:doctype: book

include::attributes.txt[]

// Attributes :https–docs-aws-amazon-com-cdk-api-v2-docs-aws-cdk-lib-Tags-html-removekey-props: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Tags.html#removekey-props :https–docs-aws-amazon-com-cdk-api-v2-docs-aws-cdk-lib-Tags-html-addkey-value-props: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Tags.html#addkey-value-props

[.topic] :info\_titleabbrev: Tags :keywords: {aws} CDK, Tags

[#tagging] = Tags and the {aws} CDK

== [abstract]

## Tags are informational key-value elements that you can add to constructs in your {aws} CDK app. A tag applied to a given construct also applies to all of its taggable children. Tags are included in the {aws} CloudFormation template synthesized from your app and are applied to the {aws} resources it deploys. You can use tags to identify and categorize resources for the following purposes:

// Content start

Tags are informational key-value elements that you can add to constructs in your {aws} CDK app. A tag applied to a given construct also applies to all of its taggable children. Tags are included in the {aws} CloudFormation template synthesized from your app and are applied to the {aws} resources it deploys. You can use tags to identify and categorize resources for the following purposes:

* Simplifying management
* Cost allocation
* Access control
* Any other purposes that you devise

= [TIP]

For more information about how you can use tags with your {aws} resources, see https://docs.aws.amazon.com/whitepapers/latest/tagging-best-practices/tagging-best-practices.html[Best Practices for Tagging {aws} Resources] in the *{aws} Whitepaper*.

====

[#tagging-use] == Using tags

The https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Tags.html[Tags] class includes the static method of(), through which you can add tags to, or remove tags from, the specified construct.

* https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Tags.html#addkey-value-props[Tags.of(<SCOPE>).add()] applies a new tag to the given construct and all of its children.
* https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Tags.html#removekey-props[Tags.of(<SCOPE>).remove()] removes a tag from the given construct and any of its children, including tags a child construct may have applied to itself.

= [NOTE]

Tagging is implemented using xref:aspects[Aspects and the {aws} CDK]. Aspects are a way to apply an operation (such as tagging) to all constructs in a given scope.

====

The following example applies the tag *key* with the value *value* to a construct.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(‘key’, ‘value’); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(‘key’, ‘value’); —

Python:: + [source,python,subs=“verbatim,attributes”] — Tags.of(my\_construct).add(“key”, “value”) —

Java:: + [source,java,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(“key”, “value”); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Tags.Of(myConstruct).Add(“key”, “value”); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.Tags\_Of(myConstruct).Add(jsii.String(“key”), jsii.String(“value”), &awscdk.TagProps{}) — ====

The following example deletes the tag *key* from a construct.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).remove(‘key’); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).remove(‘key’); —

Python:: + [source,python,subs=“verbatim,attributes”] — Tags.of(my\_construct).remove(“key”) —

Java:: + [source,java,subs=“verbatim,attributes”] — Tags.of(myConstruct).remove(“key”); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Tags.Of(myConstruct).Remove(“key”); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.Tags\_Of(myConstruct).Remove(jsii.String(“key”), &awscdk.TagProps{}) — ====

If you are using Stage constructs, apply the tag at the Stage level or below. Tags are not applied across Stage boundaries.

[#tagging-priorities] == Tag priorities

The {aws} CDK applies and removes tags recursively. If there are conflicts, the tagging operation with the highest priority wins. (Priorities are set using the optional priority property.) If the priorities of two operations are the same, the tagging operation closest to the bottom of the construct tree wins. By default, applying a tag has a priority of 100 (except for tags added directly to an {aws} CloudFormation resource, which has a priority of 50). The default priority for removing a tag is 200.

The following applies a tag with a priority of 300 to a construct.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(‘key’, ‘value’, { priority: 300 }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(‘key’, ‘value’, { priority: 300 }); —

Python:: + [source,python,subs=“verbatim,attributes”] — Tags.of(my\_construct).add(“key”, “value”, priority=300) —

Java:: + [source,java,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(“key”, “value”, TagProps.builder() .priority(300).build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Tags.Of(myConstruct).Add(“key”, “value”, new TagProps { Priority = 300 }); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.Tags\_Of(myConstruct).Add(jsii.String(“key”), jsii.String(“value”), &awscdk.TagProps{ Priority: jsii.Number(300), }) — ====

[#tagging-props] == Optional properties

Tags support https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.TagProps.html[properties] that fine-tune how tags are applied to, or removed from, resources. All properties are optional.

applyToLaunchedInstances (Python: apply\_to\_launched\_instances):: + Available for add() only. By default, tags are applied to instances launched in an Auto Scaling group. Set this property to *false* to ignore instances launched in an Auto Scaling group.

includeResourceTypes/excludeResourceTypes (Python: include\_resource\_types/exclude\_resource\_types):: + Use these to manipulate tags only on a subset of resources, based on {aws} CloudFormation resource types. By default, the operation is applied to all resources in the construct subtree, but this can be changed by including or excluding certain resource types. Exclude takes precedence over include, if both are specified.

priority:: + Use this to set the priority of this operation with respect to other Tags.add() and Tags.remove() operations. Higher values take precedence over lower values. The default is 100 for add operations (50 for tags applied directly to {aws} CloudFormation resources) and 200 for remove operations.

The following example applies the tag *tagname* with the value *value* and priority *100* to resources of type *{aws}::Xxx::Yyy* in the construct. It doesn’t apply the tag to instances launched in an Amazon EC2 Auto Scaling group or to resources of type *{aws}::Xxx::Zzz*. (These are placeholders for two arbitrary but different {aws} CloudFormation resource types.)

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(‘tagname’, ‘value’, { applyToLaunchedInstances: false, includeResourceTypes: [‘{aws}::Xxx::Yyy’], excludeResourceTypes: [‘{aws}::Xxx::Zzz’], priority: 100, }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(‘tagname’, ‘value’, { applyToLaunchedInstances: false, includeResourceTypes: [‘{aws}::Xxx::Yyy’], excludeResourceTypes: [‘{aws}::Xxx::Zzz’], priority: 100 }); —

Python:: + [source,python,subs=“verbatim,attributes”] — Tags.of(my\_construct).add(“tagname”, “value”, apply\_to\_launched\_instances=False, include\_resource\_types=[“{aws}::Xxx::Yyy”], exclude\_resource\_types=[“{aws}::Xxx::Zzz”], priority=100) —

Java:: + [source,java,subs=“verbatim,attributes”] — Tags.of(myConstruct).add(“tagname”, “value”, TagProps.builder() .applyToLaunchedInstances(false) .includeResourceTypes(Arrays.asList(“{aws}::Xxx::Yyy”)) .excludeResourceTypes(Arrays.asList(“{aws}::Xxx::Zzz”)) .priority(100).build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Tags.Of(myConstruct).Add(“tagname”, “value”, new TagProps { ApplyToLaunchedInstances = false, IncludeResourceTypes = [“{aws}::Xxx::Yyy”], ExcludeResourceTypes = [“{aws}::Xxx::Zzz”], Priority = 100 }); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.Tags\_Of(myConstruct).Add(jsii.String(“tagname”), jsii.String(“value”), &awscdk.TagProps{ ApplyToLaunchedInstances: jsii.Bool(false), IncludeResourceTypes: &[]\_\_string{jsii.String(“{aws}::Xxx:Yyy”)}, ExcludeResourceTypes: &[]\_\_string{jsii.String(“{aws}::Xxx:Zzz”)}, Priority: jsii.Number(100), }) — ====

The following example removes the tag *tagname* with priority *200* from resources of type *{aws}::Xxx::Yyy* in the construct, but not from resources of type *{aws}::Xxx::Zzz*.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).remove(‘tagname’, { includeResourceTypes: [‘{aws}::Xxx::Yyy’], excludeResourceTypes: [‘{aws}::Xxx::Zzz’], priority: 200, }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(myConstruct).remove(‘tagname’, { includeResourceTypes: [‘{aws}::Xxx::Yyy’], excludeResourceTypes: [‘{aws}::Xxx::Zzz’], priority: 200 }); —

Python:: + [source,python,subs=“verbatim,attributes”] — Tags.of(my\_construct).remove(“tagname”, include\_resource\_types=[“{aws}::Xxx::Yyy”], exclude\_resource\_types=[“{aws}::Xxx::Zzz”], priority=200,) —

Java:: + [source,java,subs=“verbatim,attributes”] — Tags.of((myConstruct).remove(“tagname”, TagProps.builder() .includeResourceTypes(Arrays.asList(“{aws}::Xxx::Yyy”)) .excludeResourceTypes(Arrays.asList(“{aws}::Xxx::Zzz”)) .priority(100).build()); ) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Tags.Of(myConstruct).Remove(“tagname”, new TagProps { IncludeResourceTypes = [“{aws}::Xxx::Yyy”], ExcludeResourceTypes = [“{aws}::Xxx::Zzz”], Priority = 100 }); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.Tags\_Of(myConstruct).Remove(jsii.String(“tagname”), &awscdk.TagProps{ IncludeResourceTypes: &[]\_\_string{jsii.String(“{aws}::Xxx:Yyy”)}, ExcludeResourceTypes: &[]\_\_string{jsii.String(“{aws}::Xxx:Zzz”)}, Priority: jsii.Number(200), }) — ====

[#tagging-example] == Example

The following example adds the tag key *StackType* with value *TheBest* to any resource created within the Stack named MarketingSystem. Then it removes it again from all resources except Amazon EC2 VPC subnets. The result is that only the subnets have the tag applied.

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

const app = new App(); const theBestStack = new Stack(app, ‘MarketingSystem’);

// Add a tag to all constructs in the stack Tags.of(theBestStack).add(‘StackType’, ‘TheBest’);

// Remove the tag from all resources except subnet resources Tags.of(theBestStack).remove(‘StackType’, { excludeResourceTypes: [‘{aws}::EC2::Subnet’] }); —

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

const app = new App(); const theBestStack = new Stack(app, ‘MarketingSystem’);

// Add a tag to all constructs in the stack Tags.of(theBestStack).add(‘StackType’, ‘TheBest’);

// Remove the tag from all resources except subnet resources Tags.of(theBestStack).remove(‘StackType’, { excludeResourceTypes: [‘{aws}::EC2::Subnet’] }); —

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import App, Stack, Tags

app = App(); the\_best\_stack = Stack(app, ‘MarketingSystem’)

= Add a tag to all constructs in the stack

Tags.of(the\_best\_stack).add(“StackType”, “TheBest”)

= Remove the tag from all resources except subnet resources

Tags.of(the\_best\_stack).remove(“StackType”, exclude\_resource\_types=[“{aws}::EC2::Subnet”]) —

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.App; import software.amazon.awscdk.Tags;

// Add a tag to all constructs in the stack Tags.of(theBestStack).add(“StackType”, “TheBest”);

// Remove the tag from all resources except subnet resources Tags.of(theBestStack).remove(“StackType”, TagProps.builder() .excludeResourceTypes(Arrays.asList(“{aws}::EC2::Subnet”)) .build()); —

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

var app = new App(); var theBestStack = new Stack(app, ‘MarketingSystem’);

// Add a tag to all constructs in the stack Tags.Of(theBestStack).Add(“StackType”, “TheBest”);

// Remove the tag from all resources except subnet resources Tags.Of(theBestStack).Remove(“StackType”, new TagProps { ExcludeResourceTypes = [“{aws}::EC2::Subnet”] }); —

Go:: + [source,go,subs=“verbatim,attributes”] — import “github.com/aws/aws-cdk-go/awscdk/v2” app := awscdk.NewApp(nil) theBestStack := awscdk.NewStack(app, jsii.String(“MarketingSystem”), &awscdk.StackProps{})

// Add a tag to all constructs in the stack awscdk.Tags\_Of(theBestStack).Add(jsii.String(“StackType”), jsii.String(“TheBest”), &awscdk.TagProps{})

// Remove the tag from all resources except subnet resources awscdk.Tags\_Of(theBestStack).Add(jsii.String(“StackType”), jsii.String(“TheBest”), &awscdk.TagProps{ ExcludeResourceTypes: &[]\*string{jsii.String(“{aws}::EC2::Subnet”)}, }) — ====

The following code achieves the same result. Consider which approach (inclusion or exclusion) makes your intent clearer.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(theBestStack).add(‘StackType’, ‘TheBest’, { includeResourceTypes: [‘{aws}::EC2::Subnet’]}); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — Tags.of(theBestStack).add(‘StackType’, ‘TheBest’, { includeResourceTypes: [‘{aws}::EC2::Subnet’]}); —

Python:: + [source,python,subs=“verbatim,attributes”] — Tags.of(the\_best\_stack).add(“StackType”, “TheBest”, include\_resource\_types=[“{aws}::EC2::Subnet”]) —

Java:: + [source,java,subs=“verbatim,attributes”] — Tags.of(theBestStack).add(“StackType”, “TheBest”, TagProps.builder() .includeResourceTypes(Arrays.asList(“{aws}::EC2::Subnet”)) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — Tags.Of(theBestStack).Add(“StackType”, “TheBest”, new TagProps { IncludeResourceTypes = [“{aws}::EC2::Subnet”] }); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.Tags\_Of(theBestStack).Add(jsii.String(“StackType”), jsii.String(“TheBest”), &awscdk.TagProps{ IncludeResourceTypes: &[]\*string{jsii.String(“{aws}::EC2::Subnet”)}, }) — ====

[#tagging-single] == Tagging single constructs

Tags.of(scope).add(key, value) is the standard way to add tags to constructs in the {aws} CDK. Its tree-walking behavior, which recursively tags all taggable resources under the given scope, is almost always what you want. Sometimes, however, you need to tag a specific, arbitrary construct (or constructs).

One such case involves applying tags whose value is derived from some property of the construct being tagged. The standard tagging approach recursively applies the same key and value to all matching resources in the scope. However, here the value could be different for each tagged construct.

Tags are implemented using xref:aspects[aspects], and the CDK calls the tag’s visit() method for each construct under the scope you specified using Tags.of(scope). We can call Tag.visit() directly to apply a tag to a single construct.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new cdk.Tag(key, value).visit(scope); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new cdk.Tag(key, value).visit(scope); —

Python:: + [source,python,subs=“verbatim,attributes”] — cdk.Tag(key, value).visit(scope) —

Java:: + [source,java,subs=“verbatim,attributes”] — Tag.Builder.create(key, value).build().visit(scope); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new Tag(key, value).Visit(scope); —

Go:: + [source,go,subs=“verbatim,attributes”] — awscdk.NewTag(key, value, &awscdk.TagProps{}).Visit(scope) — ====

You can tag all constructs under a scope but let the values of the tags derive from properties of each construct. To do so, write an aspect and apply the tag in the aspect’s visit() method as shown in the preceding example. Then, add the aspect to the desired scope using Aspects.of(scope).add(aspect).

The following example applies a tag to each resource in a stack containing the resource’s path.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — class PathTagger implements cdk.IAspect { visit(node: IConstruct) { new cdk.Tag(“aws-cdk-path”, node.node.path).visit(node); } }

stack = new MyStack(app); cdk.Aspects.of(stack).add(new PathTagger()) —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — class PathTagger { visit(node) { new cdk.Tag(“aws-cdk-path”, node.node.path).visit(node); } }

stack = new MyStack(app); cdk.Aspects.of(stack).add(new PathTagger()) —

Python:: + [source,python,subs=“verbatim,attributes”] — @jsii.implements(cdk.IAspect) class PathTagger: def visit(self, node: IConstruct): cdk.Tag(“aws-cdk-path”, node.node.path).visit(node)

stack = MyStack(app) cdk.Aspects.of(stack).add(PathTagger()) —

Java:: + [source,java,subs=“verbatim,attributes”] — final class PathTagger implements IAspect { public void visit(IConstruct node) { Tag.Builder.create(“aws-cdk-path”, node.getNode().getPath()).build().visit(node); } }

stack stack = new MyStack(app); Aspects.of(stack).add(new PathTagger()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — public class PathTagger : IAspect { public void Visit(IConstruct node) { new Tag(“aws-cdk-path”, node.Node.Path).Visit(node); } }

var stack = new MyStack(app); Aspects.Of(stack).Add(new PathTagger); — ====

= [TIP]

The logic of conditional tagging, including priorities, resource types, and so on, is built into the Tag class. You can use these features when applying tags to arbitrary resources; the tag is not applied if the conditions aren’t met. Also, the Tag class only tags taggable resources, so you don’t need to test whether a construct is taggable before applying a tag.

====