:doctype: book

include::attributes.txt[]

// Attribues [.topic] [#get-cfn-param] = Use CloudFormation parameters to get a CloudFormation value :info\_titleabbrev: Use CloudFormation parameters

// Content start

Use {aws} CloudFormation parameters within {aws} Cloud Development Kit ({aws} CDK) applications to input custom values into your synthesized CloudFormation templates at deployment.

For an introduction, see xref:parameters[Parameters and the {aws} CDK].

[#parameters-define] == Define parameters in your CDK app

Use the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.CfnParameter.html[CfnParameter] class to define a parameter. You’ll want to specify at least a type and a description for most parameters, though both are technically optional. The description appears when the user is prompted to enter the parameter’s value in the {aws} CloudFormation console. For more information on the available types, see link:https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html#parameters-section-structure-properties-type[Types].

= [NOTE]

You can define parameters in any scope. However, we recommend defining parameters at the stack level so that their logical ID doesn’t change when you refactor your code.

====

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const uploadBucketName = new CfnParameter(this, “uploadBucketName”, { type: “String”, description: “The name of the Amazon S3 bucket where uploaded files will be stored.”}); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const uploadBucketName = new CfnParameter(this, “uploadBucketName”, { type: “String”, description: “The name of the Amazon S3 bucket where uploaded files will be stored.”}); —

Python:: + [source,python,subs=“verbatim,attributes”] — upload\_bucket\_name = CfnParameter(self, “uploadBucketName”, type=“String”, description=“The name of the Amazon S3 bucket where uploaded files will be stored.”) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnParameter uploadBucketName = CfnParameter.Builder.create(this, “uploadBucketName”) .type(“String”) .description(“The name of the Amazon S3 bucket where uploaded files will be stored”) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var uploadBucketName = new CfnParameter(this, “uploadBucketName”, new CfnParameterProps { Type = “String”, Description = “The name of the Amazon S3 bucket where uploaded files will be stored” }); — ====

[#parameters-use,] == Use parameters

A CfnParameter instance exposes its value to your CDK app via a xref:tokens[token]. Like all tokens, the parameter’s token is resolved at synthesis time. But it resolves to a reference to the parameter defined in the {aws} CloudFormation template (which will be resolved at deploy time), rather than to a concrete value.

You can retrieve the token as an instance of the Token class, or in string, string list, or numeric encoding. Your choice depends on the kind of value required by the class or method that you want to use the parameter with.

==== [role=“tablist”] TypeScript:: + [cols=“1,1”, options=“header”] |=== | Property | kind of value

|=== | value | Token class instance |===

|=== | valueAsList | The token represented as a string list |===

|=== | valueAsNumber | The token represented as a number |===

|=== | valueAsString | The token represented as a string |===

JavaScript:: + [cols=“1,1”, options=“header”] |=== | Property | kind of value

|=== | value | Token class instance |===

|=== | valueAsList | The token represented as a string list |===

|=== | valueAsNumber | The token represented as a number |===

|=== | valueAsString | The token represented as a string |===

Python:: + [cols=“1,1”, options=“header”] |=== | Property | kind of value

|=== | value | Token class instance |===

|=== | value\_as\_list | The token represented as a string list |===

|=== | value\_as\_number | The token represented as a number |===

|=== | value\_as\_string | The token represented as a string |===

Java:: + [cols=“1,1”, options=“header”] |=== | Property | kind of value

|=== | getValue() | Token class instance |===

|=== | getValueAsList() | The token represented as a string list |===

|=== | getValueAsNumber() | The token represented as a number |===

|=== | getValueAsString() | The token represented as a string |===

C#:: + [cols=“1,1”, options=“header”] |=== | Property | kind of value

|=== | Value | Token class instance |===

|=== | ValueAsList | The token represented as a string list |===

|=== | ValueAsNumber | The token represented as a number |===

|ValueAsString |The token represented as a string |=== ====

For example, to use a parameter in a Bucket definition:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new Bucket(this, “amzn-s3-demo-bucket”, { bucketName: uploadBucketName.valueAsString}); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new Bucket(this, “amzn-s3-demo-bucket”, { bucketName: uploadBucketName.valueAsString}); —

Python:: + [source,python,subs=“verbatim,attributes”] — bucket = Bucket(self, “amzn-s3-demo-bucket”, bucket\_name=upload\_bucket\_name.value\_as\_string) —

Java:: + [source,java,subs=“verbatim,attributes”] — Bucket bucket = Bucket.Builder.create(this, “amzn-s3-demo-bucket”) .bucketName(uploadBucketName.getValueAsString()) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var bucket = new Bucket(this, “amzn-s3-demo-bucket”) { BucketName = uploadBucketName.ValueAsString }; — ====

[#parameters-deploy] == Deploy CDK apps containing parameters

When you deploy a generated {aws} CloudFormation template through the {aws} CloudFormation console, you will be prompted to provide the values for each parameter.

You can also provide parameter values using the CDK CLI cdk deploy command, or by specifying parameter values in your CDK project’s stack file.

[#parameters-deploy-cli] === Provide parameter values with [.noloc]cdk deploy

When you deploy using the CDK CLI cdk deploy command, you can provide parameter values at deployment with the --parameters option.

The following is an example of the cdk deploy command structure:

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

$ cdk deploy ++++++–parameters ++++++:++++++=++++++—-++++++++++++++++++++++++

If your CDK app contains a single stack, you don’t have to provide the stack logical ID argument or the stack-name value in the --parameters option. The CDK CLI will automatically find and provide these values. The following is an example that specifies an uploadbucket value for the uploadBucketName parameter of the single stack in our CDK app:

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

$ cdk deploy –parameters ++++++=++++++—-++++++++++++

[#parameters-deploy-cli-multi-stack] === Provide parameter values with cdk deploy for multi-stack applications

The following is an example CDK application in TypeScript that contains two CDK stacks. Each stack contains an Amazon S3 bucket instance and a parameter to set the Amazon S3 bucket name:

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

import \* as cdk from ‘aws-cdk-lib’; import { Construct } from ‘constructs’; import \* as s3 from ‘aws-cdk-lib/aws-s3’;

// Define the CDK app const app = new cdk.App();

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

…. // Set a default parameter name const bucketNameParam = new cdk.CfnParameter(this, ‘bucketNameParam’, { type: ‘String’, default: ‘myfirststackdefaultbucketname’ });

// Define an S3 bucket new s3.Bucket(this, ‘MyFirstBucket’, { bucketName: bucketNameParam.valueAsString }); } } ….

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

…. // Set a default parameter name const bucketNameParam = new cdk.CfnParameter(this, ‘bucketNameParam’, { type: ‘String’, default: ‘mysecondstackdefaultbucketname’ });

// Define an S3 bucket new s3.Bucket(this, ‘MySecondBucket’, { bucketName: bucketNameParam.valueAsString }); } } ….

// Instantiate the stacks new MyFirstStack(app, ‘MyFirstStack’, { stackName: ‘MyFirstDeployedStack’, });

new MySecondStack(app, ‘MySecondStack’, { stackName: ‘MySecondDeployedStack’, }); —

For CDK apps that contain multiple stacks, you can do the following:

* *Deploy one stack with parameters* – To deploy a single stack from a multi-stack application, provide the stack logical ID as an argument.
* The following is an example that deploys MySecondStack with mynewbucketname as the parameter value for bucketNameParam:

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

* $ cdk deploy ++++++–parameters ++++++=‘++++++’ —-++++++++++++++++++
* *Deploy all stacks and specify parameter values for each stack* – Provide the '\*' wildcard or the --all option to deploy all stacks. Provide the --parameters option multiple times in a single command to specify parameter values for each stack. The following is an example:

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

* $ cdk deploy <’\*‘> –parameters ++++++:++++++=’++++++’ –parameters ++++++:++++++=‘++++++’ —-++++++++++++++++++++++++++++++++++++
* *Deploy all stacks and specify parameter values for a single stack* – Provide the '\*' wildcard or the --all option to deploy all stacks. Then, specify the stack to define the parameter for in the --parameters option. The following are examples that deploys all stacks in a CDK app and specifies a parameter value for the MySecondDeployedStack {aws} CloudFormation stack. All other stacks will deploy and use the default parameter value:

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

* $ cdk deploy <’\*‘> –parameters ++++++:++++++=’++++++’ $ cdk deploy <–all> –parameters ++++++:++++++=‘++++++’ —-++++++++++++++++++++++++++++++++++++

[#parameters-deploy-cli-nested-stack] === Provide parameter values with cdk deploy for applications with nested stacks

The CDK CLI behavior when working with applications containing nested stacks is similar to multi-stack applications. The main difference is, if you want to deploy all nested stacks, use the +'\\\*\*'+ wildcard. The '\*' wildcard deploys all stacks but will not deploy nested stacks. The +'\*\*'+ wildcard deploys all stacks, including nested stacks.

The following is an example that deploys nested stacks while specifying the parameter value for one nested stack:

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

$ cdk deploy ’\*\*’ –parameters :++++++=‘++++++’ —-++++++++++++

For more information on cdk deploy command options, see xref:ref-cli-cmd-deploy[cdk deploy].