:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#configure-synth] = Configure and perform CDK stack synthesis :info\_titleabbrev: Configure and perform synthesis :keywords: {aws} CDK, {aws} CloudFormation stack, synthesis, cdk synth, {aws} CloudFormation template

== [abstract]

## Before you can deploy an {aws} Cloud Development Kit ({aws} CDK) stack, it must first be synthesized. Stack synthesis is the process of creating an {aws} CloudFormation template from a CDK stack. The CloudFormation template is what you then deploy to provision your resources on {aws}.

// Content start

Before you can deploy an {aws} Cloud Development Kit ({aws} CDK) stack, it must first be synthesized. *Stack synthesis* is the process of producing an {aws} CloudFormation template and deployment artifacts from a CDK stack. The template and artifacts are known as the *cloud assembly*. The cloud assembly is what gets deployed to provision your resources on {aws}. For more information on how deployments work, see xref:deploy-how[How {aws} CDK deployments work].

[#configure-synth-bootstrap] == How synthesis and bootstrapping work together

For your CDK apps to properly deploy, the CloudFormation templates produced during synthesis must correctly specify the resources created during bootstrapping. Therefore, bootstrapping and synthesis must complement one another for a deployment to be successful:

* Bootstrapping is a one-time process of setting up an {aws} environment for {aws} CDK deployments. It configures specific {aws} resources in your environment that are used by the CDK for deployments. These are commonly referred to as *bootstrap resources*. For instructions on bootstrapping, see xref:bootstrapping-env[Bootstrap your environment for use with the {aws} CDK].
* CloudFormation templates produced during synthesis include information on which bootstrap resources to use. During synthesis, the CDK CLI doesn’t know specifically how your {aws} environment has been bootstrapped. Instead, the CDK CLI produces CloudFormation templates based on the synthesizer you configure for each CDK stack. For a deployment to be successful, the synthesizer must produce CloudFormation templates that reference the correct bootstrap resources to use.

The CDK comes with a default synthesizer and bootstrapping configuration that are designed to work together. If you customize one, you must apply relevant customizations to the other.

[#bootstrapping-synthesizers] == How to configure CDK stack synthesis

You configure CDK stack synthesis using the synthesizer property of your link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html[Stack] instance. This property specifies how your CDK stacks will be synthesized. You provide an instance of a class that implements link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.IStackSynthesizer.html[IStackSynthesizer] or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.IReusableStackSynthesizer.html[IReusableStackSynthesizer]. Its methods will be invoked every time an asset is added to the stack or when the stack is synthesized. The following is a basic example of using this property within your stack:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyStack(this, ‘MyStack’, { // stack properties synthesizer: new DefaultStackSynthesizer({ // synthesizer properties }), }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyStack(this, ‘MyStack’, { // stack properties synthesizer: new DefaultStackSynthesizer({ // synthesizer properties }), }); —

Python:: + [source,python,subs=“verbatim,attributes”] — MyStack(self, “MyStack”, # stack properties synthesizer=DefaultStackSynthesizer( # synthesizer properties )) —

Java:: + [source,java,subs=“verbatim,attributes”] — new MyStack(app, “MyStack”, StackProps.builder() // stack properties .synthesizer(DefaultStackSynthesizer.Builder.create() // synthesizer properties .build()) .build(); ) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new MyStack(app, “MyStack”, new StackProps // stack properties { Synthesizer = new DefaultStackSynthesizer(new DefaultStackSynthesizerProps { // synthesizer properties }) }); —

Go:: + [source,go,subs=“verbatim,attributes”] — func main() { app := awscdk.NewApp(nil)

NewMyStack(app, “MyStack”, &MyStackProps{ StackProps: awscdk.StackProps{ Synthesizer: awscdk.NewDefaultStackSynthesizer(&awscdk.DefaultStackSynthesizerProps{ // synthesizer properties }), }, })

app.Synth(nil) } — ====

You can also configure a synthesizer for all CDK stacks in your CDK app using the defaultStackSynthesizer property of your App instance:

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

const app = new App({ // Configure for all stacks in this app defaultStackSynthesizer: new DefaultStackSynthesizer({ /\* … \*/ }), }); —

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

const app = new App({ // Configure for all stacks in this app defaultStackSynthesizer: new DefaultStackSynthesizer({ /\* … \*/ }), }); —

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

app = App( default\_stack\_synthesizer=DefaultStackSynthesizer( # Configure for all stacks in this app # … ) ) —

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

public class Main { public static void main(final String[] args) { App app = new App(AppProps.builder() // Configure for all stacks in this app .defaultStackSynthesizer(DefaultStackSynthesizer.Builder.create().build()) .build() ); } } —

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

namespace MyNamespace { sealed class Program { public static void Main(string[] args) { var app = new App(new AppProps { // Configure for all stacks in this app DefaultStackSynthesizer = new DefaultStackSynthesizer(new DefaultStackSynthesizerProps { // … }) }); } } } —

Go:: + [source,go,subs=“verbatim,attributes”] — package main

import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/constructs-go/constructs/v10” “github.com/aws/jsii-runtime-go” )

func main() { defer jsii.Close()

app := awscdk.NewApp(&awscdk.AppProps{ // Configure for all stacks in this app DefaultStackSynthesizer: awscdk.NewDefaultStackSynthesizer(&awscdk.DefaultStackSynthesizerProps{ // … }), }) } —- ====

By default, the {aws} CDK uses +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.DefaultStackSynthesizer.html[+DefaultStackSynthesizer`]. If you don’t configure a synthesizer, this synthesizer will be used.

If you don’t modify bootstrapping, such as making changes to the bootstrap stack or template, you don’t have to modify stack synthesis. You don’t even have to provide a synthesizer. The CDK will use the default DefaultStackSynthesizer class to configure CDK stack synthesis to properly interact with your bootstrap stack.

[#configure-synth-stack] == How to synthesize a CDK stack

To synthesize a CDK stack, use the {aws} CDK Command Line Interface ({aws} CDK CLI) cdk synth command. For more information about this command, including options that you can use with this command, see xref:ref-cli-cmd-synth[cdk synthesize].

If your CDK app contains a single stack, or to synthesize all stacks, you don’t have to provide the CDK stack name as an argument. By default, the CDK CLI will synthesize your CDK stacks into {aws} CloudFormation templates. A json formatted template for each stack is saved to the cdk.out directory. If your app contains a single stack, a yaml formatted template is printed to stdout. The following is an example:

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

$ cdk synth Resources: CDKMetadata: Type: {aws}::CDK::Metadata Properties: Analytics: v2:deflate64:H4sIAAAAAAAA/unique-identifier Metadata: aws:cdk:path: CdkAppStack/CDKMetadata/Default Condition: CDKMetadataAvailable … —

If your CDK app contains multiple stacks, you can provide the logical ID of a stack to synthesize a single stack. The following is an example:

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

## $ cdk synth MyStackName

If you don’t synthesize a stack and run cdk deploy, the CDK CLI will automatically synthesize your stack before deployment.

[#how-synth-default] == How synthesis works by default

[#how-synth-default-logical-ids] *Generated logical IDs in your {aws} CloudFormation template*:: + When you synthesize a CDK stack to produce a CloudFormation template, logical IDs are generated from the following sources, formatted as <construct-path><construct-ID><unique-hash>: +

* *Construct path* – The entire path to the construct in your CDK app. This path excludes the ID of the L1 construct, which is always Resource or Default, and the ID of the top-level stack that it’s a part of.
* *Construct ID* – The ID that you provide as the second argument when instantiating your construct.
* *Unique hash* – The {aws} CDK generates an 8 character unique hash using a deterministic hashing algorithm. This unique hash helps to ensure that logical ID values in your template are unique from one another. The deterministic behavior of this hash generation ensures that the generated logical ID value for each construct remains the same every time that you perform synthesis. The hash value will only change if you modify specific construct values such as your construct’s ID or its path.
* Logical IDs have a maximum length of 255 characters. Therefore, the {aws} CDK will truncate the construct path and construct ID if necessary to keep within that limit.
* The following is an example of a construct that defines an Amazon Simple Storage Service (Amazon S3) bucket. Here, we pass myBucket as the ID for our construct:
* ==== [role=“tablist”] TypeScript::

## [source,javascript,subs=“verbatim,attributes”]

* import \* as cdk from ‘aws-cdk-lib’; import { Construct} from ‘constructs’; import \* as s3 from ‘aws-cdk-lib/aws-s3’;

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

// Define the S3 bucket new s3.Bucket(this, ‘myBucket’, { versioned: true, removalPolicy: cdk.RemovalPolicy.DESTROY, }); } } —-

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

class MyCdkAppStack extends cdk.Stack {

constructor(scope, id, props) { super(scope, id, props);

new s3.Bucket(this, ‘myBucket’, { versioned: true, removalPolicy: cdk.RemovalPolicy.DESTROY, }); } }

== module.exports = { MyCdkAppStack }

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

class MyCdkAppStack(Stack): def *init*(self, scope: Construct, construct\_id: str, **kwargs) -> None: super().*init*(scope, construct\_id,** kwargs)

s3.Bucket(self, ‘MyBucket’, versioned=True, removal\_policy=cdk.RemovalPolicy.DESTROY ) —-

Java:: + [source,java,subs=“verbatim,attributes”] — package com.myorg;

import software.constructs.Construct; import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.services.s3.Bucket; import software.amazon.awscdk.services.s3.BucketProps; import software.amazon.awscdk.RemovalPolicy;

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

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

Bucket.Builder.create(this, "myBucket")  
 .versioned(true)  
 .removalPolicy(RemovalPolicy.DESTROY)  
 .build();

} } —- ….

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

namespace MyCdkApp { public class MyCdkAppStack : Stack { public MyCdkAppStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { new Bucket(this, “myBucket”, new BucketProps { Versioned = true, RemovalPolicy = RemovalPolicy.DESTROY }); } } } —

Go:: + [source,go,subs=“verbatim,attributes”] — package main

import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awss3” “github.com/aws/constructs-go/constructs/v10” “github.com/aws/jsii-runtime-go” )

type MyCdkAppStackProps struct { awscdk.StackProps }

func NewMyCdkAppStack(scope constructs.Construct, id string, props \*MyCdkAppStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. awss3.NewBucket(stack, jsii.String(“myBucket”), &awss3.BucketProps{ Versioned: jsii.Bool(true), RemovalPolicy: awscdk.RemovalPolicy\_DESTROY, })

return stack } ….

== // …

==== + When we run cdk synth, a logical ID in the format of myBucket<unique-hash> gets generated. The following is an example of this resource in the generated {aws} CloudFormation template: + [source,yaml,subs=“verbatim,attributes”] — Resources: myBucket5AF9C99B: Type: {aws}::S3::Bucket Properties: VersioningConfiguration: Status: Enabled UpdateReplacePolicy: Delete DeletionPolicy: Delete Metadata: aws:cdk:path: S3BucketAppStack/myBucket/Resource — + The following is an example of a custom construct named Bar that defines an Amazon S3 bucket. The Bar construct includes the custom construct Foo in its path: + ==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘aws-cdk-lib’; import { Construct } from ‘constructs’; import \* as s3 from ‘aws-cdk-lib/aws-s3’;

// Define the Bar construct export class Bar extends Construct { constructor(scope: Construct, id: string) { super(scope, id);

// Define an S3 bucket inside of Bar new s3.Bucket(this, ‘Bucket’, { versioned: true, removalPolicy: cdk.RemovalPolicy.DESTROY, } ); } }

// Define the Foo construct export class Foo extends Construct { constructor(scope: Construct, id: string) { super(scope, id);

// Create an instance of Bar inside Foo new Bar(this, ‘Bar’); } }

// Define the CDK stack export class MyCustomAppStack extends cdk.Stack { constructor(scope: Construct, id: string, props?: cdk.StackProps) { super(scope, id, props);

// Instantiate Foo construct in the stack new Foo(this, ‘Foo’); } } —-

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

// Define the Bar construct class Bar extends Construct { constructor(scope, id) { super(scope, id);

// Define an S3 bucket inside of Bar new s3.Bucket(this, ‘Bucket’, { versioned: true, removalPolicy: cdk.RemovalPolicy.DESTROY, }); } }

// Define the Foo construct class Foo extends Construct { constructor(scope, id) { super(scope, id);

// Create an instance of Bar inside Foo new Bar(this, ‘Bar’); } }

// Define the CDK stack class MyCustomAppStack extends cdk.Stack { constructor(scope, id, props) { super(scope, id, props);

// Instantiate Foo construct in the stack new Foo(this, ‘Foo’); } }

== module.exports = { MyCustomAppStack }

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

= Define the Bar construct

class Bar(Construct): def *init*(self, scope: Construct, id: str) -> None: super().*init*(scope, id)

# Define an S3 bucket inside of Bar  
 s3.Bucket(self, 'Bucket',  
 versioned=True,  
 removal\_policy=RemovalPolicy.DESTROY  
 )

= Define the Foo construct

class Foo(Construct): def *init*(self, scope: Construct, id: str) -> None: super().*init*(scope, id)

# Create an instance of Bar inside Foo  
 Bar(self, 'Bar')

= Define the CDK stack

class MyCustomAppStack(Stack): def *init*(self, scope: Construct, id: str, **kwargs) -> None: super().*init*(scope, id,** kwargs)

# Instantiate Foo construct in the stack  
 Foo(self, 'Foo') ----

Java:: In my-custom-app/src/main/java/com/myorg/Bar.java: + [source,java,subs=“verbatim,attributes”] — package com.myorg;

import software.constructs.Construct; import software.amazon.awscdk.services.s3.Bucket; import software.amazon.awscdk.services.s3.BucketProps; import software.amazon.awscdk.RemovalPolicy;

public class Bar extends Construct { public Bar(final Construct scope, final String id) { super(scope, id);

// Define an S3 bucket inside Bar  
 Bucket.Builder.create(this, "Bucket")  
 .versioned(true)  
 .removalPolicy(RemovalPolicy.DESTROY)  
 .build();

} } —- + In my-custom-app/src/main/java/com/myorg/Foo.java: + [source,java,subs=“verbatim,attributes”] —- package com.myorg;

import software.constructs.Construct;

public class Foo extends Construct { public Foo(final Construct scope, final String id) { super(scope, id);

// Create an instance of Bar inside Foo  
 new Bar(this, "Bar");

} } —- + In my-custom-app/src/main/java/com/myorg/MyCustomAppStack.java: + [source,java,subs=“verbatim,attributes”] —- package com.myorg;

import software.constructs.Construct; import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps;

public class MyCustomAppStack extends Stack { public MyCustomAppStack(final Construct scope, final String id, final StackProps props) { super(scope, id, props);

…. // Instantiate Foo construct in the stack new Foo(this, “Foo”); }

// Overload constructor in case StackProps is not provided public MyCustomAppStack(final Construct scope, final String id) { this(scope, id, null); } } —- ….

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

namespace MyCustomApp { // Define the Bar construct public class Bar : Construct { public Bar(Construct scope, string id) : base(scope, id) { // Define an S3 bucket inside Bar new Bucket(this, “Bucket”, new BucketProps { Versioned = true, RemovalPolicy = RemovalPolicy.DESTROY }); } }

…. // Define the Foo construct public class Foo : Construct { public Foo(Construct scope, string id) : base(scope, id) { // Create an instance of Bar inside Foo new Bar(this, “Bar”); } }

// Define the CDK Stack public class MyCustomAppStack : Stack { public MyCustomAppStack(Construct scope, string id, StackProps props = null) : base(scope, id, props) { // Instantiate Foo construct in the stack new Foo(this, “Foo”); } } } —- ….

Go:: + [source,go,subs=“verbatim,attributes”] — package main

import ( “github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awss3” “github.com/aws/constructs-go/constructs/v10” “github.com/aws/jsii-runtime-go” )

// Define the Bar construct type Bar struct { constructs.Construct }

func NewBar(scope constructs.Construct, id string) constructs.Construct { bar := constructs.NewConstruct(scope, &id)

…. // Define an S3 bucket inside Bar awss3.NewBucket(bar, jsii.String(“Bucket”), &awss3.BucketProps{ Versioned: jsii.Bool(true), RemovalPolicy: awscdk.RemovalPolicy\_DESTROY, })

return bar } ….

// Define the Foo construct type Foo struct { constructs.Construct }

func NewFoo(scope constructs.Construct, id string) constructs.Construct { foo := constructs.NewConstruct(scope, &id)

…. // Create an instance of Bar inside Foo NewBar(foo, “Bar”)

return foo } ….

// Define the CDK Stack type MyCustomAppStackProps struct { awscdk.StackProps }

func NewMyCustomAppStack(scope constructs.Construct, id string, props \*MyCustomAppStackProps) awscdk.Stack { stack := awscdk.NewStack(scope, &id, &props.StackProps)

…. // Instantiate Foo construct in the stack NewFoo(stack, “Foo”)

return stack } ….

// Define the CDK App func main() { app := awscdk.NewApp(nil)

…. NewMyCustomAppStack(app, “MyCustomAppStack”, &MyCustomAppStackProps{ StackProps: awscdk.StackProps{}, })

app.Synth(nil) } —- ==== + When we run cdk synth, a logical ID in the format of FooBarBucket<unique-hash> gets generated. The following is an example of this resource in the generated {aws} CloudFormation template: + [source,yaml,subs=“verbatim,attributes”] —- Resources: FooBarBucketBA3ED1FA: Type: {aws}::S3::Bucket Properties: VersioningConfiguration: Status: Enabled UpdateReplacePolicy: Delete DeletionPolicy: Delete # … —- ….

[#bootstrapping-custom-synth] == Customize CDK stack synthesis

If the default CDK synthesis behavior doesn’t suit your needs, you can customize CDK synthesis. To do this, you modify DefaultStackSynthesizer, use other available built-in synthesizers, or create your own synthesizer. For instructions, see xref:customize-synth[Customize CDK stack synthesis].

include::customize-synth.adoc[leveloffset=+1]