:doctype: book

include::attributes.txt[]

// Attributes [.topic] :info\_titleabbrev: Constructs :keywords: {aws} CDK, {aws} CloudFormation, IaC, Infrastructure as code, constructs

[#constructs] = {aws} CDK Constructs

== [abstract]

## Constructs are the basic building blocks of {aws} Cloud Development Kit ({aws} CDK) applications. A construct is a component within your application that represents one or more {aws} CloudFormation resources and their configuration. You build your application, piece by piece, by importing and configuring constructs.

// Content start

Constructs are the basic building blocks of {aws} Cloud Development Kit ({aws} CDK) applications. A construct is a component within your application that represents one or more {aws} CloudFormation resources and their configuration. You build your application, piece by piece, by importing and configuring constructs.

[#constructs-import] == Import and use constructs

Constructs are classes that you import into your CDK applications from the xref:libraries-construct[{aws} Construct Library]. You can also create and distribute your own constructs, or use constructs created by third-party developers.

Constructs are part of the Construct Programming Model (CPM). They are available to use with other tools such as CDK for [.noloc]Terraform (CDKtf), CDK for [.noloc]Kubernetes (CDK8s), and [.noloc]Projen.

Numerous third parties have also published constructs compatible with the {aws} CDK. Visit https://constructs.dev/search?q=&cdk=aws-cdk&cdkver=2&offset=0[Construct Hub] to explore the {aws} CDK construct partner ecosystem.

[#constructs-lib-levels] == Construct levels

Constructs from the {aws} Construct Library are categorized into three levels. Each level offers an increasing level of abstraction. The higher the abstraction, the easier to configure, requiring less expertise. The lower the abstraction, the more customization available, requiring more expertise.

[#constructs-lib-levels-one] *Level 1 (L1) constructs*:: L1 constructs, also known as *CFN resources*, are the lowest-level construct and offer no abstraction. Each L1 construct maps directly to a single {aws} CloudFormation resource. With L1 constructs, you import a construct that represents a specific {aws} CloudFormation resource. You then define the resource+'s properties within your construct instance. + L1 constructs are great to use when you are familiar with {aws} CloudFormation and need complete control over defining your {aws} resource properties. + In the {aws} Construct Library, L1 constructs are named starting with +Cfn+, followed by an identifier for the {aws} CloudFormation resource that it represents. For example, the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.CfnBucket.html[+CfnBucket+] construct is an L1 construct that represents an https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html[+{aws}::S3::Bucket`] {aws} CloudFormation resource. + L1 constructs are generated from the https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-resource-specification.html[{aws} CloudFormation resource specification]. If a resource exists in {aws} CloudFormation, it’ll be available in the {aws} CDK as an L1 construct. New resources or properties may take up to a week to become available in the {aws} Construct Library. For more information, see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html[{aws} resource and property types reference] in the *{aws} CloudFormation User Guide*.

[#constructs-lib-levels-two] *Level 2 (L2) constructs*:: L2 constructs, also known as *curated* constructs, are thoughtfully developed by the CDK team and are usually the most widely used construct type. L2 constructs map directly to single {aws} CloudFormation resources, similar to L1 constructs. Compared to L1 constructs, L2 constructs provide a higher-level abstraction through an intuitive intent-based API. L2 constructs include sensible default property configurations, best practice security policies, and generate a lot of the boilerplate code and glue logic for you. + L2 constructs also provide helper methods for most resources that make it simpler and quicker to define properties, permissions, event-based interactions between resources, and more. + The https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html[s3.Bucket] class is an example of an L2 construct for an Amazon Simple Storage Service (Amazon S3) bucket resource. + The {aws} Construct Library contains L2 constructs that are designated stable and ready for production use. For L2 constructs under development, they are designated as experimental and offered in a separate module.

[#constructs-lib-levels-three] *Level 3 (L3) constructs*:: L3 constructs, also known as *patterns*, are the highest-level of abstraction. Each L3 construct can contain a collection of resources that are configured to work together to accomplish a specific task or service within your application. L3 constructs are used to create entire {aws} architectures for particular use cases in your application. + To provide complete system designs, or substantial parts of a larger system, L3 constructs offer opinionated default property configurations. They are built around a particular approach toward solving a problem and providing a solution. With L3 constructs, you can create and configure multiple resources quickly, with the fewest amount of input and code. + The https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecs\_patterns.ApplicationLoadBalancedFargateService.html[ecsPatterns.ApplicationLoadBalancedFargateService] class is an example of an L3 construct that represents an {aws} Fargate service running on an Amazon Elastic Container Service (Amazon ECS) cluster and fronted by an application load balancer. + Similar to L2 constructs, L3 constructs that are ready for production use are included in the {aws} Construct Library. Those under development are offered in separate modules.

[#constructs-define,constructs-define.title] == Defining constructs

[#constructs-composition] === Composition

*Composition* is the key pattern for defining higher-level abstractions through constructs. A high-level construct can be composed from any number of lower-level constructs. From a bottom-up perspective, you use constructs to organize the individual {aws} resources that you want to deploy. You use whatever abstractions are convenient for your purpose, with as many levels as you need.

With composition, you define reusable components and share them like any other code. For example, a team can define a construct that implements the company`’s best practice for an Amazon DynamoDB table, including backup, global replication, automatic scaling, and monitoring. The team can share the construct internally with other teams, or publicly.

Teams can use constructs like any other library package. When the library is updated, developers get access to the new version’s improvements and bug fixes, similar to any other code library.

[#constructs-init] === Initialization

Constructs are implemented in classes that extend the https://docs.aws.amazon.com/cdk/api/v2/docs/constructs.Construct.html[Construct] base class. You define a construct by instantiating the class. All constructs take three parameters when they are initialized:

* *scope* – The construct’s parent or owner. This can either be a stack or another construct. Scope determines the construct’s place in the xref:apps-tree[construct tree]. You should usually pass this (self in Python), which represents the current object, for the scope.
* *id* – An xref:identifiers[identifier] that must be unique within the scope. The identifier serves as a namespace for everything that’s defined within the construct. It’s used to generate unique identifiers, such as xref:resources-physical-names[resource names] and {aws} CloudFormation logical IDs.
* Identifiers need only be unique within a scope. This lets you instantiate and reuse constructs without concern for the constructs and identifiers they might contain, and enables composing constructs into higher-level abstractions. In addition, scopes make it possible to refer to groups of constructs all at once. Examples include for https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Tag.html[tagging], or specifying where the constructs will be deployed.
* *props* – A set of properties or keyword arguments, depending on the language, that define the construct`’s initial configuration. Higher-level constructs provide more defaults, and if all prop elements are optional, you can omit the props parameter completely.

[#constructs-config] === Configuration

Most constructs accept props as their third argument (or in Python, keyword arguments), a name/value collection that defines the construct’s configuration. The following example defines a bucket with {aws} Key Management Service ({aws} KMS) encryption and static website hosting enabled. Since it does not explicitly specify an encryption key, the Bucket construct defines a new kms.Key and associates it with the bucket.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new s3.Bucket(this, ‘MyEncryptedBucket’, { encryption: s3.BucketEncryption.KMS, websiteIndexDocument: ‘index.html’ }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new s3.Bucket(this, ‘MyEncryptedBucket’, { encryption: s3.BucketEncryption.KMS, websiteIndexDocument: ‘index.html’ }); —

Python:: + [source,python,subs=“verbatim,attributes”] — s3.Bucket(self, “MyEncryptedBucket”, encryption=s3.BucketEncryption.KMS, website\_index\_document=“index.html”) —

Java:: + [source,java,subs=“verbatim,attributes”] — Bucket.Builder.create(this, “MyEncryptedBucket”) .encryption(BucketEncryption.KMS\_MANAGED) .websiteIndexDocument(“index.html”).build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new Bucket(this, “MyEncryptedBucket”, new BucketProps { Encryption = BucketEncryption.KMS\_MANAGED, WebsiteIndexDocument = “index.html” }); —

Go:: + [source,go,subs=“verbatim,attributes”] — awss3.NewBucket(stack, jsii.String(“MyEncryptedBucket”), &awss3.BucketProps{ Encryption: awss3.BucketEncryption\_KMS, WebsiteIndexDocument: jsii.String(“index.html”), }) — ====

[#constructs-interact] === Interacting with constructs

Constructs are classes that extend the base https://docs.aws.amazon.com/cdk/api/v2/docs/constructs.Construct.html[Construct] class. After you instantiate a construct, the construct object exposes a set of methods and properties that let you interact with the construct and pass it around as a reference to other parts of the system.

The {aws} CDK framework doesn’t put any restrictions on the APIs of constructs. Authors can define any API they want. However, the {aws} constructs that are included with the {aws} Construct Library, such as s3.Bucket, follow guidelines and common patterns. This provides a consistent experience across all {aws} resources.

Most {aws} constructs have a set of xref:permissions-grants[grant] methods that you can use to grant {aws} Identity and Access Management (IAM) permissions on that construct to a principal. The following example grants the IAM group data-science permission to read from the Amazon S3 bucket raw-data.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const rawData = new s3.Bucket(this, ‘raw-data’); const dataScience = new iam.Group(this, ‘data-science’); rawData.grantRead(dataScience); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const rawData = new s3.Bucket(this, ‘raw-data’); const dataScience = new iam.Group(this, ‘data-science’); rawData.grantRead(dataScience); —

Python:: + [source,python,subs=“verbatim,attributes”] — raw\_data = s3.Bucket(self, ‘raw-data’) data\_science = iam.Group(self, ‘data-science’) raw\_data.grant\_read(data\_science) —

Java:: + [source,java,subs=“verbatim,attributes”] — Bucket rawData = new Bucket(this, “raw-data”); Group dataScience = new Group(this, “data-science”); rawData.grantRead(dataScience); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var rawData = new Bucket(this, “raw-data”); var dataScience = new Group(this, “data-science”); rawData.GrantRead(dataScience); —

Go:: + [source,go,subs=“verbatim,attributes”] — rawData := awss3.NewBucket(stack, jsii.String(“raw-data”), nil) dataScience := awsiam.NewGroup(stack, jsii.String(“data-science”), nil) rawData.GrantRead(dataScience, nil) — ====

Another common pattern is for {aws} constructs to set one of the resource’s attributes from data supplied elsewhere. Attributes can include Amazon Resource Names (ARNs), names, or URLs.

The following code defines an {aws} Lambda function and associates it with an Amazon Simple Queue Service (Amazon SQS) queue through the queue’s URL in an environment variable.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const jobsQueue = new sqs.Queue(this, ‘jobs’); const createJobLambda = new lambda.Function(this, ‘create-job’, { runtime: lambda.Runtime.NODEJS\_18\_X, handler: ‘index.handler’, code: lambda.Code.fromAsset(‘./create-job-lambda-code’), environment: { QUEUE\_URL: jobsQueue.queueUrl } }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const jobsQueue = new sqs.Queue(this, ‘jobs’); const createJobLambda = new lambda.Function(this, ‘create-job’, { runtime: lambda.Runtime.NODEJS\_18\_X, handler: ‘index.handler’, code: lambda.Code.fromAsset(‘./create-job-lambda-code’), environment: { QUEUE\_URL: jobsQueue.queueUrl } }); —

Python:: + [source,python,subs=“verbatim,attributes”] — jobs\_queue = sqs.Queue(self, “jobs”) create\_job\_lambda = lambda\_.Function(self, “create-job”, runtime=lambda\_.Runtime.NODEJS\_18\_\_X, handler=“index.handler”, code=lambda\_\_.Code.from\_asset(“./create-job-lambda-code”), environment=dict( QUEUE\_URL=jobs\_queue.queue\_url ) ) —

Java:: + [source,java,subs=“verbatim,attributes”] — final Queue jobsQueue = new Queue(this, “jobs”); Function createJobLambda = Function.Builder.create(this, “create-job”) .handler(“index.handler”) .code(Code.fromAsset(“./create-job-lambda-code”)) .environment(java.util.Map.of( // Map.of is Java 9 or later “QUEUE\_URL”, jobsQueue.getQueueUrl())) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var jobsQueue = new Queue(this, “jobs”); var createJobLambda = new Function(this, “create-job”, new FunctionProps { Runtime = Runtime.NODEJS\_18\_X, Handler = “index.handler”, Code = Code.FromAsset(@“.-job-lambda-code”), Environment = new Dictionary<string, string> { [“QUEUE\_URL”] = jobsQueue.QueueUrl } }); —

Go:: + [source,go,subs=“verbatim,attributes”] — createJobLambda := awslambda.NewFunction(stack, jsii.String(“create-job”), &awslambda.FunctionProps{ Runtime: awslambda.Runtime\_NODEJS\_18\_X(), Handler: jsii.String(“index.handler”), Code: awslambda.Code\_FromAsset(jsii.String(“.-job-lambda-code”), nil), Environment: &map[string]\_\_string{ “QUEUE\_URL”: jsii.String(\_\_jobsQueue.QueueUrl()), }, }) — ====

For information about the most common API patterns in the {aws} Construct Library, see xref:resources[Resources and the {aws} CDK].

[#constructs-apps-stacks] === The app and stack construct

The https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.App.html[App] and https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html[Stack] classes from the {aws} Construct Library are unique constructs. Compared to other constructs, they don’t configure {aws} resources on their own. Instead, they are used to provide context for your other constructs. All constructs that represent {aws} resources must be defined, directly or indirectly, within the scope of a Stack construct. Stack constructs are defined within the scope of an App construct.

To learn more about CDK apps, see xref:apps[{aws} CDK apps]. To learn more about CDK stacks, see xref:stacks[Introduction to {aws} CDK stacks].

The following example defines an app with a single stack. Within the stack, an L2 construct is used to configure an Amazon S3 bucket resource.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import { App, Stack, StackProps } from ‘aws-cdk-lib’; import \* as s3 from ‘aws-cdk-lib/aws-s3’;

class HelloCdkStack extends Stack { constructor(scope: App, id: string, props?: StackProps) { super(scope, id, props);

new s3.Bucket(this, ‘MyFirstBucket’, { versioned: true }); } }

const app = new App(); new HelloCdkStack(app, “HelloCdkStack”); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const { App , Stack } = require(‘aws-cdk-lib’); const s3 = require(‘aws-cdk-lib/aws-s3’);

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

new s3.Bucket(this, ‘MyFirstBucket’, { versioned: true }); } }

const app = new App(); new HelloCdkStack(app, “HelloCdkStack”); —

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import App, Stack import aws\_cdk.aws\_s3 as s3 from constructs import Construct

class HelloCdkStack(Stack):

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

s3.Bucket(self, "MyFirstBucket", versioned=True)

….

app = App() HelloCdkStack(app, “HelloCdkStack”) —

Java:: Stack defined in HelloCdkStack.java file: + [source,java,subs=“verbatim,attributes”] — import software.constructs.Construct; import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.services.s3.\*;

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);

Bucket.Builder.create(this, "MyFirstBucket")  
 .versioned(true).build();

} } —- + ….

App defined in HelloCdkApp.java file: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.App; import software.amazon.awscdk.StackProps;

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

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

app.synth();

} } —- ….

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

namespace HelloCdkApp { internal static class Program { public static void Main(string[] args) { var app = new App(); new HelloCdkStack(app, “HelloCdkStack”); app.Synth(); } }

public class HelloCdkStack : Stack { public HelloCdkStack(Construct scope, string id, IStackProps props=null) : base(scope, id, props) { new Bucket(this, “MyFirstBucket”, new BucketProps { Versioned = true }); } } } —-

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)

…. awss3.NewBucket(stack, jsii.String(“MyFirstBucket”), &awss3.BucketProps{ Versioned: jsii.Bool(true), })

return stack } —- ==== ….

[#constructs-work] == Working with constructs

[#constructs-l1-using] === Working with L1 constructs

L1 constructs map directly to individual {aws} CloudFormation resources. You must provide the resource’s required configuration.

In this example, we create a bucket object using the CfnBucket L1 construct:

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

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

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

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

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

Go:: + [source,go,subs=“verbatim,attributes”] — awss3.NewCfnBucket(stack, jsii.String(“amzn-s3-demo-bucket”), &awss3.CfnBucketProps{ BucketName: jsii.String(“amzn-s3-demo-bucket”), }) — ====

Construct properties that aren’t simple Booleans, strings, numbers, or containers are handled differently in the supported languages.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new s3.CfnBucket(this, “amzn-s3-demo-bucket”, { bucketName: “amzn-s3-demo-bucket”, corsConfiguration: { corsRules: [{ allowedOrigins: [“\*”], allowedMethods: [“GET”] }] } }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const bucket = new s3.CfnBucket(this, “amzn-s3-demo-bucket”, { bucketName: “amzn-s3-demo-bucket”, corsConfiguration: { corsRules: [{ allowedOrigins: [“\*”], allowedMethods: [“GET”] }] } }); —

Python:: In Python, these properties are represented by types defined as inner classes of the L1 construct. For example, the optional property cors\_configuration of a CfnBucket requires a wrapper of type CfnBucket.CorsConfigurationProperty. Here we are defining cors\_configuration on a CfnBucket instance. + [source,python,subs=“verbatim,attributes”] — bucket = CfnBucket(self, “amzn-s3-demo-bucket”, bucket\_name=“amzn-s3-demo-bucket”, cors\_configuration=CfnBucket.CorsConfigurationProperty( cors\_rules=[CfnBucket.CorsRuleProperty( allowed\_origins=[“\*”], allowed\_methods=[“GET”] )] ) ) —

Java:: In Java, these properties are represented by types defined as inner classes of the L1 construct. For example, the optional property corsConfiguration of a CfnBucket requires a wrapper of type CfnBucket.CorsConfigurationProperty. Here we are defining corsConfiguration on a CfnBucket instance. + [source,java,subs=“verbatim,attributes”] — CfnBucket bucket = CfnBucket.Builder.create(this, “amzn-s3-demo-bucket”) .bucketName(“amzn-s3-demo-bucket”) .corsConfiguration(new CfnBucket.CorsConfigurationProperty.Builder() .corsRules(Arrays.asList(new CfnBucket.CorsRuleProperty.Builder() .allowedOrigins(Arrays.asList(“\*“)) .allowedMethods(Arrays.asList(”GET”)) .build())) .build()) .build(); —

C#:: In C#, these properties are represented by types defined as inner classes of the L1 construct. For example, the optional property CorsConfiguration of a CfnBucket requires a wrapper of type CfnBucket.CorsConfigurationProperty. Here we are defining CorsConfiguration on a CfnBucket instance. + [source,csharp,subs=“verbatim,attributes”] — var bucket = new CfnBucket(this, “amzn-s3-demo-bucket”, new CfnBucketProps { BucketName = “amzn-s3-demo-bucket”, CorsConfiguration = new CfnBucket.CorsConfigurationProperty { CorsRules = new object[] { new CfnBucket.CorsRuleProperty { AllowedOrigins = new string[] { “\*” }, AllowedMethods = new string[] { “GET” }, } } } }); —

Go:: In Go, these types are named using the name of the L1 construct, an underscore, and the property name. For example, the optional property CorsConfiguration of a CfnBucket requires a wrapper of type CfnBucket\_CorsConfigurationProperty. Here we are defining CorsConfiguration on a CfnBucket instance. + [source,go,subs=“verbatim,attributes”] — awss3.NewCfnBucket(stack, jsii.String(“amzn-s3-demo-bucket”), &awss3.CfnBucketProps{ BucketName: jsii.String(“amzn-s3-demo-bucket”), CorsConfiguration: &awss3.CfnBucket\_CorsConfigurationProperty{ CorsRules: []awss3.CorsRule{ awss3.CorsRule{ AllowedOrigins: jsii.Strings(“\*“), AllowedMethods: &[]awss3.HttpMethods{”GET”}, }, }, }, }) — ====

= [IMPORTANT]

You can’t use L2 property types with L1 constructs, or vice versa. When working with L1 constructs, always use the types defined for the L1 construct you’re using. Do not use types from other L1 constructs (some may have the same name, but they are not the same type).

Some of our language-specific API references currently have errors in the paths to L1 property types, or don’t document these classes at all. We hope to fix this soon. In the meantime, remember that such types are always inner classes of the L1 construct they are used with.

====

[#constructs-using] === Working with L2 constructs

In the following example, we define an Amazon S3 bucket by creating an object from the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html[Bucket] L2 construct:

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

// “this” is HelloCdkStack new s3.Bucket(this, ‘MyFirstBucket’, { versioned: true }); —

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

// “this” is HelloCdkStack new s3.Bucket(this, ‘MyFirstBucket’, { versioned: true }); —

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

= “self” is HelloCdkStack

## s3.Bucket(self, “MyFirstBucket”, versioned=True)

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

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);

Bucket.Builder.create(this, "MyFirstBucket")  
 .versioned(true).build();

} } —- ….

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

// “this” is HelloCdkStack new Bucket(this, “MyFirstBucket”, new BucketProps { Versioned = true }); —

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

// stack is HelloCdkStack awss3.NewBucket(stack, jsii.String(“MyFirstBucket”), &awss3.BucketProps{ Versioned: jsii.Bool(true), }) — ====

MyFirstBucket is not the name of the bucket that {aws} CloudFormation creates. It is a logical identifier given to the new construct within the context of your CDK app. The https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Resource.html#physicalname[physicalName] value will be used to name the {aws} CloudFormation resource.

[#constructs-work-third] == Working with third-party constructs

https://constructs.dev/search?q=&cdk=aws-cdk&cdkver=2&sort=downloadsDesc&offset=0[Construct Hub] is a resource to help you discover additional constructs from {aws}, third parties, and the open-source CDK community.

[#constructs-author] === Writing your own constructs

In addition to using existing constructs, you can also write your own constructs and let anyone use them in their apps. All constructs are equal in the {aws} CDK. Constructs from the {aws} Construct Library are treated the same as a construct from a third-party library published via [.noloc]NPM, [.noloc]Maven, or [.noloc]PyPI. Constructs published to your company’s internal package repository are also treated in the same way.

To declare a new construct, create a class that extends the https://docs.aws.amazon.com/cdk/api/v2/docs/constructs.Construct.html[Construct] base class, in the constructs package, then follow the pattern for initializer arguments.

The following example shows how to declare a construct that represents an Amazon S3 bucket. The S3 bucket sends an Amazon Simple Notification Service (Amazon SNS) notification every time someone uploads a file into it.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — export interface NotifyingBucketProps { prefix?: string; }

export class NotifyingBucket extends Construct { constructor(scope: Construct, id: string, props: NotifyingBucketProps = {}) { super(scope, id); const bucket = new s3.Bucket(this, ‘bucket’); const topic = new sns.Topic(this, ‘topic’); bucket.addObjectCreatedNotification(new s3notify.SnsDestination(topic), { prefix: props.prefix }); } } —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — class NotifyingBucket extends Construct { constructor(scope, id, props = {}) { super(scope, id); const bucket = new s3.Bucket(this, ‘bucket’); const topic = new sns.Topic(this, ‘topic’); bucket.addObjectCreatedNotification(new s3notify.SnsDestination(topic), { prefix: props.prefix }); } }

== module.exports = { NotifyingBucket }

Python:: + [source,python,subs=“verbatim,attributes”] — class NotifyingBucket(Construct):

def **init**(self, scope: Construct, id: str, \*, prefix=None): super().\_\_init\_\_(scope, id) bucket = s3.Bucket(self, “bucket”) topic = sns.Topic(self, “topic”) bucket.add\_object\_created\_notification(s3notify.SnsDestination(topic), s3.NotificationKeyFilter(prefix=prefix)) —-

Java:: + [source,java,subs=“verbatim,attributes”] — public class NotifyingBucket extends Construct {

…. public NotifyingBucket(final Construct scope, final String id) { this(scope, id, null, null); }

public NotifyingBucket(final Construct scope, final String id, final BucketProps props) { this(scope, id, props, null); }

public NotifyingBucket(final Construct scope, final String id, final String prefix) { this(scope, id, null, prefix); }

public NotifyingBucket(final Construct scope, final String id, final BucketProps props, final String prefix) { super(scope, id);

Bucket bucket = new Bucket(this, "bucket");  
Topic topic = new Topic(this, "topic");  
if (prefix != null)  
 bucket.addObjectCreatedNotification(new SnsDestination(topic),  
 NotificationKeyFilter.builder().prefix(prefix).build());

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — public class NotifyingBucketProps : BucketProps { public string Prefix { get; set; } }

public class NotifyingBucket : Construct { public NotifyingBucket(Construct scope, string id, NotifyingBucketProps props = null) : base(scope, id) { var bucket = new Bucket(this, “bucket”); var topic = new Topic(this, “topic”); bucket.AddObjectCreatedNotification(new SnsDestination(topic), new NotificationKeyFilter { Prefix = props?.Prefix }); } } —

Go:: + [source,go,subs=“verbatim,attributes”] — type NotifyingBucketProps struct { awss3.BucketProps Prefix \*string }

func NewNotifyingBucket(scope constructs.Construct, id \_\_string, props \*NotifyingBucketProps) awss3.Bucket { var bucket awss3.Bucket if props == nil { bucket = awss3.NewBucket(scope, jsii.String(\_\_id+“Bucket”), nil) } else { bucket = awss3.NewBucket(scope, jsii.String(\_\_id+“Bucket”), &props.BucketProps) } topic := awssns.NewTopic(scope, jsii.String(\_\_id+“Topic”), nil) if props == nil { bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic)) } else { bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic), &awss3.NotificationKeyFilter{ Prefix: props.Prefix, }) } return bucket } — ====

= [NOTE]

Our NotifyingBucket construct inherits not from Bucket but rather from Construct. We are using composition, not inheritance, to bundle an Amazon S3 bucket and an Amazon SNS topic together. In general, composition is preferred over inheritance when developing {aws} CDK constructs.

====

The NotifyingBucket constructor has a typical construct signature: scope, id, and props. The last argument, props, is optional (gets the default value {}) because all props are optional. (The base Construct class does not take a props argument.) You could define an instance of this construct in your app without props, for example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new NotifyingBucket(this, ‘MyNotifyingBucket’); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new NotifyingBucket(this, ‘MyNotifyingBucket’); —

Python:: + [source,python,subs=“verbatim,attributes”] — NotifyingBucket(self, “MyNotifyingBucket”) —

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

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

Go:: + [source,go,subs=“verbatim,attributes”] — NewNotifyingBucket(stack, jsii.String(“MyNotifyingBucket”), nil) — ====

Or you could use props (in Java, an additional parameter) to specify the path prefix to filter on, for example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new NotifyingBucket(this, ‘MyNotifyingBucket’, { prefix: ‘images/’ }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new NotifyingBucket(this, ‘MyNotifyingBucket’, { prefix: ‘images/’ }); —

Python:: + [source,python,subs=“verbatim,attributes”] — NotifyingBucket(self, “MyNotifyingBucket”, prefix=“images/”) —

Java:: + [source,java,subs=“verbatim,attributes”] — new NotifyingBucket(this, “MyNotifyingBucket”, “/images”); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new NotifyingBucket(this, “MyNotifyingBucket”, new NotifyingBucketProps { Prefix = “/images” }); —

Go:: + [source,go,subs=“verbatim,attributes”] — NewNotifyingBucket(stack, jsii.String(“MyNotifyingBucket”), &NotifyingBucketProps{ Prefix: jsii.String(“images/”), }) — ====

Typically, you would also want to expose some properties or methods on your constructs. It’s not very useful to have a topic hidden behind your construct, because users of your construct aren’t able to subscribe to it. Adding a topic property lets consumers access the inner topic, as shown in the following example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — export class NotifyingBucket extends Construct { public readonly topic: sns.Topic;

constructor(scope: Construct, id: string, props: NotifyingBucketProps) { super(scope, id); const bucket = new s3.Bucket(this, ‘bucket’); this.topic = new sns.Topic(this, ‘topic’); bucket.addObjectCreatedNotification(new s3notify.SnsDestination(this.topic), { prefix: props.prefix }); } } —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — class NotifyingBucket extends Construct {

constructor(scope, id, props) { super(scope, id); const bucket = new s3.Bucket(this, ‘bucket’); this.topic = new sns.Topic(this, ‘topic’); bucket.addObjectCreatedNotification(new s3notify.SnsDestination(this.topic), { prefix: props.prefix }); } }

== module.exports = { NotifyingBucket };

Python:: + [source,python,subs=“verbatim,attributes”] — class NotifyingBucket(Construct):

def **init**(self, scope: Construct, id: str, \*, prefix=None, \*\*kwargs): super().\_\_init\_\_(scope, id) bucket = s3.Bucket(self, “bucket”) self.topic = sns.Topic(self, “topic”) bucket.add\_object\_created\_notification(s3notify.SnsDestination(self.topic), s3.NotificationKeyFilter(prefix=prefix)) —-

Java:: + [source,java,subs=“verbatim,attributes”] — public class NotifyingBucket extends Construct {

…. public Topic topic = null;

public NotifyingBucket(final Construct scope, final String id) { this(scope, id, null, null); }

public NotifyingBucket(final Construct scope, final String id, final BucketProps props) { this(scope, id, props, null); }

public NotifyingBucket(final Construct scope, final String id, final String prefix) { this(scope, id, null, prefix); }

public NotifyingBucket(final Construct scope, final String id, final BucketProps props, final String prefix) { super(scope, id);

Bucket bucket = new Bucket(this, "bucket");  
topic = new Topic(this, "topic");  
if (prefix != null)  
 bucket.addObjectCreatedNotification(new SnsDestination(topic),  
 NotificationKeyFilter.builder().prefix(prefix).build());

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — public class NotifyingBucket : Construct { public readonly Topic topic;

public NotifyingBucket(Construct scope, string id, NotifyingBucketProps props = null) : base(scope, id) { var bucket = new Bucket(this, “bucket”); topic = new Topic(this, “topic”); bucket.AddObjectCreatedNotification(new SnsDestination(topic), new NotificationKeyFilter { Prefix = props?.Prefix }); } } —-

Go:: To do this in Go, we’ll need a little extra plumbing. Our original NewNotifyingBucket function returned an awss3.Bucket. We’ll need to extend Bucket to include a topic member by creating a NotifyingBucket struct. Our function will then return this type. + [source,go,subs=“verbatim,attributes”] — type NotifyingBucket struct { awss3.Bucket topic awssns.Topic }

func NewNotifyingBucket(scope constructs.Construct, id \_\_string, props \*NotifyingBucketProps) NotifyingBucket { var bucket awss3.Bucket if props == nil { bucket = awss3.NewBucket(scope, jsii.String(\_\_id+“Bucket”), nil) } else { bucket = awss3.NewBucket(scope, jsii.String(\_\_id+“Bucket”), &props.BucketProps) } topic := awssns.NewTopic(scope, jsii.String(\_\_id+“Topic”), nil) if props == nil { bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic)) } else { bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic), &awss3.NotificationKeyFilter{ Prefix: props.Prefix, }) } var nbucket NotifyingBucket nbucket.Bucket = bucket nbucket.topic = topic return nbucket } — ====

Now, consumers can subscribe to the topic, for example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const queue = new sqs.Queue(this, ‘NewImagesQueue’); const images = new NotifyingBucket(this, ‘/images’); images.topic.addSubscription(new sns\_sub.SqsSubscription(queue)); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const queue = new sqs.Queue(this, ‘NewImagesQueue’); const images = new NotifyingBucket(this, ‘/images’); images.topic.addSubscription(new sns\_sub.SqsSubscription(queue)); —

Python:: + [source,python,subs=“verbatim,attributes”] — queue = sqs.Queue(self, “NewImagesQueue”) images = NotifyingBucket(self, prefix=“Images”) images.topic.add\_subscription(sns\_sub.SqsSubscription(queue)) —

Java:: + [source,java,subs=“verbatim,attributes”] — NotifyingBucket images = new NotifyingBucket(this, “MyNotifyingBucket”, “/images”); images.topic.addSubscription(new SqsSubscription(queue)); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var queue = new Queue(this, “NewImagesQueue”); var images = new NotifyingBucket(this, “MyNotifyingBucket”, new NotifyingBucketProps { Prefix = “/images” }); images.topic.AddSubscription(new SqsSubscription(queue)); —

Go:: + [source,go,subs=“verbatim,attributes”] — queue := awssqs.NewQueue(stack, jsii.String(“NewImagesQueue”), nil) images := NewNotifyingBucket(stack, jsii.String(“MyNotifyingBucket”), &NotifyingBucketProps{ Prefix: jsii.String(“/images”), }) images.topic.AddSubscription(awssnssubscriptions.NewSqsSubscription(queue, nil)) — ====

[#constructs-learn] == Learn more

The following video provides a comprehensive overview of CDK constructs, and explains how you can use them in your CDK apps.

video::PzU-i0rJPGw[youtube,align = center,height = 390,fileref = https://www.youtube.com/embed/PzU-i0rJPGw,width = 480]