:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#migrating-v2] = Migrating from {aws} CDK v1 to {aws} CDK v2

// Content start

Version 2 of the {aws} Cloud Development Kit ({aws} CDK) is designed to make writing infrastructure as code in your preferred programming language easier. This topic describes the changes between v1 and v2 of the {aws} CDK.

= [TIP]

To identify stacks deployed with {aws} CDK v1, use the https://www.npmjs.com/package/awscdk-v1-stack-finder[awscdk-v1-stack-finder] utility.

====

The main changes from {aws} CDK v1 to CDK v2 are as follows.

* {aws} CDK v2 consolidates the stable parts of the {aws} Construct Library, including the core library, into a single package, aws-cdk-lib. Developers no longer need to install additional packages for the individual {aws} services they use. This single-package approach also means that you don’t have to synchronize the versions of the various CDK library packages.
* L1 (CfnXXXX) constructs, which represent the exact resources available in {aws} CloudFormation, are always considered stable and so are included in aws-cdk-lib.
* Experimental modules, where we’re still working with the community to develop new xref:constructs-lib-levels[L2 or L3 constructs], are not included in aws-cdk-lib. Instead, they’re distributed as individual packages. Experimental packages are named with an alpha suffix and a semantic version number. The semantic version number matches the first version of the {aws} Construct Library that they’re compatible with, also with an alpha suffix. Constructs move into aws-cdk-lib after being designated stable, permitting the main Construct Library to adhere to strict semantic versioning. +
* Stability is specified at the service level. For example, if we begin creating one or more xref:constructs-lib-levels[L2 constructs] for Amazon AppFlow, which at this writing has only L1 constructs, they first appear in a module named @aws-cdk/aws-appflow-alpha. Then, they move to aws-cdk-lib when we feel that the new constructs meet the fundamental needs of customers.
* Once a module has been designated stable and incorporated into aws-cdk-lib, new APIs are added using the “BetaN” convention described in the next bullet. +
* A new version of each experimental module is released with every release of the {aws} CDK. For the most part, however, they needn’t be kept in sync. You can upgrade aws-cdk-lib or the experimental module whenever you want. The exception is that when two or more related experimental modules depend on each other, they must be the same version.
* For stable modules to which new functionality is being added, new APIs (whether entirely new constructs or new methods or properties on an existing construct) receive a Beta1 suffix while work is in progress. (Followed by Beta2, Beta3, and so on when breaking changes are needed.) A version of the API without the suffix is added when the API is designated stable. All methods except the latest (whether beta or final) are then deprecated.
* For example, if we add a new method grantPower() to a construct, it initially appears as grantPowerBeta1(). If breaking changes are needed (for example, a new required parameter or property), the next version of the method would be named grantPowerBeta2(), and so on. When work is complete and the API is finalized, the method grantPower() (with no suffix) is added, and the BetaN methods are deprecated.
* All the beta APIs remain in the Construct Library until the next major version (3.0) release, and their signatures will not change. You’ll see deprecation warnings if you use them, so you should move to the final version of the API at your earliest convenience. However, no future {aws} CDK 2.x releases will break your application.
* The Construct class has been extracted from the {aws} CDK into a separate library, along with related types. This is done to support efforts to apply the Construct Programming Model to other domains. If you are writing your own constructs or using related APIs, you must declare the constructs module as a dependency and make minor changes to your imports. If you are using advanced features, such as hooking into the CDK app lifecycle, more changes may be needed. For full details, link:https://github.com/aws/aws-cdk-rfcs/blob/master/text/0192-remove-constructs-compat.md#release-notes[see the RFC].
* Deprecated properties, methods, and types in {aws} CDK v1.x and its Construct Library have been removed completely from the CDK v2 API. In most supported languages, these APIs produce warnings under v1.x, so you may have already migrated to the replacement APIs. A complete link:https://github.com/aws/aws-cdk/blob/master/DEPRECATED\_APIs.md[list of deprecated APIs] in CDK v1.x is available on GitHub.
* Behavior that was gated by feature flags in {aws} CDK v1.x is enabled by default in CDK v2. The earlier feature flags are no longer needed, and in most cases they’re not supported. A few are still available to let you revert to CDK v1 behavior in very specific circumstances. For more information, see xref:migrating-v2-v1-upgrade-cdk-json[Updating feature flags].
* With CDK v2, the environments you deploy into must be bootstrapped using the modern bootstrap stack. The legacy bootstrap stack (the default under v1) is no longer supported. CDK v2 furthermore requires a new version of the modern stack. To upgrade your existing environments, re-bootstrap them. It is no longer necessary to set any feature flags or environment variables to use the modern bootstrap stack.

= [IMPORTANT]

The modern bootstrap template effectively grants the permissions implied by the --cloudformation-execution-policies to any {aws} account in the --trust list. By default, this extends permissions to read and write to any resource in the bootstrapped account. Make sure to xref:bootstrapping-customizing[configure the bootstrapping stack] with policies and trusted accounts that you are comfortable with.

====

[#migrating-v2-prerequisites] == New prerequisites

Most requirements for {aws} CDK v2 are the same as for {aws} CDK v1.x. Additional requirements are listed here.

* For TypeScript developers, TypeScript 3.8 or later is required.
* A new version of the CDK Toolkit is required for use with CDK v2. Now that CDK v2 is generally available, v2 is the default version when installing the CDK Toolkit. It is backward-compatible with CDK v1 projects, so you don’t need to keep the earlier version installed unless you want to create CDK v1 projects. To upgrade, issue npm install -g aws-cdk.

[#migrating-v2-dp-upgrade] == Upgrading from {aws} CDK v2 Developer Preview

If you’re using the CDK v2 Developer Preview, you have dependencies in your project on a Release Candidate version of the {aws} CDK, such as 2.0.0-rc1. Update these to 2.0.0, then update the modules installed in your project.

==== [role=“tablist”] TypeScript:: npm install or yarn install

JavaScript:: npm install or yarn install

Python:: + [source,none,subs=“verbatim,attributes”] — python -m pip install -r requirements.txt —

Java:: + [source,none,subs=“verbatim,attributes”] — mvn package —

C#:: + [source,none,subs=“verbatim,attributes”] — dotnet restore —

Go:: + [source,none,subs=“verbatim,attributes”] — go get — ====

After updating your dependencies, issue npm update -g aws-cdk to update the CDK Toolkit to the release version.

[#migrating-v2-v1-uppgrade] == Migrating from {aws} CDK v1 to CDK v2

To migrate your app to {aws} CDK v2, first update the feature flags in cdk.json. Then update your app’s dependencies and imports as necessary for the programming language that it’s written in.

[#migrating-v2-v1-recent-v1] === Updating to a recent v1

We are seeing a number of customers upgrading from an old version of {aws} CDK v1 to the most recent version of v2 in one step. While it is certainly possible to do that, you would be both upgrading across multiple years of changes (that unfortunately may not all have had the same amount of evolution testing we have today), as well as upgrading across versions with new defaults and a different code organization.

For the safest upgrade experience and to more easily diagnose the sources of any unexpected changes, we recommend you separate those two steps: first upgrade to the latest v1 version, then make the switch to v2 afterwards.

[#migrating-v2-v1-upgrade-cdk-json] === Updating feature flags

Remove the following v1 feature flags from cdk.json if they exist, as these are all active by default in {aws} CDK v2. If their old effect is important for your infrastructure, you will need to make source code changes. See link:https://github.com/aws/aws-cdk/blob/main/packages/%40aws-cdk/cx-api/FEATURE\_FLAGS.md[the list of flags on GitHub] for more information.

* @aws-cdk/core:enableStackNameDuplicates
* aws-cdk:enableDiffNoFail
* @aws-cdk/aws-ecr-assets:dockerIgnoreSupport
* @aws-cdk/aws-secretsmanager:parseOwnedSecretName
* @aws-cdk/aws-kms:defaultKeyPolicies
* @aws-cdk/aws-s3:grantWriteWithoutAcl
* @aws-cdk/aws-efs:defaultEncryptionAtRest

A handful of v1 feature flags can be set to false in order to revert to specific {aws} CDK v1 behaviors; see xref:featureflags-disabling[Reverting to v1 behavior] or the list on GitHub for a complete reference.

For both types of flags, use the cdk diff command to inspect the changes to your synthesized template to see if the changes to any of these flags affect your infrastructure.

[#work-with-cdk-v2-cli] === CDK Toolkit compatibility

CDK v2 requires v2 or later of the CDK Toolkit. This version is backward-compatible with CDK v1 apps. Therefore, you can use a single globally installed version of CDK Toolkit with all your {aws} CDK projects, whether they use v1 or v2. An exception is that CDK Toolkit v2 only creates CDK v2 projects.

If you need to create both v1 and v2 CDK projects, *do not install CDK Toolkit v2 globally.* (Remove it if you already have it installed: npm remove -g aws-cdk.) To invoke the CDK Toolkit, use npx to run v1 or v2 of the CDK Toolkit as desired.

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

npx aws-cdk@1.x init app –language typescript npx aws-cdk@2.x init app –language typescript —

= [TIP]

Set up command line aliases so you can use the cdk and cdk1 commands to invoke the desired version of the CDK Toolkit.

[role=“tablist”] macOS/Linux:: + [source,none,subs=“verbatim,attributes”] — alias cdk1=“npx aws-cdk@1.x” alias cdk=“npx aws-cdk@2.x” —

Windows:: + [source,none,subs=“verbatim,attributes”] — doskey cdk1=npx aws-cdk@1.x $ *doskey cdk=npx aws-cdk@2.x $* — ====

[#migrating-v2-v1-upgrade-dependencies] === Updating dependencies and imports

Update your app’s dependencies, then install the new packages. Finally, update the imports in your code.

==== [role=“tablist”] TypeScript:: + *Applications*::: + For CDK apps, update package.json as follows. Remove dependencies on v1-style individual stable modules and establish the lowest version of aws-cdk-lib you require for your application (2.0.0 here). + Experimental constructs are provided in separate, independently versioned packages with names that end in alpha and an alpha version number. The alpha version number corresponds to the first release of aws-cdk-lib with which they’re compatible. Here, we have pinned aws-codestar to v2.0.0-alpha.1. + [source,json,subs=“verbatim,attributes”] — { “dependencies”: { “aws-cdk-lib”: “{caret}2.0.0”, “@aws-cdk/aws-codestar-alpha”: “2.0.0-alpha.1”, “constructs”: “{caret}10.0.0” } } —

*Construct libraries*::: + For construct libraries, establish the lowest version of aws-cdk-lib you require for your application (2.0.0 here) and update package.json as follows. + Note that aws-cdk-lib appears both as a peer dependency and a dev dependency. + [source,json,subs=“verbatim,attributes”] — { “peerDependencies”: { “aws-cdk-lib”: “{caret}2.0.0”, “constructs”: “{caret}10.0.0” }, “devDependencies”: { “aws-cdk-lib”: “{caret}2.0.0”, “constructs”: “{caret}10.0.0”, “typescript”: “~3.9.0” } } — + [NOTE] – You should perform a major version bump on your library’s version number when releasing a v2-compatible library, because this is a breaking change for library consumers. It is not possible to support both CDK v1 and v2 with a single library. To continue to support customers who are still using v1, you could maintain the earlier release in parallel, or create a new package for v2.

== It’s up to you how long you want to continue supporting {aws} CDK v1 customers. You might take your cue from the lifecycle of CDK v1 itself, which entered maintenance on June 1, 2022 and will reach end-of-life on June 1, 2023. For full details, see link:https://github.com/aws/aws-cdk-rfcs/blob/master/text/0079-cdk-2.0.md#aws-cdk-maintenance-policy[{aws} CDK Maintenance Policy].

*Both libraries and apps*::: + Install the new dependencies by running npm install or yarn install. + Change your imports to import Construct from the new constructs module, core types such as App and Stack from the top level of aws-cdk-lib, and stable Construct Library modules for the services you use from namespaces under aws-cdk-lib. + [source,javascript,subs=“verbatim,attributes”] — import { Construct } from ‘constructs’; import { App, Stack } from ‘aws-cdk-lib’; // core constructs import { aws\_s3 as s3 } from ‘aws-cdk-lib’; // stable module import \* as codestar from ‘@aws-cdk/aws-codestar-alpha’; // experimental module —

JavaScript:: + Update package.json as follows. Remove dependencies on v1-style individual stable modules and establish the lowest version of aws-cdk-lib you require for your application (2.0.0 here). + Experimental constructs are provided in separate, independently versioned packages with names that end in alpha and an alpha version number. The alpha version number corresponds to the first release of aws-cdk-lib with which they’re compatible. Here, we have pinned aws-codestar to v2.0.0-alpha.1. + [source,json,subs=“verbatim,attributes”] — { “dependencies”: { “aws-cdk-lib”: “{caret}2.0.0”, “@aws-cdk/aws-codestar-alpha”: “2.0.0-alpha.1”, “constructs”: “{caret}10.0.0” } } — + Install the new dependencies by running npm install or yarn install. + Change your app’s imports to do the following: + – \*\* Import Construct from the new constructs module \*\* Import core types, such as App and Stack, from the top level of aws-cdk-lib \*\* Import {aws} Construct Library modules from namespaces under aws-cdk-lib – + [source,javascript,subs=“verbatim,attributes”] — const { Construct } = require(‘constructs’); const { App, Stack } = require(‘aws-cdk-lib’); // core constructs const s3 = require(‘aws-cdk-lib’).aws\_s3; // stable module const codestar = require(‘@aws-cdk/aws-codestar-alpha’); // experimental module —

Python:: Update requirements.txt or the install\_requires definition in setup.py as follows. Remove dependencies on v1-style individual stable modules. + Experimental constructs are provided in separate, independently versioned packages with names that end in alpha and an alpha version number. The alpha version number corresponds to the first release of aws-cdk-lib with which they’re compatible. Here, we have pinned aws-codestar to v2.0.0alpha1. + [source,python,subs=“verbatim,attributes”] — install\_requires=[ “aws-cdk-lib>=2.0.0”, “constructs>=10.0.0”, “aws-cdk.aws-codestar-alpha>=2.0.0alpha1”, # …], — + TIP: Uninstall any other versions of {aws} CDK modules already installed in your app’s virtual environment using pip uninstall. Then Install the new dependencies with python -m pip install -r requirements.txt. +

Change your app’s imports to do the following: + – \*\* Import Construct from the new constructs module \*\* Import core types, such as App and Stack, from the top level of aws\_cdk \*\* Import {aws} Construct Library modules from namespaces under aws\_cdk – + [source,python,subs=“verbatim,attributes”] — from constructs import Construct from aws\_cdk import App, Stack # core constructs from aws\_cdk import aws\_s3 as s3 # stable module import aws\_cdk.aws\_codestar\_alpha as codestar # experimental module

= …

class MyConstruct(Construct): # …

class MyStack(Stack): # …

== s3.Bucket(…)

Java:: In pom.xml, remove all software.amazon.awscdk dependencies for stable modules and replace them with dependencies on software.constructs (for Construct) and software.amazon.awscdk. + Experimental constructs are provided in separate, independently versioned packages with names that end in alpha and an alpha version number. The alpha version number corresponds to the first release of aws-cdk-lib with which they’re compatible. Here, we have pinned aws-codestar to v2.0.0-alpha.1. + [source,xml,subs=“verbatim,attributes”] —++++++++++++software.amazon.awscdk++++++ ++++++aws-cdk-lib++++++ ++++++2.0.0++++++++++++++++++++++++software.amazon.awscdk++++++ ++++++code-star-alpha++++++ ++++++2.0.0-alpha.1++++++++++++++++++++++++software.constructs++++++ ++++++constructs++++++ ++++++10.0.0++++++++++++

’’’

* Install the new dependencies by running mvn package.
* Change your code to do the following:

|  |
| --- |
| \*\* Import Construct from the new software.constructs library |
| \*\* Import core classes, like Stack and App, from software.amazon.awscdk |
| \*\* Import service constructs from software.amazon.awscdk.services |

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

* import software.constructs.Construct; import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.App; import software.amazon.awscdk.services.s3.Bucket; import software.amazon.awscdk.services.codestar.alpha.GitHubRepository; —

C#:: The most straightforward way to upgrade the dependencies of a C# CDK application is to edit the .csproj file manually. Remove all stable Amazon.CDK.\* package references and replace them with references to the Amazon.CDK.Lib and Constructs packages. + Experimental constructs are provided in separate, independently versioned packages with names that end in alpha and an alpha version number. The alpha version number corresponds to the first release of aws-cdk-lib with which they’re compatible. Here, we have pinned aws-codestar to v2.0.0-alpha.1. + [source,xml,subs=“verbatim,attributes”] —++++++++++++++++++++++++++++++++++++

’’’

* Install the new dependencies by running dotnet restore.
* Change the imports in your source files as follows.

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

* using Constructs; // for Construct class using Amazon.CDK; // for core classes like App and Stack using Amazon.CDK.{aws}.S3; // for stable constructs like Bucket using Amazon.CDK.Codestar.Alpha; // for experimental constructs —

Go:: Issue go get to update your dependencies to the latest version and update your project’s .mod file.

====

[#migrating-v2-diff.title] == Testing your migrated app before deploying

Before deploying your stacks, use cdk diff to check for unexpected changes to the resources. Changes to logical IDs (causing replacement of resources) are *not* expected.

Expected changes include but are not limited to:

* Changes to the CDKMetadata resource.
* Updated asset hashes.
* Changes related to the new-style stack synthesis. Applies if your app used the legacy stack synthesizer in v1.
* The addition of a CheckBootstrapVersion rule.

Unexpected changes are typically not caused by upgrading to {aws} CDK v2 in itself. Usually, they’re the result of deprecated behavior that was previously changed by feature flags. This is a symptom of upgrading from a version of CDK earlier than about 1.85.x. You would see the same changes upgrading to the latest v1.x release. You can usually resolve this by doing the following:

. Upgrade your app to the latest v1.x release . Remove feature flags . Revise your code as necessary . Deploy . Upgrade to v2

= [NOTE]

If your upgraded app is undeployable after the two-stage upgrade, https://github.com/aws/aws-cdk/issues/new/choose[report the issue].

====

When you are ready to deploy the stacks in your app, consider deploying a copy first so you can test it. The easiest way to do this is to deploy it into a different Region. However, you can also change the IDs of your stacks. After testing, be sure to destroy the testing copy with cdk destroy.

[#migrating-v2-trouble.title] == Troubleshooting

*TypeScript 'from' expected or ';' expected error in imports*:: + Upgrade to TypeScript 3.8 or later.

*Run ‘cdk bootstrap’*:: + If you see an error like the following: + — ❌ MyStack failed: Error: MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment been bootstrapped? Please run ‘cdk bootstrap’ (see https://docs.aws.amazon.com/cdk/latest/guide/bootstrapping.html) at CloudFormationDeployments.validateBootstrapStackVersion (…/aws-cdk/lib/api/cloudformation-deployments.ts:323:13) at processTicksAndRejections (internal/process/task\_queues.js:97:5) MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment been bootstrapped? Please run ‘cdk bootstrap’ (see https://docs.aws.amazon.com/cdk/latest/guide/bootstrapping.html) — + {aws} CDK v2 requires an updated bootstrap stack, and furthermore, all v2 deployments require bootstrap resources. (With v1, you could deploy simple stacks without bootstrapping.) For complete details, see xref:bootstrapping[{aws} CDK bootstrapping].

[#finding-v1-stacks.title] == Finding v1 stacks

When migrating your CDK application from v1 to v2, you might want to identify the deployed {aws} CloudFormation stacks that were created using v1. To do this, run the following command:

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

## npx awscdk-v1-stack-finder

For usage details, see the awscdk-v1-stack-finder https://github.com/cdklabs/awscdk-v1-stack-finder/blob/main/README.md[README].