[#hello-world-url] == Step 7: Define your Lambda function URL

In this step, you use the addFunctionUrl helper method of the Function construct to define a Lambda function URL. To output the value of this URL at deployment, you will create an {aws} CloudFormation output using the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.CfnOutput.html[CfnOutput] construct.

Add the following to your CDK stack file:

==== [role=“tablist”] TypeScript:: Located in lib/hello-cdk-stack.ts: + [source,javascript,subs=“verbatim,attributes”] — // …

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

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

// Define the Lambda function URL resource const myFunctionUrl = myFunction.addFunctionUrl({ authType: lambda.FunctionUrlAuthType.NONE, });

// Define a CloudFormation output for your URL new cdk.CfnOutput(this, “myFunctionUrlOutput”, { value: myFunctionUrl.url, }) ….

} } —

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

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

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

// Define the Lambda function URL resource const myFunctionUrl = myFunction.addFunctionUrl({ authType: lambda.FunctionUrlAuthType.NONE, });

// Define a CloudFormation output for your URL new CfnOutput(this, “myFunctionUrlOutput”, { value: myFunctionUrl.url, }) ….

} }

== module.exports = { HelloCdkStack }

Python:: Located in hello\_cdk/hello\_cdk\_stack.py: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import ( # … CfnOutput # Import CfnOutput ) from constructs import Construct

class HelloCdkStack(Stack):

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

…. # Define the Lambda function resource # …

# Define the Lambda function URL resource

my\_function\_url = my\_function.add\_function\_url( auth\_type = \_lambda.FunctionUrlAuthType.NONE, )

# Define a CloudFormation output for your URL

CfnOutput(self, “myFunctionUrlOutput”, value=my\_function\_url.url) —- ….

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

// … // Import Lambda function URL import software.amazon.awscdk.services.lambda.FunctionUrl; import software.amazon.awscdk.services.lambda.FunctionUrlAuthType; import software.amazon.awscdk.services.lambda.FunctionUrlOptions; // Import CfnOutput import software.amazon.awscdk.CfnOutput;

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

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

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

// Define the Lambda function URL resource FunctionUrl myFunctionUrl = myFunction.addFunctionUrl(FunctionUrlOptions.builder() .authType(FunctionUrlAuthType.NONE) .build());

// Define a CloudFormation output for your URL CfnOutput.Builder.create(this, “myFunctionUrlOutput”) .value(myFunctionUrl.getUrl()) .build(); } } —- ….

C#:: Located in +src/main/java/.../HelloCdkStack.java+: + [source,csharp,subs=“verbatim,attributes”] — // …

namespace HelloCdk { public class HelloCdkStack : Stack { internal HelloCdkStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Define the Lambda function resource // …

…. // Define the Lambda function URL resource var myFunctionUrl = myFunction.AddFunctionUrl(new FunctionUrlOptions { AuthType = FunctionUrlAuthType.NONE });

// Define a CloudFormation output for your URL new CfnOutput(this, “myFunctionUrlOutput”, new CfnOutputProps { Value = myFunctionUrl.Url }); } } } —- ….

Go:: Located in hello-cdk.go: + [source,go,subs=“verbatim,attributes”] — // …

func NewHelloCdkStack(scope constructs.Construct, id string, props \*HelloCdkStackProps) 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 Lambda function URL resource myFunctionUrl := myFunction.AddFunctionUrl(&awslambda.FunctionUrlOptions{ AuthType: awslambda.FunctionUrlAuthType\_NONE, })

// Define a CloudFormation output for your URL awscdk.NewCfnOutput(stack, jsii.String(“myFunctionUrlOutput”), &awscdk.CfnOutputProps{ Value: myFunctionUrl.Url(), })

return stack }

== // …

====

= [WARNING]

To keep this tutorial simple, your Lambda function URL is defined without authentication. When deployed, this creates a publicly accessible endpoint that can be used to invoke your function. When you are done with this tutorial, follow xref:hello-world-delete[Step 12: Delete your application] to delete these resources.

====

[#hello-world-synth] == Step 8: Synthesize a CloudFormation template

In this step, you prepare for deployment by synthesizing a CloudFormation template with the CDK CLI cdk synth command. This command performs basic validation of your CDK code, runs your CDK app, and generates a CloudFormation template from your CDK stack.

If your app contains more than one stack, you must specify which stacks to synthesize. Since your app contains a single stack, the CDK CLI automatically detects the stack to synthesize.

If you don’t synthesize a template, the CDK CLI will automatically perform this step when you deploy. However, we recommend that you run this step before each deployment to check for synthesis errors.

Before synthesizing a template, you can optionally build your application to catch syntax and type errors. For instructions, see xref:hello-world-build[Step 4: Build your CDK app].

To synthesize a CloudFormation template, run the following from the root of your project:

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

## $ cdk synth

= [NOTE]

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

’’’

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

====

If successful, the CDK CLI will output a YAML–formatted CloudFormation template to stdout and save a JSON–formatted template in the cdk.out directory of your project.

The following is an example output of the CloudFormation template:

[#hello-world-synth-template] .{aws} CloudFormation template [%collapsible] ====

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

Resources: HelloWorldFunctionServiceRole++++++: 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: HelloCdkStack/HelloWorldFunction/ServiceRole/Resource HelloWorldFunction++++++: Type: {aws}::Lambda::Function Properties: Code: ZipFile: “++++++++++++

….   exports.handler = async function(event) {

\ return {  
  
 \ statusCode: 200,  
  
 \ body: JSON.stringify('Hello World!'),  
  
 \ };  
  
 \ };  
  
 \ "

Handler: index.handler Role: Fn::GetAtt: - HelloWorldFunctionServiceRole - Arn Runtime: nodejs20.x DependsOn: - HelloWorldFunctionServiceRole Metadata: aws:cdk:path: HelloCdkStack/HelloWorldFunction/Resource HelloWorldFunctionFunctionUrl: Type: {aws}::Lambda::Url Properties: AuthType: NONE TargetFunctionArn: Fn::GetAtt: - HelloWorldFunction - Arn Metadata: aws:cdk:path: HelloCdkStack/HelloWorldFunction/FunctionUrl/Resource HelloWorldFunctioninvokefunctionurl: Type: {aws}::Lambda::Permission Properties: Action: lambda:InvokeFunctionUrl FunctionName: Fn::GetAtt: - HelloWorldFunction - Arn FunctionUrlAuthType: NONE Principal: “\*” Metadata: aws:cdk:path: HelloCdkStack/HelloWorldFunction/invoke-function-url CDKMetadata: Type: {aws}::CDK::Metadata Properties: Analytics: v2:deflate64: Metadata: aws:cdk:path: HelloCdkStack/CDKMetadata/Default Condition: CDKMetadataAvailable Outputs: myFunctionUrlOutput: Value: Fn::GetAtt: - HelloWorldFunctionFunctionUrl - FunctionUrl Parameters: BootstrapVersion: Type: {aws}::SSM::Parameter::Value Default: /cdk-bootstrap//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. —- ==== ….

= [NOTE]

Every generated template contains an +{aws}::CDK::Metadata+ resource by default. The {aws} CDK team uses this metadata to gain insight into {aws} CDK usage and find ways to improve it. For details, including how to opt out of version reporting, see xref:version-reporting[Version reporting].

====

By defining a single L2 construct, the {aws} CDK creates an extensive CloudFormation template containing your Lambda resources, along with the permissions and glue logic required for your resources to interact within your application.

[#hello-world-deploy] == Step 9: Deploy your CDK stack

In this step, you use the CDK CLI cdk deploy command to deploy your CDK stack. This command retrieves your generated CloudFormation template and deploys it through {aws} CloudFormation, which provisions your resources as part of a CloudFormation stack.

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

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

$ cdk deploy

✨ Synthesis time: 2.69s

HelloCdkStack: start: Building ++++++:current\_account-current\_region HelloCdkStack: success: Built ++++++:current\_account-current\_region HelloCdkStack: start: Publishing ++++++:current\_account-current\_region HelloCdkStack: success: Published ++++++:current\_account-current\_region 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 ┌───┬───────────────────────────────────────┬────────┬──────────────────────────┬──────────────────────────────┬───────────┐ │ │ Resource │ Effect │ Action │ Principal │ Condition │ ├───┼───────────────────────────────────────┼────────┼──────────────────────────┼──────────────────────────────┼───────────┤ │ + │ ${HelloWorldFunction.Arn} │ Allow │ lambda:InvokeFunctionUrl │ \* │ │ ├───┼───────────────────────────────────────┼────────┼──────────────────────────┼──────────────────────────────┼───────────┤ │ + │ ${HelloWorldFunction/ServiceRole.Arn} │ Allow │ sts:AssumeRole │ Service:lambda.amazonaws.com │ │ └───┴───────────────────────────────────────┴────────┴──────────────────────────┴──────────────────────────────┴───────────┘ IAM Policy Changes ┌───┬───────────────────────────────────┬────────────────────────────────────────────────────────────────────────────────┐ │ │ Resource │ Managed Policy ARN │ ├───┼───────────────────────────────────┼────────────────────────────────────────────────────────────────────────────────┤ │ + │ {{aws}::Partition}:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole │ └───┴───────────────────────────────────┴────────────────────────────────────────────────────────────────────────────────┘ (NOTE: There may be security-related changes not in this list. See https://github.com/aws/aws-cdk/issues/1299)

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

Similar to cdk synth, you don’t have to specify the {aws} CDK stack since the app contains a single stack.

During deployment, the CDK CLI displays progress information as your stack is deployed. When complete, you can go to the https://console.aws.amazon.com/cloudformation/home[{aws} CloudFormation console] to view your HelloCdkStack stack. You can also go to the Lambda console to view your HelloWorldFunction resource.

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

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

… HelloCdkStack: deploying… [1/1] HelloCdkStack: creating CloudFormation changeset…

✅ HelloCdkStack

✨ Deployment time: 41.65s

Outputs: HelloCdkStack.myFunctionUrlOutput = https://++++++.lambda-url.++++++.on.aws/ Stack ARN: arn:aws:cloudformation:++++++:stack/HelloCdkStack/++++++++++++++++++++++++++++++

== ✨ Total time: 44.34s

[#hello-world-interact] == Step 10: Interact with your application on {aws}

In this step, you interact with your application on {aws} by invoking your Lambda function through the function URL. When you access the URL, your Lambda function returns the Hello World! message.

To invoke your function, access the function URL through your browser or from the command line. The following is an example:

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

$ curl https://++++++.lambda-url.++++++.on.aws/ “Hello World!”% —-++++++++++++

[#hello-world-modify] == Step 11: Modify your application

In this step, you modify the message that the Lambda function returns when invoked. You perform a diff using the CDK CLI cdk diff command to preview your changes and deploy to update your application. You then interact with your application on {aws} to see your new message.

Modify the myFunction instance in your CDK stack file as follows:

==== [role=“tablist”] TypeScript:: Located in lib/hello-cdk-stack.ts: + [source,javascript,subs=“verbatim,attributes”] — // …

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

…. // Modify the Lambda function resource const myFunction = new lambda.Function(this, “HelloWorldFunction”, { runtime: lambda.Runtime.NODEJS\_20\_X, // Provide any supported Node.js runtime handler: “index.handler”, code: lambda.Code.fromInline(exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify('Hello CDK!'), }; };), });

// … } } —- ….

JavaScript:: Located in lib/hello-cdk-stack.js: + [source,javascript,subs=“verbatim,attributes”] — // …

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

…. // Modify the Lambda function resource const myFunction = new lambda.Function(this, “HelloWorldFunction”, { runtime: lambda.Runtime.NODEJS\_20\_X, // Provide any supported Node.js runtime handler: “index.handler”, code: lambda.Code.fromInline(exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify('Hello CDK!'), }; };), });

// … ….

} }

== module.exports = { HelloCdkStack }

Python:: Located in hello\_cdk/hello\_cdk\_stack.py: + [source,python,subs=“verbatim,attributes”] —

= …

class HelloCdkStack(Stack):

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

…. # Modify the Lambda function resource my\_function = \_lambda.Function( self, “HelloWorldFunction”, runtime = \_lambda.Runtime.NODEJS\_20\_X, # Provide any supported Node.js runtime handler = “index.handler”, code = \_lambda.Code.from\_inline( ““” exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify(‘Hello CDK!’), }; }; ““” ), )

# … —-

….

Java:: Located in +src/main/java/.../HelloCdkStack.java+: + [source,java,subs=“verbatim,attributes”] — // …

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

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

…. // Modify the Lambda function resource Function myFunction = Function.Builder.create(this, “HelloWorldFunction”) .runtime(Runtime.NODEJS\_20\_X) // Provide any supported Node.js runtime .handler(“index.handler”) .code(Code.fromInline( “exports.handler = async function(event) {” + ” return {” + ” statusCode: 200,” + ” body: JSON.stringify(‘Hello CDK!’)” + ” };” + “};”)) .build();

// … } } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — // …

namespace HelloCdk { public class HelloCdkStack : Stack { internal HelloCdkStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Modify the Lambda function resource var myFunction = new Function(this, “HelloWorldFunction”, new FunctionProps { Runtime = Runtime.NODEJS\_20\_X, // Provide any supported Node.js runtime Handler = “index.handler”, Code = Code.FromInline(@” exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify(‘Hello CDK!’), }; }; “), });

// … } } } —-

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

type HelloCdkStackProps struct { awscdk.StackProps }

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

// Modify the Lambda function resource myFunction := awslambda.NewFunction(stack, jsii.String(“HelloWorldFunction”), &awslambda.FunctionProps{ Runtime: awslambda.Runtime\_NODEJS\_20\_X(), // Provide any supported Node.js runtime Handler: jsii.String(“index.handler”), Code: awslambda.Code\_FromInline(jsii.String(exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify('Hello CDK!'), }; };)), })

// …

== }

====

Currently, your code changes have not made any direct updates to your deployed Lambda resource. Your code defines the desired state of your resource. To modify your deployed resource, you will use the CDK CLI to synthesize the desired state into a new {aws} CloudFormation template. Then, you will deploy your new CloudFormation template as a change set. Change sets make only the necessary changes to reach your new desired state.

To preview your changes, run the cdk diff command. The following is an example:

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

$ cdk diff Stack HelloCdkStack 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}::Lambda::Function HelloWorldFunction HelloWorldFunction++++++└─ [~] Code └─ [~] .ZipFile: ├─ [-] exports.handler = async function(event) { return { statusCode: 200, body: JSON.stringify(‘Hello World!’), }; };++++++

└─ [+]  
 exports.handler = async function(event) {  
 return {  
 statusCode: 200,  
 body: JSON.stringify('Hello CDK!'),  
 };  
 };

== ✨ Number of stacks with differences: 1

To create this diff, the CDK CLI queries your {aws} account account for the latest {aws} CloudFormation template for the HelloCdkStack stack. Then, it compares the latest template with the template it just synthesized from your app.

To implement your changes, run the cdk deploy command. The following is an example:

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

$ cdk deploy

✨ Synthesis time: 2.12s

HelloCdkStack: start: Building ++++++:current\_account-current\_region HelloCdkStack: success: Built ++++++:current\_account-current\_region HelloCdkStack: start: Publishing ++++++:current\_account-current\_region HelloCdkStack: success: Published ++++++:current\_account-current\_region HelloCdkStack: deploying... [1/1] HelloCdkStack: creating CloudFormation changeset...++++++++++++++++++++++++

✅ HelloCdkStack

✨ Deployment time: 26.96s

Outputs: HelloCdkStack.myFunctionUrlOutput = https://++++++.lambda-url.++++++.on.aws/ Stack ARN: arn:aws:cloudformation:++++++:stack/HelloCdkStack/++++++++++++++++++++++++++++++

== ✨ Total time: 29.07s

To interact with your application, repeat xref:hello-world-interact[Step 10: Interact with your application on {aws}]. The following is an example:

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

$ curl https://++++++.lambda-url.++++++.on.aws/ “Hello CDK!”% —-++++++++++++

[#hello-world-delete] == Step 12: Delete your application

In this step, you use the CDK CLI cdk destroy command to delete your application. This command deletes the CloudFormation stack associated with your CDK stack, which includes the resources you created.

To delete your application, run the cdk destroy command and confirm your request to delete the application. The following is an example:

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

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

== ✅ HelloCdkStack: destroyed

[#hello-world-next-steps] == Next steps

Congratulations! You’ve completed this tutorial and have used the {aws} CDK to successfully create, modify, and delete resources in the {aws} Cloud. You’re now ready to begin using the {aws} CDK.

To learn more about using the {aws} CDK in your preferred programming language, see xref:work-with[Work with the {aws} CDK library].

For additional resources, see the following:

* Try the https://cdkworkshop.com/[CDK Workshop] for a more in-depth tour involving a more complex project.
* See the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-construct-library.html[API reference] to begin exploring the CDK constructs available for your favorite {aws} services.
* Visit https://constructs.dev/search?q=&cdk=aws-cdk&cdkver=2&sort=downloadsDesc&offset=0[Construct Hub] to discover constructs created by {aws} and others.
* Explore https://github.com/aws-samples/aws-cdk-examples[Examples] of using the {aws} CDK.

The {aws} CDK is an open-source project. To contribute, see to https://github.com/aws/aws-cdk/blob/main/CONTRIBUTING.md[Contributing to the {aws} Cloud Development Kit ({aws} CDK)].