AWS Cloud Development Kit

**SPL-DD-200-DVCDKP - Version 1.1.10**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

This lab introduces you to the fundamentals of the AWS Cloud Development Kit (AWS CDK). The AWS CDK is a software development framework from AWS with the purpose of making it easy to define cloud infrastructure in your favorite programming language and deploy it using AWS CloudFormation. In this lab, you write a “Hello, CDK!” AWS Lambda function and front it with an Amazon API Gateway endpoint so users can call it via an HTTP request.

Next, you are introduced to the concept of AWS CDK constructs, including writing your own construct. AWS CDK constructs allow you to bundle several infrastructure resources into reusable components. You can share these components for others to use in their applications.

Finally, you explore testing constructs and adding testing functionality to your application. In this lab, you use the AWS Cloud9 integrated development environment (IDE) for all application development tasks.

OBJECTIVES

By the end of this lab, you should be able to do the following:

* Create new AWS CDK applications.
* Define your application’s infrastructure using the AWS Construct Library.
* Deploy your AWS CDK applications to your AWS account.
* Define your own reusable constructs.
* Perform a test on a construct.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should have a basic knowledge of AWS services as defined in the AWS Cloud Practitioner Essentials course. You should be comfortable with navigating the AWS Management Console and running commands in a command line interface (CLI).

DURATION

This lab requires approximately *120* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.
* **File contents:** A code block that displays the contents of a script or file you need to run that has been pre-created for you.
* **Security:** An opportunity to incorporate security best practices.
* **Task complete:** A conclusion or summary point in the lab.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

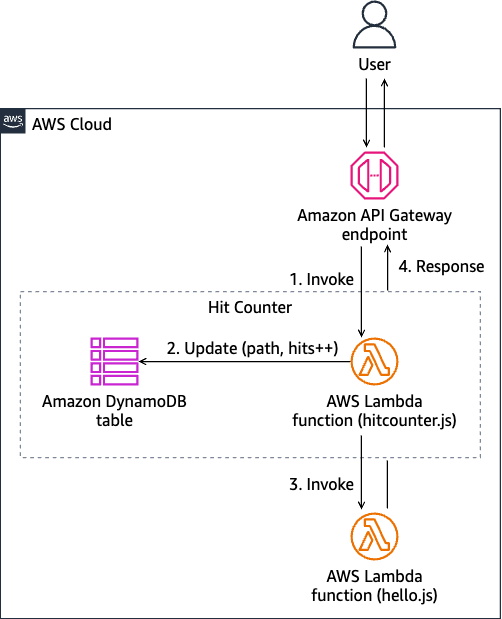
* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

LAB ENVIRONMENT

When you start the lab, the following resources are created for you:

* A virtual private cloud (VPC) with a public subnet.
* An AWS Cloud9 instance, which you use as an Integrated Development Environment (IDE) to run command line interface (CLI) commands and create and modify various files used throughout the lab.
* Several IAM roles and policies that are used by resources that you deploy, such as AWS Lambda functions.

By the end of the lab, you should have created the architecture that is shown in the following diagram:



*In the preceding diagram, a user visits an Amazon API Gateway endpoint URL, which invokes two Lambda functions. The first Lambda Function makes note of the URL path and records it in an Amazon DynamoDB table, with a counter that increases each time the same URL is visited. The second Lambda function is responsible for providing an HTTP response with a simple hello message, which is returned through the API Gateway endpoint to the user.*

SERVICES USED IN THIS LAB

**AWS Cloud Development Kit (AWS CDK)**

AWS CDK is an open source software development framework. You can use it to model and provision your cloud application resources using familiar programming languages. It provides you with high-level components that pre-configure cloud resources with proven defaults, so you can build cloud applications without needing to be an expert. With AWS CDK, you can provision your resources in a repeatable manner using AWS CloudFormation.

**Learn more**

To learn more about the AWS CDK, refer to *AWS Cloud Development Kit (AWS CDK)* in the **Additional resources** section at the ed of this lab.

**AWS Cloud9**

AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a web browser. It includes a code editor, debugger, and terminal. AWS Cloud9 comes prepackaged with essential tools for popular programming languages, including JavaScript, Python, PHP, and more. You do not need to install files or configure your development machine to start new projects.

**Learn more**

To learn more about AWS Cloud9, refer to *AWS Cloud9* in the **Additional resources** section at the ed of this lab.

**AWS Lambda**

With AWS Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to be invoked automatically by other AWS services or call it directly from any web or mobile app.

**Learn more**

To learn more about AWS Lambda, refer to *AWS Lambda* in the **Additional resources** section at the end of this lab.

**Amazon API Gateway**

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the “front door” for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication between applications. API Gateway supports containerized and serverless workloads, as well as web applications.

**Learn more**

To learn more about Amazon API Gateway, refer to *Amazon API Gateway* in the **Additional resources** section at the end of this lab.

**Amazon DynamoDB**

Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It is a fully managed, multi-region, durable database with built-in security, backup, and restore. It uses in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.

**Learn more**

To learn more about Amazon DynamoDB, refer to *Amazon DynamoDB* in the **Additional resources** section at the end of this lab.

**AWS CloudFormation**

AWS CloudFormation provides a common language for you to model and provision AWS and third-party application resources in your cloud environment.

**Learn more**

To learn more about AWS CloudFormation, refer to *AWS CloudFormation* in the **Additional resources** section at the end of this lab.

**AWS Identity and Access Management (IAM)**

With AWS Identity and Access Management (IAM), you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.

**Learn more**

To learn more about AWS IAM, refer to *AWS Identity and Access Management (IAM)* in the **Additional resources** section at the end of this lab.

AWS SERVICES NOT USED IN THIS LAB

The lab environment adheres to the [principle of least-privilege permissions](https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege). As such, the actions you can perform are limited to only those that are required to complete the lab tasks and learning objectives. Access to AWS services that are not needed to complete the lab tasks is restricted. Expect errors if you attempt to perform actions beyond those provided in the lab guide.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Connecting to and reviewing the AWS Cloud9 IDE instance**

Before you begin the lab, open the AWS Cloud9 IDE and review the interface to become familiar with its layout.

1. If you have not already done so, follow the steps in the **Start Lab** section to sign in to the AWS Management Console.
2. To open the **AWS Cloud9 console**, copy the **AwsCloud9Url** value that is listed to the left of these instructions, and then paste it into a new web browser tab.

**Note:** The AWS Cloud9 environment for this lab has already been created for you. It is named *Cloud9-Lab-IDE*.

The AWS Cloud9 IDE opens in a new web browser tab.

1. Familiarize yourself with the AWS Cloud9 IDE layout.

* The top-left section contains the various menu items.
* The file tree pane at the left of the page is where you can find the file structure of the default user’s home directory.
* Below the menu section is where you modify your files.
* Feel free to close the **Welcome** tab.
* The bottom section is the terminal session. This is where you run commands throughout the lab.

**Note:** To verify that the AWS Cloud9 environment is *ready*, check the file tree at the left of the page for folders named **cdk-primer** and **Lab-Is-Ready**. If you don’t see them, wait a few moments for the final configurations to complete.

**Task complete:** You have successfully connected to the AWS Cloud9 IDE.

**Task 2: Reviewing the empty AWS CDK project in your working directory**

In this task, you review the directories and files that you work with throughout this lab.

To save some time, the *cdk init -l typescript* command was run during the lab environment deployment process to create the empty TypeScript project in your working directory and update the configuration files required for AWS CDK V2.

 The following is an example of the standard output of the *cdk init -l typescript* command (it has already been run for you. You do not need to run this command):

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Applying project template app for typescript

# Welcome to your CDK TypeScript project!

This is a blank project for TypeScript development with CDK.

The `cdk.json` file tells the CDK Toolkit how to execute your app.

## Useful commands

\* `npm run build` compile typescript to js

\* `npm run watch` watch for changes and compile

\* `npm run test` perform the jest unit tests

\* `cdk deploy` deploy this stack to your default AWS account/region

\* `cdk diff` compare deployed stack with current state

\* `cdk synth` emits the synthesized CloudFormation template

Initializing a new git repository...

Executing npm install...

npm WARN deprecated source-map-url@0.4.1: See https://github.com/lydell/source-map-url#deprecated

npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated

npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated

npm WARN deprecated source-map-resolve@0.5.3: See https://github.com/lydell/source-map-resolve#deprecated

npm WARN deprecated sane@4.1.0: some dependency vulnerabilities fixed, support for node < 10 dropped, and newer ECMAScript syntax/features added

All done!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Newer version of CDK is available [2.15.0] \*\*\*

\*\*\* Information about upgrading from version 1.x to version 2.x is available here: https://docs.aws.amazon.com/cdk/v2/guide/migrating-v2.html \*\*\*

\*\*\* Upgrade recommended (npm install -g aws-cdk) \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The command creates the following files and subdirectories inside of the **cdk-primer** root directory:

* A hidden *.git* subdirectory and a hidden *.gitignore* file, which makes the project compatible with source control tools such as Git.
* A *bin* subdirectory, which includes a *cdk-primer.ts* file. This file contains the entry point for your AWS CDK app.
* A *lib* subdirectory, which includes a *cdk-primer-stack.ts* file. This file contains the code for your AWS CDK stack. This code is described in the next step in this procedure.
* A *node\_modules* subdirectory, which contains supporting code packages that the app and stack can use as needed.
* A *test* subdirectory, which includes a *cdk-primer-test.ts* file. This is used for Jest testing.
* A hidden *.npmignore* file, which lists the types of subdirectories and files that *npm* does not need when it builds the code.
* A *cdk.json* file, which contains information to make running the *cdk* command easier.
* A *jest.config.js* file, which is used for Jest testing.
* A *package-lock.json* file, which contains information that *npm* can use to reduce possible build and run errors.
* A *package.json* file, which contains information to make running the *npm* command easier and with possibly fewer build and run errors.
* A *README.md* file, which lists useful commands you can run with *npm* and the AWS CDK.
* A *tsconfig.json* file, which contains information to make running the *tsc* command easier and with possibly fewer build and run errors.

Next, review the main stack file, *cdk-primer/lib/cdk-primer-stack.ts*, which you work with in the coming tasks.

1. In the file tree pane, open the **cdk-primer/lib/cdk-primer-stack.ts** file.

**File contents:** The file contents should look similar to this:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

// import \* as sqs from 'aws-cdk-lib/aws-sqs';

export class CdkPrimerStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

// The code that defines your stack goes here

// example resource

// const queue = new sqs.Queue(this, 'CdkPrimerQueue', {

// visibilityTimeout: cdk.Duration.seconds(300)

// });

}

}

The *Stack*, *Construct*, and *StackProps* classes represent an AWS CloudFormation stack and its properties.

The *CdkPrimerStack* class represents the AWS CloudFormation stack for this application. It is currently empty. You add code to this stack as the lab progresses.

**Security:** Constructs automatically create required roles at construction time if you have not already specified a role for the construct to use. However, the construct only has permission to create roles in alignment with the AWS account in use. Permissions are added to the roles automatically through inline policies if you associate the construct with other constructs from the AWS Construct Library.

Depending on your security posture, this may not be allowed. To prevent constructs from updating your role’s policy, pass the object returned by *myRole.withoutPolicyUpdates()* instead of *myRole* itself.

**Learn more**

If there are IAM roles in your account that have already been created which you would like to use in your CDK application, you can use *Role.fromRoleArn* to import them. In this lab, you use pre-built roles that have the required policies and permissions for the Lambda functions to operate correctly.

To learn more, refer to *AWS Identity and Access Management Construct Library* in the **Addiitonal resources** section at the end of this lab.(https://docs.aws.amazon.com/cdk/api/latest/docs/aws-iam-readme.html).

**Task 3: Creating the Lambda directory and hello.js file**

In this task, you create a new directory named *lambda* and a new file within the *lambda* directory named *hello.js*. You then populate the *hello.js* file with code.

1. Select the **cdk-primer** directory, open the context (right-click) menu, and choose **New Folder**.

* Name the new folder

lambda

.

After creation, it should show up in-between the **bin** and **lib** folders.

1. Select the **lambda** directory, open the context (right-click) menu, and choose **New File**.

* Name the new file

hello.js

.

1. Open the **hello.js** file you just created and then copy and paste the following code snippet into the file:

exports.handler = async function(event) {

console.log("request:", JSON.stringify(event, undefined, 2));

return {

statusCode: 200,

headers: { "Content-Type": "text/plain" },

body: `Hello, CDK! You've hit ${event.path}\n`

};

};

1. To save your changes, on the **File** menu, choose **Save**.

The file you just created is a Lambda function that returns the text *Hello, CDK! You’ve hit [URL path]*. The function’s output also includes the HTTP status code and HTTP headers, which are used by API Gateway to formulate the HTTP response to the user.

1. Close the **hello.js** file.

**Task complete:** You have successfully created the directory and file that generate a Lambda function when AWS CDK deploys the CloudFormation stack.

**Task 4: Adding the Lambda function to your stack**

In this task, you modify the *cdk-primer/lib/cdk-primer-stack.ts* file in two steps. You add import statements in the first update (*TODO:1*), and then add the Lambda function in a second update (*TODO:2*).

1. Open the **cdk-primer/lib/cdk-primer-stack.ts** file.
2. Copy and paste only the code located between the **TODO:1** comments.

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

//BEGIN TODO:1 - Import the Lambda module

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

//END TODO:1

**Expected result:** The top portion of the file should now look similar to this:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

// import \* as sqs from 'aws-cdk-lib/aws-sqs';

Next, you add a *lambda.Function* construct to the *cdk-primer/lib/cdk-primer-stack.ts* file.

1. Copy and paste only the code located between the **TODO:2** comments.

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

export class CdkPrimerStack extends Stack {

constructor(scope: Construct, id: string, props?: StackProps) {

super(scope, id, props);

//BEGIN TODO:2 - Define a Lambda resource

const PrimerRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${cdk.Stack.of(this).account}:role/CDKPrimerHelloHandlerLambdaRole`,

{mutable: false},

);

const hello = new lambda.Function(this, 'HelloHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X, // run environment

code: lambda.Code.fromAsset('lambda'), // code loaded from "Lambda" directory

handler: 'hello.handler', // file is "hello", function is "handler"

role: PrimerRole,

description: 'aws:states:opt-out'

});

//END TODO:2

}

}

**Expected result:** The contents of the file should now look similar to this:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

// import \* as sqs from 'aws-cdk-lib/aws-sqs';

export class CdkPrimerStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

const PrimerRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${cdk.Stack.of(this).account}:role/CDKPrimerHelloHandlerLambdaRole`,

{mutable: false},

);

const hello = new lambda.Function(this, 'HelloHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X, // run environment

code: lambda.Code.fromAsset('lambda'), // code loaded from "Lambda" directory

handler: 'hello.handler', // file is "hello", function is "handler"

role: PrimerRole,

description: 'aws:states:opt-out'

});

// The code that defines your stack goes here

// example resource

// const queue = new sqs.Queue(this, 'CdkPrimerQueue', {

// visibilityTimeout: cdk.Duration.seconds(300)

// });

}

}

1. Save the changes to the **cdk-primer/lib/cdk-primer-stack.ts** file.

**Review of the code**

* You imported the Lambda and IAM modules.
* You created a construct to specify a pre-built role. This role has the *AWSLambdaBasicExecutionRole* policy attached to it for the Lambda function to work.
* The function uses the *NodeJS 20.x* runtime
* The handler code is loaded from the *lambda* directory which you created earlier. The path is relative to where you start the AWS CDK from, which is the project’s root directory
* The name of the handler function is *hello.handler* (*hello* is the name of the file and *handler* is the exported function name)
* The pre-built Lambda role is associated to the function with the *role* property.

**Task complete:** You have successfully added code to the *cdk-primer-stack.ts* file that instructs AWS CDK to create a Lambda function.

**Task 5: Bootstrapping the environment in the terminal**

**Consider:** Stacks that contain assets or large Lambda functions require special dedicated AWS CDK resources to be provisioned. AWS CDK only uses Amazon S3 buckets for this purpose. The *cdk bootstrap* command creates the necessary resources for you. You only need to bootstrap if you are deploying a stack that requires dedicated resources.

With the update to AWS CDK V2, the boot strapping process has changed. By default, AWS CDK now uses a pre-configured bootstrapping template to create the resources required to bootstrap successfully. To make this lab more secure, we have pre-created the infrastructure required for bootstrapping. Therefore, we have a customized template you use when running the bootstrapping command.

1. To change to the **/home/ec2-user/environment/cdk-primer** directory, run the following command:

cd ~/environment/cdk-primer

**Expected output:** None, unless there is an error.

1. To install the bootstrap stack with the customized template, run the following command:

cdk bootstrap --template customized-bootstrap-template.yaml

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using bootstrapping template from customized-bootstrap-template.yaml

Bootstrapping environment aws://111122223333/us-west-2...

Trusted accounts for deployment: (none)

Trusted accounts for lookup: (none)

Using default execution policy of 'arn:aws:iam::aws:policy/AdministratorAccess'. Pass '--cloudformation-execution-policies' to customize.

CDKToolkit: creating CloudFormation changeset...

Environment aws://111122223333/us-west-2 bootstrapped.

1. In the file tree pane, notice that a new directory named **cdk.out** was created. This directory is created as a result of the **cdk deploy** command, and is the directory where the synthesized templates are written to by default.

**Task complete:** If successful, the output displays that the stack bootstrapped without any errors.

**Task 6: Deploying the stack**

In this step, you use CloudFormation to deploy the stack.

If you want to see what resources are created in the stack you can run the *cdk* command with the *synthesize* action. It prints out the CloudFormation template that is used to create the resources in your terminal.

1. To preview the CloudFormation template, run the following command:

cdk synth

**Expected output:** The output displays the CloudFormation code that is created. To view an example of the output, expand the following **Command output example** section:

**Command output example**

1. To deploy the stack, run the following command:

cdk deploy

**Expected output:** The output should show that AWS CDK is creating various assets, including a CloudFormation stack, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Synthesis time: 16.43s

CdkPrimerStack: building assets...

[0%] start: Building 5dcceeae13d19ccc24fbf80b527b5cd04a655da5c0e8368ad045334917bfaa09:current\_account-current\_region

[0%] start: Building 1cc475c72ed9042cc353cb9ebf98647c34da78ffe4ded13565131d86419481f2:current\_account-current\_region

[50%] success: Built 5dcceeae13d19ccc24fbf80b527b5cd04a655da5c0e8368ad045334917bfaa09:current\_account-current\_region

[100%] success: Built 1cc475c72ed9042cc353cb9ebf98647c34da78ffe4ded13565131d86419481f2:current\_account-current\_region

CdkPrimerStack: assets built

CdkPrimerStack: deploying...

[0%] start: Publishing 5dcceeae13d19ccc24fbf80b527b5cd04a655da5c0e8368ad045334917bfaa09:current\_account-current\_region

[0%] start: Publishing 1cc475c72ed9042cc353cb9ebf98647c34da78ffe4ded13565131d86419481f2:current\_account-current\_region

[50%] success: Published 1cc475c72ed9042cc353cb9ebf98647c34da78ffe4ded13565131d86419481f2:current\_account-current\_region

[100%] success: Published 5dcceeae13d19ccc24fbf80b527b5cd04a655da5c0e8368ad045334917bfaa09:current\_account-current\_region

CdkPrimerStack: creating CloudFormation changeset...

CdkPrimerStack

Deployment time: 22.65s

Stack ARN:

arn:aws:cloudformation:us-west-2:111122223333:stack/CdkPrimerStack/25f70d10-65e4-11ed-acb2-022abc547f67

Total time: 39.08s

**Note:** The previous command compresses the CloudFormation template into a zip file, uploads the zip file to the Amazon S3 staging bucket that was created by the *cdk deploy* operation, and then creates a new CloudFormation stack that creates the resources specified in the template.

**Task complete:** You have successfully used AWS CDK and CloudFormation to deploy resources.

**Task 7: Reviewing stacks in the CloudFormation console**

AWS CDK apps are deployed through AWS CloudFormation. Each AWS CDK stack maps 1-to-1 with a CloudFormation stack, which means that you can use the AWS CloudFormation console to manage your stacks.

In this task, you review the stack and resources that the AWS CDK created.

1. Return to your web browser tab with the AWS Management Console. If it was closed, from the **AWS Cloud9 IDE**, choose the **AWS Cloud9 logo**, and then from the drop-down menu choose **Go To Your Dashboard**.

**Note:** You are returned to the **AWS Cloud9 Environments** page within the AWS Management Console.

1. At the top of the AWS Management Console, in the search bar, search for and choose

CloudFormation

.

1. On the **Stacks** page, review the list of stacks.

**Note:** If no stacks are listed, verify that the *Region selector* at the upper-right corner of the page matches the *labRegionName* value that is listed to the left of these instructions.

You should find the following stacks that the AWS CDK created:

* The **CDKToolkit** stack was created with the **cdk bootstrap** command and stores resources needed for its operation.
* The **CdkPrimerStack** was created with the **cdk deploy** command and deployed the Lambda function based on your construct.

**Note:** It is normal to see additional stacks, such as the AWS Cloud9 environment. They are created during the lab environment build process.

1. Choose the **CdkPrimerStack** link to view its details.
2. On the **CdkPrimerStack** page, choose the **Resources** tab.

You should see the **Logical ID** and **Physical ID** of your Lambda function resource.

**Learn more**

Did you notice the *CDKMetadata Logical ID* and wonder where it came from? To gain insight into how the AWS CDK is used, CDKMetadata collects and reports on the versions of libraries used by AWS CDK applications.

To learn more about the AWS::CDK::Metadata resource type, refer to *AWS::CDK::Metadata* in the **Additional resources** section at the end of this lab.

1. Choose the **Template** tab.

You should see the same CloudFormation template that was displayed in the terminal window when you ran the **cdk synthesize** command.

**Task complete:** You have successfully reviewed the resources deployed by AWS CDK.

**Task 8: Testing the Lambda function**

In this task, you test the Lambda function that you created with AWS CDK.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Lambda

.

1. On the **Functions** page, choose the link for the function name that starts with **CdkPrimerStack-HelloHandler**.
2. In the **Code source** section, notice that there is a **hello.js** file that contains the same construct of Lambda function code that you created in the AWS Cloud9 console and is now a Lambda function.
3. Choose the **Test** tab, and then do the following:

* For **Test event action**, choose **Create new event**.
* For **Event name**, enter

cdkHelloTest

.

* For **Event sharing settings**, choose **Private**.
* For **Template** choose **API Gateway AWS Proxy**.

1. Choose **Save**.

**Expected result:** A green *The test event cdkHelloTest was successfully saved* banner appears at the yop of the page.

1. Choose **Test**.

**Expected result:** An *Executing function: succeeded* message appears on the *Test* tab.

1. Below **Executing function: succeeded**, expand the **Details** section, which shows the results of the test.

**Expected output:** The output should show the response that the Lambda function returns, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{

"statusCode": 200,

"headers": {

"Content-Type": "text/plain"

},

"body": "Hello, CDK! You've hit /path/to/resource\n"

}

**Task complete:** You have successfully tested the Lambda function that was created using AWS CDK.

The next set of tasks are to add an API Gateway in front of your Lambda function. API Gateway exposes a public HTTP endpoint that anyone on the Internet can reach with an HTTP client, such as with the *curl* command or a web browser.

You use Lambda proxy integration mounted to the root of the API, which means that any request to any URL path is proxied directly to the Lambda function. The response from the function is returned back to the user.

**Task 9: Adding the LambdaRestApi construct to your stack**

In this task, you modify the *lib/cdk-primer-stack.ts* file to import the API Gateway module and add the LambdaRestApi construct to the stack. The changes are completed in two separate updates to the file via *TODO:3* for the first update and *TODO:4* for the second update.

1. Return to your web browser tab with the **AWS Cloud9** environment.
2. Open the **ckd-primer/lib/cdk-primer-stack.ts** file, if it is not open already.
3. To add an import statement for the APIGateway module at the beginning of the file, do the following:

* Copy and paste only the code between the **TODO:3** comments.

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

//BEGIN TODO:3 - Add API Gateway module

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

//TODO:3 End

**Expected result:** The top portion of the file should now look similar to this:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

// import \* as sqs from 'aws-cdk-lib/aws-sqs';

1. To add code to define an API endpoint and associate it with your Lambda function, do the following:

* Copy and paste only the code between the **TODO:4** comments.

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

export class CdkPrimerStack extends Stack {

constructor(scope: Construct, id: string, props?: StackProps) {

super(scope, id, props);

// The code that defines your stack goes here

const PrimerRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${Stack.of(this).account}:role/CDKPrimerHelloHandlerLambdaRole`,

{mutable: false},

);

const hello = new lambda.Function(this, 'HelloHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X, // run environment

code: lambda.Code.fromAsset('lambda'), // code loaded from "Lambda" directory

handler: 'hello.handler', // file is "hello", function is "handler"

role: PrimerRole,

description: 'aws:states:opt-out'

});

//BEGIN TODO:4 - Defines an API Gateway REST API resource backed by your "hello" function.

new apigw.LambdaRestApi(this, 'Endpoint', {

handler: hello,

cloudWatchRole: false

});

//END TODO:4

}

}

**Expected result:** The file contents should now look similar to this:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

// import \* as sqs from 'aws-cdk-lib/aws-sqs';

export class CdkPrimerStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

const PrimerRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${cdk.Stack.of(this).account}:role/CDKPrimerHelloHandlerLambdaRole`,

{mutable: false},

);

const hello = new lambda.Function(this, 'HelloHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X, // run environment

code: lambda.Code.fromAsset('lambda'), // code loaded from "Lambda" directory

handler: 'hello.handler', // file is "hello", function is "handler"

role: PrimerRole,

description: 'aws:states:opt-out'

});

new apigw.LambdaRestApi(this, 'Endpoint', {

handler: hello,

cloudWatchRole: false

});

// The code that defines your stack goes here

// example resource

// const queue = new sqs.Queue(this, 'CdkPrimerQueue', {

// visibilityTimeout: cdk.Duration.seconds(300)

// });

}

}

1. Save your changes to the **ckd-primer/lib/cdk-primer-stack.ts** file.

**Review of the code**

* With this update, you have defined an API Gateway which proxies all requests to the Lambda function.
* Adding the property *cloudWatchRole: false* keeps the AWS CDK from generating a role in the AWS CloudFormation template.

**Task complete:** You have successfully updated the *cdk-primer-stack.ts* file to include the API Gateway module and the *LambdaRestApi* construct.

**Task 10: Viewing upcoming changes**

In this task, you learn how to view the changes that will be implemented in your existing stack before actually deploying the changes.

1. To incorporate your changes from the cdk-primer-stack.ts file into the CloudFormation template, run the following command:

cdk diff

**Expected output:** The output shows that 10 new resources would be added to the CloudFormation stack if you deploy the updated changes, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Stack CdkPrimerStack

IAM Statement Changes

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

Resources

[+] AWS::ApiGateway::RestApi Endpoint EndpointEEF1FD8F

[+] AWS::ApiGateway::Deployment Endpoint/Deployment EndpointDeployment318525DA955b97b3bfc30c73b04bdea222bd23ab

[+] AWS::ApiGateway::Stage Endpoint/DeploymentStage.prod EndpointDeploymentStageprodB78BEEA0

[+] AWS::ApiGateway::Resource Endpoint/Default/{proxy+} Endpointproxy39E2174E

[+] AWS::Lambda::Permission Endpoint/Default/{proxy+}/ANY/ApiPermission.CdkPrimerStackEndpoint5AF22258.ANY..{proxy+} EndpointproxyANYApiPermissionCdkPrimerStackEndpoint5AF22258ANYproxyE9E3F32D

[+] AWS::Lambda::Permission Endpoint/Default/{proxy+}/ANY/ApiPermission.Test.CdkPrimerStackEndpoint5AF22258.ANY..{proxy+} EndpointproxyANYApiPermissionTestCdkPrimerStackEndpoint5AF22258ANYproxy25D8CDF9

[+] AWS::ApiGateway::Method Endpoint/Default/{proxy+}/ANY EndpointproxyANYC09721C5

[+] AWS::Lambda::Permission Endpoint/Default/ANY/ApiPermission.CdkPrimerStackEndpoint5AF22258.ANY.. EndpointANYApiPermissionCdkPrimerStackEndpoint5AF22258ANY94FCF28D

[+] AWS::Lambda::Permission Endpoint/Default/ANY/ApiPermission.Test.CdkPrimerStackEndpoint5AF22258.ANY.. EndpointANYApiPermissionTestCdkPrimerStackEndpoint5AF22258ANY7A6EEC4F

[+] AWS::ApiGateway::Method Endpoint/Default/ANY EndpointANY485C938B

Outputs

[+] Output Endpoint/Endpoint Endpoint8024A810: {"Value":{"Fn::Join":["",["https://",{"Ref":"EndpointEEF1FD8F"},".execute-api.",{"Ref":"AWS::Region"},".",{"Ref":"AWS::URLSuffix"},"/",{"Ref":"EndpointDeploymentStageprodB78BEEA0"},"/"]]}}

**Task complete:** You have successfully previewed the changes to the CloudFormation stack before deploying them with AWS CDK.

**Task 11: Deploying the stack**

In previous tasks, you made changes to files and reviewed the changes that will to be made to the stack using the *cdk diff* command. In this task, you run the command to deploy those changes to your stack.

1. To deploy your changes to the stack, run the following command:

cdk deploy

**Expected output:** The output displays a warning about IAM changes, with a confirmation prompt, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Synthesis time: 6.61s

This deployment will make potentially sensitive changes according to your current security approval level (--require-approval broadening).

Please confirm you intend to make the following modifications:

IAM Statement Changes

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

Do you wish to deploy these changes (y/n)?

**Note:** The output is warning you that deploying the stack entails some risk because the update makes IAM statement changes. For the app to work, you must allow these changes.

1. Enter

y

 and press **Enter** to deploy the stack and create the resources.

**Expected output:** After the deployment has completed, you should receive an **Outputs** message that includes the following details:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CdkPrimerStack: deploying...

current credentials could not be used to assume 'arn:aws:iam::111122223333:role/cdk-hnb659fds-deploy-role-111122223333-us-west-2', but are for the right account. Proceeding anyway.

[0%] start: Publishing 5dcceeae13d19ccc24fbf80b527b5cd04a655da5c0e8368ad045334917bfaa09:current\_account-current\_region

current credentials could not be used to assume 'arn:aws:iam::111122223333:role/cdk-hnb659fds-file-publishing-role-111122223333-us-west-2', but are for the right account. Proceeding anyway.

[50%] success: Published 5dcceeae13d19ccc24fbf80b527b5cd04a655da5c0e8368ad045334917bfaa09:current\_account-current\_region

[50%] start: Publishing da53485b46fc25786b0b6f375f2e310f43a9f4364da36c18baf17b0a79d0c138:current\_account-current\_region

[100%] success: Published da53485b46fc25786b0b6f375f2e310f43a9f4364da36c18baf17b0a79d0c138:current\_account-current\_region

CdkPrimerStack: creating CloudFormation changeset...

CdkPrimerStack

Deployment time: 58.66s

Outputs:

CdkPrimerStack.Endpoint8024A810 = https://w2oxtx2wnd.execute-api.us-west-2.amazonaws.com/prod/

Stack ARN:

arn:aws:cloudformation:us-west-2:111122223333:stack/CdkPrimerStack/875ab990-94c9-11ec-93a1-0a5bc91fee25

Total time: 74.29s

The *Outputs* section contains the the URL of the API Gateway endpoint, which is a CloudFormation stack output that is automatically added by the API Gateway construct.

1. Copy the **CdkPrimerStack.Endpoint** value from the output of the command, which is the URL of the API Gateway endpoint, and paste it into your favorite text editor. It looks similar to this:

*https://w2oxtx2wnd.execute-api.us-west-2.amazonaws.com/prod/*

**Task complete:** You have successfully deployed the changes to the CloudFormation stack with AWS CDK.

**Task 12: Testing your application**

In this task, you test connectivity to your application using the *curl* command and a web browser.

TEST USING THE CURL COMMAND

1. Copy the API Gateway URL from the **Outputs** section near the end of the *cdk deploy* command output, which starts with **CdkPrimerStack.Endpoint**.
2. To connect to the application using **curl**, run the following command:

* Replace the **API\_GATEWAY\_URL** placeholder value with the **API Gateway endpoint URL** that you copied to your text editor previously.

curl API\_GATEWAY\_URL

**Expected output:** The output displays the response from the Lambda function you created in a previous task, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hello, CKD! You've hit /

TEST USING A WEB BROWSER

1. Copy the **API Gateway endpoint URL** from your text editor and paste it into a new web browser tab.

**Expected result:** A webpage should load with a line of text that says the following:

Hello CDK! You've hit /

**Note:** If you received a *5xx* error from API Gateway, it is likely due to one of two issues:

* The response your function returned is not what API Gateway expects. [Go back and review the steps in Task 4 to make sure your handler returns a response that includes a statusCode, body and header fields](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-DD-200-DVCDKP-1%3A1.1.10-d5d3331b/en-US#handler_runtime).
* Your function failed for some reason. To learn why, open the *CloudWatch* console to review the most-recent log group for the string *errorMessage*.

**Task complete:** You have successfully verified that the application is responding to client requests.

Next, you write your own construct called *HitCounter*, which can be attached to any Lambda function that is used as an API Gateway backend. It counts how many requests were issued to each URL path and stores the number in an Amazon DynamoDB table.

**Note:** If you want to reuse the HitCounter construct, or make it available publicly, you can publish it to [NPM](https://npmjs.com/). You can also add it to the CDK Construct Catalog, an index of multi-language AWS CDK libraries, by including a keyword of *cdk* inside the *package.json* file before publishing.

**Task 13: Creating the HitCounter construct**

In this task, you create a new file in the *lib* directory named *hitcounter.ts* and populate it with code that creates a new construct for your application.

1. Open the context (right-click) menu for the **lib** folder, and then choose **New File**.

* Name the new file

hitcounter.ts

.

1. Open the **cdk-primer/lib/hitcounter.ts** file that you just created.
2. Copy the following code and paste it into the file:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

export interface HitCounterProps {

/\*\* the function for which you want to count url hits \*\*/

downstream: lambda.IFunction;

}

export class HitCounter extends Construct {

constructor(scope: Construct, id: string, props: HitCounterProps) {

super(scope, id);

}

}

1. Save your changes to the **cdk-primer/lib/hitcounter.ts** file.

**Review of the code**

* You declared a new construct class called *HitCounter*.
* The constructor arguments are *scope*, *id*, and *props*, and you propagate them to the *Construct* base class.
* The *props* argument is of type *HitCounterProps* which includes a single property *downstream* of type *lambda.IFunction*. This is where you are going to *plug in* the Lambda function you created in the previous tasks so it can be hit-counted.

1. Close the **hitcounter.ts** file.

**Task complete:** You have successfully created the *HitCounter* construct.

**Task 14: Creating the HitCounter Lambda handler**

In this task, you create a new file named *hitcounter.js* in the *cdk-primer/lambda* directory. You then add the handler code for the hit counter.

1. Open the context (right-click) menu for the **cdk-primer/lambda** folder, and then choose **New File**.

* Name the new file

hitcounter.js

.

1. Open the **cdk-primer/lambda/hitcounter.js** file you just created.
2. Copy and paste the following code into the file:

const { DynamoDBClient } = require('@aws-sdk/client-dynamodb');

const { DynamoDBDocumentClient, UpdateCommand } = require("@aws-sdk/lib-dynamodb");

const { InvokeCommand, LambdaClient, LogType } = require("@aws-sdk/client-lambda");

exports.handler = async function(event) {

console.log("request:", JSON.stringify(event, undefined, 2));

// create AWS SDK clients

const dynamo = new DynamoDBClient();

const docClient = DynamoDBDocumentClient.from(dynamo);

const lambdaClient = new LambdaClient({});

// update dynamo entry for "path" with hits++

const command = new UpdateCommand({

TableName: process.env.HITS\_TABLE\_NAME,

Key: { path: event.path },

UpdateExpression: 'ADD #hts :incr',

ExpressionAttributeNames:{'#hts' : 'hits'},

ExpressionAttributeValues: { ':incr': 1 }

})

const response = await docClient.send(command);

// call downstream function and capture response

const lambdaCommand = new InvokeCommand({

FunctionName: process.env.DOWNSTREAM\_FUNCTION\_NAME,

Payload: JSON.stringify(event),

LogType: LogType.Tail

});

const { Payload } = await lambdaClient.send(lambdaCommand);

const result = Buffer.from(Payload).toString();

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

// return response back to upstream caller

return JSON.parse(result);

};

1. Save your changes to the **cdk-primer/lambda/hitcounter.js** file.

**Review of the code**

Notice that this code relies on two environment variables:

* *HITS\_TABLE\_NAME* is the name of the DynamoDB table to use for storage.
* *DOWNSTREAM\_FUNCTION\_NAME* is the name of the downstream Lambda function.

Because the actual name of the table and the downstream function are decided when you deploy your app, you need to specify these values from your construct code. You do so in the next task.

1. Close the **hitcounter.js** file.

**Task complete:** You have successfully created the *HitCounter* construct.

**Task 15: Adding resources to the hit counter construct**

Next, you modify the *cdk-primer/lib/hitcounter.ts* file in three places:

* Import the DynamoDB module (*TODO:5*).
* Define the Lambda function (*TODO:6*).
* Add the DynamoDB table in your HitCounter construct (*TODO:7*).

**Note:** Copy and paste only the code in comments labeled **TODO:5**, **TODO:6**, and **TODO:7**.

1. Open the **cdk-primer/lib/hitcounter.ts** file.
2. To add import statements for the IAM and DynamoDB modules, copy and paste only the code between the **TODO:5** comments in the following code snippet:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

//BEGIN TODO:5

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

//END TODO:5

**Expected result:** The top portion of the file should look similar to this:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

1. To add a readonly handler for the Lambda function, copy and paste only the code between the **TODO:6** comments:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

export interface HitCounterProps {

/\*\* the function for which you want to count url hits \*\*/

downstream: lambda.IFunction;

}

export class HitCounter extends Construct {

//BEGIN TODO:6

/\*\* allows accessing the counter function \*/

public readonly handler: lambda.Function;

//END TODO:6

constructor(scope: Construct, id: string, props: HitCounterProps) {

super(scope, id);

}

}

**Expected result:** The file should now look similar to this:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

export interface HitCounterProps {

/\*\* the function for which you want to count url hits \*\*/

downstream: lambda.IFunction;

}

export class HitCounter extends Construct {

/\*\* allows accessing the counter function \*/

public readonly handler: lambda.Function;

constructor(scope: Construct, id: string, props: HitCounterProps) {

super(scope, id);

}

}

1. To create a Lambda function that writes to a DynamoDB table and uses a pre-built role, copy and paste only the code between the **TODO:7** comments:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

export interface HitCounterProps {

/\*\* the function for which you want to count url hits \*\*/

downstream: lambda.IFunction;

}

export class HitCounter extends Construct {

/\*\* allows accessing the counter function \*/

public readonly handler: lambda.Function;

constructor(scope: Construct, id: string, props: HitCounterProps) {

super(scope, id);

//BEGIN TODO:7

const PrimerHitCounterRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${Stack.of(this).account}:role/HelloHitCounterServiceRole`,

{mutable: false},

);

const table = new dynamodb.Table(this, 'Hits', {

partitionKey: { name: 'path', type: dynamodb.AttributeType.STRING }

});

this.handler = new lambda.Function(this, 'HitCounterHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X,

handler: 'hitcounter.handler',

role: PrimerHitCounterRole,

description: 'aws:states:opt-out',

code: lambda.Code.fromAsset('lambda'),

environment: {

DOWNSTREAM\_FUNCTION\_NAME: props.downstream.functionName,

HITS\_TABLE\_NAME: table.tableName

}

});

//END TODO:7

}

}

**Expected result:** The file contents should now look similar to this:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

export interface HitCounterProps {

/\*\* the function for which you want to count url hits \*\*/

downstream: lambda.IFunction;

}

export class HitCounter extends Construct {

/\*\* allows accessing the counter function \*/

public readonly handler: lambda.Function;

constructor(scope: Construct, id: string, props: HitCounterProps) {

super(scope, id);

const PrimerHitCounterRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${Stack.of(this).account}:role/HelloHitCounterServiceRole`,

{mutable: false},

);

const table = new dynamodb.Table(this, 'Hits', {

partitionKey: { name: 'path', type: dynamodb.AttributeType.STRING }

});

this.handler = new lambda.Function(this, 'HitCounterHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X,

handler: 'hitcounter.handler',

role: PrimerHitCounterRole,

description: 'aws:states:opt-out',

code: lambda.Code.fromAsset('lambda'),

environment: {

DOWNSTREAM\_FUNCTION\_NAME: props.downstream.functionName,

HITS\_TABLE\_NAME: table.tableName

}

});

}

}

1. Save your changes to the **cdk-primer/lib/hitcounter.ts** file.

**Review of the code**

* You added import statements for the IAM and DynamoDB modules.
* You added a construct which created a role named *PrimerHitCounterRole*.
  + This role is used by the *HitCounterHandler* function.
  + The role has the *AWSLambdaBasicExecutionRole* policy attached to it as well as a pre-built policy named *HelloHitCounterDefaultPolicy*. This policy grants the necessary *DynamoDB* permissions to read/write values to the table.
* You defined a DynamoDB table with *path* as the partition key.
* You defined a Lambda function which is bound to the *lambda/hitcounter.handler* code.
* You connected the Lambda’s environment variables to the *functionName* and *tableName* of your resources.

**Note:** The *functionName* and *tableName* properties are values that resolve when you deploy your stack. Notice that you haven’t configured these physical names when you defined the table or function, just logical IDs. This means that if you print their values during synthesis, you receive a token, which is how the AWS CDK represents these late-bound values. You should treat tokens as *opaque strings*, which means you can concatenate them together, for example, but don’t be tempted to parse them in your code.

**Task complete:** You have successfully imported the DynamoDB module, defined the Lambda function and added the DynamoDB table to your *HitCounter* construct.

**Task 16: Adding a hit counter to your stack**

In this task, you update the three parts of the *cdk-primer/lib/cdk-primer-stack.ts* file to include the hit counter. (*TODO:8*, *TODO:9*, and *TODO:10*)

1. Open the **cdk-primer/lib/cdk-primer-stack.ts** file.
2. To import the **HitCounter** to your file, copy and paste only the code between the **TODO:8** comments:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

//BEGIN TODO:8

import { HitCounter } from './hitcounter';

//END TODO:8

**Expected result:** The top portion of the file should now look similar to this:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

import { HitCounter } from './hitcounter';

// import \* as sqs from 'aws-cdk-lib/aws-sqs';

1. To add the **helloWithCounter** construct referencing the **HelloHitCounter** function to the file, copy and paste only the code between the **TODO:9** comments:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

import { HitCounter } from './hitcounter';

export class CdkPrimerStack extends Stack {

constructor(scope: App, id: string, props?: StackProps) {

super(scope, id, props);

const PrimerRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${Stack.of(this).account}:role/CDKPrimerHelloHandlerLambdaRole`,

{mutable: false},

);

const hello = new lambda.Function(this, 'HelloHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X,

code: lambda.Code.fromAsset('lambda'),

handler: 'hello.handler',

role:PrimerRole,

description: 'aws:states:opt-out'

});

//BEGIN TODO:9

const helloWithCounter = new HitCounter(this, 'HelloHitCounter', {

downstream: hello

});

//END TODO:9

// defines an API Gateway REST API resource backed by your "hello" function.

new apigw.LambdaRestApi(this, 'Endpoint', {

handler: hello,

cloudWatchRole: false

});

}

}

1. To update your API Gateway handler to use **helloWithCounter.handler** instead of **hello**, copy and paste only the code between the **TODO:10** comments:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

import { HitCounter } from './hitcounter';

export class CdkPrimerStack extends Stack {

constructor(scope: App, id: string, props?: StackProps) {

super(scope, id, props);

const PrimerRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${Stack.of(this).account}:role/CDKPrimerHelloHandlerLambdaRole`,

{mutable: false},

);

const hello = new lambda.Function(this, 'HelloHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X,

code: lambda.Code.fromAsset('lambda'),

handler: 'hello.handler',

role:PrimerRole,

description: 'aws:states:opt-out'

});

const helloWithCounter = new HitCounter(this, 'HelloHitCounter', {

downstream: hello

});

// defines an API Gateway REST API resource backed by your "hello" function.

new apigw.LambdaRestApi(this, 'Endpoint', {

//BEGIN TODO:10

handler: helloWithCounter.handler,

//END TODO:10

cloudWatchRole: false

});

}

}

**Expected results:** The file should now look similar to this:

import \* as cdk from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as DefaultStackSynthesizer from 'aws-cdk-lib';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as apigw from 'aws-cdk-lib/aws-apigateway';

import { HitCounter } from './hitcounter';

// import \* as sqs from 'aws-cdk-lib/aws-sqs';

export class CdkPrimerStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

const PrimerRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${cdk.Stack.of(this).account}:role/CDKPrimerHelloHandlerLambdaRole`,

{mutable: false},

);

const hello = new lambda.Function(this, 'HelloHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X, // run environment

code: lambda.Code.fromAsset('lambda'), // code loaded from "Lambda" directory

handler: 'hello.handler', // file is "hello", function is "handler"

role: PrimerRole,

description: 'aws:states:opt-out'

});

const helloWithCounter = new HitCounter(this, 'HelloHitCounter', {

downstream: hello

});

new apigw.LambdaRestApi(this, 'Endpoint', {

handler: helloWithCounter.handler,

cloudWatchRole: false

});

// The code that defines your stack goes here

// example resource

// const queue = new sqs.Queue(this, 'CdkPrimerQueue', {

// visibilityTimeout: cdk.Duration.seconds(300)

// });

}

}

1. Save your changes to the **cdk-primer/lib/cdk-primer-stack.ts** file.

**Review of the code**

* You imported the *HitCounter* function created in the *hitcounter.ts* file.
* You added a *helloWithCounter* construct referencing the *HelloHitCounter* function.
* You changed your API Gateway handler to *helloWithCounter.handler* instead of *hello*. This means that whenever your endpoint is hit, API Gateway will route the request to your hit counter handler, which will log the hit and relay it over to the hello function. Then, the responses will be relayed back in the reverse order all the way to the user.

**Task complete:** You have successfully added the hit counter to your stack.

**Task 17: Granting permissions to allow Lambda to read and write to DynamoDB**

In this task, you add permissions to the *cdk-primer/lib/hitcounter.ts* file. You grant *write* permissions to the HitCounter Lambda function for the DynamoDB table and *invoke* permissions to the Lambda role for the downstream function. The code to add is referenced by the *TODO:11* comments.

1. Open the **cdk-primer/lib/hitcounter.ts** file.
2. To add to add the appropriate permissions for the Lambda function, copy and paste only the code between the **TODO:11** comments to add to add the appropriate permissions for the Lambda function:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

export interface HitCounterProps {

/\*\* the function for which you want to count url hits \*\*/

downstream: lambda.Function;

}

export class HitCounter extends Construct {

/\*\* allows accessing the counter function \*/

public readonly handler: lambda.Function;

constructor(scope: Construct, id: string, props: HitCounterProps) {

super(scope, id);

const PrimerHitCounterRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${Stack.of(this).account}:role/HelloHitCounterServiceRole`,

{mutable: false},

);

const table = new dynamodb.Table(this, 'Hits', {

partitionKey: { name: 'path', type: dynamodb.AttributeType.STRING }

});

this.handler = new lambda.Function(this, 'HitCounterHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X,

handler: 'hitcounter.handler',

role: PrimerHitCounterRole,

description: 'aws:states:opt-out',

code: lambda.Code.fromAsset('lambda'),

environment: {

DOWNSTREAM\_FUNCTION\_NAME: props.downstream.functionName,

HITS\_TABLE\_NAME: table.tableName

}

});

//BEGIN TODO:11

// grant the Lambda role read/write permissions to your table

table.grantReadWriteData(this.handler);

// grant the Lambda role invoke permissions to the downstream function

props.downstream.grantInvoke(this.handler);

//END TODO:11

}

}

**Expected results:** The file should now look similar to this:

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

import \* as iam from 'aws-cdk-lib/aws-iam';

import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';

export interface HitCounterProps {

/\*\* the function for which you want to count url hits \*\*/

downstream: lambda.IFunction;

}

export class HitCounter extends Construct {

/\*\* allows accessing the counter function \*/

public readonly handler: lambda.Function;

constructor(scope: Construct, id: string, props: HitCounterProps) {

super(scope, id);

const PrimerHitCounterRole = iam.Role.fromRoleArn(

this,

'imported-role',

`arn:aws:iam::${Stack.of(this).account}:role/HelloHitCounterServiceRole`,

{mutable: false},

);

const table = new dynamodb.Table(this, 'Hits', {

partitionKey: { name: 'path', type: dynamodb.AttributeType.STRING }

});

this.handler = new lambda.Function(this, 'HitCounterHandler', {

runtime: lambda.Runtime.NODEJS\_20\_X,

handler: 'hitcounter.handler',

role: PrimerHitCounterRole,

description: 'aws:states:opt-out',

code: lambda.Code.fromAsset('lambda'),

environment: {

DOWNSTREAM\_FUNCTION\_NAME: props.downstream.functionName,

HITS\_TABLE\_NAME: table.tableName

}

});

// grant the Lambda role read/write permissions to your table

table.grantReadWriteData(this.handler);

// grant the Lambda role invoke permissions to the downstream function

props.downstream.grantInvoke(this.handler);

}

}

1. Save your changes to the **cdk-primer/lib/hitcounter.ts** file.

**Review of the code**

* The first permission grants the Lambda role (HitCounter) read/write permissions to your DynamoDB table.
* The second permission grants the Lambda role (HitCounter) permissions to invoke the downstream Lambda function.

**Task complete:** You have successfully granted *write* permissions to the HitCounter Lambda function for the DynamoDB table and *invoke* permissions to the Lambda role for the downstream function.

**Task 18: Deploying and testing updates to your stack**

In this task, you deploy the most recent changes to your stack, which causes the previous Lambda permissions to be removed and the new ones to be added.

1. To deploy the changes to your stack, run the following command:

cdk deploy

**Note:** Notice the **IAM Statement Changes**.

* Previous Lambda permissions to be removed are listed first with a minus sign [ - ] and listed in red text.
  + **4** existing statements are removed.
* New Lambda permissions to be added are listed second with a plus sign [ + ] and in green text.
  + **4** new statements are added.

**Expected output:** The output indicates that the stack update includes IAM statement changes and asks you to confirm the changes, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Synthesis time: 19.9s

CdkPrimerStack: building assets...

[0%] start: Building 1967a68f930e8a45e00391958bc85e16ad9ccbb07e2bb75a8dd5000a272b315b:current\_account-current\_region

[0%] start: Building 09a865b9af5bb346031f13b01d49a1af6bffbed06de6102cbfcad14d919c14be:current\_account-current\_region

[50%] success: Built 1967a68f930e8a45e00391958bc85e16ad9ccbb07e2bb75a8dd5000a272b315b:current\_account-current\_region

[100%] success: Built 09a865b9af5bb346031f13b01d49a1af6bffbed06de6102cbfcad14d919c14be:current\_account-current\_region

CdkPrimerStack: assets built

This deployment will make potentially sensitive changes according to your current security approval level (--require-approval broadening).

Please confirm you intend to make the following modifications:

IAM Statement Changes

[Table of changes listed here]

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

Do you wish to deploy these changes (y/n)?

1. Enter

y

 when prompted to accept the changes, and then press **Enter**.

**Expected output:** The output shows the status of the CloudFormation stack deployment, as wel as its outputs, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CdkPrimerStack: deploying...

[0%] start: Publishing 1967a68f930e8a45e00391958bc85e16ad9ccbb07e2bb75a8dd5000a272b315b:current\_account-current\_region

[0%] start: Publishing 09a865b9af5bb346031f13b01d49a1af6bffbed06de6102cbfcad14d919c14be:current\_account-current\_region

[50%] success: Published 1967a68f930e8a45e00391958bc85e16ad9ccbb07e2bb75a8dd5000a272b315b:current\_account-current\_region

[100%] success: Published 09a865b9af5bb346031f13b01d49a1af6bffbed06de6102cbfcad14d919c14be:current\_account-current\_region

CdkPrimerStack: creating CloudFormation changeset...

CdkPrimerStack

Deployment time: 84s

Outputs:

CdkPrimerStack.Endpoint8024A810 = https://ia94dzz6me.execute-api.us-west-2.amazonaws.com/prod/

Stack ARN:

arn:aws:cloudformation:us-west-2:111122223333:stack/CdkPrimerStack/25f70d10-65e4-11ed-acb2-022abc547f67

Total time: 103.9s

1. Copy the **CdkPrimerStack.Endpoint** URL from the **Outputs** section of the command output and paste it into your favorite text editor.
2. To test connectivity to the endpoint URL, run the following command:

* Replace the **OUTPUT\_URL** placeholder value with the **CdkPrimerStack.Endpoint URL** value you copied to your text editor.

curl -i OUTPUT\_URL

**Note:** You use the **-i** option to show HTTP response fields and status codes.

**Expected output:** The output displays many lines of information. The first line and the last line are the two you want to review.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HTTP/2 200

content-type: text/plain

content-length: 25

date: Wed, 23 Feb 2022 19:03:32 GMT

x-amzn-requestid: 1b9ad9e5-770f-4443-b944-1f69eed59323

x-amz-apigw-id: OAm4YFdlPHcFxSw=

x-amzn-trace-id: Root=1-62168502-16968cb836542bbe449abdee;Sampled=0

x-cache: Miss from cloudfront

via: 1.1 56c69262ecfa7873b40572ba8a323242.cloudfront.net (CloudFront)

x-amz-cf-pop: HIO50-C1

x-amz-cf-id: CNT3h\_XTZF7ygvb8d\_JvdYzbnyTDE1G5hL5dJr0fux4vFKhOoJOdqw==

Hello, CDK! You've hit /

* *HTTP/2 200* is equivalent to an *HTTP 200 Okay* message.
* *Hello, CDK! You’ve hit /* is the expected output.

1. Test a few more times in a web browser, and change the end of the URL to add entries to the hit counter. For example:

* OUTPUT\_URL/Dr-Pepper
* OUTPUT\_URL/Dr-Pepper-Zero
* OUTPUT\_URL/Cherry-Dr-Pepper

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hello, CDK! You've hit /Dr-Pepper

Hello, CDK! You've hit /Dr-Pepper-Zero

Hello, CDK! You've hit /Cherry-Dr-Pepper

**Task complete:** You have successfully created the HitCounter and it is writing to and reading from the DynamoDB table.

Next, you review the DynamoDB table to see what data the HitCounter has written to it.

1. Return to your web browser tab with the AWS Management Console.
2. At the top of the AWS Management Console, in the search bar, search for and choose

DynamoDB

.

1. In the navigation pane at the left of the page, choose **Tables**.
2. On the **Tables** page, choose the link for the table name that starts with **CdkPrimerStack-HelloHitCounter** to open the table.
3. Choose **Explore table items** to reveal a listing of paths and hits for each path.

**Expected result:** You should see table entries similar to the following:

| **path** | **hits** |
| --- | --- |
| /Cherry-Dr-Pepper | 1 |
| /Dr-Pepper-Zero | 1 |
| / | 2 |
| /Dr-Pepper | 1 |

1. Return to your web browser tab with the AWS Cloud9 IDE.
2. To connect to the API Gateway endpoint (OUTPUT\_URL) with a different path, run the following command:

* Replace the **OUTPUT\_URL** placeholder value with the **CdkPrimerStack.Endpoint URL** that you copied previously.

curl -i OUTPUT\_URL/ice-cold-Dr-Pepper

1. Return to your web browser tab with the DynamoDB **CdkPrimerStack-HelloHitCounterHits** table details page.
2. In the **Items returned** section, choose  refresh to update the list of items.
3. Verify the **/ice-cold-Dr-Pepper** entry was added successfully.

**Expected result:** You should see table entries similar to the following:

| **path** | **hits** |
| --- | --- |
| /Cherry-Dr-Pepper | 1 |
| /Dr-Pepper-Zero | 1 |
| / | 2 |
| /ice-cold-Dr-Pepper | 1 |
| /Dr-Pepper | 1 |

**Congratulations!** The **Hit Counter** is working correctly.

**Task 19: Testing constructs**

The [AWS CDK Developer Guide](https://docs.aws.amazon.com/cdk/latest/guide/testing.html) contains a guide for testing constructs. For this section of the lab, you use the *fine-grained assertions* type test.

* *Fine-grained assertions* test specific aspects of the generated AWS CloudFormation template, such as *this resource has this property with this value.* These tests help when you’re developing new features, since any code you add will cause your test to fail even if existing features still work. When this happens, your fine-grained tests reassure you that the existing functionality is unaffected.

AWS CDK ASSERT LIBRARY

You use the AWS CDK *assert (@aws-cdk/assert)* library throughout the following tasks. The library contains several helper functions for writing unit and integration tests.

**Note:** For this lab, you primarily use the *haveResource* function. This helper is used when you care that a resource of a particular type exists, regardless of its logical identifier, and that some properties are set to specific values.

**File contents:**

expect(stack).to(haveResource('AWS::CertificateManager::Certificate', {

DomainName: 'test.example.com',

// Note: some properties omitted here

ShouldNotExist: ABSENT

}));

*ABSENT* is a value to assert that a particular key in an object is not set (or set to *undefined*).

 For more information, refer to the [AWS CDK documentation](https://docs.aws.amazon.com/cdk/v2/guide/home.html).

FINE-GRAINED ASSERTION TESTS

This task assumes that you have created the HitCounter construct. The HitCounter construct creates a simple DynamoDB table. Now you install the *@aws-cdk/assert* module and create a test that validates that the table is getting created.

1. Return to your web browser tab with the AWS Cloud9 IDE.
2. Open the **cdk-primer/test/cdk-primer.test.ts** file.
3. Replace the current contents with the following code snippet:

import { Template } from 'aws-cdk-lib/assertions';

import { HitCounter } from '../lib/hitcounter';

import { Stack, App, StackProps } from 'aws-cdk-lib';

import { Construct } from 'constructs';

import \* as lambda from 'aws-cdk-lib/aws-lambda';

test('DynamoDB Table Created', () => {

const stack = new Stack();

// WHEN

new HitCounter(stack, 'MyTestConstruct', {

downstream: new lambda.Function(stack, 'TestFunction', {

runtime: lambda.Runtime.NODEJS\_20\_X,

handler: 'lambda.handler',

description: 'aws:states:opt-out',

code: lambda.Code.fromInline('test')

})

});

// THEN

Template.fromStack(stack).hasResource("AWS::DynamoDB::Table",{});

});

1. Save your changes to the **cdk-primer/test/cdk-primer.test.ts** file.

This test ensures that the synthesized stack includes a DynamoDB table.

1. To start the test, run the following command:

npx jest

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PASS test/cdk-primer.test.ts (19.124 s)

✓ DynamoDB Table Created (104 ms)

Test Suites: 1 passed, 1 total

Tests: 1 passed, 1 total

Snapshots: 0 total

Time: 19.236 s

Ran all test suites.

**Task complete:** You have successfully performed a fine-grained assertion test.

**Task 20: Cleaning up resources**

When you are done with resources you are no longer going to use, it is always best to remove those resources to keep from incurring charges. The AWS CDK has a specific way to do this.

1. To delete the **CdkPrimerStack** CloudFormation stack and all of its resources, run the following command:

cdk destroy

**Expected output:** The output asks you to verify that you want to delete *CdkPrimerStack*, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Are you sure you want to delete: CdkPrimerStack (y/n)?

1. Enter

y

 and then press **Enter**.

The stack, along with all resources in the stack, will be deleted.

**Expected output:** The output displays the resources that are currently being deleted. When finished, the output indicates that the stack has been destroyed, similar to this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CdkPrimerStack: destroying...

CdkPrimerStack: destroyed

**Task complete:** The stack and resources have been deleted.

**Conclusion**

You have successfully done the following:

* Created new AWS CDK applications.
* Defined your application’s infrastructure using the AWS Construct Library.
* Deployed your AWS CDK applications to your AWS account.
* Defined your own reusable constructs.
* Performed a test on a construct.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional resources**

For more information about the AWS CDK, here are a few things you can do from here:

* [AWS Cloud Development Kit (AWS CDK)](https://aws.amazon.com/cdk/)
* [AWS Cloud9](https://aws.amazon.com/cloud9/)
* [AWS Lambda](https://aws.amazon.com/lambda/)
* [Amazon API Gateway](https://aws.amazon.com/api-gateway/)
* [Amazon DynamoDB](https://aws.amazon.com/dynamodb/)
* [AWS CloudFormation](https://aws.amazon.com/cloudformation/)
* [AWS Identity and Access Management (IAM)](https://aws.amazon.com/iam/)
* [AWS Identity and Access Management Construct Library](https://docs.aws.amazon.com/cdk/api/latest/docs/aws-iam-readme.html)
* [AWS::CDK::Metadata](https://docs.aws.amazon.com/cdk/latest/guide/cli.html#version_reporting)
* Dive deeper into [AWS CDK Concepts](https://docs.aws.amazon.com/CDK/latest/userguide/concepts.html):
  + [Constructs](https://docs.aws.amazon.com/CDK/latest/userguide/constructs.html)
  + [Apps](https://docs.aws.amazon.com/cdk/latest/guide/apps.html)
  + [Stacks](https://docs.aws.amazon.com/cdk/latest/guide/stacks.html)
  + [Logical IDs](https://docs.aws.amazon.com/cdk/latest/guide/identifiers.html#identifiers_logical_ids)
  + [Environments](https://docs.aws.amazon.com/cdk/latest/guide/environments.html)
  + [Contexts](https://docs.aws.amazon.com/cdk/latest/guide/context.html)
  + [Assets](https://docs.aws.amazon.com/CDK/latest/userguide/assets.html)
* Explore the [AWS Construct Library](https://docs.aws.amazon.com/cdk/api/latest/docs/aws-construct-library.html), which already contains constructs for many AWS resources
* Read guidelines on how to [write your own constructs](https://docs.aws.amazon.com/cdk/latest/guide/constructs.html#constructs_author).
* Learn about [jsii](https://github.com/awslabs/jsii), the technology behind the AWS CDK’s multi-language support.
* Browse some [examples](https://github.com/aws-samples/aws-cdk-examples) on our GitHub repository