.User;
const table = dynamodb.Table.fromTableArn(this, 'ImportedTable', 'arn:
// now you can just call methods on the table
table.grantReadWriteData(user);
```

Keys:

When a table is defined, you must define it's schema using the partitionKey (required) and sortKey (optional) properties.

Billing Mode:

DynamoDB supports two billing modes:

PROVISIONED - the default mode where the table and global secondary indexes have configured read and write capacity.

PAY_PER_REQUEST - on-demand pricing and scaling. You only pay for what you use and there is no read and write capacity for the table or its global secondary indexes.

```
const table = new dynamodb.Table(this, 'Table', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  billingMode: dynamodb.BillingMode.PAY_PER_REQUEST,
});
```

Table Class:

DynamoDB supports two table classes:

STANDARD - the default mode, and is recommended for the vast majority of workloads.

STANDARD_INFREQUENT_ACCESS - optimized for tables where storage is the dominant cost.

```
const table = new dynamodb.Table(this, 'Table', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  tableClass: dynamodb.TableClass.STANDARD_INFREQUENT_ACCESS,
});
```

Amazon DynamoDB Global Tables:

You can create DynamoDB Global Tables by setting the replicationRegions property on a Table:

```
const globalTable = new dynamodb.Table(this, 'Table', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  replicationRegions: ['us-east-1', 'us-east-2', 'us-west-2'],
});
```

When doing so, a CloudFormation Custom Resource will be added to the stack in order to create the replica tables in the selected regions.

The default billing mode for Global Tables is PAY_PER_REQUEST. If you want to use PROVISIONED, you have to make sure write auto-scaling is enabled for that Table:

```
const globalTable = new dynamodb.Table(this, 'Table', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  replicationRegions: ['us-east-1', 'us-east-2', 'us-west-2'],
  billingMode: dynamodb.BillingMode.PROVISIONED,
});

globalTable.autoScaleWriteCapacity({
  minCapacity: 1,
  maxCapacity: 10,
}).scaleOnUtilization({ targetUtilizationPercent: 75 });
```

When adding a replica region for a large table, you might want to increase the timeout for the replication operation:

```
const globalTable = new dynamodb.Table(this, 'Table', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  replicationRegions: ['us-east-1', 'us-east-2', 'us-west-2'],
  replicationTimeout: Duration.hours(2), // defaults to Duration.minut
});
```

Encryption

All user data stored in Amazon DynamoDB is fully encrypted at rest. When creating a new table, you can choose to encrypt using the following customer master keys (CMK) to encrypt your table:

AWS owned CMK - By default, all tables are encrypted under an AWS owned customer master key (CMK) in the DynamoDB service account (no additional charges apply). AWS managed CMK - AWS KMS keys (one per region) are created in your account, managed, and used on your behalf by AWS DynamoDB (AWS KMS charges apply). Customer managed CMK - You have full control over the KMS key used to encrypt the DynamoDB Table (AWS KMS charges apply). Creating a Table encrypted with a customer managed CMK:

```
const table = new dynamodb.Table(this, 'MyTable', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  encryption: dynamodb.TableEncryption.CUSTOMER_MANAGED,
});

// You can access the CMK that was added to the stack on your behalf b
const tableEncryptionKey = table.encryptionKey;
```

You can also supply your own key:

```
import * as kms from '@aws-cdk/aws-kms';

const encryptionKey = new kms.Key(this, 'Key', {
  enableKeyRotation: true,
});
const table = new dynamodb.Table(this, 'MyTable', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  encryption: dynamodb.TableEncryption.CUSTOMER_MANAGED,
  encryptionKey, // This will be exposed as table.encryptionKey
});
```

Amazon S3 Construct

Define an unencrypted S3 bucket.

```
const bucket = new s3.Bucket(this, 'MyFirstBucket');
```

Bucket constructs expose the following deploy-time attributes:

- bucketArn - the ARN of the bucket (i.e. arn:aws:s3:::bucket_name)
- bucketName - the name of the bucket (i.e. bucket_name)
- bucketWebsiteUrl - the Website URL of the bucket (i.e. http://bucket_name.s3-website-us-west-1.amazonaws.com)
- bucketDomainName - the URL of the bucket (i.e. bucket_name.s3.amazonaws.com)
- bucketDualStackDomainName - the dual-stack URL of the bucket (i.e. bucket_name.s3.dualstack.eu-west-1.amazonaws.com)
- bucketRegionalDomainName - the regional URL of the bucket (i.e. bucket_name.s3.eu-west-1.amazonaws.com)
- arnForObjects(pattern) - the ARN of an object or objects within the bucket (i.e. arn:aws:s3:::bucket_name/exampleobject.png or arn:aws:s3:::bucket_name/Development/*)
- urlForObject(key) - the HTTP URL of an object within the bucket (i.e. https://s3.cn-north-1.amazonaws.com.cn/china-bucket/mykey)
- virtualHostedUrlForObject(key) - the virtual-hosted style HTTP URL of an object within the bucket (i.e. https://china-bucket-s3.cn-north-1.amazonaws.com.cn/mykey)
- s3UrlForObject(key) - the S3 URL of an object within the bucket (i.e. s3://bucket/mykey)

Encryption:

Define a KMS-encrypted bucket:

```
const bucket = new s3.Bucket(this, 'MyEncryptedBucket', {
  encryption: s3.BucketEncryption.KMS,
});

// you can access the encryption key:
assert(bucket.encryptionKey instanceof kms.Key);
```

You can also supply your own key:

```
const myKmsKey = new kms.Key(this, 'MyKey');

const bucket = new s3.Bucket(this, 'MyEncryptedBucket', {
  encryption: s3.BucketEncryption.KMS,
  encryptionKey: myKmsKey,
});

assert(bucket.encryptionKey === myKmsKey);
```

Enable KMS-SSE encryption via S3 Bucket Keys:

```
const bucket = new s3.Bucket(this, 'MyEncryptedBucket', {
  encryption: s3.BucketEncryption.KMS,
  bucketKeyEnabled: true,
});
```

Use BucketEncryption.ManagedKms to use the S3 master KMS key:

```
const bucket = new s3.Bucket(this, 'Buck', {
  encryption: s3.BucketEncryption.KMS_MANAGED,
});

assert(bucket.encryptionKey == null);
```

Permissions:

A bucket policy will be automatically created for the bucket upon the first call to addToResourcePolicy(statement):

```
const bucket = new s3.Bucket(this, 'MyBucket');
const result = bucket.addToResourcePolicy(new iam.PolicyStatement({
  actions: ['s3:GetObject'],
  resources: [bucket.arnForObjects('file.txt')],
  principals: [new iam.AccountRootPrincipal()],
}));
```

If you try to add a policy statement to an existing bucket, this method will not do anything:

```
const bucket = s3.Bucket.fromBucketName(this, 'existingBucket', 'bucket-name

// No policy statement will be added to the resource
const result = bucket.addToResourcePolicy(new iam.PolicyStatement({
  actions: ['s3:GetObject'],
  resources: [bucket.arnForObjects('file.txt')],
  principals: [new iam.AccountRootPrincipal()],
}));
```

That's because it's not possible to tell whether the bucket already has a policy attached, let alone to re-use that policy to add more statements to it. We recommend that you always check the result of the call:

```
const bucket = new s3.Bucket(this, 'MyBucket');
const result = bucket.addToResourcePolicy(new iam.PolicyStatement({
  actions: ['s3:GetObject'],
  resources: [bucket.arnForObjects('file.txt')],
  principals: [new iam.AccountRootPrincipal()],
}));

if (!result.statementAdded) {
  // Uh-oh! Someone probably made a mistake here.
}
```

The bucket policy can be directly accessed after creation to add statements or adjust the removal policy.

```
const bucket = new s3.Bucket(this, 'MyBucket');
bucket.policy?.applyRemovalPolicy(cdk.RemovalPolicy.RETAIN);
```

Most of the time, you won't have to manipulate the bucket policy directly. Instead, buckets have "grant" methods called to give prepackaged sets of permissions to other resources. For example:

```
declare const myLambda: lambda.Function;

const bucket = new s3.Bucket(this, 'MyBucket');
bucket.grantReadWrite(myLambda);
```

Amazon Simple Notification Service Construct:

Add an SNS Topic to your stack:

```
const topic = new sns.Topic(this, 'Topic', {
  displayName: 'Customer subscription topic',
});
```

Add a FIFO SNS topic with content-based de-duplication to your stack:

```
const topic = new sns.Topic(this, 'Topic', {
  contentBasedDeduplication: true,
  displayName: 'Customer subscription topic',
  fifo: true,
  topicName: 'customerTopic',
});
```

Subscriptions:

Various subscriptions can be added to the topic by calling the .addSubscription(...) method on the topic. It accepts a subscription object, default implementations of which can be found in the @aws-cdk/aws-sns-subscriptions package:

Add an HTTPS Subscription to your topic:

```
const myTopic = new sns.Topic(this, 'MyTopic');

myTopic.addSubscription(new subscriptions.UrlSubscription('https://foobar.com
```

Subscribe a queue to the topic:

```
declare const queue: sqs.Queue;
const myTopic = new sns.Topic(this, 'MyTopic');

myTopic.addSubscription(new subscriptions.SqsSubscription(queue));
```

Amazon Simple Queue Service Construct:

Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated with managing and operating message oriented middleware, and empowers developers to focus on differentiating work. Using SQS, you can send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available.

Installation:

Import to your project:

```
import * as sqs from '@aws-cdk/aws-sqs';
```

Basic usage:

Here's how to add a basic queue to your application:

```
new sqs.Queue(this, 'Queue');
```

Encryption:

If you want to encrypt the queue contents, set the encryption property. You can have the messages encrypted with a key that SQS manages for you, or a key that you can manage yourself.

```
// Use managed key
new sqs.Queue(this, 'Queue', {
  encryption: sqs.QueueEncryption.KMS_MANAGED,
});

// Use custom key
const myKey = new kms.Key(this, 'Key');

new sqs.Queue(this, 'Queue', {
  encryption: sqs.QueueEncryption.KMS,
  encryptionMasterKey: myKey,
});
```

First-In-First-Out (FIFO) queues:

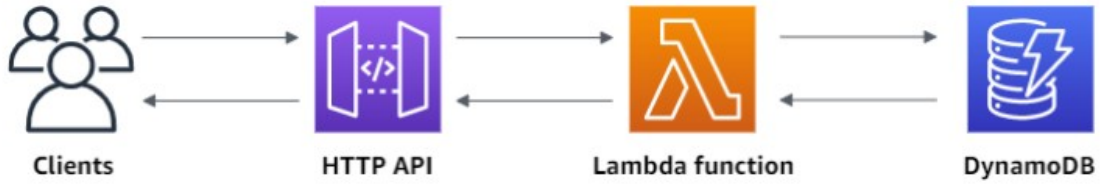
FIFO queues give guarantees on the order in which messages are dequeued, and have additional features in order to help guarantee exactly-once processing. For more information, see the SQS manual. Note that FIFO queues are not available in all AWS regions.

A queue can be made a FIFO queue by either setting fifo: true, giving it a name which ends in ".fifo", or by enabling a FIFO specific feature such as: content-based deduplication, deduplication scope or fifo throughput limit.

Build a CRUD API with Lambda and DynamoDB

First, you create a DynamoDB table using the DynamoDB console. Then you create a Lambda function using the AWS Lambda console. Next, you create an HTTP API using the API Gateway console. Lastly, you test your API.

When you invoke your HTTP API, API Gateway routes the request to your Lambda function. The Lambda function interacts with DynamoDB, and returns a response to API Gateway. API Gateway then returns a response to you.



To complete this exercise, you need an AWS account and an AWS Identity and Access Management user with console access. For more information, see Prerequisites for getting started with API Gateway.

In this tutorial, you use the AWS Management Console. For an AWS SAM template that creates this API and all related resources, see template.yaml.

Step 1: Create a DynamoDB table

To create a DynamoDB table

1. Open the DynamoDB console at https://console.aws.amazon.com/dynamodb/.
2. Choose **Create table**.
3. For **Table name**, enter `http-crud-tutorial-items`.
4. For **Partition key**, enter `id`.
5. Choose **Create table**.

Step 2: Create a Lambda function:

You create a Lambda function for the backend of your API. This Lambda function creates, reads, updates, and deletes items from DynamoDB. The function uses events from API Gateway to determine how to interact with DynamoDB. For simplicity this tutorial uses a single Lambda function. As a best practice, you should create separate functions for each route.

To create a Lambda function

1. Sign in to the Lambda console at https://console.aws.amazon.com/lambda.
2. Choose **Create function**.
3. For **Function name**, enter `http-crud-tutorial-function`.
4. Under **Permissions** choose **Change default execution role**.
5. Select **Create a new role from AWS policy templates**.
6. For **Role name**, enter `http-crud-tutorial-role`.
7. For **Policy templates**, choose `Simple microservice permissions`. This policy grants the Lambda function permission to interact with DynamoDB.

> **Note**
> This tutorial uses a managed policy for simplicity. As a best practice, you should create your own IAM policy to grant the minimum permissions required.

8. Choose **Create function**.
9. Open `index.mjs` in the console's code editor, and replace its contents with the following code. Choose **Deploy** to update your function.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamo";
import {
  DynamoDBDocumentClient,
  ScanCommand,
  PutCommand,
  GetCommand,
  DeleteCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

const dynamo = DynamoDBDocumentClient.from(client);

const tableName = "http-crud-tutorial-items";

export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

  try {
    switch (event.routeKey) {
      case "DELETE /items/{id}":
        await dynamo.send(
          new DeleteCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = `Deleted item ${event.pathParameters.id}`;
        break;
      case "GET /items/{id}":
        body = await dynamo.send(
          new GetCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
```

```
            },
          })
        );
        body = body.Item;
        break;
      case "GET /items":
        body = await dynamo.send(
          new ScanCommand({ TableName: tableName })
        );
        body = body.Items;
        break;
      case "PUT /items":
        let requestJSON = JSON.parse(event.body);
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
            Item: {
              id: requestJSON.id,
              price: requestJSON.price,
              name: requestJSON.name,
            },
          })
        );
        body = `Put item ${requestJSON.id}`;
        break;
      default:
        throw new Error(`Unsupported route: "$
{event.routeKey}"`);
    }
  } catch (err) {
    statusCode = 400;
    body = err.message;
  } finally {
    body = JSON.stringify(body);
  }

  return {
    statusCode,
    body,
    headers,
  };
};
```

Step 3: Create an HTTP API:

The HTTP API provides an HTTP endpoint for your Lambda function. In this step, you create an empty API. In the following steps, you configure routes and integrations to connect your API and your Lambda function.

To create an HTTP API

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose Create API, and then for HTTP API, choose Build.
- For API name, enter http-crud-tutorial-api.
- Choose Next.
- For Configure routes, choose Next to skip route creation. You create routes later.
- Review the stage that API Gateway creates for you, and then choose Next.
- Choose Create.

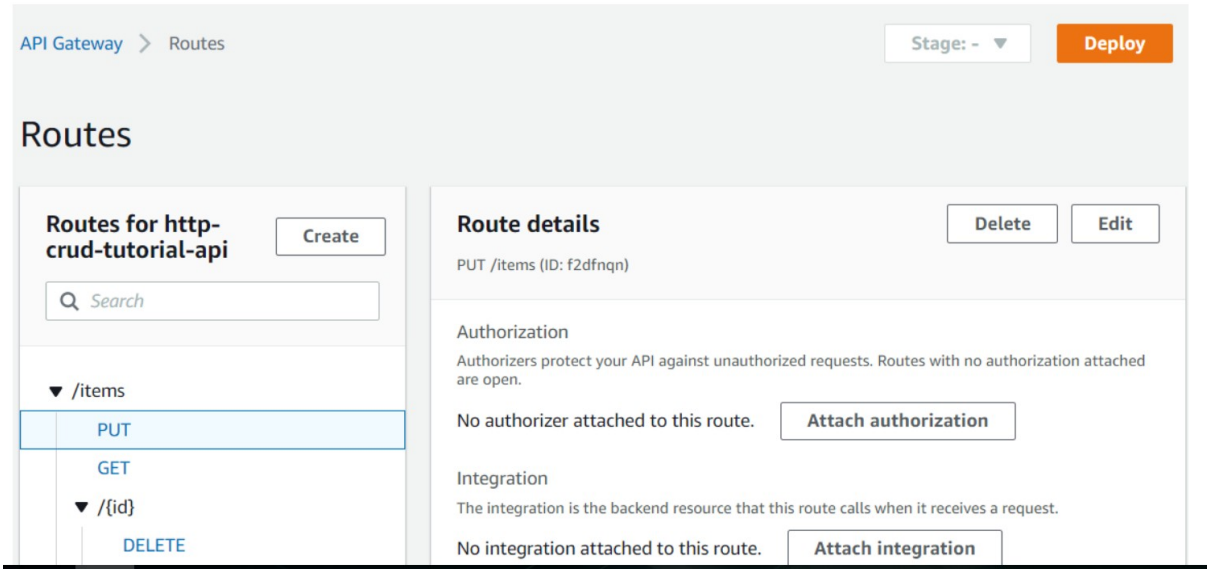
Step 4: Create routes:

Routes are a way to send incoming API requests to backend resources. Routes consist of two parts: an HTTP method and a resource path, for example, GET /items. For this example API, we create four routes:

- GET /items/{id}
- GET /items
- PUT /items
- DELETE /items/{id}

To create routes

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose your API.
- Choose Routes.
- Choose Create.
- For Method, choose GET.
- For the path, enter /items/{id}. The {id} at the end of the path is a path parameter that API Gateway retrieves from the request path when a client makes a request.
- Choose Create.
- Repeat steps 4-7 for GET /items, DELETE /items/{id}, and PUT /items.

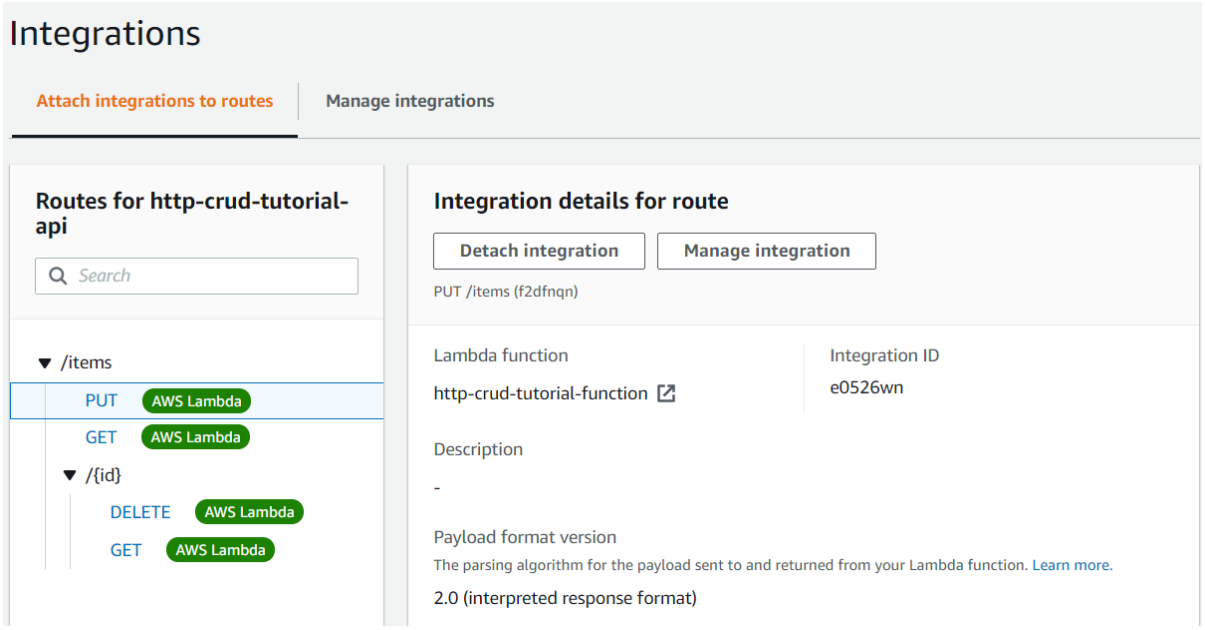


Step 6: Attach your integration to routes:

For this example API, you use the same Lambda integration for all routes. After you attach the integration to all of the API's routes, your Lambda function is invoked when a client calls any of your routes.

To attach integrations to routes

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose your API.
- Choose Integrations.
- Choose a route.
- Under Choose an existing integration, choose http-crud-tutorial-function.
- Choose Attach integration.
- Repeat steps 4-6 for all routes.



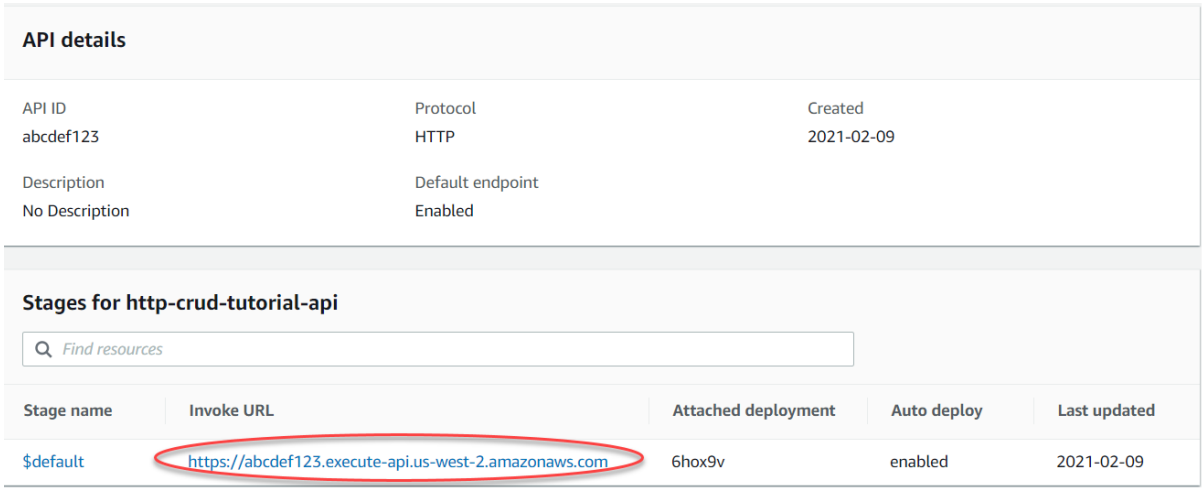
Step 7: Test your API:

To get the URL to invoke your API

Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.

Choose your API.

Note your API's invoke URL. It appears under Invoke URL on the Details page.



Copy your API's invoke URL.

The full URL looks like https://abcdef123.execute-api.us-west-2.amazonaws.com.

To create or update an item

Use the following command to create or update an item. The command includes a request body with the item's ID, price, and name.

curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\": \"123\", \"price\": 12345, \"name\": \"myitem\"}" https://abcdef123.execute-api.us-west-2.amazonaws.com/items

To get all items

Use the following command to list all items.

curl https://abcdef123.execute-api.us-west-2.amazonaws.com/items

To get an item

Use the following command to get an item by its ID.

curl https://abcdef123.execute-api.us-west-2.amazonaws.com/items/123

To delete an item

Use the following command to delete an item.

curl -X "DELETE" https://abcdef123.execute-api.us-west-2.amazonaws.com/items/123

Get all items to verify that the item was deleted.

curl https://abcdef123.execute-api.us-west-2.amazonaws.com/items

Step 8: Clean up:

To delete a DynamoDB table:

- Open the DynamoDB console at https://console.aws.amazon.com/dynamodb/.
- Select your table.
- Choose Delete table.
- Confirm your choice, and choose Delete.

To delete an HTTP API:

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- On the APIs page, select an API. Choose Actions, and then choose Delete.
- Choose Delete.

To delete a Lambda function:

- Sign in to the Lambda console at https://console.aws.amazon.com/lambda.
- On the Functions page, select a function. Choose Actions, and then choose Delete.
- Choose Delete.

To delete a Lambda function's log group:

- In the Amazon CloudWatch console, open the Log groups page.
- On the Log groups page, select the function's log group (/aws/lambda/http-crud-tutorial-function). Choose Actions, and then choose Delete log group.
- Choose Delete.

To delete a Lambda function's execution role:

- In the AWS Identity and Access Management console, open the Roles page.
- Select the function's role, for example, http-crud-tutorial-role.
- Choose Delete role.
- Choose Yes, delete.

Function and Class Components

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element. We call such components “function components” because they are literally JavaScript functions.

You can also use an ES6 class to define a component:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Rendering a Component

Previously, we only encountered React elements that represent DOM tags:

```
const element = <div />;
```

However, elements can also represent user-defined components:

```
const element = <Welcome name="Sara" />;
```

When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object “props”.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

References

https://aws.amazon.com/cdk/resources/

https://polaris.shopify.com/

https://polaris.shopify.com/components

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_appmesh.Backend.html

https://docs.aws.amazon.com/pdfs/cdk/v2/guide/awscdk.pdf#getting_started