# Examples and Tutorials

Real-world examples and step-by-step tutorials for AWS CDK applications.

## 🎯 Featured Examples

### Serverless Applications

* [Serverless Example](serverless_example.md) - Complete serverless application walkthrough

## 📚 Tutorial Categories

### Web Applications

* Examples of web applications built with CDK
* Frontend and backend integration patterns
* Static site hosting with CDK

### Serverless Patterns

* [Serverless Example](serverless_example.md) - Lambda, API Gateway, DynamoDB
* Event-driven architectures
* Microservices with CDK

### Data and Analytics

* Data pipelines and processing
* Analytics and reporting solutions
* Real-time data streaming

### Container Applications

* ECS and EKS deployment patterns
* Containerized application examples
* Multi-service architectures

## 🚀 Getting Started with Examples

### For Beginners

1. Start with [Serverless Example](serverless_example.md) - comprehensive tutorial
2. Review the code and understand the patterns
3. Modify the example for your use case

### For Intermediate Users

* Explore multiple examples to understand different patterns
* Combine concepts from different examples
* Build upon existing examples

## 🔧 How to Use Examples

1. **Study the Code** - Understand the CDK constructs used
2. **Deploy and Test** - Run the examples in your AWS account
3. **Modify and Extend** - Adapt examples to your needs
4. **Learn Patterns** - Extract reusable patterns for your projects

## 📋 Prerequisites

Before working with examples: - Complete [Getting Started](../01-getting-started/) tutorials - Understand [Core Concepts](../02-core-concepts/) - Have AWS environment set up and bootstrapped

## 💡 Best Practices for Examples

* Always review costs before deploying examples
* Clean up resources after testing
* Adapt security settings for your requirements
* Use examples as learning tools, not production templates

## 🔗 Related Sections

* [Getting Started](../01-getting-started/) - Foundation tutorials
* [Development](../03-development/) - Best practices for building
* [Advanced](../05-advanced/) - Advanced patterns and techniques

## 🤝 Contributing Examples

Have a great CDK example to share? Consider contributing: - Ensure examples follow CDK best practices - Include clear documentation and README - Test examples thoroughly before sharing

include::attributes.txt[]

// Attributes

[.topic] [#configure-access-sso-example-cli] = Example: Authenticate with IAM Identity Center automatic token refresh for use with the {aws} CDK CLI :info\_titleabbrev: Example: Authenticate with IAM Identity Center automatic token refresh :keywords: {aws}, {aws} CDK, {aws} CDK CLI, Programmatic access, Security credentials, IAM, IAM Identity Center

== [abstract]

## In this example, we configure the {aws} Command Line Interface to authenticate our user with the {aws} IAM Identity Center token provider configuration. The SSO token provider configuration lets the {aws} CLI automatically retrieve refreshed authentication tokens to generate short-term credentials that we can use with the {aws} Cloud Development Kit ({aws} CDK) Command Line Interface ({aws} CDK CLI).

// Content start

In this example, we configure the {aws} Command Line Interface ({aws} CLI) to authenticate our user with the {aws} IAM Identity Center token provider configuration. The SSO token provider configuration lets the {aws} CLI automatically retrieve refreshed authentication tokens to generate short-term credentials that we can use with the {aws} Cloud Development Kit ({aws} CDK) Command Line Interface ({aws} CDK CLI).

[#configure-access-sso-example-cli-prerequisites] == Prerequisites

This example assumes that the following prerequisites have been completed:

* Prerequisites required to get set up with {aws} and install our starting CLI tools. For more information, see xref:configure-access-prerequisites[Prerequisites].
* IAM Identity Center has been set up by our organization as the method of managing users.
* At least one user has been created in IAM Identity Center.

[#configure-access-sso-example-cli-configure] == Step 1: Configure the {aws} CLI

For detailed instructions on this step, see https://docs.aws.amazon.com/cli/latest/userguide/sso-configure-profile-token.html[Configure the {aws} CLI to use IAM Identity Center token provider credentials with automatic authentication refresh] in the *{aws} Command Line Interface User Guide*.

We sign in to the {aws} access portal provided by our organization to gather our IAM Identity Center information. This includes the *SSO start URL* and *SSO Region*.

Next, we use the {aws} CLI aws configure sso command to configure an IAM Identity Center profile and sso-session on our local machine:

== [source,bash,subs=“verbatim,attributes”]

$ aws configure sso SSO session name (Recommended): ++++++SSO start URL [None]: <https://my-sso-portal.awsapps.com/start> SSO region [None]: ++++++SSO registration scopes [sso:account:access]: ++++++—-++++++++++++++++++

The {aws} CLI attempts to open our default browser to begin the login process for our IAM Identity Center account. If the {aws} CLI is unable to open our browser, instructions are provided to manually start the login process. This process associates the IAM Identity Center session with our current {aws} CLI session.

After establishing our session, the {aws} CLI displays the {aws} accounts available to us:

== [source,none,subs=“verbatim,attributes”]

There are 2 {aws} accounts available to you.

DeveloperAccount, developer-account-admin@example.com (<123456789011>) ProductionAccount, production-account-admin@example.com (<123456789022>) — \_\_\_\_

We use the arrow keys to select our *DeveloperAccount*.

Next, the {aws} CLI displays the IAM roles available to us from our selected account:

== [source,none,subs=“verbatim,attributes”]

Using the account ID <123456789011> There are 2 roles available to you.

ReadOnly FullAccess — \_\_\_\_

We use the arrow keys to select *FullAccess*.

Next, the {aws} CLI prompts us to complete configuration by specifying a default output format, default {aws} Region, and name for our profile:

== [source,none,subs=“verbatim,attributes”]

CLI default client Region [None]: ++++++++++++CLI default output format [None]: ++++++++++++CLI profile name [123456789011\_FullAccess]: ++++++++++++—-++++++++++++++++++++++++++++++++++++

The {aws} CLI displays a final message, showing how to use the named profile with the {aws} CLI:

== [source,none,subs=“verbatim,attributes”]

To use this profile, specify the profile name using –profile, as shown:

== aws s3 ls –profile ++++++++++++

After completing this step, our config file will look like the following:

== [source,none,subs=“verbatim,attributes”]

[profile ++++++] sso\_session = ++++++sso\_account\_id = <123456789011> sso\_role\_name = ++++++region = ++++++output = ++++++++++++++++++++++++++++++++++++

[sso-session ++++++] sso\_region = ++++++sso\_start\_url = <https://my-sso-portal.awsapps.com/start> sso\_registration\_scopes = —-++++++++++++

We can now use this sso-session and named profile to request security credentials.

[#configure-access-sso-example-cli-credentials] == Step 2: Use the {aws} CLI to generate security credentials

For detailed instructions on this step, see https://docs.aws.amazon.com/cli/latest/userguide/sso-using-profile.html[Use an IAM Identity Center named profile] in the *{aws} Command Line Interface User Guide*.

We use the {aws} CLI aws sso login command to request security credentials for our profile:

== [source,bash,subs=“verbatim,attributes”]

$ aws sso login –profile ++++++—-++++++

The {aws} CLI attempts to open our default browser and verifies our IAM log in. If we are not currently signed into IAM Identity Center, we will be prompted to complete the sign in process. If the {aws} CLI is unable to open our browser, instructions are provided to manually start the authorization process.

After successfully logging in, the {aws} CLI caches our IAM Identity Center session credentials. These credentials include an expiration timestamp. When they expire, the {aws} CLI will request that we sign in to IAM Identity Center again.

Using valid IAM Identity Center credentials, the {aws} CLI securely retrieves {aws} credentials for the IAM role specified in our profile. From here, we can use the {aws} CDK CLI with our credentials.

[#configure-access-sso-example-cli-cdk] == Step 3: Use the CDK CLI

With any CDK CLI command, we use the xref:ref-cli-cmd-options-profile[--profile] option to specify the named profile that we generated credentials for. If our credentials are valid, the CDK CLI will successfully perform the command. The following is an example:

== [source,none,subs=“verbatim,attributes”]

$ cdk diff –profile ++++++Stack CdkAppStack Hold on while we create a read-only change set to get a diff with accurate replacement information (use –no-change-set to use a less accurate but faster template-only diff) Resources [-] {aws}::S3::Bucket amzn-s3-demo-bucket amzn-s3-demo-bucket5AF9C99B destroy++++++

Outputs [-] Output BucketRegion: {“Value”:{“Ref”:“{aws}::Region”}}

== ✨ Number of stacks with differences: 1

When our credentials expire, an error message like the following will display:

== [source,none,subs=“verbatim,attributes”]

$ cdk diff –profile ++++++Stack CdkAppStack++++++

== Unable to resolve {aws} account to use. It must be either configured when you define your CDK Stack, or through the environment

To refresh our credentials, we use the {aws} CLI aws sso login command:

== [source,bash,subs=“verbatim,attributes”]

$ aws sso login –profile ++++++—-++++++

:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#ecs-example] = Example: Create an {aws} Fargate service using the {aws} CDK :info\_titleabbrev: Example: Create a Fargate service

== [abstract]

## Learn how to create an {aws} Fargate service.

// Content start

In this example, we show you how to create an {aws} Fargate service running on an Amazon Elastic Container Service (Amazon ECS) cluster that’s fronted by an internet-facing Application Load Balancer from an image on Amazon ECR.

Amazon ECS is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster. You can host your cluster on serverless infrastructure that’s managed by Amazon ECS by launching your services or tasks using the Fargate launch type. For more control, you can host your tasks on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances that you manage by using the Amazon EC2 launch type.

In this example, we launch some services using the Fargate launch type. If you’ve used the {aws} Management Console to create a Fargate service, you know that there are many steps to follow to accomplish that task. {aws} has several tutorials and documentation topics that walk you through creating a Fargate service, including:

* link:https://aws.amazon.com/getting-started/tutorials/deploy-docker-containers[How to Deploy Docker Containers - {aws}]
* link:https://docs.aws.amazon.com/AmazonECS/latest/developerguide/get-set-up-for-amazon-ecs.html[Setting Up with Amazon ECS]
* link:https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS\_GetStarted.html[Getting Started with Amazon ECS Using Fargate]

This example creates a similar Fargate service using the {aws} CDK.

The Amazon ECS construct used in this example helps you use {aws} services by providing the following benefits:

* Automatically configures a load balancer.
* Automatically opens a security group for load balancers. This enables load balancers to communicate with instances without having to explicitly create a security group.
* Automatically orders dependency between the service and the load balancer attaching to a target group, where the {aws} CDK enforces the correct order of creating the listener before an instance is created.
* Automatically configures user data on automatically scaling groups. This creates the correct configuration to associate a cluster to AMIs.
* Validates parameter combinations early. This exposes {aws} CloudFormation issues earlier, thus saving deployment time. For example, depending on the task, it’s easy to improperly configure the memory settings. Previously, we would not encounter an error until we deployed our app. But now the {aws} CDK can detect a misconfiguration and emit an error when we synthesize our app.
* Automatically adds permissions for Amazon Elastic Container Registry (Amazon ECR) if we use an image from Amazon ECR.
* Automatically scales. The {aws} CDK supplies a method so we can auto scale instances when we use an Amazon EC2 cluster. This happens automatically when we use an instance in a Fargate cluster.
* In addition, the {aws} CDK prevents an instance from being deleted when automatic scaling tries to stop an instance, but either a task is running or is scheduled on that instance.
* Previously, we had to create a Lambda function to have this functionality.
* Provides asset support, so that we can deploy a source from our machine to Amazon ECS in one step. Previously, to use an application source, we had to perform several manual steps, such as uploading to Amazon ECR and creating a Docker image.

= [IMPORTANT]

The ApplicationLoadBalancedFargateService constructs we’ll be using includes numerous {aws} components, some of which have non-trivial costs if left provisioned in our {aws} account, even if we don’t use them. Be sure to clean up (cdk destroy) if you follow along with this example.

====

[#ecs-example-initialize] == Create a CDK project

We start by creating a CDK project. This is a directory that stores our {aws} CDK code, including our CDK app.

==== [role=“tablist”] TypeScript:: + [source,none,subs=“verbatim,attributes”] — mkdir MyEcsConstruct cd MyEcsConstruct cdk init –language typescript —

JavaScript:: + [source,none,subs=“verbatim,attributes”] — mkdir MyEcsConstruct cd MyEcsConstruct cdk init –language javascript —

Python:: + [source,none,subs=“verbatim,attributes”] — mkdir MyEcsConstruct cd MyEcsConstruct cdk init –language python source .venv/bin/activate # On Windows, run ‘.’ instead pip install -r requirements.txt —

Java:: + [source,java,subs=“verbatim,attributes”] — mkdir MyEcsConstruct cd MyEcsConstruct cdk init –language java — + We may now import the Maven project into our IDE.

C#:: + [source,none,subs=“verbatim,attributes”] — mkdir MyEcsConstruct cd MyEcsConstruct cdk init –language csharp — + We may now open src/MyEcsConstruct.sln in Visual Studio. ====

Next, we run the app and confirm that it creates an empty stack.

== [source,none,subs=“verbatim,attributes”]

## cdk synth

[#ecs-example-create-fargate-service] == Create a Fargate service

There are two different ways that we can run our container tasks with Amazon ECS:

* Use the Fargate launch type, where Amazon ECS manages the physical machines that oour containers are running on for us.
* Use the EC2 launch type, where we do the managing, such as specifying automatic scaling.

For this example, we’ll create a Fargate service running on an Amazon ECS cluster, fronted by an internet-facing Application Load Balancer.

We add the following {aws} Construct Library module imports to our *stack file*:

==== [role=“tablist”] TypeScript:: File: lib/my\_ecs\_construct-stack.ts + [source,javascript,subs=“verbatim,attributes”] — import \* as ec2 from “aws-cdk-lib/aws-ec2”; import \* as ecs from “aws-cdk-lib/aws-ecs”; import \* as ecs\_patterns from “aws-cdk-lib/aws-ecs-patterns”; —

JavaScript:: File: lib/my\_ecs\_construct-stack.js + [source,javascript,subs=“verbatim,attributes”] — const ec2 = require(“aws-cdk-lib/aws-ec2”); const ecs = require(“aws-cdk-lib/aws-ecs”); const ecs\_patterns = require(“aws-cdk-lib/aws-ecs-patterns”); —

Python:: File: my\_ecs\_construct/my\_ecs\_construct\_stack.py + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import (aws\_ec2 as ec2, aws\_ecs as ecs, aws\_ecs\_patterns as ecs\_patterns) —

Java:: File: src/main/java/com/myorg/MyEcsConstructStack.java + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.services.ec2.\_; import software.amazon.awscdk.services.ecs.\_; import software.amazon.awscdk.services.ecs.patterns.\*; —

C#:: File: src/MyEcsConstruct/MyEcsConstructStack.cs + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK.{aws}.EC2; using Amazon.CDK.{aws}.ECS; using Amazon.CDK.{aws}.ECS.Patterns; — ====

Within our stack, we add the following code:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const vpc = new ec2.Vpc(this, “MyVpc”, { maxAzs: 3 // Default is all AZs in region });

…. const cluster = new ecs.Cluster(this, “MyCluster”, { vpc: vpc });

// Create a load-balanced Fargate service and make it public new ecs\_patterns.ApplicationLoadBalancedFargateService(this, “MyFargateService”, { cluster: cluster, // Required cpu: 512, // Default is 256 desiredCount: 6, // Default is 1 taskImageOptions: { image: ecs.ContainerImage.fromRegistry(“amazon/amazon-ecs-sample”) }, memoryLimitMiB: 2048, // Default is 512 publicLoadBalancer: true // Default is true }); —- ….

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const vpc = new ec2.Vpc(this, “MyVpc”, { maxAzs: 3 // Default is all AZs in region });

…. const cluster = new ecs.Cluster(this, “MyCluster”, { vpc: vpc });

// Create a load-balanced Fargate service and make it public new ecs\_patterns.ApplicationLoadBalancedFargateService(this, “MyFargateService”, { cluster: cluster, // Required cpu: 512, // Default is 256 desiredCount: 6, // Default is 1 taskImageOptions: { image: ecs.ContainerImage.fromRegistry(“amazon/amazon-ecs-sample”) }, memoryLimitMiB: 2048, // Default is 512 publicLoadBalancer: true // Default is true }); —- ….

Python:: + [source,python,subs=“verbatim,attributes”] — vpc = ec2.Vpc(self, “MyVpc”, max\_azs=3) # default is all AZs in region

…. cluster = ecs.Cluster(self, “MyCluster”, vpc=vpc)

ecs\_patterns.ApplicationLoadBalancedFargateService(self, "MyFargateService",  
 cluster=cluster, # Required  
 cpu=512, # Default is 256  
 desired\_count=6, # Default is 1  
 task\_image\_options=ecs\_patterns.ApplicationLoadBalancedTaskImageOptions(  
 image=ecs.ContainerImage.from\_registry("amazon/amazon-ecs-sample")),  
 memory\_limit\_mib=2048, # Default is 512  
 public\_load\_balancer=True) # Default is True ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — Vpc vpc = Vpc.Builder.create(this, “MyVpc”) .maxAzs(3) // Default is all AZs in region .build();

…. Cluster cluster = Cluster.Builder.create(this, “MyCluster”) .vpc(vpc).build();

// Create a load-balanced Fargate service and make it public  
ApplicationLoadBalancedFargateService.Builder.create(this, "MyFargateService")  
 .cluster(cluster) // Required  
 .cpu(512) // Default is 256  
 .desiredCount(6) // Default is 1  
 .taskImageOptions(  
 ApplicationLoadBalancedTaskImageOptions.builder()  
 .image(ContainerImage.fromRegistry("amazon/amazon-ecs-sample"))  
 .build())  
 .memoryLimitMiB(2048) // Default is 512  
 .publicLoadBalancer(true) // Default is true  
 .build(); ----

….

C#:: + [source,csharp,subs=“verbatim,attributes”] — var vpc = new Vpc(this, “MyVpc”, new VpcProps { MaxAzs = 3 // Default is all AZs in region });

…. var cluster = new Cluster(this, “MyCluster”, new ClusterProps { Vpc = vpc });

// Create a load-balanced Fargate service and make it public  
 new ApplicationLoadBalancedFargateService(this, "MyFargateService",  
 new ApplicationLoadBalancedFargateServiceProps  
 {  
 Cluster = cluster, // Required  
 DesiredCount = 6, // Default is 1  
 TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions  
 {  
 Image = ContainerImage.FromRegistry("amazon/amazon-ecs-sample")  
 },  
 MemoryLimitMiB = 2048, // Default is 256  
 PublicLoadBalancer = true // Default is true  
 }  
 ); ---- ====

….

Next, we validate our code by running the following to synthesize our stack:

== [source,none,subs=“verbatim,attributes”]

## cdk synth

The stack is hundreds of lines, so we won’t show it here. The stack should contain one default instance, a private subnet and a public subnet for the three Availability Zones, and a security group.

To deploy the stack, we run the following:

== [source,none,subs=“verbatim,attributes”]

## cdk deploy

{aws} CloudFormation displays information about the dozens of steps that it takes as it deploys our app.

Once deployment completes, we have successfully created a Fargate powered Amazon ECS service to run a Docker image.

[#ecs-example-destroy] == Clean up

As a general maintenance best practice, and to minimize unnecessary costs, we delete our stack when complete:

== [source,none,subs=“verbatim,attributes”]

## cdk destroy

:doctype: book

include::attributes.txt[]

// Attributes

:https–docs-aws-amazon-com-cdk-api-v2-docs-aws-cdk-lib-aws-apigateway-readme-html: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_apigateway-readme.html :https–docs-aws-amazon-com-cdk-api-v2-docs-aws-cdk-lib-aws-lambda-readme-html: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda-readme.html

[.topic] [#serverless-example] = Tutorial: Create a serverless Hello World application :info\_titleabbrev: Tutorial: Serverless Hello World application :info\_abstract: In this tutorial, you use the {aws} CDK to create a simple serverless Hello World application that implements a basic API backend.

== [abstract]

## In this tutorial, you use the {aws} CDK to create a simple serverless [.noloc]Hello World application that implements a basic API backend.

// Content start

In this tutorial, you use the {aws} Cloud Development Kit ({aws} CDK) to create a simple serverless Hello World application that implements a basic API backend consisting of the following:

* *Amazon API Gateway REST API* – Provides an HTTP endpoint that is used to invoke your function through an HTTP GET request.
* *{aws} Lambda function* – Function that returns a Hello World! message when invoked with the HTTP endpoint.
* *Integrations and permissions* – Configuration details and permissions for your resources to interact with one another and perform actions, such as writing logs to Amazon CloudWatch.

The following diagram shows the components of this application:

image::./images/serverless-example-01.png[Diagram of a Lambda function that is invoked when you send a GET request to the API Gateway endpoint.,scaledwidth=100%]

For this tutorial, you will create and interact with your application in the following steps:

. Create an {aws} CDK project. . Define a Lambda function and API Gateway REST API using L2 constructs from the {aws} Construct Library. . Deploy your application to the {aws} Cloud. . Interact with your application in the {aws} Cloud. . Delete the sample application from the {aws} Cloud.

[#serverless-example-pre] == Prerequisites

Before starting this tutorial, complete the following:

* Create an {aws} account and have the {aws} Command Line Interface ({aws} CLI) installed and configured.
* Install Node.js and npm.
* Install the CDK Toolkit globally, using npm install -g aws-cdk.

For more information, see xref:getting-started[Getting started with the {aws} CDK].

We also recommend a basic understanding of the following:

* xref:home[What is the {aws} CDK?] for a basic introduction to the {aws} CDK.
* xref:core-concepts[Learn {aws} CDK core concepts] for an overview of core concepts of the {aws} CDK.

[#serverless-example-project] == Step 1: Create a CDK project

In this step, you create a new CDK project using the {aws} CDK CLI cdk init command.

*To create a CDK project*:: + . From a starting directory of your choice, create and navigate to a project directory named cdk-hello-world on your machine: + [source,none,subs=“verbatim,attributes”] — $ mkdir cdk-hello-world && cd cdk-hello-world — + . Use the cdk init command to create a new project in your preferred programming language: + ==== [role=“tablist”] TypeScript::: + [source,none,subs=“verbatim,attributes”] — $ cdk init –language typescript — + Install {aws} CDK libraries: + [source,none,subs=“verbatim,attributes”] — $ npm install aws-cdk-lib constructs —

JavaScript::: + [source,none,subs=“verbatim,attributes”] — $ cdk init –language javascript — + Install {aws} CDK libraries: + [source,none,subs=“verbatim,attributes”] — $ npm install aws-cdk-lib constructs —

Python::: + [source,none,subs=“verbatim,attributes”] — $ cdk init –language python — + Activate the virtual environment: + [source,none,subs=“verbatim,attributes”] — $ source .venv/bin/activate # On Windows, run ‘.’ instead — + Install {aws} CDK libraries and project dependencies: + [source,none,subs=“verbatim,attributes”] — (.venv)$ python3 -m pip install -r requirements.txt —

Java::: + [source,none,subs=“verbatim,attributes”] — $ cdk init –language java — + Install {aws} CDK libraries and project dependencies: + [source,none,subs=“verbatim,attributes”] — $ mvn package —

C#::: + [source,none,subs=“verbatim,attributes”] — $ cdk init –language csharp — + Install {aws} CDK libraries and project dependencies: + [source,none,subs=“verbatim,attributes”] — $ dotnet restore src —

Go::: + [source,none,subs=“verbatim,attributes”] — $ cdk init –language go — + Install project dependencies: + [source,none,subs=“verbatim,attributes”] — $ go get github.com/aws/aws-cdk-go/awscdk/v2 $ go get github.com/aws/aws-cdk-go/awscdk/v2/awslambda $ go get github.com/aws/aws-cdk-go/awscdk/v2/awsapigateway $ go mod tidy — ==== +

The CDK CLI creates a project with the following structure: + ==== [role=“tablist”] TypeScript::: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world ├── .git ├── .gitignore ├── .npmignore ├── README.md ├── bin │ └── cdk-hello-world.ts ├── cdk.json ├── jest.config.js ├── lib │ └── cdk-hello-world-stack.ts ├── node\_modules ├── package-lock.json ├── package.json ├── test │ └── cdk-hello-world.test.ts └── tsconfig.json —

JavaScript::: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world ├── .git ├── .gitignore ├── .npmignore ├── README.md ├── bin │ └── cdk-hello-world.js ├── cdk.json ├── jest.config.js ├── lib │ └── cdk-hello-world-stack.js ├── node\_modules ├── package-lock.json ├── package.json └── test └── cdk-hello-world.test.js —

Python::: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world ├── .git ├── .gitignore ├── .venv ├── README.md ├── app.py ├── cdk.json ├── cdk\_hello\_world │ ├── *init*.py │ └── cdk\_hello\_world\_stack.py ├── requirements-dev.txt ├── requirements.txt ├── source.bat └── tests —

Java::: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world ├── .git ├── .gitignore ├── README.md ├── cdk.json ├── pom.xml ├── src │ ├── main │ │ └── java │ │ └── com │ │ └── myorg │ │ ├── CdkHelloWorldApp.java │ │ └── CdkHelloWorldStack.java └── target —

C#::: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world ├── .git ├── .gitignore ├── README.md ├── cdk.json └── src ├── CdkHelloWorld │ ├── CdkHelloWorld.csproj │ ├── CdkHelloWorldStack.cs │ ├── GlobalSuppressions.cs │ └── Program.cs └── CdkHelloWorld.sln —

Go::: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world ├── .git ├── .gitignore ├── README.md ├── cdk-hello-world.go ├── cdk-hello-world\_test.go ├── cdk.json ├── go.mod └── go.sum — ====

The CDK CLI automatically creates a CDK app that contains a single stack. The CDK app instance is created from the +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.App.html[App]+ class. The following is a portion of your CDK application file:

==== [role=“tablist”] TypeScript:: Located in bin/cdk-hello-world.ts: + [source,javascript,subs=“verbatim,attributes”] — #!/usr/bin/env node import ‘source-map-support/register’; import \* as cdk from ‘aws-cdk-lib’; import { CdkHelloWorldStack } from ‘../lib/cdk-hello-world-stack’;

const app = new cdk.App(); new CdkHelloWorldStack(app, ‘CdkHelloWorldStack’, { }); —

JavaScript:: Located in bin/cdk-hello-world.js: + [source,javascript,subs=“verbatim,attributes”] — #!/usr/bin/env node const cdk = require(‘aws-cdk-lib’); const { CdkHelloWorldStack } = require(‘../lib/cdk-hello-world-stack’); const app = new cdk.App(); new CdkHelloWorldStack(app, ‘CdkHelloWorldStack’, { }); —

Python:: Located in app.py: + [source,python,subs=“verbatim,attributes”] — #!/usr/bin/env python3 import os import aws\_cdk as cdk from cdk\_hello\_world.cdk\_hello\_world\_stack import CdkHelloWorldStack

app = cdk.App() CdkHelloWorldStack(app, “CdkHelloWorldStack”,) app.synth() —

Java:: Located in +src/main/java/.../CdkHelloWorldApp.java+: + [source,java,subs=“verbatim,attributes”] — package com.myorg;

import software.amazon.awscdk.App; import software.amazon.awscdk.Environment; import software.amazon.awscdk.StackProps;

import java.util.Arrays;

public class JavaApp { public static void main(final String[] args) { App app = new App();

…. new JavaStack(app, “JavaStack”, StackProps.builder() .build());

app.synth();

} } —- ….

C#:: Located in src/CdkHelloWorld/Program.cs: + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using System; using System.Collections.Generic; using System.Linq;

namespace CdkHelloWorld { sealed class Program { public static void Main(string[] args) { var app = new App(); new CdkHelloWorldStack(app, “CdkHelloWorldStack”, new StackProps {

});  
 app.Synth();  
 }

} } —-

Go:: Located in cdk-hello-world.go: + [source,go,subs=“verbatim,attributes”] — package main import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/constructs-go/constructs/v10” “github.com/aws/jsii-runtime-go” )

// …

func main() { defer jsii.Close() app := awscdk.NewApp(nil) NewCdkHelloWorldStack(app, “CdkHelloWorldStack”, &CdkHelloWorldStackProps{ awscdk.StackProps{ Env: env(), }, }) app.Synth(nil) }

func env() \*awscdk.Environment { return nil } — ====

[#serverless-example-function] == Step 2: Create your Lambda function

Within your CDK project, create a lambda directory that includes a new hello.js file. The following is an example:

==== [role=“tablist”] TypeScript:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ mkdir lambda && cd lambda $ touch hello.js — + The following should now be added to your CDK project: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world └── lambda └── hello.js —

JavaScript:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ mkdir lambda && cd lambda $ touch hello.js — + The following should now be added to your CDK project: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world └── lambda └── hello.js —

Python:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ mkdir lambda && cd lambda $ touch hello.js — + The following should now be added to your CDK project: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world └── lambda └── hello.js —

Java:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ mkdir -p src/main/resources/lambda $ cd src/main/resources/lambda $ touch hello.js — + The following should now be added to your CDK project: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world └── src └── main └──resources └──lambda └──hello.js —

C#:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ mkdir lambda && cd lambda $ touch hello.js — + The following should now be added to your CDK project: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world └── lambda └── hello.js —

Go:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ mkdir lambda && cd lambda $ touch hello.js — + The following should now be added to your CDK project: + [source,none,subs=“verbatim,attributes”] — cdk-hello-world └── lambda └── hello.js — ====

= [NOTE]

To keep this tutorial simple, we use a JavaScript Lambda function for all CDK programming languages.

====

Define your Lambda function by adding the following to the newly created file:

== [source,javascript,subs=“verbatim,attributes”]

exports.handler = async (event) => { return { statusCode: 200, headers: { “Content-Type”: “text/plain” }, body: JSON.stringify({ message: “Hello, World!” }), }; }; —

[#serverless-example-constructs] == Step 3: Define your constructs

In this step, you will define your Lambda and API Gateway resources using {aws} CDK L2 constructs.

Open the project file that defines your CDK stack. You will modify this file to define your constructs. The following is an example of your starting stack file:

==== [role=“tablist”] TypeScript:: Located in lib/cdk-hello-world-stack.ts: + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘aws-cdk-lib’; import { Construct } from ‘constructs’;

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

// Your constructs will go here

} } —

JavaScript:: Located in lib/cdk-hello-world-stack.js: + [source,javascript,subs=“verbatim,attributes”] — const { Stack, Duration } = require(‘aws-cdk-lib’); const lambda = require(‘aws-cdk-lib/aws-lambda’); const apigateway = require(‘aws-cdk-lib/aws-apigateway’);

class CdkHelloWorldStack extends Stack {

constructor(scope, id, props) { super(scope, id, props);

// Your constructs will go here

} }

== module.exports = { CdkHelloWorldStack }

Python:: Located in cdk\_hello\_world/cdk\_hello\_world\_stack.py: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import Stack from constructs import Construct

class CdkHelloWorldStack(Stack): def *init*(self, scope: Construct, construct\_id: str, **kwargs) -> None: super().*init*(scope, construct\_id,** kwargs)

// Your constructs will go here ----

Java:: Located in +src/main/java/.../CdkHelloWorldStack.java+: + [source,java,subs=“verbatim,attributes”] — package com.myorg;

import software.constructs.Construct; import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps;

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

…. public CdkHelloWorldStack(final Construct scope, final String id, final StackProps props) { super(scope, id, props);

// Your constructs will go here

} } —- ….

C#:: Located in src/CdkHelloWorld/CdkHelloWorldStack.cs: + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Constructs;

namespace CdkHelloWorld { public class CdkHelloWorldStack : Stack { internal CdkHelloWorldStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Your constructs will go here } } } —

Go:: Located at cdk-hello-world.go: + [source,go,subs=“verbatim,attributes”] — package main

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

type CdkHelloWorldStackProps struct { awscdk.StackProps }

func NewCdkHelloWorldStack(scope constructs.Construct, id string, props \*CdkHelloWorldStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. // Your constructs will go here

return stack } ….

func main() {

// …

}

func env() \*awscdk.Environment {

return nil

== }

====

In this file, the {aws} CDK is doing the following:

* Your CDK stack instance is instantiated from the +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html[Stack]+ class.
* The +link:https://docs.aws.amazon.com/cdk/api/v2/docs/constructs-readme.html[Constructs]+ base class is imported and provided as the scope or parent of your stack instance.

[#serverless-example-constructs-lambda] === Define your Lambda function resource

To define your Lambda function resource, you import and use the +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda-readme.html[aws-lambda]+ L2 construct from the {aws} Construct Library.

Modify your stack file as follows:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘aws-cdk-lib’; import { Construct } from ‘constructs’; // Import Lambda L2 construct import \* as lambda from ‘aws-cdk-lib/aws-lambda’;

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

// Define the Lambda function resource const helloWorldFunction = new lambda.Function(this, ‘HelloWorldFunction’, { runtime: lambda.Runtime.NODEJS\_20\_X, // Choose any supported Node.js runtime code: lambda.Code.fromAsset(‘lambda’), // Points to the lambda directory handler: ‘hello.handler’, // Points to the ‘hello’ file in the lambda directory }); } } —-

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const { Stack, Duration } = require(‘aws-cdk-lib’); // Import Lambda L2 construct const lambda = require(‘aws-cdk-lib/aws-lambda’);

class CdkHelloWorldStack extends Stack { constructor(scope, id, props) { super(scope, id, props);

// Define the Lambda function resource const helloWorldFunction = new lambda.Function(this, ‘HelloWorldFunction’, { runtime: lambda.Runtime.NODEJS\_20\_X, // Choose any supported Node.js runtime code: lambda.Code.fromAsset(‘lambda’), // Points to the lambda directory handler: ‘hello.handler’, // Points to the ‘hello’ file in the lambda directory }); } }

== module.exports = { CdkHelloWorldStack }

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import ( Stack, # Import Lambda L2 construct aws\_lambda as \_lambda, )

= …

class CdkHelloWorldStack(Stack):

…. def **init**(self, scope: Construct, construct\_id: str, \*\*kwargs) -> None: super().\_\_init\_\_(scope, construct\_id, \*\*kwargs)

# Define the Lambda function resource  
hello\_world\_function = \_lambda.Function(  
 self,  
 "HelloWorldFunction",  
 runtime = \_lambda.Runtime.NODEJS\_20\_X, # Choose any supported Node.js runtime  
 code = \_lambda.Code.from\_asset("lambda"), # Points to the lambda directory  
 handler = "hello.handler", # Points to the 'hello' file in the lambda directory  
) ---- + NOTE: We import the `aws\_lambda` module as `\\_lambda` because `lambda` is a build-in identifier in Python.

….

Java:: + [source,java,subs=“verbatim,attributes”] — // … // Import Lambda L2 construct import software.amazon.awscdk.services.lambda.Code; import software.amazon.awscdk.services.lambda.Function; import software.amazon.awscdk.services.lambda.Runtime;

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

…. public CdkHelloWorldStack(final Construct scope, final String id, final StackProps props) { super(scope, id, props);

// Define the Lambda function resource  
Function helloWorldFunction = Function.Builder.create(this, "HelloWorldFunction")  
 .runtime(Runtime.NODEJS\_20\_X) // Choose any supported Node.js runtime  
 .code(Code.fromAsset("src/main/resources/lambda")) // Points to the lambda directory  
 .handler("hello.handler") // Points to the 'hello' file in the lambda directory  
 .build();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — // … // Import Lambda L2 construct using Amazon.CDK.{aws}.Lambda;

namespace CdkHelloWorld { public class CdkHelloWorldStack : Stack { internal CdkHelloWorldStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Define the Lambda function resource var helloWorldFunction = new Function(this, “HelloWorldFunction”, new FunctionProps { Runtime = Runtime.NODEJS\_20\_X, // Choose any supported Node.js runtime Code = Code.FromAsset(“lambda”), // Points to the lambda directory Handler = “hello.handler” // Points to the ‘hello’ file in the lambda directory }); } } } —

Go:: + [source,go,subs=“verbatim,attributes”] — package main

import ( // … // Import Lambda L2 construct “github.com/aws/aws-cdk-go/awscdk/v2/awslambda” // Import S3 assets construct “github.com/aws/aws-cdk-go/awscdk/v2/awss3assets” // … )

// …

func NewCdkHelloWorldStack(scope constructs.Construct, id string, props \*CdkHelloWorldStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. // Define the Lambda function resource helloWorldFunction := awslambda.NewFunction(stack, jsii.String(“HelloWorldFunction”), &awslambda.FunctionProps{ Runtime: awslambda.Runtime\_NODEJS\_20\_X(), // Choose any supported Node.js runtime Code: awslambda.Code\_FromAsset(jsii.String(“lambda”), &awss3assets.AssetOptions{}), // Points to the lambda directory Handler: jsii.String(“hello.handler”), // Points to the ‘hello’ file in the lambda directory })

return stack } ….

== // …

====

Here, you create a Lambda function resource and define the following properties:

* runtime – The environment the function runs in. Here, we use Node.js version 20.x.
* code – The path to the function code on your local machine.
* handler – The name of the specific file that contains your function code.

[#serverless-example-constructs-api] === Define your API Gateway REST API resource

To define your API Gateway [.noloc]REST API resource, you import and use the +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_apigateway-readme.html[aws-apigateway]+ L2 construct from the {aws} Construct Library.

Modify your stack file as follows:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // … //Import API Gateway L2 construct import \* as apigateway from ‘aws-cdk-lib/aws-apigateway’;

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

…. // …

// Define the API Gateway resource const api = new apigateway.LambdaRestApi(this, ‘HelloWorldApi’, { handler: helloWorldFunction, proxy: false, });

// Define the ‘/hello’ resource with a GET method const helloResource = api.root.addResource(‘hello’); helloResource.addMethod(‘GET’); } } —- ….

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // … // Import API Gateway L2 construct const apigateway = require(‘aws-cdk-lib/aws-apigateway’);

class CdkHelloWorldStack extends Stack { constructor(scope, id, props) { super(scope, id, props);

…. // …

// Define the API Gateway resource const api = new apigateway.LambdaRestApi(this, ‘HelloWorldApi’, { handler: helloWorldFunction, proxy: false, });

// Define the ‘/hello’ resource with a GET method const helloResource = api.root.addResource(‘hello’); helloResource.addMethod(‘GET’); }; }; ….

== // …

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import ( # … # Import API Gateway L2 construct aws\_apigateway as apigateway, ) from constructs import Construct

class CdkHelloWorldStack(Stack):

…. def **init**(self, scope: Construct, construct\_id: str, \*\*kwargs) -> None: super().\_\_init\_\_(scope, construct\_id, \*\*kwargs)

# ...  
  
# Define the API Gateway resource  
api = apigateway.LambdaRestApi(  
 self,  
 "HelloWorldApi",  
 handler = hello\_world\_function,  
 proxy = False,  
)  
  
# Define the '/hello' resource with a GET method  
hello\_resource = api.root.add\_resource("hello")  
hello\_resource.add\_method("GET") ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — // … // Import API Gateway L2 construct import software.amazon.awscdk.services.apigateway.LambdaRestApi; import software.amazon.awscdk.services.apigateway.Resource;

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

…. public CdkHelloWorldStack(final Construct scope, final String id, final StackProps props) { super(scope, id, props);

// ...  
  
// Define the API Gateway resource  
LambdaRestApi api = LambdaRestApi.Builder.create(this, "HelloWorldApi")  
 .handler(helloWorldFunction)  
 .proxy(false) // Turn off default proxy integration  
 .build();  
  
// Define the '/hello' resource and its GET method  
Resource helloResource = api.getRoot().addResource("hello");  
helloResource.addMethod("GET");

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — // … // Import API Gateway L2 construct using Amazon.CDK.{aws}.APIGateway;

namespace CdkHelloWorld { public class CdkHelloWorldStack : Stack { internal CdkHelloWorldStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // …

…. // Define the API Gateway resource var api = new LambdaRestApi(this, “HelloWorldApi”, new LambdaRestApiProps { Handler = helloWorldFunction, Proxy = false });

// Add a '/hello' resource with a GET method  
 var helloResource = api.Root.AddResource("hello");  
 helloResource.AddMethod("GET");  
}

} } —- ….

Go:: + [source,go,subs=“verbatim,attributes”] — // …

import ( // … // Import Api Gateway L2 construct “github.com/aws/aws-cdk-go/awscdk/v2/awsapigateway” // … )

// …

func NewCdkHelloWorldStack(scope constructs.Construct, id string, props \*CdkHelloWorldStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. // Define the Lambda function resource // …

// Define the API Gateway resource api := awsapigateway.NewLambdaRestApi(stack, jsii.String(“HelloWorldApi”), &awsapigateway.LambdaRestApiProps{ Handler: helloWorldFunction, Proxy: jsii.Bool(false), })

// Add a ‘/hello’ resource with a GET method helloResource := api.Root().AddResource(jsii.String(“hello”), &awsapigateway.ResourceOptions{}) helloResource.AddMethod(jsii.String(“GET”), awsapigateway.NewLambdaIntegration(helloWorldFunction, &awsapigateway.LambdaIntegrationOptions{}), &awsapigateway.MethodOptions{})

return stack } ….

== // …

====

Here, you create an API Gateway REST API resource, along with the following:

* An integration between the REST API and your Lambda function, allowing the API to invoke your function. This includes the creation of a Lambda permission resource.
* A new resource or path named hello that is added to the root of the API endpoint. This creates a new endpoint that adds /hello to your base URL.
* A GET method for the hello resource. When a GET request is sent to the /hello endpoint, the Lambda function is invoked and its response is returned.

[#serverless-example-deploy-prepare] == Step 4: Prepare your application for deployment

In this step you prepare your application for deployment by building, if necessary, and performing basic validation with the {aws} CDK CLI cdk synth command.

If necessary, build your application:

==== [role=“tablist”] TypeScript:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ npm run build —

JavaScript:: Building is not required.

Python:: Building is not required.

Java:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ mvn package —

C#:: From the root of your project, run the following: + [source,none,subs=“verbatim,attributes”] — $ dotnet build src —

Go:: Building is not required. ====

Run cdk synth to synthesize an {aws} CloudFormation template from your CDK code. By using L2 constructs, many of the configuration details required by {aws} CloudFormation to facilitate the interaction between your Lambda function and REST API are provisioned for you by the {aws} CDK.

From the root of your project, run the following:

== [source,none,subs=“verbatim,attributes”]

## $ cdk synth

= [NOTE]

If you receive an error like the following, verify that you are in the cdk-hello-world directory and try again:

’’’

## –app is required either in command-line, in cdk.json or in ~/.cdk.json

====

If successful, the {aws} CDK CLI will output the {aws} CloudFormation template in YAML format at the command prompt. A JSON formatted template is also saved in the cdk.out directory.

The following is an example output of the {aws} CloudFormation template:

[#serverless-example-deploy-cfn] .{aws} CloudFormation template [%collapsible] ====

== [source,yaml,subs=“verbatim,attributes”]

Resources: HelloWorldFunctionServiceRoleunique-identifier: Type: {aws}::IAM::Role Properties: AssumeRolePolicyDocument: Statement: - Action: sts:AssumeRole Effect: Allow Principal: Service: lambda.amazonaws.com Version: “2012-10-17” ManagedPolicyArns: - Fn::Join: - “” - - “arn:” - Ref: {aws}::Partition - :iam::aws:policy/service-role/AWSLambdaBasicExecutionRole Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/ServiceRole/Resource HelloWorldFunctionunique-identifier: Type: {aws}::Lambda::Function Properties: Code: S3Bucket: Fn::Sub: cdk-unique-identifier-assets-${\{aws}::AccountId}-${{aws}::Region} S3Key: unique-identifier.zip Handler: hello.handler Role: Fn::GetAtt: - HelloWorldFunctionServiceRoleunique-identifier - Arn Runtime: nodejs20.x DependsOn: - HelloWorldFunctionServiceRoleunique-identifier Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/Resource aws:asset:path: asset.unique-identifier aws:asset:is-bundled: false aws:asset:property: Code HelloWorldApiunique-identifier: Type: {aws}::ApiGateway::RestApi Properties: Name: HelloWorldApi Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Resource HelloWorldApiDeploymentunique-identifier: Type: {aws}::ApiGateway::Deployment Properties: Description: Automatically created by the RestApi construct RestApiId: Ref: HelloWorldApiunique-identifier DependsOn: - HelloWorldApihelloGETunique-identifier - HelloWorldApihellounique-identifier Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Deployment/Resource HelloWorldApiDeploymentStageprod012345ABC: Type: {aws}::ApiGateway::Stage Properties: DeploymentId: Ref: HelloWorldApiDeploymentunique-identifier RestApiId: Ref: HelloWorldApiunique-identifier StageName: prod Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/DeploymentStage.prod/Resource HelloWorldApihellounique-identifier: Type: {aws}::ApiGateway::Resource Properties: ParentId: Fn::GetAtt: - HelloWorldApiunique-identifier - RootResourceId PathPart: hello RestApiId: Ref: HelloWorldApiunique-identifier Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/Resource HelloWorldApihelloGETApiPermissionCdkHelloWorldStackHelloWorldApiunique-identifier: Type: {aws}::Lambda::Permission Properties: Action: lambda:InvokeFunction FunctionName: Fn::GetAtt: - HelloWorldFunctionunique-identifier - Arn Principal: apigateway.amazonaws.com SourceArn: Fn::Join: - “” - - “arn:” - Ref: {aws}::Partition - “:execute-api:” - Ref: {aws}::Region - “:” - Ref: {aws}::AccountId - “:” - Ref: HelloWorldApi9E278160 - / - Ref: HelloWorldApiDeploymentStageprodunique-identifier - /GET/hello Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/ApiPermission.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello HelloWorldApihelloGETApiPermissionTestCdkHelloWorldStackHelloWorldApiunique-identifier: Type: {aws}::Lambda::Permission Properties: Action: lambda:InvokeFunction FunctionName: Fn::GetAtt: - HelloWorldFunctionunique-identifier - Arn Principal: apigateway.amazonaws.com SourceArn: Fn::Join: - “” - - “arn:” - Ref: {aws}::Partition - “:execute-api:” - Ref: {aws}::Region - “:” - Ref: {aws}::AccountId - “:” - Ref: HelloWorldApiunique-identifier - /test-invoke-stage/GET/hello Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/ApiPermission.Test.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello HelloWorldApihelloGETunique-identifier: Type: {aws}::ApiGateway::Method Properties: AuthorizationType: NONE HttpMethod: GET Integration: IntegrationHttpMethod: POST Type: AWS\_PROXY Uri: Fn::Join: - “” - - “arn:” - Ref: {aws}::Partition - “:apigateway:” - Ref: {aws}::Region - :lambda:path/2015-03-31/functions/ - Fn::GetAtt: - HelloWorldFunctionunique-identifier - Arn - /invocations ResourceId: Ref: HelloWorldApihellounique-identifier RestApiId: Ref: HelloWorldApiunique-identifier Metadata: aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/Resource CDKMetadata: Type: {aws}::CDK::Metadata Properties: Analytics: v2:deflate64:unique-identifier Metadata: aws:cdk:path: CdkHelloWorldStack/CDKMetadata/Default Condition: CDKMetadataAvailable Outputs: HelloWorldApiEndpointunique-identifier: Value: Fn::Join: - “” - - https:// - Ref: HelloWorldApiunique-identifier - .execute-api. - Ref: {aws}::Region - “.” - Ref: {aws}::URLSuffix - / - Ref: HelloWorldApiDeploymentStageprodunique-identifier - / Conditions: CDKMetadataAvailable: Fn::Or: - Fn::Or: - Fn::Equals: - Ref: {aws}::Region - af-south-1 - Fn::Equals: - Ref: {aws}::Region - ap-east-1 - Fn::Equals: - Ref: {aws}::Region - ap-northeast-1 - Fn::Equals: - Ref: {aws}::Region - ap-northeast-2 - Fn::Equals: - Ref: {aws}::Region - ap-south-1 - Fn::Equals: - Ref: {aws}::Region - ap-southeast-1 - Fn::Equals: - Ref: {aws}::Region - ap-southeast-2 - Fn::Equals: - Ref: {aws}::Region - ca-central-1 - Fn::Equals: - Ref: {aws}::Region - cn-north-1 - Fn::Equals: - Ref: {aws}::Region - cn-northwest-1 - Fn::Or: - Fn::Equals: - Ref: {aws}::Region - eu-central-1 - Fn::Equals: - Ref: {aws}::Region - eu-north-1 - Fn::Equals: - Ref: {aws}::Region - eu-south-1 - Fn::Equals: - Ref: {aws}::Region - eu-west-1 - Fn::Equals: - Ref: {aws}::Region - eu-west-2 - Fn::Equals: - Ref: {aws}::Region - eu-west-3 - Fn::Equals: - Ref: {aws}::Region - il-central-1 - Fn::Equals: - Ref: {aws}::Region - me-central-1 - Fn::Equals: - Ref: {aws}::Region - me-south-1 - Fn::Equals: - Ref: {aws}::Region - sa-east-1 - Fn::Or: - Fn::Equals: - Ref: {aws}::Region - us-east-1 - Fn::Equals: - Ref: {aws}::Region - us-east-2 - Fn::Equals: - Ref: {aws}::Region - us-west-1 - Fn::Equals: - Ref: {aws}::Region - us-west-2 Parameters: BootstrapVersion: Type: {aws}::SSM::Parameter::Value++++++Default: /cdk-bootstrap/hnb659fds/version Description: Version of the CDK Bootstrap resources in this environment, automatically retrieved from SSM Parameter Store. [cdk:skip] Rules: CheckBootstrapVersion: Assertions: - Assert: Fn::Not: - Fn::Contains: - - “1” - “2” - “3” - “4” - “5” - Ref: BootstrapVersion AssertDescription: CDK bootstrap stack version 6 required. Please run ‘cdk bootstrap’ with a recent version of the CDK CLI. —- ====++++++

By using L2 constructs, you define a few properties to configure your resources and use helper methods to integrate them together. The {aws} CDK configures the majority of your {aws} CloudFormation resources and properties required to provision your application.

[#serverless-example-deploy] == Step 5: Deploy your application

In this step, you use the {aws} CDK CLI cdk deploy command to deploy your application. The {aws} CDK works with the {aws} CloudFormation service to provision your resources.

= [IMPORTANT]

You must perform a one-time bootstrapping of your {aws} environment before deployment. For instructions, see xref:bootstrapping-env[Bootstrap your environment for use with the {aws} CDK].

====

From the root of your project, run the following. Confirm changes if prompted:

== [source,none,subs=“verbatim,attributes”]

$ cdk deploy

✨ Synthesis time: 2.44s

…

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

When deployment completes, the {aws} CDK CLI will output your endpoint URL. Copy this URL for the next step. The following is an example:

== [source,none,subs=“verbatim,attributes”]

… ✅ HelloWorldStack

✨ Deployment time: 45.37s

Outputs: HelloWorldStack.HelloWorldApiEndpointunique-identifier = https://++++++.execute-api.++++++.amazonaws.com/prod/ Stack ARN: arn:aws:cloudformation:region:account-id:stack/HelloWorldStack/unique-identifier ... —-++++++++++++

[#serverless-example-interact] == Step 6: Interact with your application

In this step, you initiate a GET request to your API endpoint and receive your Lambda function response.

Locate your endpoint URL from the previous step and add the /hello path. Then, using your browser or command prompt, send a GET request to your endpoint. The following is an example:

== [source,none,subs=“verbatim,attributes”]

$ curl https://++++++.execute-api.++++++.amazonaws.com/prod/hello {“message”:“Hello World!”}% —-++++++++++++

Congratulations, you have successfully created, deployed, and interacted with your application using the {aws} CDK!

[#serverless-example-delete] == Step 7: Delete your application

In this step, you use the {aws} CDK CLI to delete your application from the {aws} Cloud.

To delete your application, run cdk destroy. When prompted, confirm your request to delete the application:

== [source,none,subs=“verbatim,attributes”]

$ cdk destroy Are you sure you want to delete: CdkHelloWorldStack (y/n)? y CdkHelloWorldStack: destroying… [1/1] … ✅ CdkHelloWorldStack: destroyed —

[#serverless-example-troubleshooting] == Troubleshooting

[#serverless-example-trobuleshooting-error1] === Error: {“[.code]message”: “[.code]Internal server error”}%

When invoking the deployed Lambda function, you receive this error. This error could occur for multiple reasons.

*To troubleshoot further*:: + Use the {aws} CLI to invoke your Lambda function. + . Modify your stack file to capture the output value of your deployed Lambda function name. The following is an example: + [source,javascript,subs=“verbatim,attributes”] — …

class CdkHelloWorldStack extends Stack { constructor(scope, id, props) { super(scope, id, props);

…. // Define the Lambda function resource // …

new CfnOutput(this, ‘HelloWorldFunctionName’, { value: helloWorldFunction.functionName, description: ‘JavaScript Lambda function’ });

// Define the API Gateway resource // … } } —- + . Deploy your application again. The {aws} CDK CLI will output the value of your deployed Lambda function name: + [source,none,subs=“verbatim,attributes”] —- $ cdk deploy ….

✨ Synthesis time: 0.29s … ✅ CdkHelloWorldStack

✨ Deployment time: 20.36s

Outputs: … CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-HelloWorldFunctionunique-identifier … — + . Use the {aws} CLI to invoke your Lambda function in the {aws} Cloud and output the response to a text file: + [source,none,subs=“verbatim,attributes”] — $ aws lambda invoke –function-name CdkHelloWorldStack-HelloWorldFunctionunique-identifier output.txt — + . Check output.txt to see your results. +

*Possible cause: API Gateway resource is defined incorrectly in your stack file*::: + If output.txt shows a successful Lambda function response, the issue could be with how you defined your API Gateway REST API. The {aws} CLI invokes your Lambda directly, not through your endpoint. Check your code to ensure it matches this tutorial. Then, deploy again. + *Possible cause: Lambda resource is defined incorrectly in your stack file*::: If output.txt returns an error, the issue could be with how you defined your Lambda function. Check your code to ensure it matches this tutorial. Then deploy again.

include::attributes.txt[]

// Page attributes

[.topic] [#toolkit-library-examples] = Advanced CDK Toolkit Library examples :info\_titleabbrev: Advanced examples :keywords: CDK Toolkit Library, Examples, {aws} CDK, {aws} Cloud Development Kit ({aws} CDK), deploy, {aws} CloudFormation, Infrastructure as Code

== [abstract]

## Learn how to use advanced features of the {aws} CDK Toolkit Library through practical examples. This guide provides detailed code samples for error handling, deployment monitoring, and cloud assembly management that build upon the basic concepts covered in other sections.

// Content start

Learn how to use advanced features of the {aws} CDK Toolkit Library through practical examples. This guide provides detailed code samples for error handling, deployment monitoring, and cloud assembly management that build upon the basic concepts covered in other sections.

[#toolkit-library-examples-integration] == Integrating features

The following example demonstrates how to combine cloud assembly sources, custom io host implementation, and deployment options:

== [source,typescript,subs=“verbatim,attributes”]

import { Toolkit, StackSelectionStrategy, IIoHost } from ‘@aws-cdk/toolkit-lib’;

async function deployApplication(appPath, environment, options = {}) { // Create toolkit with custom message handling const toolkit = new Toolkit({ ioHost: { notify: async (msg) => { // Add environment to all messages console.log(+[${environment}][${msg.time}] ${msg.level}: ${msg.message}+); }, requestResponse: async (msg) => { // In production environments, use default responses if (environment === ‘production’) { console.log(Auto-approving for production: ${msg.message}); return msg.defaultResponse; }

// For other environments, implement custom approval logic  
 return promptForApproval(msg);

} } as IIoHost });

try { // Create cloud assembly source from the CDK app console.log(+Creating cloud assembly source from ${appPath}+); const cloudAssemblySource = await toolkit.fromCdkApp(appPath);

…. // Synthesize the cloud assembly console.log(Synthesizing cloud assembly); const cloudAssembly = await toolkit.synth(cloudAssemblySource);

try { // Deploy with environment-specific options console.log(Deploying to ${environment} environment); return await toolkit.deploy(cloudAssembly, { stacks: options.stacks || { strategy: StackSelectionStrategy.ALL\_STACKS }, parameters: options.parameters || {}, tags: { Environment: environment, DeployedBy: ‘CDK-Toolkit-Library’, DeployTime: new Date().toISOString() } }); } finally { // Always dispose when done await cloudAssembly.dispose(); } } catch (error) { console.error(Deployment to ${environment} failed:, error); throw error; } } ….

// Example usage await deployApplication(‘ts-node app.ts’, ‘staging’, { parameters: { MyStack: { InstanceType: ‘t3.small’ } } }); —

[#toolkit-library-examples-progress] == Tracking deployment progress

Track deployment progress with detailed status updates:

== [source,typescript,subs=“verbatim,attributes”]

import { Toolkit, StackSelectionStrategy, IIoHost } from ‘@aws-cdk/toolkit-lib’;

// Create a progress tracker class DeploymentTracker { private startTime: Date; private resources = new Map<string, string>();

constructor() { this.startTime = new Date(); }

onStackEvent(stackName: string, event: string, timestamp: string) { // Calculate elapsed time if needed, or use the provided timestamp const elapsed = (new Date().getTime() - this.startTime.getTime()) / 1000; console.log(+[${timestamp}] (${elapsed.toFixed(1)}s elapsed) Stack ${stackName}: ${event}+); }

onResourceEvent(resourceId: string, status: string) { this.resources.set(resourceId, status); this.printProgress(); }

private printProgress() { console.log(‘Status:’); for (const [id, status] of this.resources.entries()) { console.log(+- ${id}: ${status}+); } console.log(); } }

// Use the tracker with the toolkit const tracker = new DeploymentTracker(); const toolkit = new Toolkit({ ioHost: { notify: async (msg) => { if (msg.code.startsWith(‘CDK\_DEPLOY’)) { // Track deployment events if (msg.data && ‘stackName’ in msg.data) { tracker.onStackEvent(msg.data.stackName, msg.message, msg.time); } } else if (msg.code.startsWith(‘CDK\_RESOURCE’)) { // Track resource events if (msg.data && ‘resourceId’ in msg.data) { tracker.onResourceEvent(msg.data.resourceId, msg.message); } } } } as IIoHost });

// Example usage with progress tracking async function deployWithTracking(cloudAssemblySource: any) { try { // Synthesize the cloud assembly const cloudAssembly = await toolkit.synth(cloudAssemblySource);

try { // Deploy using the cloud assembly await toolkit.deploy(cloudAssembly, { stacks: { strategy: StackSelectionStrategy.ALL\_STACKS } }); } finally { // Always dispose when done await cloudAssembly.dispose(); } } catch (error) { // Display the error message console.error(“Operation failed:”, error.message); throw error; } } —-

[#toolkit-library-examples-error] == Handling errors with recovery

Implement robust error handling with recovery strategies:

== [source,typescript,subs=“verbatim,attributes”]

import { Toolkit, ToolkitError, StackSelectionStrategy } from ‘@aws-cdk/toolkit-lib’;

async function deployWithRetry(toolkit: Toolkit, cloudAssemblySource: any) { try { // Synthesize the cloud assembly const cloudAssembly = await toolkit.synth(cloudAssemblySource);

try { // Deploy using the cloud assembly await toolkit.deploy(cloudAssembly, { stacks: { strategy: StackSelectionStrategy.ALL\_STACKS } }); } finally { // Always dispose when done await cloudAssembly.dispose(); } } catch (error) { // Simply show the error to the user console.error(“Operation failed:”, error.message); throw error; } }

// Example usage try { await deployWithRetry(toolkit, cloudAssemblySource); } catch (error) { console.error(“Operation failed:”, error.message); process.exit(1); } —

[#toolkit-library-examples-cicd] == Integrating with CI/CD pipelines

Integrate the CDK Toolkit Library into a CI/CD pipeline:

== [source,typescript,subs=“verbatim,attributes”]

import { Toolkit, StackSelectionStrategy, IIoHost } from ‘@aws-cdk/toolkit-lib’; import \* as fs from ‘fs’; import \* as path from ‘path’;

async function cicdDeploy() { // Create a non-interactive toolkit for CI/CD environments const toolkit = new Toolkit({ ioHost: { notify: async (msg) => { // Write to both console and log file const logMessage = ${msg.time} [${msg.level}] ${msg.message}; console.log(logMessage);

// Append to deployment log file  
 fs.appendFileSync('deployment.log', logMessage + '\n');

}, requestResponse: async (msg) => { // Always use default responses in CI/CD console.log(Auto-responding to: ${msg.message} with: ${msg.defaultResponse}); return msg.defaultResponse; } } as IIoHost });

// Determine environment from CI/CD variables const environment = process.env.DEPLOYMENT\_ENV || ‘development’;

// Load environment-specific parameters const paramsPath = path.join(process.cwd(), +params.${environment}.json+); const parameters = fs.existsSync(paramsPath) ? JSON.parse(fs.readFileSync(paramsPath, ‘utf8’)) : {};

try { // Use pre-synthesized cloud assembly from build step const cloudAssemblySource = await toolkit.fromAssemblyDirectory(‘cdk.out’);

…. // Synthesize the cloud assembly const cloudAssembly = await toolkit.synth(cloudAssemblySource);

try { // Deploy with CI/CD specific options const result = await toolkit.deploy(cloudAssembly, { stacks: { strategy: StackSelectionStrategy.ALL\_STACKS }, parameters, tags: { Environment: environment, BuildId: process.env.BUILD\_ID || ‘unknown’, CommitHash: process.env.COMMIT\_HASH || ‘unknown’ } });

// Write outputs to a file for subsequent pipeline steps fs.writeFileSync( ‘stack-outputs.json’, JSON.stringify(result.outputs, null, 2) );

return result; } finally { // Always dispose when done await cloudAssembly.dispose(); } } catch (error) { // Display the error message console.error(“Operation failed:”, error.message); process.exit(1); } } ….

// Run the CI/CD deployment cicdDeploy().then(() => { console.log(‘CI/CD deployment completed successfully’); }); —

[#toolkit-library-examples-resources] == Additional resources

For more detailed information on specific components used in these examples, refer to:

* xref:toolkit-library-configure-ca[Managing cloud assembly sources] - Learn how to create and manage cloud assembly sources.
* xref:toolkit-library-configure-messages[Configuring messages and interactions] - Detailed guide on customizing the IIoHost interface.