:doctype: book

include::attributes.txt[]

// Attributes [.topic] [#use-cfn-template] = Import an existing {aws} CloudFormation template :info\_titleabbrev: Import an {aws} CloudFormation template

// Content start

Import resources from an {aws} CloudFormation template into your {aws} Cloud Development Kit ({aws} CDK) applications by using the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include-readme.html[cloudformation-include.CfnInclude] construct to convert resources to L1 constructs.

After import, you can work with these resources in your app in the same way that you would if they were originally defined in {aws} CDK code. You can also use these L1 constructs within higher-level {aws} CDK constructs. For example, this can let you use the L2 permission grant methods with the resources they define.

The cloudformation-include.CfnInclude construct essentially adds an {aws} CDK API wrapper to any resource in your {aws} CloudFormation template. Use this capability to import your existing {aws} CloudFormation templates to the {aws} CDK a piece at a time. By doing this, you can manage your existing resources using {aws} CDK constructs to utilize the benefits of higher-level abstractions. You can also use this feature to vend your {aws} CloudFormation templates to {aws} CDK developers by providing an {aws} CDK construct API.

= [NOTE]

{aws} CDK v1 also included link:https://docs.aws.amazon.com/cdk/api/latest/docs/aws-cdk-lib.CfnInclude.html[aws-cdk-lib.CfnInclude], which was previously used for the same general purpose. However, it lacks much of the functionality of cloudformation-include.CfnInclude.

====

[#use-cfn-template-import]] == Import an {aws} CloudFormation template

The following is a sample {aws} CloudFormation template that we will use to provide examples in this topic. Copy and save the template as my-template.json to follow along. After working through these examples, you can explore further by using any of your existing deployed {aws} CloudFormation templates. You can obtain them from the {aws} CloudFormation console.

== [source,json,subs=“verbatim,attributes”]

{ “Resources”: { “amzn-s3-demo-bucket”: { “Type”: “{aws}::S3::Bucket”, “Properties”: { “BucketName”: “amzn-s3-demo-bucket”, } } } } —

You can work with either JSON or YAML templates. We recommend JSON if available since YAML parsers can vary slightly in what they accept.

The following is an example of how to import the sample template into your {aws} CDK app using cloudformation-include. Templates are imported within the context of an CDK stack.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘aws-cdk-lib’; import \* as cfninc from ‘aws-cdk-lib/cloudformation-include’; import { Construct } from ‘constructs’;

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

const template = new cfninc.CfnInclude(this, ‘Template’, { templateFile: ‘my-template.json’, }); } } —-

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

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

const template = new cfninc.CfnInclude(this, ‘Template’, { templateFile: ‘my-template.json’, }); } }

== module.exports = { MyStack }

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk as cdk from aws\_cdk import cloudformation\_include as cfn\_inc from constructs import Construct

class MyStack(cdk.Stack):

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

template = cfn\_inc.CfnInclude(self, "Template",  
 template\_file="my-template.json") ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.cloudformation.include.CfnInclude; import software.constructs.Construct;

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

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

CfnInclude template = CfnInclude.Builder.create(this, "Template")  
 .templateFile("my-template.json")  
 .build();

} } —- ….

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

namespace MyApp { public class MyStack : Stack { internal MyStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { var template = new cfnInc.CfnInclude(this, “Template”, new cfnInc.CfnIncludeProps { TemplateFile = “my-template.json” }); } } } — ====

By default, importing a resource preserves the resource’s original logical ID from the template. This behavior is suitable for importing an {aws} CloudFormation template into the {aws} CDK, where logical IDs must be retained. {aws} CloudFormation needs this information to recognize these imported resources as the same resources from the {aws} CloudFormation template.

If you are developing an {aws} CDK construct wrapper for the template so that it can be used by other {aws} CDK developers, have the {aws} CDK generate new resource IDs instead. By doing this, the construct can be used multiple times in a stack without name conflicts. To do this, set the preserveLogicalIds property to false when importing the template. The following is an example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const template = new cfninc.CfnInclude(this, ‘MyConstruct’, { templateFile: ‘my-template.json’, preserveLogicalIds: false }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const template = new cfninc.CfnInclude(this, ‘MyConstruct’, { templateFile: ‘my-template.json’, preserveLogicalIds: false }); —

Python:: + [source,python,subs=“verbatim,attributes”] — template = cfn\_inc.CfnInclude(self, “Template”, + template\_file=“my-template.json”, preserve\_logical\_ids=False) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnInclude template = CfnInclude.Builder.create(this, “Template”) .templateFile(“my-template.json”) .preserveLogicalIds(false) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var template = new cfnInc.CfnInclude(this, “Template”, new cfn\_inc.CfnIncludeProps { TemplateFile = “my-template.json”, PreserveLogicalIds = false }); — ====

To put imported resources under the control of your {aws} CDK app, add the stack to the App:

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

const app = new cdk.App(); new MyStack(app, ‘MyStack’); —

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

const app = new cdk.App(); new MyStack(app, ‘MyStack’); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk as cdk from mystack.my\_stack import MyStack

app = cdk.App() MyStack(app, “MyStack”) —

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

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

new MyStack(app, "MyStack");

} } —-

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

namespace CdkApp { sealed class Program { public static void Main(string[] args) { var app = new App(); new MyStack(app, “MyStack”); } } } — ====

To verify that there won’t be any unintended changes to the {aws} resources in the stack, you can perform a diff. Use the {aws} CDK CLI cdk diff command and omit any {aws} CDK-specific metadata. The following is an example:

== [source,none,subs=“verbatim,attributes”]

## cdk diff –no-version-reporting –no-path-metadata –no-asset-metadata

After you import an {aws} CloudFormation template, the {aws} CDK app should become the source of truth for your imported resources. To make changes to your resources, modify them in your {aws} CDK app and deploy with the {aws} CDK CLI cdk deploy command.

[#use-cfn-template-cfninclude-access] == Access imported resources

The name template in the example code represents the imported {aws} CloudFormation template. To access a resource from it, use the object’s link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrresourcelogicalid[getResource()] method. To access the returned resource as a specific kind of resource, cast the result to the desired type. This isn’t necessary in Python or JavaScript. The following is an example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const cfnBucket = template.getResource(‘amzn-s3-demo-bucket’) as s3.CfnBucket; —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const cfnBucket = template.getResource(‘amzn-s3-demo-bucket’); —

Python:: + [source,python,subs=“verbatim,attributes”] — cfn\_bucket = template.get\_resource(“amzn-s3-demo-bucket”) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnBucket cfnBucket = (CfnBucket)template.getResource(“amzn-s3-demo-bucket”); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var cfnBucket = (CfnBucket)template.GetResource(“amzn-s3-demo-bucket”); — ====

From this example, cfnBucket is now an instance of the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html[aws-s3.CfnBucket] class. This is an L1 construct that represents the corresponding {aws} CloudFormation resource. You can treat it like any other resource of its type. For example, you can get its ARN value with the bucket.attrArn property.

To wrap the L1 CfnBucket resource in an L2 link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html[aws-s3.CfnBucket] instance instead, use the static methods link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#static-fromwbrbucketwbrarnscope-id-bucketarn[fromBucketArn()], link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#static-fromwbrbucketwbrattributesscope-id-attrs[fromBucketAttributes()], or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#static-fromwbrbucketwbrnamescope-id-bucketname[fromBucketName()]. Usually, the fromBucketName() method is most convenient. The following is an example:

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

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

Python:: + [source,python,subs=“verbatim,attributes”] — bucket = s3.Bucket.from\_bucket\_name(self, “Bucket”, cfn\_bucket.ref) —

Java:: + [source,java,subs=“verbatim,attributes”] — Bucket bucket = (Bucket)Bucket.fromBucketName(this, “Bucket”, cfnBucket.getRef()); —

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

Other L2 constructs have similar methods for creating the construct from an existing resource.

When you wrap an L1 construct in an L2 construct, it doesn’t create a new resource. From our example, we are not creating a second S3; bucket. Instead, the new Bucket instance encapsulates the existing CfnBucket.

From the example, the bucket is now an L2 Bucket construct that behaves like any other L2 construct. For example, you can grant an {aws} Lambda function write access to the bucket by using the bucket’s convenient link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#grantwbrwriteidentity-objectskeypattern[grantWrite()] method. You don’t have to define the necessary {aws} Identity and Access Management (IAM) policy manually. The following is an example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — bucket.grantWrite(lambdaFunc); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — bucket.grantWrite(lambdaFunc); —

Python:: + [source,python,subs=“verbatim,attributes”] — bucket.grant\_write(lambda\_func) —

Java:: + [source,java,subs=“verbatim,attributes”] — bucket.grantWrite(lambdaFunc); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — bucket.GrantWrite(lambdaFunc); — ====

[#use-cfn-template-cfninclude-params] == Replace parameters

If your {aws} CloudFormation template contains parameters, you can replace them with build time values at import by using the parameters property. In the following example, we replace the UploadBucket parameter with the ARN of a bucket defined elsewhere in our {aws} CDK code.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const template = new cfninc.CfnInclude(this, ‘Template’, { templateFile: ‘my-template.json’, parameters: { ‘UploadBucket’: bucket.bucketArn, }, }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const template = new cfninc.CfnInclude(this, ‘Template’, { templateFile: ‘my-template.json’, parameters: { ‘UploadBucket’: bucket.bucketArn, }, }); —

Python:: + [source,python,subs=“verbatim,attributes”] — template = cfn\_inc.CfnInclude(self, “Template”, + template\_file=“my-template.json”, parameters=dict(UploadBucket=bucket.bucket\_arn) ) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnInclude template = CfnInclude.Builder.create(this, “Template”) .templateFile(“my-template.json”) .parameters(java.util.Map.of( // Map.of requires Java 9+ “UploadBucket”, bucket.getBucketArn())) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var template = new cfnInc.CfnInclude(this, “Template”, new cfnInc.CfnIncludeProps { TemplateFile = “my-template.json”, Parameters = new Dictionary<string, string> { { “UploadBucket”, bucket.BucketArn } } }); — ====

[#use-cfn-template-cfninclude-other] == Import other template elements

You can import any {aws} CloudFormation template element, not just resources. The imported elements become a part of the {aws} CDK stack. To import these elements, use the following methods of the CfnInclude object:

* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrconditionconditionname[getCondition()] – {aws} CloudFormation https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/conditions-section-structure.html[conditions].
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrhookhooklogicalid[getHook()] – {aws} CloudFormation https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/blue-green.html[hooks] for blue/green deployments.
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrmappingmappingname[getMapping()] – {aws} CloudFormation https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/mappings-section-structure.html[mappings].
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbroutputlogicalid[getOutput()] – {aws} CloudFormation https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/outputs-section-structure.html[outputs].
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrparameterparametername[getParameter()] – {aws} CloudFormation https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html[parameters].
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrrulerulename[getRule()] – {aws} CloudFormation link:https://docs.aws.amazon.com/servicecatalog/latest/adminguide/reference-template\_constraint\_rules.html[rules] for {aws} Service Catalog templates.

Each of these methods return an instance of a class that represents the specific type of {aws} CloudFormation element. These objects are mutable. Changes that you make to them will appear in the template that gets generated from the {aws} CDK stack. The following is an example that imports a parameter from the template and modifies its default value:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const param = template.getParameter(‘MyParameter’); param.default = “{aws} CDK” —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const param = template.getParameter(‘MyParameter’); param.default = “{aws} CDK” —

Python:: + [source,python,subs=“verbatim,attributes”] — param = template.get\_parameter(“MyParameter”) param.default = “{aws} CDK” —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnParameter param = template.getParameter(“MyParameter”); param.setDefaultValue(“{aws} CDK”) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var cfnBucket = (CfnBucket)template.GetResource(“amzn-s3-demo-bucket”); var param = template.GetParameter(“MyParameter”); param.Default = “{aws} CDK”; — ====

[#use-cfn-template-cfninclude-nested] == Import nested stacks

You can import https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-nested-stacks.html[nested stacks] by specifying them either when you import their main template, or at some later point. The nested template must be stored in a local file, but referenced as a NestedStack resource in the main template. Also, the resource name used in the {aws} CDK code must match the name used for the nested stack in the main template.

Given this resource definition in the main template, the following code shows how to import the referenced nested stack both ways.

== [source,json,subs=“verbatim,attributes”]

“NestedStack”: { “Type”: “{aws}::CloudFormation::Stack”, “Properties”: { “TemplateURL”: “https://my-s3-template-source.s3.amazonaws.com/nested-stack.json” } } —

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // include nested stack when importing main stack const mainTemplate = new cfninc.CfnInclude(this, ‘MainStack’, { templateFile: ‘main-template.json’, loadNestedStacks: { ‘NestedStack’: { templateFile: ‘nested-template.json’, }, }, });

// or add it some time after importing the main stack const nestedTemplate = mainTemplate.loadNestedStack(‘NestedTemplate’, { templateFile: ‘nested-template.json’, }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // include nested stack when importing main stack const mainTemplate = new cfninc.CfnInclude(this, ‘MainStack’, { templateFile: ‘main-template.json’, loadNestedStacks: { ‘NestedStack’: { templateFile: ‘nested-template.json’, }, }, });

// or add it some time after importing the main stack const nestedTemplate = mainTemplate.loadNestedStack(‘NestedStack’, { templateFile: ‘my-nested-template.json’, }); —

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

= include nested stack when importing main stack

main\_template = cfn\_inc.CfnInclude(self, “MainStack”, + template\_file=“main-template.json”, load\_nested\_stacks=dict(NestedStack= cfn\_inc.CfnIncludeProps(template\_file=“nested-template.json”)))

= or add it some time after importing the main stack

nested\_template = main\_template.load\_nested\_stack(“NestedStack”, template\_file=“nested-template.json”) —

Java:: + [source,java,subs=“verbatim,attributes”] — CfnInclude mainTemplate = CfnInclude.Builder.create(this, “MainStack”) .templateFile(“main-template.json”) .loadNestedStacks(java.util.Map.of( // Map.of requires Java 9+ “NestedStack”, CfnIncludeProps.builder() .templateFile(“nested-template.json”).build())) .build();

// or add it some time after importing the main stack IncludedNestedStack nestedTemplate = mainTemplate.loadNestedStack(“NestedTemplate”, CfnIncludeProps.builder() .templateFile(“nested-template.json”) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // include nested stack when importing main stack var mainTemplate = new cfnInc.CfnInclude(this, “MainStack”, new cfnInc.CfnIncludeProps { TemplateFile = “main-template.json”, LoadNestedStacks = new Dictionary<string, cfnInc.ICfnIncludeProps> { { “NestedStack”, new cfnInc.CfnIncludeProps { TemplateFile = “nested-template.json” } } } });

// or add it some time after importing the main stack var nestedTemplate = mainTemplate.LoadNestedStack(“NestedTemplate”, new cfnInc.CfnIncludeProps { TemplateFile = ‘nested-template.json’ }); — ====

You can import multiple nested stacks with either methods. When importing the main template, you provide a mapping between the resource name of each nested stack and its template file. This mapping can contain any number of entries. To do it after the initial import, call loadNestedStack() once for each nested stack.

After importing a nested stack, you can access it using the main template’s link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrnestedwbrstacklogicalid[getNestedStack()] method.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const nestedStack = mainTemplate.getNestedStack(‘NestedStack’).stack; —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const nestedStack = mainTemplate.getNestedStack(‘NestedStack’).stack; —

Python:: + [source,python,subs=“verbatim,attributes”] — nested\_stack = main\_template.get\_nested\_stack(“NestedStack”).stack —

Java:: + [source,java,subs=“verbatim,attributes”] — NestedStack nestedStack = mainTemplate.getNestedStack(“NestedStack”).getStack(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var nestedStack = mainTemplate.GetNestedStack(“NestedStack”).Stack; — ====

The getNestedStack() method returns an link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.cloudformation\_include.CfnInclude.html#getwbrnestedwbrstacklogicalid[IncludedNestedStack] instance. From this instance, you can access the {aws} CDK https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.NestedStack.html[NestedStack] instance via the stack property, as shown in the example. You can also access the original {aws} CloudFormation template object via includedTemplate, from which you can load resources and other {aws} CloudFormation elements.