include::attributes.txt[]

// Attributes

[.topic] [#stack-how-to-create-multiple-stacks] = Example: Create a CDK app with multiple stacks :info\_titleabbrev: Example: CDK app with multiple stacks

// Content start

You can create an {aws} Cloud Development Kit ({aws} CDK) application containing multiple xref:stacks[stacks]. When you deploy the {aws} CDK app, each stack becomes its own {aws} CloudFormation template. You can also synthesize and deploy each stack individually using the {aws} CDK CLI cdk deploy command.

In this example, we cover the following:

* How to extend the Stack class to accept new properties or arguments.
* How to use properties to determine which resources the stack contains and their configuration.
* How to instantiate multiple stacks from this class.

The example in this topic uses a Boolean property, named encryptBucket (Python: encrypt\_bucket). It indicates whether an Amazon S3 bucket should be encrypted. If so, the stack enables encryption using a key managed by {aws} Key Management Service ({aws} KMS). The app creates two instances of this stack, one with encryption and one without.

[#cdk-how-to-create-multiple-stacks-prereqs] == Prerequisites

This example assumes that all xref:getting-started[getting started] steps have been completed.

[#cdk-how-to-create-multiple-stacks-create] == Create a CDK project

First, we create a CDK project using the CDK CLI:

==== [role=“tablist”] TypeScript:: + [source,none,subs=“verbatim,attributes”] — mkdir multistack cd multistack cdk init –language=typescript —

JavaScript:: + [source,none,subs=“verbatim,attributes”] — mkdir multistack cd multistack cdk init –language=javascript —

Python:: + [source,none,subs=“verbatim,attributes”] — mkdir multistack cd multistack cdk init –language=python source .venv/bin/activate # On Windows, run ‘.’ instead pip install -r requirements.txt —

Java:: + [source,none,subs=“verbatim,attributes”] — mkdir multistack cd multistack cdk init –language=java — + You can import the resulting Maven project into your Java IDE.

C#:: + [source,none,subs=“verbatim,attributes”] — mkdir multistack cd multistack cdk init –language=csharp — + You can open the file src/Pipeline.sln in Visual Studio. ====

[#cdk-how-to-create-multiple-stacks-extend-stackprops] == Add an optional parameter

The props argument of the Stack constructor fulfills the interface StackProps. In this example, we want the stack to accept an additional property to tell us whether to encrypt the Amazon S3 bucket. To do this, we create an interface or class that includes the property. This allows the compiler to make sure that the property has a Boolean value and enables auto-completion for it in your IDE.

We open our *stack file* in our IDE or editor and add the new interface, class, or argument. New lines are highlighted in bold:

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

interface MultiStackProps extends cdk.StackProps { encryptBucket?: boolean; }

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

// The code that defines our stack goes here } } —-

JavaScript:: File: lib/multistack-stack.js + JavaScript doesn’t have an interface feature; we don’t need to add any code. + [source,javascript,subs=“verbatim,attributes”] — const cdk = require(‘aws-cdk-stack’);

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

// The code that defines our stack goes here } }

== module.exports = { MultistackStack }

Python:: File: multistack/multistack\_stack.py + Python does not have an interface feature, so we’ll extend our stack to accept the new property by adding a keyword argument. + [source,python,subs=“verbatim,attributes”] — import aws\_cdk as cdk from constructs import Construct

class MultistackStack(cdk.Stack):

…. # The Stack class doesn’t know about our encrypt\_bucket parameter, # so accept it separately and pass along any other keyword arguments. def **init**(self, scope: Construct, id: str, \*, encrypt\_bucket=False, \*\*kwargs) -> None: super().\_\_init\_\_(scope, id, \*\*kwargs)

# The code that defines our stack goes here ----

….

Java:: File: src/main/java/com/myorg/MultistackStack.java + It’s more complicated than we really want to get into to extend a props type in Java. Instead, write the stack’s constructor to accept an optional Boolean parameter. Because props is an optional argument, we’ll write an additional constructor that lets you skip it. It will default to false. + [source,java,subs=“verbatim,attributes”] — package com.myorg;

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

import software.amazon.awscdk.services.s3.Bucket;

public class MultistackStack extends Stack { // additional constructors to allow props and/or encryptBucket to be omitted public MultistackStack(final Construct scope, final String id, boolean encryptBucket) { this(scope, id, null, encryptBucket); }

…. public MultistackStack(final Construct scope, final String id) { this(scope, id, null, false); }

public MultistackStack(final Construct scope, final String id, final StackProps props, final boolean encryptBucket) { super(scope, id, props);

// The code that defines our stack goes here

} } —- ….

C#:: File: src/Multistack/MultistackStack.cs + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using constructs;

namespace Multistack {

…. public class MultiStackProps : StackProps { public bool? EncryptBucket { get; set; } }

public class MultistackStack : Stack { public MultistackStack(Construct scope, string id, MultiStackProps props) : base(scope, id, props) { // The code that defines our stack goes here } } } —- ==== ….

The new property is optional. If encryptBucket (Python: encrypt\_bucket) is not present, its value is undefined, or the local equivalent. The bucket will be unencrypted by default.

[#cdk-how-to-create-multiple-stacks-define-stack] == Define the stack class

Next, we define our stack class, using our new property. New code is highlighted in bold:

==== [role=“tablist”] TypeScript:: File: lib/multistack-stack.ts + [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’;

interface MultistackProps extends cdk.StackProps { encryptBucket?: boolean; }

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

// Add a Boolean property “encryptBucket” to the stack constructor. // If true, creates an encrypted bucket. Otherwise, the bucket is unencrypted. // Encrypted bucket uses KMS-managed keys (SSE-KMS). if (props && props.encryptBucket) { new s3.Bucket(this, “MyGroovyBucket”, { encryption: s3.BucketEncryption.KMS\_MANAGED, removalPolicy: cdk.RemovalPolicy.DESTROY }); } else { new s3.Bucket(this, “MyGroovyBucket”, { removalPolicy: cdk.RemovalPolicy.DESTROY}); } } } —-

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

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

// Add a Boolean property “encryptBucket” to the stack constructor. // If true, creates an encrypted bucket. Otherwise, the bucket is unencrypted. // Encrypted bucket uses KMS-managed keys (SSE-KMS). if ( props && props.encryptBucket) { new s3.Bucket(this, “MyGroovyBucket”, { encryption: s3.BucketEncryption.KMS\_MANAGED, removalPolicy: cdk.RemovalPolicy.DESTROY }); } else { new s3.Bucket(this, “MyGroovyBucket”, { removalPolicy: cdk.RemovalPolicy.DESTROY}); } } }

== module.exports = { MultistackStack }

Python:: File: multistack/multistack\_stack.py + [source,python,subs=“verbatim,attributes”] — import aws\_cdk as cdk from constructs import Construct from aws\_cdk import aws\_s3 as s3

class MultistackStack(cdk.Stack):

…. # The Stack class doesn’t know about our encrypt\_bucket parameter, # so accept it separately and pass along any other keyword arguments. def **init**(self, scope: Construct, id: str, \*, encrypt\_bucket=False, \*\*kwargs) -> None: super().\_\_init\_\_(scope, id, \*\*kwargs)

# Add a Boolean property "encryptBucket" to the stack constructor.  
# If true, creates an encrypted bucket. Otherwise, the bucket is unencrypted.  
# Encrypted bucket uses KMS-managed keys (SSE-KMS).  
if encrypt\_bucket:  
 s3.Bucket(self, "MyGroovyBucket",  
 encryption=s3.BucketEncryption.KMS\_MANAGED,  
 removal\_policy=cdk.RemovalPolicy.DESTROY)  
else:  
 s3.Bucket(self, "MyGroovyBucket",  
 removal\_policy=cdk.RemovalPolicy.DESTROY) ----

….

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

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

import software.amazon.awscdk.services.s3.Bucket; import software.amazon.awscdk.services.s3.BucketEncryption;

public class MultistackStack extends Stack { // additional constructors to allow props and/or encryptBucket to be omitted public MultistackStack(final Construct scope, final String id, boolean encryptBucket) { this(scope, id, null, encryptBucket); }

…. public MultistackStack(final Construct scope, final String id) { this(scope, id, null, false); }

// main constructor public MultistackStack(final Construct scope, final String id, final StackProps props, final boolean encryptBucket) { super(scope, id, props);

// Add a Boolean property "encryptBucket" to the stack constructor.  
// If true, creates an encrypted bucket. Otherwise, the bucket is  
// unencrypted. Encrypted bucket uses KMS-managed keys (SSE-KMS).  
if (encryptBucket) {  
 Bucket.Builder.create(this, "MyGroovyBucket")  
 .encryption(BucketEncryption.KMS\_MANAGED)  
 .removalPolicy(RemovalPolicy.DESTROY).build();  
} else {  
 Bucket.Builder.create(this, "MyGroovyBucket")  
 .removalPolicy(RemovalPolicy.DESTROY).build();  
}

} } —- ….

C#:: File: src/Multistack/MultistackStack.cs + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Amazon.CDK.{aws}.S3;

namespace Multistack {

…. public class MultiStackProps : StackProps { public bool? EncryptBucket { get; set; } }

public class MultistackStack : Stack { public MultistackStack(Construct scope, string id, IMultiStackProps props = null) : base(scope, id, props) { // Add a Boolean property “EncryptBucket” to the stack constructor. // If true, creates an encrypted bucket. Otherwise, the bucket is unencrypted. // Encrypted bucket uses KMS-managed keys (SSE-KMS). if (props?.EncryptBucket ?? false) { new Bucket(this, “MyGroovyBucket”, new BucketProps { Encryption = BucketEncryption.KMS\_MANAGED, RemovalPolicy = RemovalPolicy.DESTROY }); } else { new Bucket(this, “MyGroovyBucket”, new BucketProps { RemovalPolicy = RemovalPolicy.DESTROY }); } } } } —- ==== ….

[#stack-how-to-create-multiple-stacks-create-stacks] == Create two stack instances

In our *application file*, we add the code to instantiate two separate stacks. We delete the existing MultistackStack definition and define our two stacks. New code is highlight in bold:

==== [role=“tablist”] TypeScript:: File: bin/multistack.ts + [source,javascript,subs=“verbatim,attributes”] — #!/usr/bin/env node import ‘source-map-support/register’; import \* as cdk from ‘aws-cdk-lib’; import { MultistackStack } from ‘../lib/multistack-stack’;

const app = new cdk.App();

new MultistackStack(app, “MyWestCdkStack”, { env: {region: “us-west-1”}, encryptBucket: false });

new MultistackStack(app, “MyEastCdkStack”, { env: {region: “us-east-1”}, encryptBucket: true });

== app.synth();

JavaScript:: File: bin/multistack.js + [source,javascript,subs=“verbatim,attributes”] — #!/usr/bin/env node const cdk = require(‘aws-cdk-lib’); const { MultistackStack } = require(‘../lib/multistack-stack’);

const app = new cdk.App();

new MultistackStack(app, “MyWestCdkStack”, { env: {region: “us-west-1”}, encryptBucket: false });

new MultistackStack(app, “MyEastCdkStack”, { env: {region: “us-east-1”}, encryptBucket: true });

== app.synth();

Python:: File: ./app.py + [source,python,subs=“verbatim,attributes”] — #!/usr/bin/env python3

import aws\_cdk as cdk

from multistack.multistack\_stack import MultistackStack

app = cdk.App() MultistackStack(app, “MyWestCdkStack”, env=cdk.Environment(region=“us-west-1”), encrypt\_bucket=False)

MultistackStack(app, “MyEastCdkStack”, env=cdk.Environment(region=“us-east-1”), encrypt\_bucket=True)

== app.synth()

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

import software.amazon.awscdk.App; import software.amazon.awscdk.Environment; import software.amazon.awscdk.StackProps;

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

…. new MultistackStack(app, “MyWestCdkStack”, StackProps.builder() .env(Environment.builder() .region(“us-west-1”) .build()) .build(), false);

new MultistackStack(app, "MyEastCdkStack", StackProps.builder()  
 .env(Environment.builder()  
 .region("us-east-1")  
 .build())  
 .build(), true);  
  
app.synth();

} } —- ….

C#:: File: src/Multistack/Program.cs + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK;

namespace Multistack { class Program { static void Main(string[] args) { var app = new App();

…. new MultistackStack(app, “MyWestCdkStack”, new MultiStackProps { Env = new Environment { Region = “us-west-1” }, EncryptBucket = false });

new MultistackStack(app, "MyEastCdkStack", new MultiStackProps  
 {  
 Env = new Environment { Region = "us-east-1" },  
 EncryptBucket = true  
 });  
  
 app.Synth();  
}

} } —- ==== ….

This code uses the new encryptBucket (Python: encrypt\_bucket) property on the MultistackStack class to instantiate the following:

* One stack with an encrypted Amazon S3 bucket in the us-east-1 {aws} Region.
* One stack with an unencrypted Amazon S3 bucket in the us-west-1 {aws} Region.

[#cdk-how-to-create-multiple-stacks-synth-deploy] == Synthesize and deploy the stack

Next, we can deploy stacks from the app. First, we synthesize an {aws} CloudFormation template for MyEastCdkStack. This is the stack in us-east-1 with the encrypted Amazon S3 bucket.

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

## $ cdk synth MyEastCdkStack

To deploy this stack to our {aws} environment, we can issue one of the following commands. The first command uses our default {aws} profile to obtain the credentials to deploy the stack. The second uses a profile that we specify. For PROFILE\_NAME, we can substitute the name of an {aws} CLI profile that contains appropriate credentials for deploying to the us-east-1 {aws} Region.

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

## $ cdk deploy MyEastCdkStack

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

$ cdk deploy MyEastCdkStack –profile=++++++—-++++++

[#cdk-how-to-create-multiple-stacks-destroy-stack] == Clean up

To avoid charges for resources that we deployed, we destroy the stack using the following command:

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

## cdk destroy MyEastCdkStack

The destroy operation fails if there is anything stored in the stack’s bucket. There shouldn’t be, since we only created the bucket. If we did put something in the bucket, we must delete the bucket contents before destroying the stack. We can use the {aws} Management Console or the {aws} CLI to delete the bucket contents.