:doctype: book

include::attributes.txt[]

// Attributes [.topic] [#cfn-layer] = Customize constructs from the {aws} Construct Library :info\_titleabbrev: Customize constructs

== [abstract]

## Customize constructs from the {aws} Construct Library through escape hatches, raw overrides, and custom resources.

// Content start

Customize constructs from the {aws} Construct Library through escape hatches, raw overrides, and custom resources.

[#develop-customize-escape] == Use escape hatches

The {aws} Construct Library provides xref:constructs[constructs] of varying levels of abstraction.

At the highest level, your {aws} CDK application and the stacks in it are themselves abstractions of your entire cloud infrastructure, or significant chunks of it. They can be parameterized to deploy them in different environments or for different needs.

Abstractions are powerful tools for designing and implementing cloud applications. The {aws} CDK gives you the power not only to build with its abstractions, but also to create new abstractions. Using the existing open-source L2 and L3 constructs as guidance, you can build your own L2 and L3 constructs to reflect your own organization’s best practices and opinions.

No abstraction is perfect, and even good abstractions cannot cover every possible use case. During development, you may find a construct that almost fits your needs, requiring a small or large customization.

For this reason, the {aws} CDK provides ways to *break out* of the construct model. This includes moving to a lower-level abstraction or to a different model entirely. Escape hatches let you *escape* the {aws} CDK paradigm and customize it in ways that suit your needs. Then, you can wrap your changes in a new construct to abstract away the underlying complexity and provide a clean API for other developers.

The following are examples of situations where you can use escape hatches:

* An {aws} service feature is available through {aws} CloudFormation, but there are no L2 constructs for it.
* An {aws} service feature is available through {aws} CloudFormation, and there are L2 constructs for the service, but these don’t yet expose the feature. Because L2 constructs are curated by the CDK team, they may not be immediately available for new features.
* The feature is not yet available through {aws} CloudFormation at all.
* To determine whether a feature is available through {aws} CloudFormation, see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html[{aws} Resource and Property Types Reference].

[#develop-customize-escape-l1] === Develop escape hatches for L1 constructs

If L2 constructs are not available for the service, you can use the automatically generated L1 constructs. These resources can be recognized by their name starting with Cfn, such as CfnBucket or CfnRole. You instantiate them exactly as you would use the equivalent {aws} CloudFormation resource.

For example, to instantiate a low-level Amazon S3 bucket L1 with analytics enabled, you would write something like the following.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new s3.CfnBucket(this, ‘amzn-s3-demo-bucket’, { analyticsConfigurations: [ { id: ‘Config’, // … + } ] }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new s3.CfnBucket(this, ‘amzn-s3-demo-bucket’, { analyticsConfigurations: [ { id: ‘Config’ // … + } ] }); —

Python:: + [source,python,subs=“verbatim,attributes”] — s3.CfnBucket(self, “amzn-s3-demo-bucket”, analytics\_configurations: [ dict(id=“Config”, # … ) ] ) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnBucket.Builder.create(this, “amzn-s3-demo-bucket”) .analyticsConfigurations(Arrays.asList(java.util.Map.of( // Java 9 or later “id”, “Config”, // … ))).build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new CfnBucket(this, ‘amzn-s3-demo-bucket’, new CfnBucketProps { AnalyticsConfigurations = new Dictionary<string, string> { [“id”] = “Config”, // … } }); — ====

There might be rare cases where you want to define a resource that doesn’t have a corresponding CfnXxx class. This could be a new resource type that hasn’t yet been published in the {aws} CloudFormation resource specification. In cases like this, you can instantiate the cdk.CfnResource directly and specify the resource type and properties. This is shown in the following example.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new cdk.CfnResource(this, ‘amzn-s3-demo-bucket’, { type: ‘{aws}::S3::Bucket’, properties: { // Note the PascalCase here! These are CloudFormation identifiers. AnalyticsConfigurations: [ { Id: ‘Config’, // … } ] } }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new cdk.CfnResource(this, ‘amzn-s3-demo-bucket’, { type: ‘{aws}::S3::Bucket’, properties: { // Note the PascalCase here! These are CloudFormation identifiers. AnalyticsConfigurations: [ { Id: ‘Config’ // … } ] } }); —

Python:: + [source,python,subs=“verbatim,attributes”] — cdk.CfnResource(self, ‘amzn-s3-demo-bucket’, type=“{aws}::S3::Bucket”, properties=dict( # Note the PascalCase here! These are CloudFormation identifiers. “AnalyticsConfigurations”: [ { “Id”: “Config”, # … } ] ) ) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnResource.Builder.create(this, “amzn-s3-demo-bucket”) .type(“{aws}::S3::Bucket”) .properties(java.util.Map.of( // Map.of requires Java 9 or later // Note the PascalCase here! These are CloudFormation identifiers “AnalyticsConfigurations”, Arrays.asList( java.util.Map.of(“Id”, “Config”, // … )))) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new CfnResource(this, “amzn-s3-demo-bucket”, new CfnResourceProps { Type = “{aws}::S3::Bucket”, Properties = new Dictionary<string, object> { // Note the PascalCase here! These are CloudFormation identifiers [“AnalyticsConfigurations”] = new Dictionary<string, string>[] { new Dictionary<string, string> { [“Id”] = “Config” } } } }); — ====

[#develop-customize-escape-l2] === Develop escape hatches for L2 constructs

If an L2 construct is missing a feature or you’re trying to work around an issue, you can modify the L1 construct that’s encapsulated by the L2 construct.

All L2 constructs contain within them the corresponding L1 construct. For example, the high-level Bucket construct wraps the low-level CfnBucket construct. Because the CfnBucket corresponds directly to the {aws} CloudFormation resource, it exposes all features that are available through {aws} CloudFormation.

The basic approach to get access to the L1 construct is to use construct.node.defaultChild (Python: default\_child), cast it to the right type (if necessary), and modify its properties. Again, let’s take the example of a Bucket.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // Get the CloudFormation resource const cfnBucket = bucket.node.defaultChild as s3.CfnBucket;

// Change its properties cfnBucket.analyticsConfiguration = [ { id: ‘Config’, // … + }]; —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // Get the CloudFormation resource const cfnBucket = bucket.node.defaultChild;

// Change its properties cfnBucket.analyticsConfiguration = [ { id: ‘Config’ // … + }]; —

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

= Get the CloudFormation resource

cfn\_bucket = bucket.node.default\_child

= Change its properties

cfn\_bucket.analytics\_configuration = [ { “id”: “Config”, # … }] —

Java:: + [source,java,subs=“verbatim,attributes”] — // Get the CloudFormation resource CfnBucket cfnBucket = (CfnBucket)bucket.getNode().getDefaultChild();

cfnBucket.setAnalyticsConfigurations( Arrays.asList(java.util.Map.of( // Java 9 or later “Id”, “Config”, // … )); ) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // Get the CloudFormation resource var cfnBucket = (CfnBucket)bucket.Node.DefaultChild;

cfnBucket.AnalyticsConfigurations = new List++++++{ new Dictionary<string, string> { [“Id”] = “Config”, // … } }; — ====++++++

You can also use this object to change {aws} CloudFormation options such as Metadata and UpdatePolicy.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — cfnBucket.cfnOptions.metadata = { MetadataKey: ‘MetadataValue’ }; —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — cfnBucket.cfnOptions.metadata = { MetadataKey: ‘MetadataValue’ }; —

Python:: + [source,python,subs=“verbatim,attributes”] — cfn\_bucket.cfn\_options.metadata = { “MetadataKey”: “MetadataValue” } —

Java:: + [source,java,subs=“verbatim,attributes”] — cfnBucket.getCfnOptions().setMetadata(java.util.Map.of( // Java 9+ “MetadataKey”, “Metadatavalue”)); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — cfnBucket.CfnOptions.Metadata = new Dictionary<string, object> { [“MetadataKey”] = “Metadatavalue” }; — ====

[#develop-customize-unescape] == Use un-escape hatches

The {aws} CDK also provides the capability to go *up* an abstraction level, which we might refer to as an “un-escape” hatch. If you have an L1 construct, such as CfnBucket, you can create a new L2 construct (Bucket in this case) to wrap the L1 construct.

This is convenient when you create an L1 resource but want to use it with a construct that requires an L2 resource. It’s also helpful when you want to use convenience methods like .grantXxxxx() that aren’t available on the L1 construct.

You move to the higher abstraction level using a static method on the L2 class called .fromCfnXxxxx()–for example, Bucket.fromCfnBucket() for Amazon S3 buckets. The L1 resource is the only parameter.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — b1 = new s3.CfnBucket(this, “buck09”, { … }); b2 = s3.Bucket.fromCfnBucket(b1); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — b1 = new s3.CfnBucket(this, “buck09”, { …} ); b2 = s3.Bucket.fromCfnBucket(b1); —

Python:: + [source,python,subs=“verbatim,attributes”] — b1 = s3.CfnBucket(self, “buck09”, …) b2 = s3.from\_cfn\_bucket(b1) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnBucket b1 = CfnBucket.Builder.create(this, “buck09”) // …. .build(); IBucket b2 = Bucket.fromCfnBucket(b1); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var b1 = new CfnBucket(this, “buck09”, new CfnBucketProps { … }); var v2 = Bucket.FromCfnBucket(b1); — ====

L2 constructs created from L1 constructs are proxy objects that refer to the L1 resource, similar to those created from resource names, ARNs, or lookups. Modifications to these constructs do not affect the final synthesized {aws} CloudFormation template (since you have the L1 resource, however, you can modify that instead). For more information on proxy objects, see xref:resources-external[Referencing resources in your {aws} account].

To avoid confusion, do not create multiple L2 constructs that refer to the same L1 construct. For example, if you extract the CfnBucket from a Bucket using the technique in the xref:develop-customize-escape-l2[previous section], you shouldn’t create a second Bucket instance by calling Bucket.fromCfnBucket() with that CfnBucket. It actually works as you’d expect (only one +{aws}::S3::Bucket+ is synthesized) but it makes your code more difficult to maintain.

[#develop-customize-override] == Use raw overrides

If there are properties that are missing from the L1 construct, you can bypass all typing using raw overrides. This also makes it possible to delete synthesized properties.

Use one of the addOverride methods (Python: add\_override) methods, as shown in the following example.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // Get the CloudFormation resource const cfnBucket = bucket.node.defaultChild as s3.CfnBucket;

// Use dot notation to address inside the resource template fragment cfnBucket.addOverride(‘Properties.VersioningConfiguration.Status’, ‘NewStatus’); cfnBucket.addDeletionOverride(‘Properties.VersioningConfiguration.Status’);

// use index (0 here) to address an element of a list cfnBucket.addOverride(‘Properties.Tags.0.Value’, ‘NewValue’); cfnBucket.addDeletionOverride(‘Properties.Tags.0’);

// addPropertyOverride is a convenience function for paths starting with “Properties.” cfnBucket.addPropertyOverride(‘VersioningConfiguration.Status’, ‘NewStatus’); cfnBucket.addPropertyDeletionOverride(‘VersioningConfiguration.Status’); cfnBucket.addPropertyOverride(‘Tags.0.Value’, ‘NewValue’); cfnBucket.addPropertyDeletionOverride(‘Tags.0’); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // Get the CloudFormation resource const cfnBucket = bucket.node.defaultChild ;

// Use dot notation to address inside the resource template fragment cfnBucket.addOverride(‘Properties.VersioningConfiguration.Status’, ‘NewStatus’); cfnBucket.addDeletionOverride(‘Properties.VersioningConfiguration.Status’);

// use index (0 here) to address an element of a list cfnBucket.addOverride(‘Properties.Tags.0.Value’, ‘NewValue’); cfnBucket.addDeletionOverride(‘Properties.Tags.0’);

// addPropertyOverride is a convenience function for paths starting with “Properties.” cfnBucket.addPropertyOverride(‘VersioningConfiguration.Status’, ‘NewStatus’); cfnBucket.addPropertyDeletionOverride(‘VersioningConfiguration.Status’); cfnBucket.addPropertyOverride(‘Tags.0.Value’, ‘NewValue’); cfnBucket.addPropertyDeletionOverride(‘Tags.0’); —

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

= Get the CloudFormation resource

cfn\_bucket = bucket.node.default\_child

= Use dot notation to address inside the resource template fragment

cfn\_bucket.add\_override(“Properties.VersioningConfiguration.Status”, “NewStatus”) cfn\_bucket.add\_deletion\_override(“Properties.VersioningConfiguration.Status”)

= use index (0 here) to address an element of a list

cfn\_bucket.add\_override(“Properties.Tags.0.Value”, “NewValue”) cfn\_bucket.add\_deletion\_override(“Properties.Tags.0”)

= addPropertyOverride is a convenience function for paths starting with “Properties.”

cfn\_bucket.add\_property\_override(“VersioningConfiguration.Status”, “NewStatus”) cfn\_bucket.add\_property\_deletion\_override(“VersioningConfiguration.Status”) cfn\_bucket.add\_property\_override(“Tags.0.Value”, “NewValue”) cfn\_bucket.add\_property\_deletion\_override(“Tags.0”) —

Java:: + [source,java,subs=“verbatim,attributes”] — // Get the CloudFormation resource CfnBucket cfnBucket = (CfnBucket)bucket.getNode().getDefaultChild();

// Use dot notation to address inside the resource template fragment cfnBucket.addOverride(“Properties.VersioningConfiguration.Status”, “NewStatus”); cfnBucket.addDeletionOverride(“Properties.VersioningConfiguration.Status”);

// use index (0 here) to address an element of a list cfnBucket.addOverride(“Properties.Tags.0.Value”, “NewValue”); cfnBucket.addDeletionOverride(“Properties.Tags.0”);

// addPropertyOverride is a convenience function for paths starting with “Properties.” cfnBucket.addPropertyOverride(“VersioningConfiguration.Status”, “NewStatus”); cfnBucket.addPropertyDeletionOverride(“VersioningConfiguration.Status”); cfnBucket.addPropertyOverride(“Tags.0.Value”, “NewValue”); cfnBucket.addPropertyDeletionOverride(“Tags.0”); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // Get the CloudFormation resource var cfnBucket = (CfnBucket)bucket.node.defaultChild;

// Use dot notation to address inside the resource template fragment cfnBucket.AddOverride(“Properties.VersioningConfiguration.Status”, “NewStatus”); cfnBucket.AddDeletionOverride(“Properties.VersioningConfiguration.Status”);

// use index (0 here) to address an element of a list cfnBucket.AddOverride(“Properties.Tags.0.Value”, “NewValue”); cfnBucket.AddDeletionOverride(“Properties.Tags.0”);

// addPropertyOverride is a convenience function for paths starting with “Properties.” cfnBucket.AddPropertyOverride(“VersioningConfiguration.Status”, “NewStatus”); cfnBucket.AddPropertyDeletionOverride(“VersioningConfiguration.Status”); cfnBucket.AddPropertyOverride(“Tags.0.Value”, “NewValue”); cfnBucket.AddPropertyDeletionOverride(“Tags.0”); — ====

[#develop-customize-custom] == Use custom resources

If the feature isn’t available through {aws} CloudFormation, but only through a direct API call, you must write an {aws} CloudFormation Custom Resource to make the API call you need. You can use the {aws} CDK to write custom resources and wrap them into a regular construct interface. From the perspective of a consumer of your construct, the experience will feel native.

Building a custom resource involves writing a Lambda function that responds to a resource’s CREATE, UPDATE, and DELETE lifecycle events. If your custom resource needs to make only a single API call, consider using the https://github.com/awslabs/aws-cdk/tree/master/packages/%40aws-cdk/custom-resources[AwsCustomResource]. This makes it possible to perform arbitrary SDK calls during an {aws} CloudFormation deployment. Otherwise, you should write your own Lambda function to perform the work you need to get done.

The subject is too broad to cover completely here, but the following links should get you started:

* https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-custom-resources.html[Custom Resources]
* https://github.com/aws-samples/aws-cdk-examples/tree/master/typescript/custom-resource/[Custom-Resource Example]
* For a more fully fledged example, see the https://github.com/awslabs/aws-cdk/blob/master/packages/@aws-cdk/aws-certificatemanager/lib/dns-validated-certificate.ts [DnsValidatedCertificate] class in the CDK standard library. This is implemented as a custom resource.