:doctype: book

include::attributes.txt[]

// Attributes [.topic] [#aspects] = Aspects and the {aws} CDK :info\_titleabbrev: Aspects :keywords: {aws} CDK, aspects

== [abstract]

## Aspects are a way to apply an operation to all constructs in a given scope. The aspect could modify the constructs, such as by adding tags. Or it could verify something about the state of the constructs, such as making sure that all buckets are encrypted.

// Content start

Aspects are a way to apply an operation to all constructs in a given scope. The aspect could modify the constructs, such as by adding tags. Or it could verify something about the state of the constructs, such as making sure that all buckets are encrypted.

To apply an aspect to a construct and all constructs in the same scope, call +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Aspects.html#static-ofscope[Aspects].of(<SCOPE>).add()+ with a new aspect, as shown in the following example.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — Aspects.of(myConstruct).add(new SomeAspect(…)); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — Aspects.of(myConstruct).add(new SomeAspect(…)); —

Python:: + [source,python,subs=“verbatim,attributes”] — Aspects.of(my\_construct).add(SomeAspect(…)) —

Java:: + [source,java,subs=“verbatim,attributes”] — Aspects.of(myConstruct).add(new SomeAspect(…)); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Aspects.Of(myConstruct).add(new SomeAspect(…)); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.Aspects\_Of(stack).Add(awscdk.NewTag(…)) — ====

The {aws} CDK uses aspects to xref:tagging[tag resources], but the framework can also be used for other purposes. For example, you can use it to validate or change the {aws} CloudFormation resources that are defined for you by higher-level constructs.

[#aspects-detail] == Aspects in detail

Aspects employ the link:https://en.wikipedia.org/wiki/Visitor\_pattern[visitor pattern]. An aspect is a class that implements the following interface.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — interface IAspect { visit(node: IConstruct): void;} —

JavaScript:: JavaScript doesn’t have interfaces as a language feature. Therefore, an aspect is simply an instance of a class having a visit method that accepts the node to be operated on.

Python:: Python doesn’t have interfaces as a language feature. Therefore, an aspect is simply an instance of a class having a visit method that accepts the node to be operated on.

Java:: + [source,java,subs=“verbatim,attributes”] — public interface IAspect { public void visit(Construct node); } —

C#:: + [source,csharp,subs=“verbatim,attributes”] — public interface IAspect { void Visit(IConstruct node); } —

Go:: + [source,go,subs=“verbatim,attributes”] — type IAspect interface { Visit(node constructs.IConstruct) } — ====

When you call +Aspects.of(<SCOPE>).add(...)+, the construct adds the aspect to an internal list of aspects. You can obtain the list with Aspects.of(<SCOPE>).

During the xref:deploy-how-synth-app[prepare phase], the {aws} CDK calls the visit method of the object for the construct and each of its children in top-down order.

The visit method is free to change anything in the construct. In strongly typed languages, cast the received construct to a more specific type before accessing construct-specific properties or methods.

Aspects don’t propagate across Stage construct boundaries, because Stages are self-contained and immutable after definition. Apply aspects on the Stage construct itself (or lower) if you want them to visit constructs inside the Stage.

[#aspects-example] == Example

The following example validates that all buckets created in the stack have versioning enabled. The aspect adds an error annotation to the constructs that fail the validation. This results in the synth operation failing and prevents deploying the resulting cloud assembly.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — class BucketVersioningChecker implements IAspect { public visit(node: IConstruct): void { // See that we’re dealing with a CfnBucket if (node instanceof s3.CfnBucket) {

// Check for versioning property, exclude the case where the property // can be a token (IResolvable). if (!node.versioningConfiguration || (!Tokenization.isResolvable(node.versioningConfiguration) && node.versioningConfiguration.status !== ‘Enabled’)) { Annotations.of(node).addError(‘Bucket versioning is not enabled’); } } } }

// Later, apply to the stack Aspects.of(stack).add(new BucketVersioningChecker()); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — class BucketVersioningChecker { visit(node) { // See that we’re dealing with a CfnBucket if ( node instanceof s3.CfnBucket) {

// Check for versioning property, exclude the case where the property // can be a token (IResolvable). if (!node.versioningConfiguration || !Tokenization.isResolvable(node.versioningConfiguration) && node.versioningConfiguration.status !== ‘Enabled’)) { Annotations.of(node).addError(‘Bucket versioning is not enabled’); } } } }

// Later, apply to the stack Aspects.of(stack).add(new BucketVersioningChecker()); —

Python:: + [source,python,subs=“verbatim,attributes”] — @jsii.implements(cdk.IAspect) class BucketVersioningChecker:

def visit(self, node): # See that we’re dealing with a CfnBucket if isinstance(node, s3.CfnBucket):

# Check for versioning property, exclude the case where the property # can be a token (IResolvable). if (not node.versioning\_configuration or not Tokenization.is\_resolvable(node.versioning\_configuration) and node.versioning\_configuration.status != “Enabled”): Annotations.of(node).add\_error(‘Bucket versioning is not enabled’)

= Later, apply to the stack

## Aspects.of(stack).add(BucketVersioningChecker())

Java:: + [source,java,subs=“verbatim,attributes”] — public class BucketVersioningChecker implements IAspect { @Override public void visit(Construct node) { // See that we’re dealing with a CfnBucket if (node instanceof CfnBucket) { CfnBucket bucket = (CfnBucket)node; Object versioningConfiguration = bucket.getVersioningConfiguration(); if (versioningConfiguration == null || !Tokenization.isResolvable(versioningConfiguration.toString()) && !versioningConfiguration.toString().contains(“Enabled”)) Annotations.of(bucket.getNode()).addError(“Bucket versioning is not enabled”); } } }

// Later, apply to the stack Aspects.of(stack).add(new BucketVersioningChecker()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — class BucketVersioningChecker : Amazon.Jsii.Runtime.Deputy.DeputyBase, IAspect { public void Visit(IConstruct node) { // See that we’re dealing with a CfnBucket if (node is CfnBucket) { var bucket = (CfnBucket)node; if (bucket.VersioningConfiguration is null || !Tokenization.IsResolvable(bucket.VersioningConfiguration) && !bucket.VersioningConfiguration.ToString().Contains(“Enabled”)) Annotations.Of(bucket.Node).AddError(“Bucket versioning is not enabled”); } } }

// Later, apply to the stack Aspects.Of(stack).add(new BucketVersioningChecker()); — ====