:doctype: book

include::attributes.txt[]

// Attributes [.topic] :info\_titleabbrev: Resources :keywords: {aws} CDK, {aws} Cloud Development Kit ({aws} CDK), {aws} CloudFormation, IaC, Infrastructure as code, {aws}, {aws} resources

[#resources] = Resources and the {aws} CDK

== [abstract]

## *Resources* are what you configure to use {aws} services in your applications. Resources are a feature of {aws} CloudFormation. By configuring resources and their properties in a CloudFormation template, you can deploy to CloudFormation to provision your resources. With the {aws} Cloud Development Kit ({aws} CDK), you can configure resources through constructs. You then deploy your CDK app, which involves synthesizing a {aws} CloudFormation template and deploying to {aws} CloudFormation to provision your resources.

// Content start

*Resources* are what you configure to use {aws} services in your applications. Resources are a feature of {aws} CloudFormation. By configuring resources and their properties in a {aws} CloudFormation template, you can deploy to {aws} CloudFormation to provision your resources. With the {aws} Cloud Development Kit ({aws} CDK), you can configure resources through constructs. You then deploy your CDK app, which involves synthesizing a {aws} CloudFormation template and deploying to {aws} CloudFormation to provision your resources.

[#resources-configure] == Configuring resources using constructs

As described in xref:constructs[{aws} CDK Constructs], the {aws} CDK provides a rich class library of constructs, called *constructs*, that represent all {aws} resources.

To create an instance of a resource using its corresponding construct, pass in the scope as the first argument, the logical ID of the construct, and a set of configuration properties (props). For example, here’s how to create an Amazon SQS queue with {aws} KMS encryption using the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_sqs.Queue.html[sqs.Queue] construct from the {aws} Construct Library.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as sqs from ‘@aws-cdk/aws-sqs’;

new sqs.Queue(this, ‘MyQueue’, { encryption: sqs.QueueEncryption.KMS\_MANAGED }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const sqs = require(‘@aws-cdk/aws-sqs’);

new sqs.Queue(this, ‘MyQueue’, { encryption: sqs.QueueEncryption.KMS\_MANAGED }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_sqs as sqs

== sqs.Queue(self, “MyQueue”, encryption=sqs.QueueEncryption.KMS\_MANAGED)

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.services.sqs.\*;

Queue.Builder.create(this, “MyQueue”).encryption( QueueEncryption.KMS\_MANAGED).build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK.{aws}.SQS;

new Queue(this, “MyQueue”, new QueueProps { Encryption = QueueEncryption.KMS\_MANAGED }); —

Go:: + [source,go,subs=“verbatim,attributes”] — import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/jsii-runtime-go”  
sqs “github.com/aws/aws-cdk-go/awscdk/v2/awssqs” )

sqs.NewQueue(stack, jsii.String(“MyQueue”), &sqs.QueueProps{ Encryption: sqs.QueueEncryption\_KMS\_MANAGED, }) — ====

Some configuration props are optional, and in many cases have default values. In some cases, all props are optional, and the last argument can be omitted entirely.

[#resources-attributes] === Resource attributes

Most resources in the {aws} Construct Library expose attributes, which are resolved at deployment time by {aws} CloudFormation. Attributes are exposed in the form of properties on the resource classes with the type name as a prefix. The following example shows how to get the URL of an Amazon SQS queue using the queueUrl (Python: queue\_url) property.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as sqs from ‘@aws-cdk/aws-sqs’;

const queue = new sqs.Queue(this, ‘MyQueue’); const url = queue.queueUrl; // => A string representing a deploy-time value —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const sqs = require(‘@aws-cdk/aws-sqs’);

const queue = new sqs.Queue(this, ‘MyQueue’); const url = queue.queueUrl; // => A string representing a deploy-time value —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_sqs as sqs

queue = sqs.Queue(self, “MyQueue”) url = queue.queue\_url # => A string representing a deploy-time value —

Java:: + [source,java,subs=“verbatim,attributes”] — Queue queue = new Queue(this, “MyQueue”); String url = queue.getQueueUrl(); // => A string representing a deploy-time value —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var queue = new Queue(this, “MyQueue”); var url = queue.QueueUrl; // => A string representing a deploy-time value —

Go:: + [source,go,subs=“verbatim,attributes”] — import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/jsii-runtime-go”  
sqs “github.com/aws/aws-cdk-go/awscdk/v2/awssqs” )

queue := sqs.NewQueue(stack, jsii.String(“MyQueue”), &sqs.QueueProps{}) url := queue.QueueUrl() // => A string representing a deploy-time value — ====

See xref:tokens[Tokens and the {aws} CDK] for information about how the {aws} CDK encodes deploy-time attributes as strings.

[#resources-referencing] == Referencing resources

When configuring resources, you will often have to reference properties of another resource. The following are examples:

* An Amazon Elastic Container Service (Amazon ECS) resource requires a reference to the cluster on which it runs.
* An Amazon CloudFront distribution requires a reference to the Amazon Simple Storage Service (Amazon S3) bucket containing the source code.

You can reference resources in any of the following ways:

* By passing a resource defined in your CDK app, either in the same stack or in a different one
* By passing a proxy object referencing a resource defined in your {aws} account, created from a unique identifier of the resource (such as an ARN)

If the property of a construct represents a construct for another resource, its type is that of the interface type of the construct. For example, the Amazon ECS construct takes a property cluster of type ecs.ICluster. Another example, is the CloudFront distribution construct that takes a property sourceBucket (Python: source\_bucket) of type s3.IBucket.

You can directly pass any resource object of the proper type defined in the same {aws} CDK app. The following example defines an Amazon ECS cluster and then uses it to define an Amazon ECS service.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const cluster = new ecs.Cluster(this, ‘Cluster’, { /*…*/ });

== const service = new ecs.Ec2Service(this, ‘Service’, { cluster: cluster });

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const cluster = new ecs.Cluster(this, ‘Cluster’, { /*…*/ });

== const service = new ecs.Ec2Service(this, ‘Service’, { cluster: cluster });

Python:: + [source,python,subs=“verbatim,attributes”] — cluster = ecs.Cluster(self, “Cluster”)

== service = ecs.Ec2Service(self, “Service”, cluster=cluster)

Java:: + [source,java,subs=“verbatim,attributes”] — Cluster cluster = new Cluster(this, “Cluster”); Ec2Service service = new Ec2Service(this, “Service”, new Ec2ServiceProps.Builder().cluster(cluster).build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var cluster = new Cluster(this, “Cluster”); var service = new Ec2Service(this, “Service”, new Ec2ServiceProps { Cluster = cluster }); —

Go:: + [source,go,subs=“verbatim,attributes”] — import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/jsii-runtime-go” + ecs “github.com/aws/aws-cdk-go/awscdk/v2/awsecs” )

cluster := ecs.NewCluster(stack, jsii.String(“MyCluster”), &ecs.ClusterProps{}) service := ecs.NewEc2Service(stack, jsii.String(“MyService”), &ecs.Ec2ServiceProps{ Cluster: cluster, }) — ====

[#resource-stack] === Referencing resources in a different stack

You can refer to resources in a different stack as long as they are defined in the same app and are in the same {aws} environment. The following pattern is generally used:

* Store a reference to the construct as an attribute of the stack that produces the resource. (To get a reference to the current construct’s stack, use Stack.of(this).)
* Pass this reference to the constructor of the stack that consumes the resource as a parameter or a property. The consuming stack then passes it as a property to any construct that needs it.

The following example defines a stack stack1. This stack defines an Amazon S3 bucket and stores a reference to the bucket construct as an attribute of the stack. Then the app defines a second stack, stack2, which accepts a bucket at instantiation. stack2 might, for example, define an {aws} Glue Table that uses the bucket for data storage.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const prod = { account: ‘123456789012’, region: ‘us-east-1’ };

const stack1 = new StackThatProvidesABucket(app, ‘Stack1’, { env: prod });

// stack2 will take a property { bucket: IBucket } const stack2 = new StackThatExpectsABucket(app, ‘Stack2’, { bucket: stack1.bucket, env: prod }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const prod = { account: ‘123456789012’, region: ‘us-east-1’ };

const stack1 = new StackThatProvidesABucket(app, ‘Stack1’, { env: prod });

// stack2 will take a property { bucket: IBucket } const stack2 = new StackThatExpectsABucket(app, ‘Stack2’, { bucket: stack1.bucket, env: prod }); —

Python:: + [source,python,subs=“verbatim,attributes”] — prod = core.Environment(account=“123456789012”, region=“us-east-1”)

stack1 = StackThatProvidesABucket(app, “Stack1”, env=prod)

= stack2 will take a property “bucket”

## stack2 = StackThatExpectsABucket(app, “Stack2”, bucket=stack1.bucket, env=prod)

Java:: + [source,java,subs=“verbatim,attributes”] — // Helper method to build an environment static Environment makeEnv(String account, String region) { return Environment.builder().account(account).region(region) .build(); }

App app = new App();

Environment prod = makeEnv(“123456789012”, “us-east-1”);

StackThatProvidesABucket stack1 = new StackThatProvidesABucket(app, “Stack1”, StackProps.builder().env(prod).build());

// stack2 will take an argument “bucket” StackThatExpectsABucket stack2 = new StackThatExpectsABucket(app, “Stack,”, StackProps.builder().env(prod).build(), stack1.bucket); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Amazon.CDK.Environment makeEnv(string account, string region) { return new Amazon.CDK.Environment { Account = account, Region = region }; }

var prod = makeEnv(account: “123456789012”, region: “us-east-1”);

var stack1 = new StackThatProvidesABucket(app, “Stack1”, new StackProps { Env = prod });

// stack2 will take a property “bucket” var stack2 = new StackThatExpectsABucket(app, “Stack2”, new StackProps { Env = prod, bucket = stack1.Bucket}); — ====

If the {aws} CDK determines that the resource is in the same environment, but in a different stack, it automatically synthesizes {aws} CloudFormation https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-exports.html[exports] in the producing stack and an https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference-importvalue.html[Fn::ImportValue] in the consuming stack to transfer that information from one stack to the other.

[#resources-deadlock] ==== Resolving dependency deadlocks

Referencing a resource from one stack in a different stack creates a dependency between the two stacks. This makes sure that they’re deployed in the right order. After the stacks are deployed, this dependency is concrete. After that, removing the use of the shared resource from the consuming stack can cause an unexpected deployment failure. This happens if there is another dependency between the two stacks that force them to be deployed in the same order. It can also happen without a dependency if the producing stack is simply chosen by the CDK Toolkit to be deployed first. The {aws} CloudFormation export is removed from the producing stack because it’s no longer needed, but the exported resource is still being used in the consuming stack because its update is not yet deployed. Therefore, deploying the producer stack fails.

To break this deadlock, remove the use of the shared resource from the consuming stack. (This removes the automatic export from the producing stack.) Next, manually add the same export to the producing stack using exactly the same logical ID as the automatically generated export. Remove the use of the shared resource in the consuming stack and deploy both stacks. Then, remove the manual export (and the shared resource if it’s no longer needed) and deploy both stacks again. The stack’s https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#exportwbrvalueexportedvalue-options[exportValue()] method is a convenient way to create the manual export for this purpose. (See the example in the linked method reference.)

[#resources-external] === Referencing resources in your {aws} account

Suppose you want to use a resource already available in your {aws} account in your {aws} CDK app. This might be a resource that was defined through the console, an {aws} SDK, directly with {aws} CloudFormation, or in a different {aws} CDK application. You can turn the resource’s ARN (or another identifying attribute, or group of attributes) into a proxy object. The proxy object serves as a reference to the resource by calling a static factory method on the resource’s class.

When you create such a proxy, the external resource *does not* become a part of your {aws} CDK app. Therefore, changes you make to the proxy in your {aws} CDK app do not affect the deployed resource. The proxy can, however, be passed to any {aws} CDK method that requires a resource of that type.

The following example shows how to reference a bucket based on an existing bucket with the ARN arn:aws:s3:::amzn-s3-demo-bucket1, and an Amazon Virtual Private Cloud based on an existing VPC having a specific ID.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // Construct a proxy for a bucket by its name (must be same account) s3.Bucket.fromBucketName(this, ‘MyBucket’, ‘amzn-s3-demo-bucket1’);

// Construct a proxy for a bucket by its full ARN (can be another account) s3.Bucket.fromBucketArn(this, ‘MyBucket’, ‘arn:aws:s3:::amzn-s3-demo-bucket1’);

// Construct a proxy for an existing VPC from its attribute(s) ec2.Vpc.fromVpcAttributes(this, ‘MyVpc’, { vpcId: ‘vpc-1234567890abcde’, }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // Construct a proxy for a bucket by its name (must be same account) s3.Bucket.fromBucketName(this, ‘MyBucket’, ‘amzn-s3-demo-bucket1’);

// Construct a proxy for a bucket by its full ARN (can be another account) s3.Bucket.fromBucketArn(this, ‘MyBucket’, ‘arn:aws:s3:::amzn-s3-demo-bucket1’);

// Construct a proxy for an existing VPC from its attribute(s) ec2.Vpc.fromVpcAttributes(this, ‘MyVpc’, { vpcId: ‘vpc-1234567890abcde’ }); —

Python:: + [source,python,subs=“verbatim,attributes”] —

= Construct a proxy for a bucket by its name (must be same account)

s3.Bucket.from\_bucket\_name(self, “MyBucket”, “amzn-s3-demo-bucket1”)

= Construct a proxy for a bucket by its full ARN (can be another account)

s3.Bucket.from\_bucket\_arn(self, “MyBucket”, “arn:aws:s3:::amzn-s3-demo-bucket1”)

= Construct a proxy for an existing VPC from its attribute(s)

## ec2.Vpc.from\_vpc\_attributes(self, “MyVpc”, vpc\_id=“vpc-1234567890abcdef”)

Java:: + [source,java,subs=“verbatim,attributes”] — // Construct a proxy for a bucket by its name (must be same account) Bucket.fromBucketName(this, “MyBucket”, “amzn-s3-demo-bucket1”);

// Construct a proxy for a bucket by its full ARN (can be another account) Bucket.fromBucketArn(this, “MyBucket”, “arn:aws:s3:::amzn-s3-demo-bucket1”);

// Construct a proxy for an existing VPC from its attribute(s) Vpc.fromVpcAttributes(this, “MyVpc”, VpcAttributes.builder() .vpcId(“vpc-1234567890abcdef”).build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // Construct a proxy for a bucket by its name (must be same account) Bucket.FromBucketName(this, “MyBucket”, “amzn-s3-demo-bucket1”);

// Construct a proxy for a bucket by its full ARN (can be another account) Bucket.FromBucketArn(this, “MyBucket”, “arn:aws:s3:::amzn-s3-demo-bucket1”);

// Construct a proxy for an existing VPC from its attribute(s) Vpc.FromVpcAttributes(this, “MyVpc”, new VpcAttributes { VpcId = “vpc-1234567890abcdef” }); —

Go:: + [source,go,subs=“verbatim,attributes”] — // Define a proxy for a bucket by its name (must be same account) s3.Bucket\_FromBucketName(stack, jsii.String(“MyBucket”), jsii.String(“amzn-s3-demo-bucket1”))

// Define a proxy for a bucket by its full ARN (can be another account) s3.Bucket\_FromBucketArn(stack, jsii.String(“MyBucket”), jsii.String(“arn:aws:s3:::amzn-s3-demo-bucket1”))

// Define a proxy for an existing VPC from its attributes ec2.Vpc\_FromVpcAttributes(stack, jsii.String(“MyVpc”), &ec2.VpcAttributes{ VpcId: jsii.String(“vpc-1234567890abcde”), }) — ====

Let’s take a closer look at the https://docs.aws.amazon.com/cdk/api/v1/docs/@aws-cdk\_aws-ec2.Vpc.html#static-fromwbrlookupscope-id-options [`Vpc.fromLookup()`] method. Because the ec2.Vpc construct is complex, there are many ways you might want to select the VPC to be used with your CDK app. To address this, the VPC construct has a fromLookup static method (Python: from\_lookup) that lets you look up the desired Amazon VPC by querying your {aws} account at synthesis time.

To use Vpc.fromLookup(), the system that synthesizes the stack must have access to the account that owns the Amazon VPC. This is because the CDK Toolkit queries the account to find the right Amazon VPC at synthesis time.

Furthermore, Vpc.fromLookup() works only in stacks that are defined with an explicit *account* and *region* (see xref:environments[Environments for the {aws} CDK]). If the {aws} CDK tries to look up an Amazon VPC from an xref:stack-api[environment-agnostic stack], the CDK Toolkit doesn’t know which environment to query to find the VPC.

You must provide Vpc.fromLookup() attributes sufficient to uniquely identify a VPC in your {aws} account. For example, there can only ever be one default VPC, so it’s sufficient to specify the VPC as the default.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — ec2.Vpc.fromLookup(this, ‘DefaultVpc’, { isDefault: true }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — ec2.Vpc.fromLookup(this, ‘DefaultVpc’, { isDefault: true }); —

Python:: + [source,python,subs=“verbatim,attributes”] — ec2.Vpc.from\_lookup(self, “DefaultVpc”, is\_default=True) —

Java:: + [source,java,subs=“verbatim,attributes”] — Vpc.fromLookup(this, “DefaultVpc”, VpcLookupOptions.builder() .isDefault(true).build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Vpc.FromLookup(this, id = “DefaultVpc”, new VpcLookupOptions { IsDefault = true }); —

Go:: + [source,go,subs=“verbatim,attributes”] — ec2.Vpc\_FromLookup(this, jsii.String(“DefaultVpc”), &ec2.VpcLookupOptions{ IsDefault: jsii.Bool(true), }) — ====

You can also use the tags property to query for VPCs by tag. You can add tags to the Amazon VPC at the time of its creation by using {aws} CloudFormation or the {aws} CDK. You can edit tags at any time after creation by using the {aws} Management Console, the {aws} CLI, or an {aws} SDK. In addition to any tags you add yourself, the {aws} CDK automatically adds the following tags to all VPCs it creates.

* *Name* – The name of the VPC.
* *aws-cdk:subnet-name* – The name of the subnet.
* *aws-cdk:subnet-type* – The type of the subnet: Public, Private, or Isolated.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — ec2.Vpc.fromLookup(this, ‘PublicVpc’, {tags: {‘aws-cdk:subnet-type’: “Public”}}); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — ec2.Vpc.fromLookup(this, ‘PublicVpc’, {tags: {‘aws-cdk:subnet-type’: “Public”}}); —

Python:: + [source,python,subs=“verbatim,attributes”] — ec2.Vpc.from\_lookup(self, “PublicVpc”, tags={“aws-cdk:subnet-type”: “Public”}) —

Java:: + [source,java,subs=“verbatim,attributes”] — Vpc.fromLookup(this, “PublicVpc”, VpcLookupOptions.builder() .tags(java.util.Map.of(“aws-cdk:subnet-type”, “Public”)) // Java 9 or later .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Vpc.FromLookup(this, id: “PublicVpc”, new VpcLookupOptions { Tags = new Dictionary<string, string> { [“aws-cdk:subnet-type”] = “Public” } }); —

Go:: + [source,go,subs=“verbatim,attributes”] — ec2.Vpc\_FromLookup(this, jsii.String(“DefaultVpc”), &ec2.VpcLookupOptions{ Tags: &map[string]\*string{“aws-cdk:subnet-type”: jsii.String(“Public”)}, }) — ====

Results of Vpc.fromLookup() are cached in the project’s cdk.context.json file. (See xref:context[Context values and the {aws} CDK].) Commit this file to version control so that your app will continue to refer to the same Amazon VPC. This works even if you later change the attributes of your VPCs in a way that would result in a different VPC being selected. This is particularly important if you’re deploying the stack in an environment that doesn’t have access to the {aws} account that defines the VPC, such as xref:cdk-pipeline[CDK Pipelines].

Although you can use an external resource anywhere you’d use a similar resource defined in your {aws} CDK app, you cannot modify it. For example, calling addToResourcePolicy (Python: add\_to\_resource\_policy) on an external s3.Bucket does nothing.

[#resources-physical-names] == Resource physical names

The logical names of resources in {aws} CloudFormation are different from the names of resources that are shown in the {aws} Management Console after they’re deployed by {aws} CloudFormation. The {aws} CDK calls these final names *physical names*.

For example, {aws} CloudFormation might create the Amazon S3 bucket with the logical ID Stack2MyBucket4DD88B4F and the physical name stack2MyBucket4dd88b4f-iuv1rbv9z3to.

You can specify a physical name when creating constructs that represent resources by using the property <resourceType>Name. The following example creates an Amazon S3 bucket with the physical name amzn-s3-demo-bucket.

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

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

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

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

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

Go:: + [source,go,subs=“verbatim,attributes”] — bucket := s3.NewBucket(this, jsii.String(“MyBucket”), &s3.BucketProps{ BucketName: jsii.String(“amzn-s3-demo-bucket”), }) — ====

Assigning physical names to resources has some disadvantages in {aws} CloudFormation. Most importantly, any changes to deployed resources that require a resource replacement, such as changes to a resource’s properties that are immutable after creation, will fail if a resource has a physical name assigned. If you end up in that state, the only solution is to delete the {aws} CloudFormation stack, then deploy the {aws} CDK app again. See the https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-name.html[{aws} CloudFormation documentation] for details.

In some cases, such as when creating an {aws} CDK app with cross-environment references, physical names are required for the {aws} CDK to function correctly. In those cases, if you don’t want to bother with coming up with a physical name yourself, you can let the {aws} CDK name it for you. To do so, use the special value PhysicalName.GENERATE\_IF\_NEEDED, as follows.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new s3.Bucket(this, ‘MyBucket’, { bucketName: core.PhysicalName.GENERATE\_IF\_NEEDED, }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new s3.Bucket(this, ‘MyBucket’, { bucketName: core.PhysicalName.GENERATE\_IF\_NEEDED }); —

Python:: + [source,python,subs=“verbatim,attributes”] — bucket = s3.Bucket(self, “MyBucket”, bucket\_name=core.PhysicalName.GENERATE\_IF\_NEEDED) —

Java:: + [source,java,subs=“verbatim,attributes”] — Bucket bucket = Bucket.Builder.create(this, “MyBucket”) .bucketName(PhysicalName.GENERATE\_IF\_NEEDED).build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var bucket = new Bucket(this, “MyBucket”, new BucketProps { BucketName = PhysicalName.GENERATE\_IF\_NEEDED }); —

Go:: + [source,go,subs=“verbatim,attributes”] — bucket := s3.NewBucket(this, jsii.String(“MyBucket”), &s3.BucketProps{ BucketName: awscdk.PhysicalName\_GENERATE\_IF\_NEEDED(), }) — ====

[#resources-identifiers] == Passing unique resource identifiers

Whenever possible, you should pass resources by reference, as described in the previous section. However, there are cases where you have no other choice but to refer to a resource by one of its attributes. Example use cases include the following:

* When you are using low-level {aws} CloudFormation resources.
* When you need to expose resources to the runtime components of an {aws} CDK application, such as when referring to Lambda functions through environment variables.

These identifiers are available as attributes on the resources, such as the following.

==== [role=“tablist”] TypeScript:: + [source,none,subs=“verbatim,attributes”] — bucket.bucketName lambdaFunc.functionArn securityGroup.groupArn —

JavaScript:: + [source,none,subs=“verbatim,attributes”] — bucket.bucketName lambdaFunc.functionArn securityGroup.groupArn —

Python:: + [source,none,subs=“verbatim,attributes”] — bucket.bucket\_name lambda\_func.function\_arn security\_group\_arn —

Java:: The Java {aws} CDK binding uses getter methods for attributes. + [source,java,subs=“verbatim,attributes”] — bucket.getBucketName() lambdaFunc.getFunctionArn() securityGroup.getGroupArn() —

C#:: + [source,none,subs=“verbatim,attributes”] — bucket.BucketName lambdaFunc.FunctionArn securityGroup.GroupArn —

Go:: + [source,none,subs=“verbatim,attributes”] — bucket.BucketName() fn.FunctionArn() — ====

The following example shows how to pass a generated bucket name to an {aws} Lambda function.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new s3.Bucket(this, ‘Bucket’);

new lambda.Function(this, ‘MyLambda’, { // … environment: { BUCKET\_NAME: bucket.bucketName, }, }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new s3.Bucket(this, ‘Bucket’);

new lambda.Function(this, ‘MyLambda’, { // … environment: { BUCKET\_NAME: bucket.bucketName } }); —

Python:: + [source,python,subs=“verbatim,attributes”] — bucket = s3.Bucket(self, “Bucket”)

== lambda.Function(self, “MyLambda”, environment=dict(BUCKET\_NAME=bucket.bucket\_name))

Java:: + [source,java,subs=“verbatim,attributes”] — final Bucket bucket = new Bucket(this, “Bucket”);

Function.Builder.create(this, “MyLambda”) .environment(java.util.Map.of( // Java 9 or later “BUCKET\_NAME”, bucket.getBucketName())) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var bucket = new Bucket(this, “Bucket”);

new Function(this, “MyLambda”, new FunctionProps { Environment = new Dictionary<string, string> { [“BUCKET\_NAME”] = bucket.BucketName } }); —

Go:: + [source,go,subs=“verbatim,attributes”] — bucket := s3.NewBucket(this, jsii.String(“Bucket”), &s3.BucketProps{}) lambda.NewFunction(this, jsii.String(“MyLambda”), &lambda.FunctionProps{ Environment: &map[string]\*string{“BUCKET\_NAME”: bucket.BucketName()}, }) — ====

[#resources-grants] == Granting permissions between resources

Higher-level constructs make least-privilege permissions achievable by offering simple, intent-based APIs to express permission requirements. For example, many L2 constructs offer grant methods that you can use to grant an entity (such as an IAM role or user) permission to work with the resource, without having to manually create IAM permission statements.

The following example creates the permissions to allow a Lambda function’s execution role to read and write objects to a particular Amazon S3 bucket. If the Amazon S3 bucket is encrypted with an {aws} KMS key, this method also grants permissions to the Lambda function’s execution role to decrypt with the key.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — if (bucket.grantReadWrite(func).success) { // … } —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — if ( bucket.grantReadWrite(func).success) { // … } —

Python:: + [source,python,subs=“verbatim,attributes”] — if bucket.grant\_read\_write(func).success: # … —

Java:: + [source,java,subs=“verbatim,attributes”] — if (bucket.grantReadWrite(func).getSuccess()) { // … } —

C#:: + [source,csharp,subs=“verbatim,attributes”] — if (bucket.GrantReadWrite(func).Success) { // … } —

Go:: + [source,go,subs=“verbatim,attributes”] — if \*bucket.GrantReadWrite(function, nil).Success() { // … } — ====

The grant methods return an iam.Grant object. Use the success attribute of the Grant object to determine whether the grant was effectively applied (for example, it may not have been applied on xref:resources-referencing[external resources]). You can also use the assertSuccess (Python: assert\_success) method of the Grant object to enforce that the grant was successfully applied.

If a specific grant method isn’t available for the particular use case, you can use a generic grant method to define a new grant with a specified list of actions.

The following example shows how to grant a Lambda function access to the Amazon DynamoDB CreateBackup action.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — table.grant(func, ‘dynamodb:CreateBackup’); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — table.grant(func, ‘dynamodb:CreateBackup’); —

Python:: + [source,python,subs=“verbatim,attributes”] — table.grant(func, “dynamodb:CreateBackup”) —

Java:: + [source,java,subs=“verbatim,attributes”] — table.grant(func, “dynamodb:CreateBackup”); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — table.Grant(func, “dynamodb:CreateBackup”); —

Go:: + [source,go,subs=“verbatim,attributes”] — table := dynamodb.NewTable(this, jsii.String(“MyTable”), &dynamodb.TableProps{}) table.Grant(function, jsii.String(“dynamodb:CreateBackup”)) — ====

Many resources, such as Lambda functions, require a role to be assumed when executing code. A configuration property enables you to specify an iam.IRole. If no role is specified, the function automatically creates a role specifically for this use. You can then use grant methods on the resources to add statements to the role.

The grant methods are built using lower-level APIs for handling with IAM policies. Policies are modeled as https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyDocument.html[PolicyDocument] objects. Add statements directly to roles (or a construct’s attached role) using the addToRolePolicy method (Python: add\_to\_role\_policy), or to a resource’s policy (such as a Bucket policy) using the addToResourcePolicy (Python: add\_to\_resource\_policy) method.

[#resources-metrics] == Resource metrics and alarms

Many resources emit CloudWatch metrics that can be used to set up monitoring dashboards and alarms. Higher-level constructs have metric methods that let you access the metrics without looking up the correct name to use.

The following example shows how to define an alarm when the ApproximateNumberOfMessagesNotVisible of an Amazon SQS queue exceeds 100.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as cw from ‘@aws-cdk/aws-cloudwatch’; import \* as sqs from ‘@aws-cdk/aws-sqs’; import { Duration } from ‘@aws-cdk/core’;

const queue = new sqs.Queue(this, ‘MyQueue’);

const metric = queue.metricApproximateNumberOfMessagesNotVisible({ label: ‘Messages Visible (Approx)’, period: Duration.minutes(5), // … }); metric.createAlarm(this, ‘TooManyMessagesAlarm’, { comparisonOperator: cw.ComparisonOperator.GREATER\_THAN\_THRESHOLD, threshold: 100, // … }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const cw = require(‘@aws-cdk/aws-cloudwatch’); const sqs = require(‘@aws-cdk/aws-sqs’); const { Duration } = require(‘@aws-cdk/core’);

const queue = new sqs.Queue(this, ‘MyQueue’);

const metric = queue.metricApproximateNumberOfMessagesNotVisible({ label: ‘Messages Visible (Approx)’, period: Duration.minutes(5) // … }); metric.createAlarm(this, ‘TooManyMessagesAlarm’, { comparisonOperator: cw.ComparisonOperator.GREATER\_THAN\_THRESHOLD, threshold: 100 // … }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_cloudwatch as cw import aws\_cdk.aws\_sqs as sqs from aws\_cdk.core import Duration

queue = sqs.Queue(self, “MyQueue”) metric = queue.metric\_approximate\_number\_of\_messages\_not\_visible( label=“Messages Visible (Approx)”, period=Duration.minutes(5), # … ) metric.create\_alarm(self, “TooManyMessagesAlarm”, comparison\_operator=cw.ComparisonOperator.GREATER\_THAN\_THRESHOLD, threshold=100, # … ) —

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.core.Duration; import software.amazon.awscdk.services.sqs.Queue; import software.amazon.awscdk.services.cloudwatch.Metric; import software.amazon.awscdk.services.cloudwatch.MetricOptions; import software.amazon.awscdk.services.cloudwatch.CreateAlarmOptions; import software.amazon.awscdk.services.cloudwatch.ComparisonOperator;

Queue queue = new Queue(this, “MyQueue”);

Metric metric = queue .metricApproximateNumberOfMessagesNotVisible(MetricOptions.builder() .label(“Messages Visible (Approx)”) .period(Duration.minutes(5)).build());

metric.createAlarm(this, “TooManyMessagesAlarm”, CreateAlarmOptions.builder() .comparisonOperator(ComparisonOperator.GREATER\_THAN\_THRESHOLD) .threshold(100) // … .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — using cdk = Amazon.CDK; using cw = Amazon.CDK.{aws}.CloudWatch; using sqs = Amazon.CDK.{aws}.SQS;

var queue = new sqs.Queue(this, “MyQueue”); var metric = queue.MetricApproximateNumberOfMessagesNotVisible(new cw.MetricOptions { Label = “Messages Visible (Approx)”, Period = cdk.Duration.Minutes(5), // … }); metric.CreateAlarm(this, “TooManyMessagesAlarm”, new cw.CreateAlarmOptions { ComparisonOperator = cw.ComparisonOperator.GREATER\_THAN\_THRESHOLD, Threshold = 100, // .. }); —

Go:: + [source,go,subs=“verbatim,attributes”] — import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/jsii-runtime-go” cw “github.com/aws/aws-cdk-go/awscdk/v2/awscloudwatch” sqs “github.com/aws/aws-cdk-go/awscdk/v2/awssqs” )

queue := sqs.NewQueue(this, jsii.String(“MyQueue”), &sqs.QueueProps{}) metric := queue.MetricApproximateNumberOfMessagesNotVisible(&cw.MetricOptions{ Label: jsii.String(“Messages Visible (Approx)”), Period: awscdk.Duration\_Minutes(jsii.Number(5)), })

metric.CreateAlarm(this, jsii.String(“TooManyMessagesAlarm”), &cw.CreateAlarmOptions{ ComparisonOperator: cw.ComparisonOperator\_GREATER\_THAN\_THRESHOLD, Threshold: jsii.Number(100), }) — ====

If there is no method for a particular metric, you can use the general metric method to specify the metric name manually.

Metrics can also be added to CloudWatch dashboards. See https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_cloudwatch-readme.html[CloudWatch].

[#resources-traffic] == Network traffic

In many cases, you must enable permissions on a network for an application to work, such as when the compute infrastructure needs to access the persistence layer. Resources that establish or listen for connections expose methods that enable traffic flows, including setting security group rules or network ACLs.

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ec2.IConnectable.html[IConnectable] resources have a connections property that is the gateway to network traffic rules configuration.

You enable data to flow on a given network path by using allow methods. The following example enables HTTPS connections to the web and incoming connections from the Amazon EC2 Auto Scaling group fleet2.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as asg from ‘@aws-cdk/aws-autoscaling’; import \* as ec2 from ‘@aws-cdk/aws-ec2’;

const fleet1: asg.AutoScalingGroup = asg.AutoScalingGroup(/*…*/);

// Allow surfing the (secure) web fleet1.connections.allowTo(new ec2.Peer.anyIpv4(), new ec2.Port({ fromPort: 443, toPort: 443 }));

const fleet2: asg.AutoScalingGroup = asg.AutoScalingGroup(/*…*/); fleet1.connections.allowFrom(fleet2, ec2.Port.AllTraffic()); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const asg = require(‘@aws-cdk/aws-autoscaling’); const ec2 = require(‘@aws-cdk/aws-ec2’);

const fleet1 = asg.AutoScalingGroup();

// Allow surfing the (secure) web fleet1.connections.allowTo(new ec2.Peer.anyIpv4(), new ec2.Port({ fromPort: 443, toPort: 443 }));

const fleet2 = asg.AutoScalingGroup(); fleet1.connections.allowFrom(fleet2, ec2.Port.AllTraffic()); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_autoscaling as asg import aws\_cdk.aws\_ec2 as ec2

fleet1 = asg.AutoScalingGroup( … )

= Allow surfing the (secure) web

fleet1.connections.allow\_to(ec2.Peer.any\_ipv4(), ec2.Port(PortProps(from\_port=443, to\_port=443)))

fleet2 = asg.AutoScalingGroup( … ) fleet1.connections.allow\_from(fleet2, ec2.Port.all\_traffic()) —

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.services.autoscaling.AutoScalingGroup; import software.amazon.awscdk.services.ec2.Peer; import software.amazon.awscdk.services.ec2.Port;

AutoScalingGroup fleet1 = AutoScalingGroup.Builder.create(this, “MyFleet”) /\* … \*/.build();

// Allow surfing the (secure) Web fleet1.getConnections().allowTo(Peer.anyIpv4(), Port.Builder.create().fromPort(443).toPort(443).build());

AutoScalingGroup fleet2 = AutoScalingGroup.Builder.create(this, “MyFleet2”) /\* … \*/.build(); fleet1.getConnections().allowFrom(fleet2, Port.allTraffic()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — using cdk = Amazon.CDK; using asg = Amazon.CDK.AWS.AutoScaling; using ec2 = Amazon.CDK.AWS.EC2;

// Allow surfing the (secure) Web var fleet1 = new asg.AutoScalingGroup(this, “MyFleet”, new asg.AutoScalingGroupProps { /\* … \*/ }); fleet1.Connections.AllowTo(ec2.Peer.AnyIpv4(), new ec2.Port(new ec2.PortProps { FromPort = 443, ToPort = 443 }));

var fleet2 = new asg.AutoScalingGroup(this, “MyFleet2”, new asg.AutoScalingGroupProps { /\* … \*/ }); fleet1.Connections.AllowFrom(fleet2, ec2.Port.AllTraffic()); —

Go:: + [source,go,subs=“verbatim,attributes”] — import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/jsii-runtime-go” autoscaling “github.com/aws/aws-cdk-go/awscdk/v2/awsautoscaling” ec2 “github.com/aws/aws-cdk-go/awscdk/v2/awsec2” )

fleet1 := autoscaling.NewAutoScalingGroup(this, jsii.String(“MyFleet1”), &autoscaling.AutoScalingGroupProps{}) fleet1.Connections().AllowTo(ec2.Peer\_AnyIpv4(),ec2.NewPort(&ec2.PortProps{ FromPort: jsii.Number(443), ToPort: jsii.Number(443) }),jsii.String(“secure web”))

fleet2 := autoscaling.NewAutoScalingGroup(this, jsii.String(“MyFleet2”), &autoscaling.AutoScalingGroupProps{}) fleet1.Connections().AllowFrom(fleet2, ec2.Port\_AllTraffic(),jsii.String(“all traffic”)) — ====

Certain resources have default ports associated with them. Examples include the listener of a load balancer on the public port, and the ports on which the database engine accepts connections for instances of an Amazon RDS database. In such cases, you can enforce tight network control without having to manually specify the port. To do so, use the allowDefaultPortFrom and allowToDefaultPort methods (Python: allow\_default\_port\_from, allow\_to\_default\_port).

The following example shows how to enable connections from any IPV4 address, and a connection from an Auto Scaling group to access a database.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — listener.connections.allowDefaultPortFromAnyIpv4(‘Allow public access’);

== fleet.connections.allowToDefaultPort(rdsDatabase, ‘Fleet can access database’);

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — listener.connections.allowDefaultPortFromAnyIpv4(‘Allow public access’);

== fleet.connections.allowToDefaultPort(rdsDatabase, ‘Fleet can access database’);

Python:: + [source,python,subs=“verbatim,attributes”] — listener.connections.allow\_default\_port\_from\_any\_ipv4(“Allow public access”)

== fleet.connections.allow\_to\_default\_port(rds\_database, “Fleet can access database”)

Java:: + [source,java,subs=“verbatim,attributes”] — listener.getConnections().allowDefaultPortFromAnyIpv4(“Allow public access”);

== fleet.getConnections().AllowToDefaultPort(rdsDatabase, “Fleet can access database”);

C#:: + [source,csharp,subs=“verbatim,attributes”] — listener.Connections.AllowDefaultPortFromAnyIpv4(“Allow public access”);

== fleet.Connections.AllowToDefaultPort(rdsDatabase, “Fleet can access database”);

Go:: + [source,go,subs=“verbatim,attributes”] — listener.Connections().AllowDefaultPortFromAnyIpv4(jsii.String(“Allow public Access”)) fleet.Connections().AllowToDefaultPort(rdsDatabase, jsii.String(“Fleet can access database”)) — ====

[#resources-events] == Event handling

Some resources can act as event sources. Use the addEventNotification method (Python: add\_event\_notification) to register an event target to a particular event type emitted by the resource. In addition to this, addXxxNotification methods offer a simple way to register a handler for common event types.

The following example shows how to trigger a Lambda function when an object is added to an Amazon S3 bucket.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as s3nots from ‘@aws-cdk/aws-s3-notifications’;

const handler = new lambda.Function(this, ‘Handler’, { /*…*/ }); const bucket = new s3.Bucket(this, ‘Bucket’); bucket.addObjectCreatedNotification(new s3nots.LambdaDestination(handler)); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const s3nots = require(‘@aws-cdk/aws-s3-notifications’);

const handler = new lambda.Function(this, ‘Handler’, { /*…*/ }); const bucket = new s3.Bucket(this, ‘Bucket’); bucket.addObjectCreatedNotification(new s3nots.LambdaDestination(handler)); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_s3\_notifications as s3\_nots

handler = lambda\_.Function(self, “Handler”, …) bucket = s3.Bucket(self, “Bucket”) bucket.add\_object\_created\_notification(s3\_nots.LambdaDestination(handler)) —

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.services.s3.Bucket; import software.amazon.awscdk.services.lambda.Function; import software.amazon.awscdk.services.s3.notifications.LambdaDestination;

Function handler = Function.Builder.create(this, “Handler”)/\* … \*/.build(); Bucket bucket = new Bucket(this, “Bucket”); bucket.addObjectCreatedNotification(new LambdaDestination(handler)); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — using lambda = Amazon.CDK.{aws}.Lambda; using s3 = Amazon.CDK.{aws}.S3; using s3Nots = Amazon.CDK.{aws}.S3.Notifications;

var handler = new lambda.Function(this, “Handler”, new lambda.FunctionProps { .. }); var bucket = new s3.Bucket(this, “Bucket”); bucket.AddObjectCreatedNotification(new s3Nots.LambdaDestination(handler)); —

Go:: + [source,go,subs=“verbatim,attributes”] — import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/jsii-runtime-go” s3 “github.com/aws/aws-cdk-go/awscdk/v2/awss3” s3nots “github.com/aws/aws-cdk-go/awscdk/v2/awss3notifications”  
)

handler := lambda.NewFunction(this, jsii.String(“MyFunction”), &lambda.FunctionProps{}) bucket := s3.NewBucket(this, jsii.String(“Bucket”), &s3.BucketProps{}) bucket.AddObjectCreatedNotification(s3nots.NewLambdaDestination(handler), nil) — ====

[#resources-removal] == Removal policies

Resources that maintain persistent data, such as databases, Amazon S3 buckets, and Amazon ECR registries, have a *removal policy*. The removal policy indicates whether to delete persistent objects when the {aws} CDK stack that contains them is destroyed. The values specifying the removal policy are available through the RemovalPolicy enumeration in the {aws} CDK core module.

= [NOTE]

Resources besides those that store data persistently might also have a removalPolicy that is used for a different purpose. For example, a Lambda function version uses a removalPolicy attribute to determine whether a given version is retained when a new version is deployed. These have different meanings and defaults compared to the removal policy on an Amazon S3 bucket or DynamoDB table.

====

[cols=“1,1”, options=“header”] |=== |Value |Meaning

|=== | RemovalPolicy.RETAIN | Keep the contents of the resource when destroying the stack (default). The resource is orphaned from the stack and must be deleted manually. If you attempt to re-deploy the stack while the resource still exists, you will receive an error message due to a name conflict. |===

|=== | RemovalPolicy.DESTROY | The resource will be destroyed along with the stack. |===

{aws} CloudFormation does not remove Amazon S3 buckets that contain files even if their removal policy is set to DESTROY. Attempting to do so is an {aws} CloudFormation error. To have the {aws} CDK delete all files from the bucket before destroying it, set the bucket’s autoDeleteObjects property to true.

Following is an example of creating an Amazon S3 bucket with RemovalPolicy of DESTROY and autoDeleteOjbects set to true.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘@aws-cdk/core’; import \* as s3 from ‘@aws-cdk/aws-s3’;

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

const bucket = new s3.Bucket(this, ‘Bucket’, { removalPolicy: cdk.RemovalPolicy.DESTROY, autoDeleteObjects: true }); } } —-

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const cdk = require(‘@aws-cdk/core’); const s3 = require(‘@aws-cdk/aws-s3’);

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

const bucket = new s3.Bucket(this, ‘Bucket’, { removalPolicy: cdk.RemovalPolicy.DESTROY, autoDeleteObjects: true }); } }

== module.exports = { CdkTestStack }

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.core as cdk import aws\_cdk.aws\_s3 as s3

class CdkTestStack(cdk.stack): def *init*(self, scope: cdk.Construct, id: str, **kwargs): super().*init*(scope, id,** kwargs)

bucket = s3.Bucket(self, "Bucket",  
 removal\_policy=cdk.RemovalPolicy.DESTROY,  
 auto\_delete\_objects=True) ----

Java:: + [source,java,subs=“verbatim,attributes”] — software.amazon.awscdk.core.\_; import software.amazon.awscdk.services.s3.\_;

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

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

Bucket.Builder.create(this, "Bucket")  
 .removalPolicy(RemovalPolicy.DESTROY)  
 .autoDeleteObjects(true).build();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Amazon.CDK.{aws}.S3;

public CdkTestStack(Construct scope, string id, IStackProps props) : base(scope, id, props) { new Bucket(this, “Bucket”, new BucketProps { RemovalPolicy = RemovalPolicy.DESTROY, AutoDeleteObjects = true }); } —

Go:: + [source,go,subs=“verbatim,attributes”] — import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/jsii-runtime-go” s3 “github.com/aws/aws-cdk-go/awscdk/v2/awss3” )

s3.NewBucket(this, jsii.String(“Bucket”), &s3.BucketProps{ RemovalPolicy: awscdk.RemovalPolicy\_DESTROY, AutoDeleteObjects: jsii.Bool(true), }) — ====

You can also apply a removal policy directly to the underlying {aws} CloudFormation resource via the applyRemovalPolicy() method. This method is available on some stateful resources that do not have a removalPolicy property in their L2 resource’s props. Examples include the following:

* {aws} CloudFormation stacks
* Amazon Cognito user pools
* Amazon DocumentDB database instances
* Amazon EC2 volumes
* Amazon OpenSearch Service domains
* Amazon FSx file systems
* Amazon SQS queues

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const resource = bucket.node.findChild(‘Resource’) as cdk.CfnResource; resource.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const resource = bucket.node.findChild(‘Resource’); resource.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY); —

Python:: + [source,python,subs=“verbatim,attributes”] — resource = bucket.node.find\_child(‘Resource’) resource.apply\_removal\_policy(cdk.RemovalPolicy.DESTROY); —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnResource resource = (CfnResource)bucket.node.findChild(“Resource”); resource.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY); —

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

var resource = (CfnResource)bucket.node.findChild(‘Resource’); resource.ApplyRemovalPolicy(cdk.RemovalPolicy.DESTROY); — ====

= [NOTE]

The {aws} CDK’s RemovalPolicy translates to {aws} CloudFormation’s DeletionPolicy. However, the default in {aws} CDK is to retain the data, which is the opposite of the {aws} CloudFormation default.

====