:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#troubleshooting] = Troubleshooting common {aws} CDK issues :info\_titleabbrev: {aws} CDK troubleshooting

// Content start

This topic describes how to troubleshoot the following issues with the {aws} CDK.

* xref:troubleshooting-toolkit[After updating the {aws} CDK the {aws} CDK Toolkit (CLI) reports a mismatch with the {aws} Construct Library]
* xref:troubleshooting-nobucket[When deploying my {aws} CDK stack I receive a NoSuchBucket error]
* xref:troubleshooting-forbidden-null[When deploying my {aws} CDK stack I receive a forbidden: null message]
* xref:troubleshooting-app-required[When synthesizing an {aws} CDK stack I get the message –app is required either in command-line in cdk.json or in ~/.cdk.json]
* xref:troubleshooting-resource-count[When synthesizing an {aws} CDK stack I receive an error because the {aws} CloudFormation template contains too many resources]
* xref:troubleshooting-availability-zones[I specified three (or more) Availability Zones for my Auto Scaling group or VPC but it was only deployed in two]
* xref:troubleshooting-resource-not-deleted[My S3 bucket DynamoDB table or other resource is not deleted when I issue cdk destroy]

[#troubleshooting-toolkit] == After updating the {aws} CDK, the {aws} CDK Toolkit (CLI) reports a mismatch with the {aws} Construct Library

The version of the {aws} CDK Toolkit (which provides the cdk command) must be at least equal to the version of the main {aws} Construct Library module, aws-cdk-lib. The Toolkit is intended to be backward compatible. The latest 2.x version of the toolkit can be used with any 1.x or 2.x release of the library. For this reason, we recommend you install this component globally and keep it up to date.

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

## npm update -g aws-cdk

If you need to work with multiple versions of the {aws} CDK Toolkit, install a specific version of the toolkit locally in your project folder.

If you are using TypeScript or JavaScript, your project directory already contains a versioned local copy of the CDK Toolkit.

If you are using another language, use npm to install the {aws} CDK Toolkit, omitting the -g flag and specifying the desired version. For example:

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

## npm install aws-cdk@2.0

To run a locally installed {aws} CDK Toolkit, use the command npx aws-cdk instead of only cdk. For example:

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

## npx aws-cdk deploy MyStack

npx aws-cdk runs the local version of the {aws} CDK Toolkit if one exists. It falls back to the global version when a project doesn’t have a local installation. You may find it convenient to set up a shell alias to make sure cdk is always invoked this way.

==== [role=“tablist”] macOS/Linux:: + [source,none,subs=“verbatim,attributes”] — alias cdk=“npx aws-cdk” —

Windows:: + [source,none,subs=“verbatim,attributes”] — doskey cdk=npx aws-cdk $\* — ====

[#troubleshooting-nobucket] == When deploying my {aws} CDK stack, I receive a NoSuchBucket error

Your {aws} environment has not been bootstrapped, and so does not have an Amazon S3 bucket to hold resources during deployment. You can create the staging bucket and other required resources with the following command:

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

## cdk bootstrap aws://ACCOUNT-NUMBER/REGION

To avoid generating unexpected {aws} charges, the {aws} CDK does not automatically bootstrap any environment. You must explicitly bootstrap each environment into which you will deploy.

By default, the bootstrap resources are created in the Region or Regions that are used by stacks in the current {aws} CDK application. Alternatively, they are created in the Region specified in your local {aws} profile (set by aws configure), using that profile’s account. You can specify a different account and Region on the command line as follows. (You must specify the account and Region if you are not in an app’s directory.)

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

## cdk bootstrap aws://ACCOUNT-NUMBER/REGION

For more information, see xref:bootstrapping[{aws} CDK bootstrapping].

[#troubleshooting-forbidden-null] == When deploying my {aws} CDK stack, I receive a forbidden: null message

You are deploying a stack that requires bootstrap resources, but are using an IAM role or account that lacks permission to write to it. (The staging bucket is used when deploying stacks that contain assets or that synthesize an {aws} CloudFormation template larger than 50K.) Use an account or role that has permission to perform the action s3:\* against the bucket mentioned in the error message.

[#troubleshooting-app-required] == When synthesizing an {aws} CDK stack, I get the message --app is required either in command-line, in cdk.json or in ~/.cdk.json

This message usually means that you aren’t in the main directory of your {aws} CDK project when you issue cdk synth. The file cdk.json in this directory, created by the cdk init command, contains the command line needed to run (and thereby synthesize) your {aws} CDK app. For a TypeScript app, for example, the default cdk.json looks something like this:

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

{ “app”: “npx ts-node bin/my-cdk-app.ts” } —

We recommend issuing cdk commands only in your project’s main directory, so the {aws} CDK toolkit can find cdk.json there and successfully run your app.

If this isn’t practical for some reason, the {aws} CDK Toolkit looks for the app’s command line in two other locations:

* In cdk.json in your home directory
* On the cdk synth command itself using the -a option

For example, you might synthesize a stack from a TypeScript app as follows.

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

## cdk synth –app “npx ts-node my-cdk-app.ts” MyStack

[#troubleshooting-resource-count] == When synthesizing an {aws} CDK stack, I receive an error because the {aws} CloudFormation template contains too many resources

The {aws} CDK generates and deploys {aws} CloudFormation templates. {aws} CloudFormation has a hard limit on the number of resources a stack can contain. With the {aws} CDK, you can run up against this limit more quickly than you might expect.

= [NOTE]

The {aws} CloudFormation resource limit is 500 at this writing. See https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cloudformation-limits.html[{aws} CloudFormation quotas] for the current resource limit.

====

The {aws} Construct Library’s higher-level, intent-based constructs automatically provision any auxiliary resources that are needed for logging, key management, authorization, and other purposes. For example, granting one resource access to another generates any IAM objects needed for the relevant services to communicate.

In our experience, real-world use of intent-based constructs results in 1–5 {aws} CloudFormation resources per construct, though this can vary. For serverless applications, 5–8 {aws} resources per API endpoint is typical.

Patterns, which represent a higher level of abstraction, let you define even more {aws} resources with even less code. The {aws} CDK code in xref:ecs-example[Example: Create an {aws} Fargate service using the {aws} CDK], for example, generates more than 50 {aws} CloudFormation resources while defining only three constructs!

Exceeding the {aws} CloudFormation resource limit is an error during {aws} CloudFormation synthesis. The {aws} CDK issues a warning if your stack exceeds 80% of the limit. You can use a different limit by setting the maxResources property on your stack, or disable validation by setting maxResources to 0.

= [TIP]

You can get an exact count of the resources in your synthesized output using the following utility script. (Since every {aws} CDK developer needs Node.js, the script is written in JavaScript.)

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

// rescount.js - count the resources defined in a stack // invoke with: node rescount.js ++++++// e.g. node rescount.js cdk.out/MyStack.template.json++++++

import \* as fs from ‘fs’; const path = process.argv[2];

if (path) fs.readFile(path, ‘utf8’, function(err, contents) { console.log(err ? +${err}+ : +${Object.keys(JSON.parse(contents).Resources).length} resources defined in ${path}+); }); else console.log(“Please specify the path to the stack’s output .json file”); — ====

As your stack’s resource count approaches the limit, consider re-architecting to reduce the number of resources your stack contains: for example, by combining some Lambda functions, or by breaking your stack into multiple stacks. The CDK supports xref:resource-stack[references between stacks], so you can separate your app’s functionality into different stacks in whatever way makes the most sense to you.

= [NOTE]

{aws} CloudFormation experts often suggest the use of nested stacks as a solution to the resource limit. The {aws} CDK supports this approach via the xref:stack-nesting[NestedStack] construct.

====

[#troubleshooting-availability-zones] == I specified three (or more) Availability Zones for my Auto Scaling group or VPC, but it was only deployed in two

To get the number of Availability Zones that you request, specify the account and Region in the stack’s env property. If you do not specify both, the {aws} CDK, by default, synthesizes the stack as environment-agnostic. You can then deploy the stack to a specific Region using {aws} CloudFormation. Because some Regions have only two Availability Zones, an environment-agnostic template doesn’t use more than two.

= [NOTE]

In the past, Regions have occasionally launched with only one Availability Zone. Environment-agnostic {aws} CDK stacks cannot be deployed to such Regions. At this writing, however, all {aws} Regions have at least two AZs.

====

You can change this behavior by overriding your stack’s link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#availabilityzones[availabilityZones] (Python: availability\_zones) property to explicitly specify the zones that you want to use.

For more information about specifying a stack’s account and region at synthesis time, while retaining the flexibility to deploy to any region, see xref:environments[Environments for the {aws} CDK].

[#troubleshooting-resource-not-deleted] == My S3 bucket, DynamoDB table, or other resource is not deleted when I issue cdk destroy

By default, resources that can contain user data have a removalPolicy (Python: removal\_policy) property of RETAIN, and the resource is not deleted when the stack is destroyed. Instead, the resource is orphaned from the stack. You must then delete the resource manually after the stack is destroyed. Until you do, redeploying the stack fails. This is because the name of the new resource being created during deployment conflicts with the name of the orphaned resource.

If you set a resource’s removal policy to DESTROY, that resource will be deleted when the stack is destroyed.

==== [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 CdkTestStack extends cdk.Stack { constructor(scope: Construct, id: string, props?: cdk.StackProps) { super(scope, id, props);

const bucket = new s3.Bucket(this, ‘Bucket’, { 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 CdkTestStack extends cdk.Stack { constructor(scope, id, props) { super(scope, id, props);

const bucket = new s3.Bucket(this, ‘Bucket’, { removalPolicy: cdk.RemovalPolicy.DESTROY }); } }

== module.exports = { CdkTestStack }

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

class CdkTestStack(cdk.stack): def *init*(self, scope: Construct, id: str, **kwargs): super().*init*(scope, id,** kwargs)

bucket = s3.Bucket(self, "Bucket",  
 removal\_policy=cdk.RemovalPolicy.DESTROY) ----

Java:: + [source,java,subs=“verbatim,attributes”] — software.amazon.awscdk.\_; import software.amazon.awscdk.services.s3.\_; import software.constructs;

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

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

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

} } —- ….

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

public CdkTestStack(Construct scope, string id, IStackProps props) : base(scope, id, props) { new Bucket(this, “Bucket”, new BucketProps { RemovalPolicy = RemovalPolicy.DESTROY }); } — ====

= [NOTE]

{aws} CloudFormation cannot delete a non-empty Amazon S3 bucket. If you set an Amazon S3 bucket’s removal policy to DESTROY, and it contains data, attempting to destroy the stack will fail because the bucket cannot be deleted. You can have the {aws} CDK delete the objects in the bucket before attempting to destroy it by setting the bucket’s autoDeleteObjects prop to true.

====