# Deployment Guide

Everything you need to deploy AWS CDK applications to AWS environments.

## 🚀 Deployment Essentials

### Environment Setup

* [Bootstrapping](bootstrapping.md) - Prepare AWS environments for CDK
* [Bootstrap Environment](bootstrapping-env.md) - Environment-specific bootstrapping
* [Bootstrap Customizing](bootstrapping-customizing.md) - Custom bootstrap templates
* [Bootstrap Troubleshooting](bootstrapping-troubleshoot.md) - Common bootstrap issues

### Configuration and Access

* [Configure Access](configure-access.md) - Set up AWS credentials and permissions
* [Configure Environment](configure-env.md) - Environment configuration
* [Configure Synthesis](configure-synth.md) - Synthesis configuration options
* [SSO CLI Example](configure-access-sso-example-cli.md) - AWS SSO setup example

### Deployment Process

* [Deployment Chapter](chapter-deploy.md) - Overview of deployment process
* [CLI Reference](cli.md) - CDK CLI commands and options
* [Deploy Troubleshooting](deploy-troubleshoot.md) - Common deployment issues

### Permissions and Security

* [Permissions](permissions.md) - IAM permissions for CDK deployments
* [Usage Data](usage-data.md) - Understanding CDK usage metrics

## 🎯 Quick Start Deployment

### First-Time Setup

1. [**Configure Access**](configure-access.md) - Set up AWS credentials
2. [**Bootstrapping**](bootstrapping.md) - Prepare your AWS environment
3. [**CLI Usage**](cli.md) - Learn essential CDK CLI commands

### Environment-Specific Deployment

1. [**Configure Environment**](configure-env.md) - Set up multiple environments
2. [**Bootstrap Environment**](bootstrapping-env.md) - Bootstrap specific environments
3. [**Deployment Chapter**](chapter-deploy.md) - Deploy to different environments

### Troubleshooting

* [**Bootstrap Issues**](bootstrapping-troubleshoot.md) - Fix bootstrap problems
* [**Deploy Issues**](deploy-troubleshoot.md) - Resolve deployment failures

## 📋 Pre-Deployment Checklist

* ☐ AWS credentials configured
* ☐ Target environment bootstrapped
* ☐ Necessary permissions assigned
* ☐ Application tested locally
* ☐ CloudFormation templates synthesized

## 🔗 Related Sections

* [Getting Started](../01-getting-started/) - Initial setup
* [Development](../03-development/) - Prepare your application
* [Advanced](../05-advanced/) - Advanced deployment patterns

include::attributes.txt[]

// Attributes

[.topic] [#bootstrapping-customizing] = Customize {aws} CDK bootstrapping :info\_titleabbrev: Customize bootstrapping :keywords: {aws} CDK, {aws} Cloud Development Kit ({aws} CDK), {aws} account, {aws} Region, Bootstrapping, Bootstrap, Environment, Customize

== [abstract]

## You can customize {aws} Cloud Development Kit ({aws} CDK) bootstrapping by using the {aws} CDK Command Line Interface ({aws} CDK CLI) or by modifying and deploying the {aws} CloudFormation bootstrap template.

// Content start

You can customize {aws} Cloud Development Kit ({aws} CDK) bootstrapping by using the {aws} CDK Command Line Interface ({aws} CDK CLI) or by modifying and deploying the {aws} CloudFormation bootstrap template.

For an introduction to bootstrapping, see xref:bootstrapping[{aws} CDK bootstrapping].

[#bootstrapping-customizing-cli] == Use the CDK CLI to customize bootstrapping

The following are a few examples of how you can customize bootstrapping by using the CDK CLI. For a list of all cdk bootstrap options, see xref:ref-cli-cmd-bootstrap[cdk bootstrap](#cdk-bootstrap).

[#bootstrapping-customizing-cli-s3-name] *Override the name of the Amazon S3 bucket*:: Use the --bootstrap-bucket-name option to override the default Amazon S3 bucket name. This may require that you modify template synthesis. For more information, see xref:bootstrapping-custom-synth[Customize CDK stack synthesis].

[#bootstrapping-customizing-keys] *Modify server-side encryption keys for the Amazon S3 bucket*:: By default, the Amazon S3 bucket in the bootstrap stack is configure to use {aws} managed keys for server-side encryption. To use an existing customer managed key, use the --bootstrap-kms-key-id option and provide a value for the {aws} Key Management Service ({aws} KMS) key to use. If you want more control over the encryption key, provide --bootstrap-customer-key to use a customer managed key.

[#bootstrapping-customizing-cli-deploy-role] *Attach managed policies to the deployment role assumed by {aws} CloudFormation*:: By default, stacks are deployed with full administrator permissions using the AdministratorAccess policy. To use your own managed policies, use the --cloudformation-execution-policies option and provide the ARNs of the managed policies to attach to the deployment role. + To provide multiple policies, pass them a single string, separated by commas: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap –cloudformation-execution-policies “arn:aws:iam::aws:policy/AWSLambda\_FullAccess,arn:aws:iam::aws:policy/AWSCodeDeployFullAccess” — + To avoid deployment failures, be sure that the policies you specify are sufficient for any deployments that you will perform into the environment being bootstrapped. + [#bootstrapping-customizing-cli-qualifier] *Change the qualifier that is added to the names of resources in your bootstrap stack*:: By default, the hnb659fds qualifier is added to the physical ID of resources in your bootstrap stack. To change this value, use the --qualifier option. + This modification is useful when provisioning multiple bootstrap stacks in the same environment to avoid name clashes. + Changing the qualifier is intended for name isolation between automated tests of the CDK itself. Unless you can very precisely scope down the IAM permissions given to the CloudFormation execution role, there are no permission isolation benefits to having two different bootstrap stacks in a single account. Therefore, there’s usually no need to change this value. + When you change the qualifier, your CDK app must pass the changed value to the stack synthesizer. For more information, see xref:bootstrapping-custom-synth[Customize CDK stack synthesis]. + [#bootstrapping-customizing-cli-tags] *Add tags to the bootstrap stack*:: Use the --tags option in the format of KEY=VALUE to add CloudFormation tags to your bootstrap stack. + [#bootstrapping-customizing-cli-accounts-deploy] *Specify additional {aws} accounts that can deploy into the environment being bootstrapped*:: Use the --trust option to provide additional {aws} accounts that are allowed to deploy into the environment being bootstrapped. By default, the account performing the bootstrapping will always be trusted. + This option is useful when you are bootstrapping an environment that a CDK [.noloc]Pipeline from another environment will deploy into. + When you use this option, you must also provide --cloudformation-execution-policies. + To add trusted accounts to an existing bootstrap stack, you must specify all of the accounts to trust, including those that you may have previously provided. If you only provide new accounts to trust, the previously trusted accounts will be removed. + The following is an example that trusts two accounts: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap aws://123456789012/us-west-2 –trust 234567890123 –trust 987654321098 –cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess ⏳ Bootstrapping environment aws://123456789012/us-west-2… Trusted accounts for deployment: 234567890123, 987654321098 Trusted accounts for lookup: (none) Execution policies: arn:aws:iam::aws:policy/AdministratorAccess CDKToolkit: creating CloudFormation changeset… ✅ Environment aws://123456789012/us-west-2 bootstrapped. —

[#bootstrapping-customizing-cli-accounts-lookup] *Specify additional {aws} accounts that can look up information in the environment being bootstrapped*:: + Use the --trust-for-lookup option to specify {aws} accounts that are allowed to look up context information from the environment being bootstrapped. This option is useful to give accounts permission to synthesize stacks that will be deployed into the environment, without actually giving them permission to deploy those stacks directly.

[#bootstrapping-customizing-cli-protection] *Enable termination protection for the bootstrap stack*:: If a bootstrap stack is deleted, the {aws} resources that were originally provisioned in the environment will also be deleted. After your environment is bootstrapped, we recommend that you don’t delete and recreate the environment’s bootstrap stack, unless you are intentionally doing so. Instead, try to update the bootstrap stack to a new version by running the cdk bootstrap command again. + Use the --termination-protection option to manage termination protection settings for the bootstrap stack. By enabling termination protection, you prevent the bootstrap stack and its resources from being accidentally deleted. This is especially important if you use CDK [.noloc]Pipelines since there is no general recovery option if you accidentally delete the bootstrap stack. + After enabling termination protection, you can use the {aws} CLI or {aws} CloudFormation console to verify. + *To enable termination protection*::: + . Run the following command to enable termination protection on a new or existing bootstrap stack: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap –termination-protection — + . Use the {aws} CLI or CloudFormation console to verify. The following is an example, using the {aws} CLI. If you modified your bootstrap stack name, replace CDKToolkit with your stack name: + [source,none,subs=“verbatim,attributes”] — $ aws cloudformation describe-stacks –stack-name ++++++–query “Stacks[0].EnableTerminationProtection” true —-++++++

[#bootstrapping-customizing-template] == Modify the default bootstrap template

When you need more customization than the CDK CLI can provide, you can modify the bootstrap template as needed. Then, deploy the template to bootstrap your environment.

*To modify and deploy the default bootstrap template*:: + . Obtain the default bootstrap template using the --show-template option. By default, the CDK CLI will output the template in your terminal window. You can modify the CDK CLI command to save the template to your local machine. The following is an example: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap –show-template > +++<my-bootstrap-template.yaml>+++—- . Modify the bootstrap template as needed. Any changes that you make should adhere to the bootstrapping template contract. For more information on the bootstrapping template contract, see xref:bootstrapping-contract[Follow the bootstrap contract]. + To ensure that your customizations are not accidentally overwritten later by someone running cdk bootstrap using the default template, change the default value of the BootstrapVariant template parameter. The CDK CLI will only allow overwriting the bootstrap stack with templates that have the same BootstrapVariant and an equal or higher version than the template that is currently deployed. + . Deploy your modified template using your preferred {aws} CloudFormation deployment method. The following is an example that uses the CDK CLI: + [source,none,subs=“verbatim,attributes”] —- $ cdk bootstrap –template +++<my-bootstrap-template.yaml>+++—-+++</my-bootstrap-template.yaml>++++++</my-bootstrap-template.yaml>+++

[#bootstrapping-contract] == Follow the bootstrap contract

For your CDK apps to properly deploy, the CloudFormation templates produced during synthesis must correctly specify the resources created during bootstrapping. These resources are commonly referred to as *bootstrap resources*. Bootstrapping creates resources in your {aws} environment that are used by the {aws} CDK to perform deployments and manage application assets. Synthesis produces CloudFormation templates from each CDK stack in your application. These templates don’t just define the {aws} resources that will be provisioned from your application. They also specify the bootstrap resources to use during deployment.

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 that you configure. Therefore, when you customize bootstrapping, you may need to customize synthesis. For instructions on customizing synthesis, see xref:bootstrapping-custom-synth[Customize CDK stack synthesis]. The purpose is to ensure that your synthesized CloudFormation templates are compatible with your bootstrapped environment. This compatibility is referred to as the *bootstrap contract*.

The simplest method to customize stack synthesis is by modifying the DefaultStackSynthesizer class in your Stack instance. If you require customization beyond what this class can offer, you can write your own synthesizer as a class that implements +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.IStackSynthesizer.html[IStackSynthesizer]+ (perhaps deriving from DefaultStackSynthesizer).

When you customize bootstrapping, follow the bootstrap template contract to remain compatible with DefaultStackSynthesizer. If you modify bootstrapping beyond the bootstrap template contract, you will need to write your own synthesizer.

[#bootstrapping-contract-versioning] === Versioning

The bootstrap template should contain a resource to create an Amazon EC2 Systems Manager (SSM) parameter with a well-known name and an output to reflect the template’s version:

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

Resources: CdkBootstrapVersion: Type: {aws}::SSM::Parameter Properties: Type: String Name: Fn::Sub: ‘/cdk-bootstrap/${Qualifier}/version’ Value: 4 Outputs: BootstrapVersion: Value: Fn::GetAtt: [CdkBootstrapVersion, Value] —

[#bootstrapping-contract-roles] === Roles

The DefaultStackSynthesizer requires five IAM roles for five different purposes. If you are not using the default roles, you must specify your IAM role ARNs within your DefaultStackSynthesizer object. The roles are as follows:

* The *deployment role* is assumed by the CDK CLI and by {aws} CodePipeline to deploy into an environment. Its AssumeRolePolicy controls who can deploy into the environment. In the template, you can see the permissions that this role needs.
* The *lookup role* is assumed by the CDK CLI to perform context lookups in an environment. Its AssumeRolePolicy controls who can deploy into the environment. The permissions this role needs can be seen in the template.
* The *file publishing role* and the *image publishing role* are assumed by the CDK CLI and by {aws} CodeBuild projects to publish assets into an environment. They’re used to write to the Amazon S3 bucket and the Amazon ECR repository, respectively. These roles require write access to these resources.
* *The {aws} CloudFormation execution role* is passed to {aws} CloudFormation to perform the actual deployment. Its permissions are the permissions that the deployment will execute under. The permissions are passed to the stack as a parameter that lists managed policy ARNs.

[#bootstrapping-contract-outputs] === Outputs

The CDK CLI requires that the following CloudFormation outputs exist on the bootstrap stack:

* BucketName – The name of the file asset bucket.
* BucketDomainName – The file asset bucket in domain name format.
* BootstrapVersion – The current version of the bootstrap stack.

[#bootstrapping-contract-history] === Template history

The bootstrap template is versioned and evolves over time with the {aws} CDK itself. If you provide your own bootstrap template, keep it up to date with the canonical default template. You want to make sure that your template continues to work with all CDK features. For more information, see xref:bootstrap-template-history[Bootstrap template version history].

:doctype: book

include::attributes.txt[]

// Attributes [.topic] [#bootstrapping-env] = Bootstrap your environment for use with the {aws} CDK :info\_titleabbrev: Bootstrap your environment :keywords: {aws} CDK, {aws} Cloud Development Kit ({aws} CDK), {aws} account, {aws} Region, Bootstrapping, Bootstrap, Environment

== [abstract]

## Bootstrap your {aws} environment before using the {aws} Cloud Development Kit ({aws} CDK) to deploy CDK stacks into your environment.

// Content start

Bootstrap your {aws} environment to prepare it for {aws} Cloud Development Kit ({aws} CDK) stack deployments.

* For an introduction to environments, see xref:environments[Environments for the {aws} CDK].
* For an introduction to bootstrapping, see xref:bootstrapping[{aws} CDK bootstrapping].

[#bootstrapping-howto] == How to bootstrap your environment

You can use the {aws} CDK Command Line Interface ({aws} CDK CLI) or your preferred {aws} CloudFormation deployment tool to bootstrap your environment.

[#bootstrapping-howto-cli] *Use the CDK CLI*:: + You can use the CDK CLI cdk bootstrap command to bootstrap your environment. This is the method that we recommend if you don’t require significant modifications to bootstrapping. + *Bootstrap from any working directory*::: + To bootstrap from any working directory, provide the environment to bootstrap as a command line argument. The following is an example: + [source,bash,subs=“verbatim,attributes”] — $ cdk bootstrap <aws://123456789012/us-east-1> — + [TIP] – If you don’t have your {aws} account number, you can get it from the {aws} Management Console. You can also use the following {aws} CLI command to display your default account information, including your account number:

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

## $ aws sts get-caller-identity

If you have named profiles in your {aws} config and credentials files, use the --profile option to retrieve account information for a specific profile. The following is an example:

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

$ aws sts get-caller-identity –profile ++++++—-++++++

To display the default Region, use the aws configure get command:

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

$ aws configure get region $ aws configure get region –profile ++++++—- – +++++++

When providing an argument, the aws:// prefix is optional. The following is valid: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap <123456789012/us-east-1> — + To bootstrap multiple environments at the same time, provide multiple arguments: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap <aws://123456789012/us-east-1> <aws://123456789012/us-east-2> —

*Bootstrap from the parent directory of a CDK project*::: + You can run cdk bootstrap from the parent directory of a CDK project containing a cdk.json file. If you don't provide an environment as an argument, the CDK CLI will obtain environment information from default sources, such as yourconfigandcredentialsfiles or any environment information specified for your CDK stack. + When you bootstrap from the parent directory of a CDK project, environments provided from command line arguments take precedence over other sources. + To bootstrap an environment that is specified in yourconfigandcredentialsfiles, use the–profile` option: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap –profile ++++++—- +++++++

For more information on the cdk bootstrap command and supported options, see xref:ref-cli-cmd-bootstrap[cdk bootstrap](#cdk-bootstrap).

[#bootstrapping-howto-cfn] *Use any {aws} CloudFormation tool*:: + You can copy the https://github.com/aws/aws-cdk-cli/blob/main/packages/aws-cdk/lib/api/bootstrap/bootstrap-template.yaml[bootstrap template] from the *aws-cdk-cli GitHub repository* or obtain the template with the cdk bootstrap --show-template command. Then, use any {aws} CloudFormation tool to deploy the template into your environment. + With this method, you can use {aws} CloudFormation StackSets or {aws} Control Tower. You can also use the {aws} CloudFormation console or the {aws} Command Line Interface ({aws} CLI). You can make modifications to your template before you deploy it. This method may be more flexible and suitable for large-scale deployments. + The following is an example of using the --show-template option to retrieve and save the bootstrap template to your local machine: + ==== [role=“tablist”] macOS/Linux:: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap –show-template > bootstrap-template.yaml —

Windows:: On Windows, PowerShell must be used to preserve the encoding of the template. + [source,none,subs=“verbatim,attributes”] — powershell “cdk bootstrap –show-template | Out-File -encoding utf8 bootstrap-template.yaml” — ==== +

= [NOTE]

If CDK notices are appearing in your {aws} CloudFormation template output, provide the --no-notices option with your command.

==== + To deploy this template using the CDK CLI, you can run the following: + [source,none,subs=“verbatim,attributes”] — $ cdk bootstrap –template +++<bootstrap-template.yaml>+++—- + The following is an example of using the {aws} CLI to deploy the template: + ==== [role=“tablist”] macOS/Linux:: + [source,none,subs=“verbatim,attributes”] —- aws cloudformation create-stack  –stack-name CDKToolkit  –template-body file://bootstrap-template.yaml  –capabilities CAPABILITY\_NAMED\_IAM  –region ++++++—-+++++++++</bootstrap-template.yaml>+++

Windows:: + [source,none,subs=“verbatim,attributes”] — aws cloudformation create-stack {caret} –stack-name CDKToolkit {caret} –template-body file://bootstrap-template.yaml {caret} –capabilities CAPABILITY\_NAMED\_IAM {caret} –region ++++++—- ==== +++++++

For information on using CloudFormation StackSets to bootstrap multiple environments, see https://aws.amazon.com/blogs/mt/bootstrapping-multiple-aws-accounts-for-aws-cdk-using-cloudformation-stacksets/[Bootstrapping multiple {aws} accounts for {aws} CDK using CloudFormation StackSets] in the *{aws} Cloud Operations & Migrations Blog*.

[#bootstrapping-env-when] == When to bootstrap your environment

You must bootstrap each {aws} environment before you deploy into the environment. We recommend that you proactively bootstrap each environment that you plan to use. You can do this before you plan on actually deploying CDK apps into the environment. By proactively bootstrapping your environments, you prevent potential future issues such as Amazon S3 bucket name conflicts or deploying CDK apps into environments that haven’t been bootstrapped.

It’s okay to bootstrap an environment more than once. If an environment has already been bootstrapped, the bootstrap stack will be upgraded if necessary. Otherwise, nothing will happen.

If you attempt to deploy a CDK stack into an environment that hasn`’t been bootstrapped, you will see an error like the following:

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

$ cdk deploy

✨ Synthesis time: 2.02s

== ❌ Deployment failed: Error: BootstrapExampleStack: 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)

[#bootstrapping-env-when-update] *Update your bootstrap stack*:: + Periodically, the CDK team will update the bootstrap template to a new version. When this happens, we recommend that you update your bootstrap stack. If you haven`’t customized the bootstrapping process, you can update your bootstrap stack by following the same steps that you took to originally bootstrap your environment. For more information, see xref:bootstrap-template-history[Bootstrap template version history].

[#bootstrapping-env-default] == Default resources created during bootstrapping

[#bootstrapping-env-roles] *IAM roles created during bootstrapping*:: + By default, bootstrapping provisions the following {aws} Identity and Access Management (IAM) roles in your environment: + –

* CloudFormationExecutionRole
* DeploymentActionRole
* FilePublishingRole
* ImagePublishingRole
* {blank}
* == LookupRole
* [#bootstrapping-env-roles-cfn] CloudFormationExecutionRole:::
* This IAM role is a CloudFormation service role that grants CloudFormation permission to perform stack deployments on your behalf. This role gives CloudFormation permission to perform {aws} API calls in your account, including deploying stacks.
* By using a service role, the permissions provisioned for the service role determine what actions can be performed on your CloudFormation resources. Without this service role, the security credentials you provide with the CDK CLI would determine what CloudFormation is allowed to do.
* [#bootstrapping-env-roles-deploy] DeploymentActionRole:::
* This IAM role grants permission to perform deployments into your environment. It is assumed by the CDK CLI during deployments.
* By using a role for deployments, you can perform cross-account deployments since the role can be assumed by {aws} identities in a different account.
* [#bootstrapping-env-roles-s3] FilePublishingRole:::
* This IAM role grants permission to perform actions against the bootstrapped Amazon Simple Storage Service (Amazon S3) bucket, including uploading and deleting assets. It is assumed by the CDK CLI during deployments.
* [#bootstrapping-env-roles-ecr] ImagePublishingRole:::
* This IAM role grants permission to perform actions against the bootstrapped Amazon Elastic Container Registry (Amazon ECR) repository. It is assumed by the CDK CLI during deployments.
* [#bootstrapping-env-roles-lookup] LookupRole:::
* This IAM role grants readOnly permission to look up xref:context[context values] from the {aws} environment. It is assumed by the CDK CLI when performing tasks such as template synthesis and deployments.

[#bootstrapping-env-default-id] *Resource IDs created during bootstrapping*:: + When you deploy the default bootstrap template, physical IDs for bootstrap resources are created using the following structure: cdk-<qualifier>-<description>-<account-ID>-<Region>. + –

* *Qualifier* – A nine character unique string value of hnb659fds. The actual value has no significance.
* *Description* – A short description of the resource. For example, container-assets.
* *Account ID* – The {aws} account ID of the environment.
* {blank}
* == *Region* – The {aws} Region of the environment.
* The following is an example physical ID of the Amazon S3 staging bucket created during bootstrapping: cdk-hnb659fds-assets-012345678910-us-west-1.

[#bootstrapping-env-permissions] == Permissions to use when bootstrapping your environment

When bootstrapping an {aws} environment, the IAM identity performing the bootstrapping must have at least the following permissions:

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

{ “Version”: “2012-10-17”, “Statement”: [ { “Effect”: “Allow”, “Action”: [ “cloudformation:**”, ”ecr:**”, “ssm:**”, ”s3:**”, “iam:\_\_” ], “Resource”: “\_\_” } ] } —

Over time, the bootstrap stack, including the resources that are created and permissions they require, may change. With future changes, you may need to modify the permissions required to bootstrap an environment.

[#bootstrapping-env-customize] == Customize bootstrapping

If the default bootstrap template doesn`’t suit your needs, you can customize the bootstrapping of resources into your environment in the following ways:

* Use command line options with the cdk bootstrap command – This method is best for making small, specific changes that are supported through command line options.
* Modify the default bootstrap template and deploy it – This method is best for making complex changes or if you want complete control over the configuration of resources provisioned during bootstrapping.

For more information on customizing bootstrapping, see xref:bootstrapping-customizing[Customize {aws} CDK bootstrapping].

[#bootstrapping-env-pipelines] == Bootstrapping with CDK Pipelines

If you are using CDK Pipelines to deploy into another account’s environment, and you receive a message like the following:

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

## Policy contains a statement with one or more invalid principals

This error message means that the appropriate IAM roles do not exist in the other environment. The most likely cause is that the environment has not been bootstrapped. Bootstrap the environment and try again.

[#bootstrapping-env-pipelines-protect] *Protecting your bootstrap stack from deletion*:: + If a bootstrap stack is deleted, the {aws} resources that were originally provisioned in the environment to support CDK deployments will also be deleted. This will cause the pipeline to stop working. If this happens, there is no general solution for recovery. + After your environment is bootstrapped, do not delete and recreate the environment’s bootstrap stack. Instead, try to update the bootstrap stack to a new version by running the cdk bootstrap command again. + To protect against accidental deletion of your bootstrap stack, we recommend that you provide the --termination-protection option with the cdk bootstrap command to enable termination protection. You can enable termination protection on new or existing bootstrap stacks. For instructions on enabling termination protection, see xref:bootstrapping-customizing-cli-protection[Enable termination protection for the bootstrap stack].

[#bootstrap-template-history] == Bootstrap template version history

The bootstrap template is versioned and evolves over time with the {aws} CDK itself. If you provide your own bootstrap template, keep it up to date with the canonical default template. You want to make sure that your template continues to work with all CDK features.

= [NOTE]

Earlier versions of the bootstrap template created an {aws} KMS key in each bootstrapped environment by default. To avoid charges for the KMS key, re-bootstrap these environments using --no-bootstrap-customer-key. The current default is no KMS key, which helps avoid these charges.

====

This section contains a list of the changes made in each version.

[cols=“1,1,1”, options=“header”] |=== | Template version | {aws} CDK version | Changes

|=== | *1* | 1.40.0 | Initial version of template with Bucket, Key, Repository, and Roles. |===

|=== | *2* | 1.45.0 | Split asset publishing role into separate file and image publishing roles. |===

|=== | *3* | 1.46.0 | Add FileAssetKeyArn export to be able to add decrypt permissions to asset consumers. |===

|=== | *4* | 1.61.0 | {aws} KMS permissions are now implicit via Amazon S3 and no longer require FileAssetKeyArn. Add CdkBootstrapVersion SSM parameter so the bootstrap stack version can be verified without knowing the stack name. |===

|=== | *5* | 1.87.0 | Deployment role can read SSM parameter. |===

|=== | *6* | 1.108.0 | Add lookup role separate from deployment role. |===

|=== | *6* | 1.109.0 | Attach aws-cdk:bootstrap-role tag to deployment, file publishing, and image publishing roles. |===

|=== | *7* | 1.110.0 | Deployment role can no longer read Buckets in the target account directly. (However, this role is effectively an administrator, and could always use its {aws} CloudFormation permissions to make the bucket readable anyway). |===

|=== | *8* | 1.114.0 | The lookup role has full read-only permissions to the target environment, and has a aws-cdk:bootstrap-role tag as well. |===

|=== | *9* | 2.1.0 | Fixes Amazon S3 asset uploads from being rejected by commonly referenced encryption SCP. |===

|=== | *10* | 2.4.0 | Amazon ECR ScanOnPush is now enabled by default. |===

|=== | *11* | 2.18.0 | Adds policy allowing Lambda to pull from Amazon ECR repos so it survives re-bootstrapping. |===

|=== | *12* | 2.20.0 | Adds support for experimental cdk import. |===

|=== | *13* | 2.25.0 | Makes container images in bootstrap-created Amazon ECR repositories immutable. |===

|=== | *14* | 2.34.0 | Turns off Amazon ECR image scanning at the repository level by default to allow bootstrapping Regions that do not support image scanning. |===

|=== | *15* | 2.60.0 | KMS keys cannot be tagged. |===

|=== | *16* | 2.69.0 | Addresses Security Hub finding link:https://docs.aws.amazon.com/securityhub/latest/userguide/kms-controls.html#kms-2[KMS.2]. |===

|=== | *17* | 2.72.0 | Addresses Security Hub finding link:https://docs.aws.amazon.com/securityhub/latest/userguide/ecr-controls.html#ecr-3[ECR.3]. |===

|=== | *18* | 2.80.0 | Reverted changes made for version 16 as they don’t work in all partitions and are are not recommended. |===

|=== | *19* | 2.106.1 | Reverted changes made to version 18 where AccessControl property was removed from the template. (https://github.com/aws/aws-cdk/issues/27964[#27964]) |===

|=== | *20* | 2.119.0 | Add ssm:GetParameters action to the {aws} CloudFormation deploy IAM role. For more information, see link:https://github.com/aws/aws-cdk/pull/28336/files#diff-4fdac38426c4747aa17d515b01af4994d3d2f12c34f7b6655f24328259beb7bf[#28336]. |===

|=== | *21* | 2.149.0 | Add condition to the file publishing role. |===

|=== | *22* | 2.160.0 | Add sts:TagSession permissions to the trust policy of bootstrap IAM roles. |===

|=== | *23* | 2.161.0 | Add cloudformation:RollbackStack and cloudformation:ContinueUpdateRollback permissions to the trust policy of the deploy IAM role. This provides permissions for the cdk rollback command. |===

|=== | *24* | 2.165.0 | Change the duration of days that noncurrent objects in the bootstrap bucket will be retained, from 365 to 30 days. Since the new cdk gc command introduces the ability to delete objects in the bootstrap bucket, this new behavior ensures that deleted objects remain in the bootstrap bucket for 30 days instead of 365 days. For more information on this change, see aws-cdk PR https://github.com/aws/aws-cdk/pull/31949[#31949]. |===

|=== | *25* | 2.165.0 | Add support to the bootstrap bucket for the removal of incomplete multipart uploads. Incomplete multipart uploads will be deleted after 1 day. For more information on this change, see aws-cdk PR https://github.com/aws/aws-cdk/pull/31956[#31956]. |===

|=== | *26* | 2.1002.0 | Add two deletion-related policies (UpdateReplacePolicy and DeletionPolicy to the FileAssetsBucketEncryptionKey) resource. These policies ensure that the old {aws} KMS key resource will be properly deleted when the bootstrap stack is updated or deleted. For more information on this change, see aws-cdk-cli PR https://github.com/aws/aws-cdk-cli/pull/100[#100]. |===

|=== | *27* | 2.1003.0 | Add new Amazon ECR resource policy to grant Amazon EMR Serverless specific permissions for retrieving container images. For more information on this change, see aws-cdk-cli PR https://github.com/aws/aws-cdk-cli/pull/112[#112]. |===

[#bootstrapping-template] == Upgrade from legacy to modern bootstrap template

The {aws} CDK v1 supported two bootstrapping templates, legacy and modern. CDK v2 supports only the modern template. For reference, here are the high-level differences between these two templates.

[cols=“1h,1,1”, options=“header”] |=== | Feature | Legacy (v1 only) | Modern (v1 and v2)

|=== | Cross-account deployments | Not allowed | Allowed |===

|=== | {aws} CloudFormation Permissions | Deploys using current user’s permissions (determined by {aws} profile, environment variables, etc.) | Deploys using the permissions specified when the bootstrap stack was provisioned (for example, by using --trust) |===

|=== | Versioning | Only one version of bootstrap stack is available | Bootstrap stack is versioned; new resources can be added in future versions, and {aws} CDK apps can require a minimum version |===

|=== | Resources\* | Amazon S3 bucket | a |===

* Amazon S3 bucket
* {aws} KMS key
* IAM roles
* Amazon ECR repository
* SSM parameter for versioning

|=== | {aws} KMS key | IAM roles | Amazon ECR repository |===

[cols=2\*] |=== | // | SSM parameter for versioning |===

|=== | Resource naming | Automatically generated | Deterministic |===

|=== | Bucket encryption | Default key | {aws} managed key by default. You can customize to use a customer managed key. |===

* *We will add additional resources to the bootstrap template as needed.*

An environment that was bootstrapped using the legacy template must be upgraded to use the modern template for CDK v2 by re-bootstrapping. Re-deploy all {aws} CDK applications in the environment at least once before deleting the legacy bucket.

[#bootstrapping-securityhub] == Address Security Hub Findings

If you are using {aws} Security Hub, you may see findings reported on some of the resources created by the {aws} CDK bootstrapping process. Security Hub findings help you find resource configurations you should double-check for accuracy and safety. We have reviewed these specific resource configurations with {aws} Security and are confident they do not constitute a security problem.

[#bootstrapping-securityhub-kms2] *[KMS.2] IAM principals should not have IAM inline policies that allow decryption actions on all KMS keys*:: + The *deploy role* (DeploymentActionRole) grants permission to read encrypted data, which is necessary for cross-account deployments with CDK Pipelines. Policies in this role do not grant permission to all data. It only grants permission to read encrypted data from Amazon S3 and {aws} KMS, and only when those resources allow it through their bucket or key policy. + The following is a snippet of these two statements in the *deploy role* from the bootstrap template: + [source,yaml,subs=“verbatim,attributes”] — DeploymentActionRole: Type: {aws}::IAM::Role Properties: … Policies: - PolicyDocument: Statement: … - Sid: PipelineCrossAccountArtifactsBucket Effect: Allow Action: - s3:GetObject *- s3:GetBucket* - s3:List *- s3:Abort* - s3:DeleteObject *- s3:PutObject* Resource: “*” Condition: StringNotEquals: s3:ResourceAccount: Ref: {aws}::AccountId - Sid: PipelineCrossAccountArtifactsKey Effect: Allow Action: - kms:Decrypt - kms:DescribeKey - kms:Encrypt - kms:ReEncrypt* - kms:GenerateDataKey\* Resource: “\_” Condition: StringEquals: kms:ViaService: Fn::Sub: s3.${{aws}::Region}.amazonaws.com … — + [#bootstrapping-securityhub-kms2-why] \*Why does Security Hub flag this?\_::: + The policies contain a Resource: \\* combined with a Condition clause. Security Hub flags the \* wildcard. This wildcard is used because at the time the account is bootstrapped, the {aws} KMS key created by CDK Pipelines for the CodePipeline artifact bucket does not exist yet, and therefore, can't be referenced on the bootstrap template by ARN. In addition, Security Hub does not consider theConditionclause when raising this flag. ThisConditionrestrictsResource: *+ to requests made from the same {aws} account of the {aws} KMS key. These requests must come from Amazon S3 in the same {aws} Region as the {aws} KMS key. + [#bootstrapping-securityhub-kms2-decide] \*Do I need to fix this finding?\*::: + As long as you have not modified the {aws} KMS key on your bootstrap template to be overly permissive, the \_deploy role\_ does not allow more access than it needs. Therefore, it is not necessary to fix this finding. + [#bootstrapping-securityhub-kms2-fix] \*What if I want to fix this finding?\*::: + How you fix this finding depends on whether or not you will be using CDK Pipelines for cross-account deployments. + \*To fix the Security Hub finding and use CDK Pipelines for cross-account deployments\*:::: + . If you have not done so, deploy the CDK bootstrap stack using the +cdk bootstrapcommand. . If you have not done so, create and deploy your CDK [.noloc]```Pipeline```+. For instructions, see xref:cdk-pipeline[Continuous integration and delivery (CI/CD) using CDK Pipelines]. . Obtain the {aws} KMS key ARN of the CodePipeline artifact bucket. This resource is created during pipeline creation. . Obtain a copy of the CDK bootstrap template to modify it. The following is an example, using the {aws} CDK CLI: + [source,bash,subs="verbatim,attributes"] ---- $ cdk bootstrap --show-template > bootstrap-template.yaml ---- . Modify the template by replacing +Resource:* of thePipelineCrossAccountArtifactsKey` statement with your ARN value. . Deploy the template to update your bootstrap stack. The following is an example, using the CDK CLI: + [source,bash,subs=“verbatim,attributes”] — $ cdk bootstrap aws://++++++/++++++–template bootstrap-template.yaml —- +++++++++++++

*To fix the Security Hub finding if you’re not using CDK Pipelines for cross-account deployments*:::: + . Obtain a copy of the CDK bootstrap template to modify it. The following is an example, using the CDK CLI: + [source,bash,subs=“verbatim,attributes”] — $ cdk bootstrap –show-template > bootstrap-template.yaml — . Delete the PipelineCrossAccountArtifactsBucket and PipelineCrossAccountArtifactsKey statements from the template. . Deploy the template to update your bootstrap stack. The following is an example, using the CDK CLI: + [source,bash,subs=“verbatim,attributes”] — $ cdk bootstrap aws://++++++/++++++–template bootstrap-template.yaml —-++++++++++++

[#bootstrapping-env-considerations] == Considerations

Since bootstrapping provisions resources in your environment, you may incur {aws} charges when those resources are used with the {aws} CDK.

include::bootstrapping-customizing.adoc[leveloffset=+1]

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

include::bootstrapping-troubleshoot.adoc[leveloffset=+1]

include::attributes.txt[]

// Attributes

[.topic] [#bootstrapping-troubleshoot] = Troubleshoot {aws} CDK bootstrapping issues :info\_titleabbrev: Troubleshoot bootstrapping :keywords: {aws} CDK, Bootstrapping, Troubleshoot

== [abstract]

## Troubleshoot common issues when bootstrapping your environment with the {aws} Cloud Development Kit ({aws} CDK).

// Content start

Troubleshoot common issues when bootstrapping your environment with the {aws} Cloud Development Kit ({aws} CDK).

* For an introduction to bootstrapping, see xref:bootstrapping[{aws} CDK bootstrapping].
* For instructions on bootstrapping, see xref:bootstrapping-env[Bootstrap your environment for use with the {aws} CDK].

[#bootstrapping-troubleshoot-s3-bucket-name] == When bootstrapping using the default template, you get a ‘CREATE\_FAILED’ error for the Amazon S3 bucket

When bootstrapping using the {aws} CDK Command Line Interface (CDK CLI) cdk bootstrap command with the default bootstrap template, you receive the following error:

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

CREATE\_FAILED | {aws}::S3::Bucket | ++++++already exists —-++++++

Before troubleshooting, ensure that you are using the latest version of the CDK CLI.

* To check your version, run cdk --version.
* To install the latest version, run npm install -g aws-cdk.

After installing the latest version, try bootstrapping your environment again. If you still receive the same error, continue with troubleshooting.

[#bootstrapping-troubleshoot-s3-bucket-name-causes] === Common causes

When you bootstrap your environment, the {aws} CDK generates physical IDs for your bootstrap resources. For more information, see xref:bootstrapping-env-default-id[Resource IDs created during bootstrapping].

Unlike the other bootstrap resources, Amazon S3 bucket names are global. This means that each bucket name must be unique across all {aws} accounts in all {aws} Regions within a partition. For more information, see https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingBucket.html[Buckets overview] in the *Amazon S3 User Guide*. Therefore, the most common cause of this error is that the physical ID generated as your bucket name already exists somewhere within the partition. This could be within your account or another account.

The following is an example bucket name: cdk-hnb659fds-assets-012345678910-us-west-1. While unlikely, due to the qualifier and account ID being a part of the name, it is possible that this name for an Amazon S3 bucket is used by another {aws} account. Since bucket names are globally scoped, it can`’t be used by you if its used by a different account in the same partition. Most likely, a bucket with the same name exists somewhere in your account. This could be in the Region you are attempting to bootstrap, or another Region.

Generally, the resolution is to locate this bucket in your account and determine what to do with it, or customize bootstrapping to create bootstrap resources of a different name.

[#bootstrapping-troubleshoot-s3-bucket-name-resolution] === Resolution

First, determine if a bucket with the same name exists within your {aws} account. Using an {aws} identity with valid permissions to lookup Amazon S3 buckets in your account, you can do this in the following ways:

* Use the {aws} Command Line Interface ({aws} CLI) aws s3 ls command to view a list of all your buckets.
* Look up bucket names for each Region in your account using the https://console.aws.amazon.com/s3[Amazon S3 console].

If a bucket with the same name exists, determine if it’s being used. If it’s not being used, consider deleting the bucket and attempting to bootstrap your environment again.

If a bucket with the same name exists and you don’t want to delete it, determine if it’s already associated with a bootstrap stack in your account. You may have to check multiple Regions. The Region in the Amazon S3 bucket name doesn’t necessarily mean that the bucket is in that Region. To check if it’s associated with the CDKToolkit bootstrap stack, you can do either of the following for each Region:

* Use the {aws} CLI aws cloudformation describe-stack-resources --stack-name <CDKToolkit> --region <Region> command to view the resources in your bootstrap stack and check if the bucket is listed.
* In the https://console.aws.amazon.com/cloudformation[{aws} CloudFormation console], locate the CDKToolkit stack. Then, on the *Resources* tab, check if the bucket exists.

If the bucket is associated with a bootstrap stack, determine if the bootstrap stack is in the same Region that you are attempting to bootstrap. If it is, your environment is already bootstrapped and you should be able to start using the CDK to deploy applications into your environment. If the Amazon S3 bucket is associated with a bootstrap stack in a different Region, you`’ll need to determine what to do. Possible resolutions include renaming the existing Amazon S3 bucket, deleting the current Amazon S3 bucket if its not being used, or using a new name for the Amazon S3 bucket you are attempting to create.

If you are unable to locate an Amazon S3 bucket with the same name in your account, it may exist in a different account. To resolve this, you`’ll need to customize bootstrapping to create new names for all of your bootstrap resources or for just your Amazon S3 bucket. To create new names for all bootstrap resources, you can modify the qualifier. To create a new name for only your Amazon S3 bucket, you can provide a new bucket name.

To customize bootstrapping, you can use options with the CDK CLI cdk bootstrap command or by modifying the bootstrap template. For instructions, see xref:bootstrapping-customizing[Customize {aws} CDK bootstrapping].

If you customize bootstrapping, you will need to apply the same changes to synthesis before you can properly deploy an application. For instructions, see xref:bootstrapping-custom-synth[Customize CDK stack synthesis].

For example, you can provide a new qualifier with cdk bootstrap:

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

$ cdk bootstrap –qualifier ++++++—-++++++

The following is an example Amazon S3 bucket name that will be created with this modification: cdk-abcde0123-assets-012345678910-us-west-1. All bootstrap resources created during bootstrapping will use this qualifier.

When developing your CDK app, you must specify your custom qualifier in your synthesizer. This helps the CDK with identifying your bootstrap resources during synthesis and deployment. The following is an example of customizing the default synthesizer for your stack instance:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyStack(this, ‘MyStack’, { synthesizer: new DefaultStackSynthesizer({ qualifier: ‘abcde0123’, }), }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyStack(this, ‘MyStack’, { synthesizer: new DefaultStackSynthesizer({ qualifier: ‘abcde0123’, }), }) —

Python:: + [source,python,subs=“verbatim,attributes”] — MyStack(self, “MyStack”, synthesizer=DefaultStackSynthesizer( qualifier=“abcde0123” )) —

Java:: + [source,java,subs=“verbatim,attributes”] — new MyStack(app, “MyStack”, StackProps.builder() .synthesizer(DefaultStackSynthesizer.Builder.create() .qualifier(“abcde0123”) .build()) .build(); ) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new MyStack(app, “MyStack”, new StackProps { Synthesizer = new DefaultStackSynthesizer(new DefaultStackSynthesizerProps { Qualifier = “abcde0123” }) }); —

Go:: + [source,go,subs=“verbatim,attributes”] — func NewMyStack(scope constructs.Construct, id string, props \*MyStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. synth := awscdk.NewDefaultStackSynthesizer(&awscdk.DefaultStackSynthesizerProps{ Qualifier: jsii.String(“abcde0123”), })

stack.SetSynthesizer(synth)

return stack } —- ….

You can also specify the new qualifier in the cdk.json file of your CDK project:

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

{ “app”: “…”, “context”: { “@aws-cdk/core:bootstrapQualifier”: “abcde0123” } } —

To modify only the Amazon S3 bucket name, you can use the xref:ref-cli-cmd-bootstrap-options-bootstrap-bucket-name[--bootstrap-bucket-name] option. The following is an example:

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

$ cdk bootstrap –bootstrap-bucket-name ‘++++++’ —- ====++++++

When developing your CDK app, you must specify your new bucket name in your synthesizer. The following is an example of customizing the default synthesizer for your stack instance:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyStack(this, ‘MyStack’, { synthesizer: new DefaultStackSynthesizer({ fileAssetsBucketName: ‘my-new-bucket-name’, }), }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyStack(this, ‘MyStack’, { synthesizer: new DefaultStackSynthesizer({ fileAssetsBucketName: ‘my-new-bucket-name’, }), }) —

Python:: + [source,python,subs=“verbatim,attributes”] — MyStack(self, “MyStack”, synthesizer=DefaultStackSynthesizer( file\_assets\_bucket\_name=‘my-new-bucket-name’ )) —

Java:: + [source,java,subs=“verbatim,attributes”] — new MyStack(app, “MyStack”, StackProps.builder() .synthesizer(DefaultStackSynthesizer.Builder.create() .fileAssetsBucketName(“my-new-bucket-name”) .build()) .build(); ) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — new MyStack(app, “MyStack”, new StackProps { Synthesizer = new DefaultStackSynthesizer(new DefaultStackSynthesizerProps { FileAssetsBucketName = “my-new-bucket-name” }) }); —

Go:: + [source,go,subs=“verbatim,attributes”] — func NewMyStack(scope constructs.Construct, id string, props \*MyStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. synth := awscdk.NewDefaultStackSynthesizer(&awscdk.DefaultStackSynthesizerProps{ FileAssetsBucketName: jsii.String(“my-new-bucket-name”), })

stack.SetSynthesizer(synth)

return stack } —- ==== ….

[#bootstrapping-troubleshoot-s3-bucket-name-prevention] === Prevention

We recommend that you proactively bootstrap each {aws} environment that you plan to use. For more information, see xref:bootstrapping-env-when[When to bootstrap your environment]. Specifically for the Amazon S3 bucket naming issue, this will create Amazon S3 buckets in each {aws} environment and prevent others from using your Amazon S3 bucket name.

include::attributes.txt[]

// Attributes [.topic] :info\_titleabbrev: Bootstrapping :keywords: {aws} CDK, {aws} Cloud Development Kit ({aws} CDK), {aws} account, {aws} Region, Bootstrapping, Bootstrap, Environment

[#bootstrapping] = {aws} CDK bootstrapping

== [abstract]

## *Bootstrapping* is the process of preparing your {aws} environment for usage with the {aws} Cloud Development Kit ({aws} CDK). Before you deploy a CDK stack into an {aws} environment, the environment must first be bootstrapped.

// Content start

*Bootstrapping* is the process of preparing your {aws} environment for usage with the {aws} Cloud Development Kit ({aws} CDK). Before you deploy a CDK stack into an {aws} environment, the environment must first be bootstrapped.

[#bootstrapping-what] == What is bootstrapping?

Bootstrapping prepares your {aws} environment by provisioning specific {aws} resources in your environment that are used by the {aws} CDK. These resources are commonly referred to as your *bootstrap resources*. They include the following:

* *Amazon Simple Storage Service (Amazon S3) bucket* – Used to store your CDK project files, such as {aws} Lambda function code and assets.
* *Amazon Elastic Container Registry (Amazon ECR) repository* – Used primarily to store [.noloc]Docker images.
* *{aws} Identity and Access Management (IAM) roles* – Configured to grant permissions needed by the {aws} CDK to perform deployments. For more information about the IAM roles created during bootstrapping, see xref:bootstrapping-env-roles[IAM roles created during bootstrapping].

[#bootstrapping-how] == How does bootstrapping work?

Resources and their configuration that are used by the CDK are defined in an {aws} CloudFormation template. This template is created and managed by the CDK team. For the latest version of this template, see https://github.com/aws/aws-cdk-cli/blob/main/packages/aws-cdk/lib/api/bootstrap/bootstrap-template.yaml[bootstrap-template.yaml] in the *aws-cdk-cli [.noloc]GitHub repository*.

To bootstrap an environment, you use the {aws} CDK Command Line Interface ({aws} CDK CLI) cdk bootstrap command. The CDK CLI retrieves the template and deploys it to {aws} CloudFormation as a stack, known as the *bootstrap stack*. By default, the stack name is CDKToolkit. By deploying this template, CloudFormation provisions the resources in your environment. After deployment, the bootstrap stack will appear in the {aws} CloudFormation console of your environment.

You can also customize bootstrapping by modifying the template or by using CDK CLI options with the cdk bootstrap command.

{aws} environments are independent. Each environment that you want to use with the {aws} CDK must first be bootstrapped.

[#bootstrapping-learn] == Learn more

For instructions on bootstrapping your environment, see xref:bootstrapping-env[Bootstrap your environment for use with the {aws} CDK].

include::attributes.txt[]

// Attributes

:https–docs-aws-amazon-com-cdk-api-v2-docs-aws-cdk-lib-aws-cloudformation-readme-html: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_cloudformation-readme.html :https–docs-aws-amazon-com-cdk-api-v2-docs-aws-cdk-lib-readme-html-stack-synthesizers: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib-readme.html#stack-synthesizers [.topic] [#deploy] = Deploy {aws} CDK applications :keywords: {aws} CDK, IaC, Infrastructure as code, {aws}, {aws} Cloud, serverless, modern applications

== [abstract]

## An {aws} Cloud Development Kit ({aws} CDK) deployment is the process of provisioning your infrastructure on {aws}.

// Content start

An {aws} Cloud Development Kit ({aws} CDK) deployment is the process of provisioning your infrastructure on {aws}.

[#deploy-how] == How {aws} CDK deployments work

The {aws} CDK utilizes the {aws} CloudFormation service to perform deployments. Before you deploy, you synthesize your CDK stacks. This creates a CloudFormation template and deployment artifacts for each CDK stack in your app. Deployments are initiated from a local development machine or from a *continuous integration and continuous delivery (CI/CD)* environment. During deployment, assets are uploaded to the bootstrapped resources and the CloudFormation template is submitted to CloudFormation to provision your {aws} resources.

For a deployment to be successful, the following is required:

* The {aws} CDK Command Line Interface ({aws} CDK CLI) must be provided with valid permissions.
* The {aws} environment must be bootstrapped.
* The {aws} CDK must know the bootstrapped resources to upload assets into.

[#deploy-prerequisites] == Prerequisites for CDK deployments

Before you can deploy an {aws} CDK application, you must complete the following:

* Configure security credentials for the CDK CLI.
* Bootstrap your {aws} environment.
* Configure an {aws} environment for each of your CDK stacks.
* Develop your CDK app.

[#deploy-prerequisites-creds] *Configure security credentials*:: + To use the CDK CLI to interact with {aws}, you must configure security credentials on your local machine. For instructions, see xref:configure-access[Configure security credentials for the {aws} CDK CLI].

[#deploy-prerequisites-bootstrap] *Bootstrap your {aws} environment*:: + A deployment is always associated with one or more {aws} xref:environments[environments]. Before you can deploy, the environment must first be xref:bootstrapping[bootstrapped]. Bootstrapping provisions resources in your environment that the CDK uses to perform and manage deployments. These resources include an Amazon Simple Storage Service (Amazon S3) bucket and Amazon Elastic Container Registry (Amazon ECR) repository to store and manage xref:assets[assets]. These resources also include {aws} Identity and Access Management (IAM) roles that are used to provide permissions during development and deployment. + We recommend that you use the {aws} CDK Command Line Interface ({aws} CDK CLI) cdk bootstrap command to bootstrap your environment. You can customize bootstrapping or manually create these resources in your environment if necessary. For instructions, see xref:bootstrapping-env[Bootstrap your environment for use with the {aws} CDK].

[#deploy-prerequisites-env] *Configure {aws} environments*:: + Each CDK stack must be associated with an environment to determine where the stack is deployed to. For instructions, see xref:configure-env[Configure environments to use with the {aws} CDK].

[#deploy-prerequisites-develop] *Develop your CDK app*:: + Within a CDK xref:projects[project], you create and develop your CDK app. Within your app, you create one or more CDK xref:stacks[stacks]. Within your stacks, you import and use xref:constructs[constructs] from the {aws} Construct Library to define your infrastructure. Before you can deploy, your CDK app must contain at least one stack.

[#deploy-how-synth] == CDK app synthesis

To perform synthesis, we recommend that you use the CDK CLI cdk synth command. The cdk deploy command will also perform synthesis before initiating deployment. However, by using cdk synth, you can validate your CDK app and catch errors before initiating deployment.

Synthesis behavior is determined by the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib-readme.html#stack-synthesizers[stack synthesizer] that you configure for your CDK stack. If you don't configure a synthesizer,https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.DefaultStackSynthesizer.html[DefaultStackSynthesizer]` will be used. You can also configure and customize synthesis to meet your needs. For instructions, see xref:configure-synth[Configure and perform CDK stack synthesis].

For your synthesized CloudFormation template to deploy successfully into your environment, it must be compatible with how your environment was bootstrapped. For example, your CloudFormation template must specify the correct Amazon S3 bucket to deploy assets into. If you use the default method of bootstrapping your environment, the default stack synthesizer will work. If you customize CDK behavior, such as customizing bootstrapping or synthesis, CDK deployment behavior may vary.

[#deploy-how-synth-app] *The app lifecycle*:: + When you perform synthesis, your CDK app is run through the following phases, known as the *app lifecycle*: + *Construction (or Initialization)*::: + Your code instantiates all of the defined constructs and then links them together. In this stage, all of the constructs (app, stacks, and their child constructs) are instantiated and the constructor chain is run. Most of your app code is run in this stage. + *Preparation*::: All constructs that have implemented the prepare method participate in a final round of modifications, to set up their final state. The preparation phase happens automatically. As a user, you don’t see any feedback from this phase. It’s rare to need to use the “prepare” hook, and generally not recommended. Be very careful when mutating the construct tree during this phase, because the order of operations could impact behavior. + During this phase, once the construct tree has been built, any xref:aspects[aspects] that you have configured are applied as well. + *Validation*::: + All constructs that have implemented the validate method can validate themselves to ensure that they’re in a state that will correctly deploy. You will get notified of any validation failures that happen during this phase. Generally, we recommend performing validation as soon as possible (usually as soon as you get some input) and throwing exceptions as early as possible. Performing validation early improves reliability as stack traces will be more accurate, and ensures that your code can continue to execute safely. + *Synthesis*::: + This is the final stage of running your CDK app. It’s triggered by a call to app.synth(), and it traverses the construct tree and invokes the synthesize method on all constructs. Constructs that implement synthesize can participate in synthesis and produce deployment artifacts to the resulting cloud assembly. These artifacts include CloudFormation templates, {aws} Lambda application bundles, file and Docker image assets, and other deployment artifacts. In most cases, you won’t need to implement the synthesize method.

[#deploy-how-synth-run] *Running your app*:: + The CDK CLI needs to know how to run your CDK app. If you created the project from a template using the cdk init command, your app’s cdk.json file includes an app key. This key specifies the necessary command for the language that the app is written in. If your language requires compilation, the command line performs this step before running the app automatically. + ==== [role=“tablist”] TypeScript:: + [source,json,subs=“verbatim,attributes”] — { “app”: “npx ts-node –prefer-ts-exts bin/my-app.ts” } —

JavaScript:: + [source,json,subs=“verbatim,attributes”] — { “app”: “node bin/my-app.js” } —

Python:: + [source,json,subs=“verbatim,attributes”] — { “app”: “python app.py” } —

Java:: + [source,json,subs=“verbatim,attributes”] — { “app”: “mvn -e -q compile exec:java” } —

C#:: + [source,json,subs=“verbatim,attributes”] — { “app”: “dotnet run -p src/MyApp/MyApp.csproj” } —

Go:: + [source,json,subs=“verbatim,attributes”] — { “app”: “go mod download && go run my-app.go” } — ==== + If you didn’t create your project using the CDK CLI, or if you want to override the command line given in cdk.json, you can provide the xref:ref-cli-cmd-options-app[--app] option when running the cdk command.

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

$ cdk –app ‘++++++’ ++++++... —-++++++++++++

The <executable> part of the command indicates the command that should be run to execute your CDK application. Use quotation marks as shown, since such commands contain spaces. The <cdk-command> is a subcommand like synth or deploy that tells the CDK CLI what you want to do with your app. Follow this with any additional options needed for that subcommand.

The CDK CLI can also interact directly with an already-synthesized cloud assembly. To do that, pass the directory in which the cloud assembly is stored in --app. The following example lists the stacks defined in the cloud assembly stored under ./my-cloud-assembly.

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

## $ cdk –app <./my-cloud-assembly> ls

[#deploy-how-synth-assemblies] *Cloud assemblies*:: + The call to app.synth() is what tells the {aws} CDK to synthesize a cloud assembly from an app. Typically you don’t interact directly with cloud assemblies. They are files that include everything needed to deploy your app to a cloud environment. For example, it includes an {aws} CloudFormation template for each stack in your app. It also includes a copy of any file assets or Docker images that you reference in your app. + See the https://github.com/aws/aws-cdk/blob/master/design/cloud-assembly.md[cloud assembly specification] for details on how cloud assemblies are formatted. + To interact with the cloud assembly that your {aws} CDK app creates, you typically use the {aws} CDK CLI. However, any tool that can read the cloud assembly format can be used to deploy your app.

[#deploy-how-deploy] == Deploy your application

To deploy your application, we recommend that you use the CDK CLI cdk deploy command to initiate deployments or to configure automated deployments.

When you run cdk deploy, the CDK CLI initiates cdk synth to prepare for deployment. The following diagram illustrates the app lifecycle in the context of a deployment:

image::./images/app-lifecycle\_cdk-flowchart.png[Flowchart of the {aws} CDK app lifecycle.,scaledwidth=100%]

During deployment, the CDK CLI takes the cloud assembly produced by synthesis and deploys it to an {aws} environment. Assets are uploaded to Amazon S3 and Amazon ECR and the CloudFormation template is submitted to {aws} CloudFormation for deployment.

By the time the {aws} CloudFormation deployment phase starts, your CDK app has already finished running and exited. This has the following implications:

* The CDK app can’t respond to events that happen during deployment, such as a resource being created or the whole deployment finishing. To run code during the deployment phase, you must inject it into the {aws} CloudFormation template as a xref:develop-customize-custom[custom resource]. For more information about adding a custom resource to your app, see the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_cloudformation-readme.html[{aws} CloudFormation module], or the link:https://github.com/aws-samples/aws-cdk-examples/tree/master/typescript/custom-resource/[custom-resource] example. You can also configure the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.triggers-readme.html[Triggers] module to run code during deployments.
* The CDK app might have to work with values that can’t be known at the time it runs. For example, if the {aws} CDK app defines an Amazon S3 bucket with an automatically generated name, and you retrieve the bucket.bucketName (Python: bucket\_name) attribute, that value is not the name of the deployed bucket. Instead, you get a Token value. To determine whether a particular value is available, call cdk.isUnresolved(value) (Python: is\_unresolved). See xref:tokens[Tokens and the {aws} CDK] for details.

[#deploy-how-deploy-permissions] *Deployment permissions*:: + Before deployment can be performed, permissions must be established. The following diagram illustrates the permissions that are used during a default deployment, when using the default bootstrapping process and stack synthesizer: + image::./images/default-deploy-process\_cdk\_flowchart.png[Flowchart of the default {aws} CDK deployment process.,scaledwidth=100%] + *Actor initiates deployment*::: + Deployments are initiated by an *actor*, using the CDK CLI. An actor can either be a person, or a service such as {aws} CodePipeline. + If necessary, the CDK CLI runs cdk synth when you run cdk deploy. During synthesis, the {aws} identity assumes the LookupRole to perform context lookups in the {aws} environment. + *Permissions are established*::: + First, the actor’s security credentials are used to authenticate to {aws} and obtain the first IAM identity in the process. For human actors, how security credentials are configured and obtained depends on how you or your organization manages users. For more information, see xref:configure-access[Configure security credentials for the {aws} CDK CLI]. For service actors, such as CodePipeline, an IAM execution role is assumed and used. + Next, the IAM roles created in your {aws} environment during bootstrapping are used to establish permissions to perform the actions needed for deployment. For more information about these roles and what they grant permissions for, see xref:bootstrapping-env-roles[IAM roles created during bootstrapping]. This process includes the following: + –

* The {aws} identity assumes the DeploymentActionRole role and passes the CloudFormationExecutionRole role to CloudFormation, ensuring that CloudFormation assumes the role when it performs any actions in your {aws} environment. DeploymentActionRole grants permission to perform deployments into your environment and CloudFormationExecutionRole determines what actions CloudFormation can perform.
* The {aws} identity assumes the FilePublishingRole, which determines the actions that can be performed on the Amazon S3 bucket created during bootstrapping.
* The {aws} identity assumes the ImagePublishingRole, which determines the actions that can be performed on the Amazon ECR repository created during bootstrapping.
* {blank}
* == If necessary, the {aws} identity assumes the LookupRole to perform context lookups in the {aws} environment. This action may also be performed during template synthesis.
* *Deployment is performed*:::
* During deployment, the CDK CLI reads the bootstrap version parameter to confirm the bootstrap version number. {aws} CloudFormation also reads this parameter at deployment time to confirm. If permissions across the deployment workflow are valid, deployment is performed. Assets are uploaded to the bootstrapped resources and the CloudFormation template produced at synthesis is deployed using the CloudFormation service as a CloudFormation stack to provision your resources.

include::policy-validation-synthesis.adoc[leveloffset=+1]

include::create-cdk-pipeline.adoc[leveloffset=+1]

include::build-containers.adoc[leveloffset=+1]

include::deploy-troubleshoot.adoc[leveloffset=+1]

:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#cli] = {aws} CDK CLI reference

// Content start

The {aws} Cloud Development Kit ({aws} CDK) Command Line Interface ({aws} CDK CLI), also known as the CDK Toolkit, is the primary tool for interacting with your {aws} CDK app. It executes your app, interrogates the application model you defined, and produces and deploys the {aws} CloudFormation templates generated by the {aws} CDK. It also provides other features useful for creating and working with {aws} CDK projects. This topic contains information about common use cases of the CDK CLI.

The CDK CLI is installed with the Node Package Manager. In most cases, we recommend installing it globally.

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

npm install -g aws-cdk # install latest version npm install -g aws-cdk@X.YY.Z # install specific version —

= [TIP]

If you regularly work with multiple versions of the {aws} CDK, consider installing a matching version of the CDK CLI in individual CDK projects. To do this, omit -g from the npm install command. Then use npx aws-cdk to invoke it. This runs the local version if one exists, falling back to a global version if not.

====

[#cli-commands] == CDK CLI commands

All CDK CLI commands start with cdk, which is followed by a subcommand (list, synthesize, deploy, etc.). Some subcommands have a shorter version (ls, synth, etc.) that is equivalent. Options and arguments follow the subcommand in any order.

For a description of all subcommands, options, and arguments, see xref:ref-cli-cmd[{aws} CDK CLI command reference].

[#cli-options] == Specify options and their values

Command line options begin with two hyphens (--). Some frequently used options have single-letter synonyms that begin with a single hyphen (for example, --app has a synonym -a). The order of options in an CDK CLI command is not important.

All options accept a value, which must follow the option name. The value may be separated from the name by white space or by an equals sign =. The following two options are equivalent.

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

–toolkit-stack-name MyBootstrapStack –toolkit-stack-name=MyBootstrapStack —

Some options are flags (Booleans). You may specify true or false as their value. If you do not provide a value, the value is taken to be true. You may also prefix the option name with no- to imply false.

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

= sets staging flag to true

–staging –staging=true –staging true

= sets staging flag to false

–no-staging –staging=false –staging false —

A few options, namely --context, --parameters, --plugin, --tags, and --trust, may be specified more than once to specify multiple values. These are noted as having [array] type in the CDK CLI help. For example:

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

## cdk bootstrap –tags costCenter=0123 –tags responsibleParty=jdoe

[#cli-help] == Built-in help

The CDK CLI has integrated help. You can see general help about the utility and a list of the provided subcommands by issuing:

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

## cdk –help

To see help for a particular subcommand, for example deploy, specify it before the --help flag.

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

## cdk deploy –help

Issue cdk version to display the version of the CDK CLI. Provide this information when requesting support.

[#version-reporting] == Version reporting

To gain insight into how the {aws} CDK is used, the constructs used by {aws} CDK applications are collected and reported by using a resource identified as +{aws}::CDK::Metadata+. To learn more, see xref:usage-data[Configure {aws} CDK usage data reporting].

[#cli-auth] == Authentication with {aws}

There are different ways in which you can configure programmatic access to {aws} resources, depending on the environment and the {aws} access available to you.

To choose your method of authentication and configure it for the CDK CLI, see xref:configure-access[Configure security credentials for the {aws} CDK CLI].

The recommended approach for new users developing locally, who are not given a method of authentication by their employer, is to set up {aws} IAM Identity Center. This method includes installing the {aws} CLI for ease of configuration and for regularly signing in to the {aws} access portal. If you choose this method, your environment should contain the following elements after you complete the procedure for https://docs.aws.amazon.com/sdkref/latest/guide/access-sso.html[IAM Identity Center authentication] in the *{aws} SDKs and Tools Reference Guide*:

* The {aws} CLI, which you use to start an {aws} access portal session before you run your application.
* A https://docs.aws.amazon.com/sdkref/latest/guide/file-format.html[shared {aws} config file] having a [default] profile with a set of configuration values that can be referenced from the {aws} CDK. To find the location of this file, see https://docs.aws.amazon.com/sdkref/latest/guide/file-location.html[Location of the shared files] in the *{aws} SDKs and Tools Reference Guide*.
* The shared config file sets the https://docs.aws.amazon.com/sdkref/latest/guide/feature-region.html[region] setting. This sets the default {aws} Region the {aws} CDK and CDK CLI use for {aws} requests.
* The CDK CLI uses the profile’s link:https://docs.aws.amazon.com/sdkref/latest/guide/feature-sso-credentials.html#feature-sso-credentials-profile[SSO token provider configuration] to acquire credentials before sending requests to {aws}. The sso\_role\_name value, which is an IAM role connected to an IAM Identity Center permission set, should allow access to the {aws} services used in your application.
* The following sample config file shows a default profile set up with SSO token provider configuration. The profile’s sso\_session setting refers to the named link:https://docs.aws.amazon.com/sdkref/latest/guide/file-format.html#section-session[sso-session section]. The sso-session section contains settings to initiate an {aws} access portal session.

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

* [default] sso\_session = ++++++sso\_account\_id = <111122223333> sso\_role\_name = ++++++region = ++++++output = ++++++++++++++++++++++++++++++

[sso-session ++++++] sso\_region = ++++++sso\_start\_url = <https://provided-domain.awsapps.com/start> sso\_registration\_scopes = sso:account:access —-++++++++++++

[#accessportal] === Start an {aws} access portal session

Before accessing {aws} services, you need an active {aws} access portal session for the CDK CLI to use IAM Identity Center authentication to resolve credentials. Depending on your configured session lengths, your access will eventually expire and the CDK CLI will encounter an authentication error. Run the following command in the {aws} CLI to sign in to the {aws} access portal.

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

## aws sso login

If your SSO token provider configuration is using a named profile instead of the default profile, the command is aws sso login --profile <NAME>. Also specify this profile when issuing cdk commands using the --profile option or the AWS\_PROFILE environment variable.

To test if you already have an active session, run the following {aws} CLI command.

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

## aws sts get-caller-identity

The response to this command should report the IAM Identity Center account and permission set configured in the shared config file.

= [NOTE]

If you already have an active {aws} access portal session and run aws sso login, you will not be required to provide credentials.

The sign in process may prompt you to allow the {aws} CLI access to your data. Since the {aws} CLI is built on top of the SDK for Python, permission messages may contain variations of the botocore name.

====

[#cli-environment] == Specify Region and other configuration

The CDK CLI needs to know the {aws} Region that you’re deploying into and how to authenticate with {aws}. This is needed for deployment operations and to retrieve context values during synthesis. Together, your account and Region make up the *environment*.

Region may be specified using environment variables or in configuration files. These are the same variables and files used by other {aws} tools such as the {aws} CLI and the various {aws} SDKs. The CDK CLI looks for this information in the following order.

* The AWS\_DEFAULT\_REGION environment variable.
* A named profile defined in the standard {aws} config file and specified using the --profile option on cdk commands.
* The [default] section of the standard {aws} config file.

Besides specifying {aws} authentication and a Region in the [default] section, you can also add one or more [profile <NAME>] sections, where <NAME> is the name of the profile. For more information about named profiles, see https://docs.aws.amazon.com/sdkref/latest/guide/file-format.html[Shared config and credentials files] in the *{aws} SDKs and Tools Reference Guide*.

The standard {aws} config file is located at ~/.aws/config (macOS/Linux) or %USERPROFILE%\.aws\config (Windows). For details and alternate locations, see https://docs.aws.amazon.com/sdkref/latest/guide/file-location.html[Location of the shared config and credentials files] in the *{aws} SDKs and Tools Reference Guide*

The environment that you specify in your {aws} CDK app by using the stack’s env property is used during synthesis. It’s used to generate an environment-specific {aws} CloudFormation template, and during deployment, it overrides the account or Region specified by one of the preceding methods. For more information, see xref:environments[Environments for the {aws} CDK].

= [NOTE]

The {aws} CDK uses credentials from the same source files as other {aws} tools and SDKs, including the https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html[{aws} Command Line Interface]. However, the {aws} CDK might behave somewhat differently from these tools. It uses the {aws} SDK for JavaScript under the hood. For complete details on setting up credentials for the {aws} SDK for JavaScript, see https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-credentials.html[Setting credentials].

====

You may optionally use the --role-arn (or -r) option to specify the ARN of an IAM role that should be used for deployment. This role must be assumable by the {aws} account being used.

[#cli-app-command] == Specify the app command

Many features of the CDK CLI require one or more {aws} CloudFormation templates be synthesized, which in turn requires running your application. The {aws} CDK supports programs written in a variety of languages. Therefore, it uses a configuration option to specify the exact command necessary to run your app. This option can be specified in two ways.

First, and most commonly, it can be specified using the app key inside the file cdk.json. This is in the main directory of your {aws} CDK project. The CDK CLI provides an appropriate command when creating a new project with cdk init. Here is the cdk.json from a fresh TypeScript project, for instance.

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

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

The CDK CLI looks for cdk.json in the current working directory when attempting to run your app. Because of this, you might keep a shell open in your project’s main directory for issuing CDK CLI commands.

The CDK CLI also looks for the app key in ~/.cdk.json (that is, in your home directory) if it can’t find it in ./cdk.json. Adding the app command here can be useful if you usually work with CDK code in the same language.

If you are in some other directory, or to run your app using a command other than the one in cdk.json, use the --app (or -a) option to specify it.

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

## cdk –app “npx ts-node bin/hello-cdk.ts” ls

When deploying, you may also specify a directory containing synthesized cloud assemblies, such as cdk.out, as the value of --app. The specified stacks are deployed from this directory; the app is not synthesized.

[#cli-stacks] == Specify stacks

Many CDK CLI commands (for example, cdk deploy) work on stacks defined in your app. If your app contains only one stack, the CDK CLI assumes you mean that one if you don’t specify a stack explicitly.

Otherwise, you must specify the stack or stacks you want to work with. You can do this by specifying the desired stacks by ID individually on the command line. Recall that the ID is the value specified by the second argument when you instantiate the stack.

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

## cdk synth PipelineStack LambdaStack

You may also use wildcards to specify IDs that match a pattern.

* ? matches any single character
* \\* matches any number of characters (\* alone matches all stacks)
* +\*\*+ matches everything in a hierarchy

You may also use the --all option to specify all stacks.

If your app uses xref:cdk-pipeline[CDK Pipelines], the CDK CLI understands your stacks and stages as a hierarchy. Also, the --all option and the \\* wildcard only match top-level stacks. To match all the stacks, use +\\*\*+. Also use +\*\*+ to indicate all the stacks under a particular hierarchy.

When using wildcards, enclose the pattern in quotes, or escape the wildcards with \. If you don’t, your shell may try to expand the pattern to the names of files in the current directory. At best, this won’t do what you expect; at worst, you could deploy stacks you didn’t intend to. This isn’t strictly necessary on Windows because cmd.exe does not expand wildcards, but is good practice nonetheless.

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

cdk synth “*Stack” # PipelineStack, LambdaStack, etc. cdk synth ‘Stack?’ # StackA, StackB, Stack1, etc. cdk synth*  # All stacks in the app, or all top-level stacks in a CDK Pipelines app cdk synth ‘*’ # All stacks in a CDK Pipelines app cdk synth ’PipelineStack/Prod/*’ # All stacks in Prod stage in a CDK Pipelines app —

= [NOTE]

The order in which you specify the stacks is not necessarily the order in which they will be processed. The CDK CLI accounts for dependencies between stacks when deciding the order in which to process them. For example, let’s say that one stack uses a value produced by another (such as the ARN of a resource defined in the second stack). In this case, the second stack is synthesized before the first one because of this dependency. You can add dependencies between stacks manually using the stack’s +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#addwbrdependencytarget-reason[addDependency()]+ method.

====

[#cli-bootstrap] == Bootstrap your {aws} environment

Deploying stacks with the CDK requires special dedicated {aws} CDK resources to be provisioned. The cdk bootstrap command creates the necessary resources for you. You only need to bootstrap if you are deploying a stack that requires these dedicated resources. See xref:bootstrapping[{aws} CDK bootstrapping] for details.

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

## cdk bootstrap

If issued with no arguments, as shown here, the cdk bootstrap command synthesizes the current app and bootstraps the environments its stacks will be deployed to. If the app contains environment-agnostic stacks, which don’t explicitly specify an environment, the default account and Region are bootstrapped, or the environment specified using --profile.

Outside of an app, you must explicitly specify the environment to be bootstrapped. You may also do so to bootstrap an environment that’s not specified in your app or local {aws} profile. Credentials must be configured (e.g. in ~/.aws/credentials) for the specified account and Region. You may specify a profile that contains the required credentials.

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

cdk bootstrap ++++++/++++++# e.g. cdk bootstrap 1111111111/us-east-1 cdk bootstrap –profile test 1111111111/us-east-1 —-++++++++++++

= [IMPORTANT]

Each environment (account/region combination) to which you deploy such a stack must be bootstrapped separately.

====

You may incur {aws} charges for what the {aws} CDK stores in the bootstrapped resources. Additionally, if you use --bootstrap-customer-key, an {aws} KMS key will be created, which also incurs charges per environment.

= [NOTE]

Earlier versions of the bootstrap template created a KMS key by default. To avoid charges, re-bootstrap using --no-bootstrap-customer-key.

====

= [NOTE]

CDK CLI v2 does not support the original bootstrap template, dubbed the legacy template, used by default with CDK v1.

====

= [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.

====

[#cli-init] == Create a new app

To create a new app, create a directory for it, then, inside the directory, issue cdk init.

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

mkdir my-cdk-app cd my-cdk-app cdk init ++++++–language ++++++—-++++++++++++

The supported languages (++++++) are:++++++

[cols=“1,1”, options=“header”] |=== | Code | Language

|=== | typescript | TypeScript |===

|=== | javascript | JavaScript |===

|=== | python | Python |===

|=== | java | Java |===

|=== | csharp | C# |===

|=== | Go | Go |===++++++is an optional template. If the desired template is *app*, the default, you may omit it. The available templates are: [cols=“1,1”, options=“header”] |=== | Template | Description |app (default) |Creates an empty {aws} CDK app. |sample-app |Creates an {aws} CDK app with a stack containing an Amazon SQS queue and an Amazon SNS topic. |=== The templates use the name of the project folder to generate names for files and classes inside your new app. [#cli-list] == List stacks To see a list of the IDs of the stacks in your {aws} CDK application, enter one of the following equivalent commands: [source,none,subs=“verbatim,attributes”] —- cdk list cdk ls —- If your application contains xref:cdk-pipeline[CDK Pipelines] stacks, the CDK CLI displays stack names as paths according to their location in the pipeline hierarchy. (For example, PipelineStack, PipelineStack/Prod, and PipelineStack/Prod/MyService.) If your app contains many stacks, you can specify full or partial stack IDs of the stacks to be listed. For more information, see xref:cli-stacks[Specify stacks]. Add the --long flag to see more information about the stacks, including the stack names and their environments ({aws} account and Region). [#cli-synth] == Synthesize stacks The cdk synthesize command (almost always abbreviated synth) synthesizes a stack defined in your app into a CloudFormation template. [source,none,subs=“verbatim,attributes”] —- cdk synth # if app contains only one stack cdk synth MyStack cdk synth Stack1 Stack2 cdk synth “*” # all stacks in app —- [NOTE] ==== The CDK CLI actually runs your app and synthesizes fresh templates before most operations (such as when deploying or comparing stacks). These templates are stored by default in the cdk.out directory. The cdk synth command simply prints the generated templates for one or more specified stacks. ==== See cdk synth --help for all available options. A few of the most frequently used options are covered in the following section. [#cli-specify-context#] === Specify context values Use the --context or -c option to pass xref:context[runtime context] values to your CDK app. [source,none,subs=”verbatim,attributes”] —- # specify a single context value cdk synth –context key=value MyStack # specify multiple context values (any number) cdk synth –context key1=value1 –context key2=value2 MyStack —- When deploying multiple stacks, the specified context values are normally passed to all of them. If you want, you can specify different values for each stack by prefixing the stack name to the context value. [source,none,subs=”verbatim,attributes”] —- # different context values for each stack cdk synth –context Stack1:key=value Stack2:key=value Stack1 Stack2 —- [#cli-specify-format] === Specify display format By default, the synthesized template is displayed in YAML format. Add the --json flag to display it in JSON format instead. [source,none,subs=”verbatim,attributes”] —- cdk synth –json MyStack —- [#cli-specify-output] === Specify the output directory Add the --output (-o) option to write the synthesized templates to a directory other than cdk.out. [source,none,subs=”verbatim,attributes”] —- cdk synth –output=~/templates —- [#cli-deploy] == Deploy stacks The cdk deploy subcommand deploys one or more specified stacks to your {aws} account. [source,none,subs=”verbatim,attributes”] —- cdk deploy # if app contains only one stack cdk deploy MyStack cdk deploy Stack1 Stack2 cdk deploy ”*” # all stacks in app —- [NOTE] ==== The CDK CLI runs your app and synthesizes fresh {aws} CloudFormation templates before deploying anything. Therefore, most command line options you can use with cdk synth (for example, --context) can also be used with cdk deploy. ==== See cdk deploy --help for all available options. A few of the most useful options are covered in the following section. [#cli-deploy-nosynth] === Skip synthesis The cdk deploy command normally synthesizes your app’s stacks before deploying to make sure that the deployment reflects the latest version of your app. If you know that you haven’t changed your code since your last cdk synth, you can suppress the redundant synthesis step when deploying. To do so, specify your project’s cdk.out directory in the --app option. [source,none,subs=“verbatim,attributes”] —- cdk deploy –app cdk.out StackOne StackTwo —- [#cli-deploy-norollback] === Disable rollback {aws} CloudFormation has the ability to roll back changes so that deployments are atomic. This means that they either succeed or fail as a whole. The {aws} CDK inherits this capability because it synthesizes and deploys {aws} CloudFormation templates. Rollback makes sure that your resources are in a consistent state at all times, which is vital for production stacks. However, while you’re still developing your infrastructure, some failures are inevitable, and rolling back failed deployments can slow you down. For this reason, the CDK CLI lets you disable rollback by adding --no-rollback to your cdk deploy command. With this flag, failed deployments are not rolled back. Instead, resources deployed before the failed resource remain in place, and the next deployment starts with the failed resource. You’ll spend a lot less time waiting for deployments and a lot more time developing your infrastructure. [#cli-deploy-hotswap] === Hot swapping Use the --hotswap flag with cdk deploy to attempt to update your {aws} resources directly instead of generating an {aws} CloudFormation change set and deploying it. Deployment falls back to {aws} CloudFormation deployment if hot swapping is not possible. Currently hot swapping supports Lambda functions, Step Functions state machines, and Amazon ECS container images. The --hotswap flag also disables rollback (i.e., implies --no-rollback). [IMPORTANT] ==== Hot-swapping is not recommended for production deployments. ==== [#cli-deploy-watch] === Watch mode The CDK CLI’s watch mode (cdk deploy --watch, or cdk watch for short) continuously monitors your CDK app’s source files and assets for changes. It immediately performs a deployment of the specified stacks when a change is detected. By default, these deployments use the --hotswap flag, which fast-tracks deployment of changes to Lambda functions. It also falls back to deploying through {aws} CloudFormation if you have changed infrastructure configuration. To have cdk watch always perform full {aws} CloudFormation deployments, add the --no-hotswap flag to cdk watch. Any changes made while cdk watch is already performing a deployment are combined into a single deployment, which begins as soon as the in-progress deployment is complete. Watch mode uses the "watch" key in the project’s cdk.json to determine which files to monitor. By default, these files are your application files and assets, but this can be changed by modifying the "include" and "exclude" entries in the "watch" key. The following cdk.json file shows an example of these entries. [source,json,subs=“verbatim,attributes”] —- { “app”: “mvn -e -q compile exec:java”, “watch”: { “include”: “src/main/\*\*“,”exclude”: “target/*” } } —- cdk watch executes the "build" command from cdk.json to build your app before synthesis. If your deployment requires any commands to build or package your Lambda code (or anything else that’s not in your CDK app), add it here. Git-style wildcards, both \\* and \\*\*, can be used in the "watch" and "build" keys. Each path is interpreted relative to the parent directory of cdk.json. The default value of include is \\*\*/\*, meaning all files and directories in the project root directory. exclude is optional. [IMPORTANT] ==== Watch mode is not recommended for production deployments. ==== [#cli-specify-parameters] === Specify {aws} CloudFormation parameters The CDK CLI supports specifying {aws} CloudFormation xref:parameters[parameters] at deployment. You may provide these on the command line following the --parameters flag. [source,none,subs=”verbatim,attributes”] —- cdk deploy MyStack –parameters uploadBucketName=UploadBucket —- To define multiple parameters, use multiple --parameters flags. [source,none,subs=”verbatim,attributes”] —- cdk deploy MyStack –parameters uploadBucketName=UpBucket –parameters downloadBucketName=DownBucket —- If you are deploying multiple stacks, you can specify a different value of each parameter for each stack. To do so, prefix the name of the parameter with the stack name and a colon. Otherwise, the same value is passed to all stacks. [source,none,subs=”verbatim,attributes”] —- cdk deploy MyStack YourStack –parameters MyStack:uploadBucketName=UploadBucket –parameters YourStack:uploadBucketName=UpBucket —- By default, the {aws} CDK retains values of parameters from previous deployments and uses them in later deployments if they are not specified explicitly. Use the --no-previous-parameters flag to require all parameters to be specified. [#cli-specify-outputs-file] === Specify outputs file If your stack declares {aws} CloudFormation outputs, these are normally displayed on the screen at the conclusion of deployment. To write them to a file in JSON format, use the --outputs-file flag. [source,none,subs=”verbatim,attributes”] —- cdk deploy –outputs-file outputs.json MyStack —- [#cli-security] === Approve security-related changes To protect you against unintended changes that affect your security posture, the CDK CLI prompts you to approve security-related changes before deploying them. You can specify the level of change that requires approval: [source,none,subs=”verbatim,attributes”] —- cdk deploy –require-approval ++++++—- +++<LEVEL>+++ can be one of the following: [cols=”1,1”, options=”header”] |=== | Term | Meaning |never |Approval is never required |any-change |Requires approval on any IAM or security-group-related change |broadening (default) |Requires approval when IAM statements or traffic rules are added; removals don’t require approval |=== The setting can also be configured in the cdk.json file. [source,json,subs=”verbatim,attributes”] —- { ”app”: ”...”, ”requireApproval”: ”never” } —- [#cli-diff] == Compare stacks The cdk diff command compares the current version of a stack (and its dependencies) defined in your app with the already-deployed versions, or with a saved {aws} CloudFormation template, and displays a list of changes. —- Stack HelloCdkStack IAM Statement Changes ┌───┬──────────────────────────────┬────────┬──────────────────────────────┬──────────────────────────────┬───────────┐ │ │ Resource │ Effect │ Action │ Principal │ Condition │ ├───┼──────────────────────────────┼────────┼──────────────────────────────┼──────────────────────────────┼───────────┤ │ + │ ${Custom::S3AutoDeleteObject │ Allow │ sts:AssumeRole │ Service:lambda.amazonaws.com │ │ │ │ sCustomResourceProvider/Role │ │ │ │ │ │ │ .Arn} │ │ │ │ │ ├───┼──────────────────────────────┼────────┼──────────────────────────────┼──────────────────────────────┼───────────┤ │ + │ ${MyFirstBucket.Arn} │ Allow │ s3:DeleteObject\* │ \{aws}:${Custom::S3AutoDeleteOb │ │ │ │ ${MyFirstBucket.Arn}/* │ │ s3:GetBucket\* │ jectsCustomResourceProvider/ │ │ │ │ │ │ s3:GetObject\* │ Role.Arn} │ │ │ │ │ │ s3:List\* │ │ │ └───┴──────────────────────────────┴────────┴──────────────────────────────┴──────────────────────────────┴───────────┘ IAM Policy Changes ┌───┬────────────────────────────────────────────────────────┬────────────────────────────────────────────────────────┐ │ │ Resource │ Managed Policy ARN │ ├───┼────────────────────────────────────────────────────────┼────────────────────────────────────────────────────────┤ │ + │ ${Custom::S3AutoDeleteObjectsCustomResourceProvider/Ro │ {"Fn::Sub":"arn:${{aws}::Partition}:iam::aws:policy/serv │ │ │ le} │ ice-role/AWSLambdaBasicExecutionRole”} │ └───┴────────────────────────────────────────────────────────┴────────────────────────────────────────────────────────┘ (NOTE: There may be security-related changes not in this list. See https://github.com/aws/aws-cdk/issues/1299) Parameters [+] Parameter AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/S3Bucket AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3BucketBF7A7F3F: {“Type”:“String”,“Description”:“S3 bucket for asset "4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392"”} [+] Parameter AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/S3VersionKey AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3VersionKeyFAF93626: {“Type”:“String”,“Description”:“S3 key for asset version "4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392"”} [+] Parameter AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/ArtifactHash AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392ArtifactHashE56CD69A: {“Type”:“String”,“Description”:“Artifact hash for asset "4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392"”} Resources [+] {aws}::S3::BucketPolicy MyFirstBucket/Policy MyFirstBucketPolicy3243DEFD [+] Custom::S3AutoDeleteObjects MyFirstBucket/AutoDeleteObjectsCustomResource MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E [+] {aws}::IAM::Role Custom::S3AutoDeleteObjectsCustomResourceProvider/Role CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092 [+] {aws}::Lambda::Function Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F [~] {aws}::S3::Bucket MyFirstBucket MyFirstBucketB8884501 ├─ [~] DeletionPolicy │ ├─ [-] Retain │ └─ [+] Delete └─ [~] UpdateReplacePolicy ├─ [-] Retain └─ [+] Delete —- To compare your app’s stacks with the existing deployment: [source,none,subs=“verbatim,attributes”] —- cdk diff MyStack —- To compare your app’s stacks with a saved CloudFormation template: [source,none,subs=“verbatim,attributes”] —- cdk diff –template ~/stacks/MyStack.old MyStack —- [#cli-import] == Import existing resources into a stack You can use the cdk import command to bring resources under the management of CloudFormation for a particular {aws} CDK stack. This is useful if you are migrating to {aws} CDK, or are moving resources between stacks or changing their logical id. cdk import uses https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/resource-import.html[CloudFormation resource imports]. See the https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/resource-import-supported-resources.html[list of resources that can be imported here]. To import an existing resource into a {aws} CDK stack, follow the following steps: \* Make sure the resource is not currently being managed by any other CloudFormation stack. If it is, first set the removal policy to RemovalPolicy.RETAIN in the stack the resource is currently in and perform a deployment. Then, remove the resource from the stack and perform another deployment. This process will make sure that the resource is no longer managed by CloudFormation but does not delete it. \* Run a cdk diff to make sure there are no pending changes to the {aws} CDK stack you want to import resources into. The only changes allowed in an “import” operation are the addition of new resources which you want to import. \* Add constructs for the resources you want to import to your stack. For example, if you want to import an Amazon S3 bucket, add something like new s3.Bucket(this, 'ImportedS3Bucket', {});. Do not make any modifications to any other resource. + You must also make sure to exactly model the state that the resource currently has into the definition. For the example of the bucket, be sure to include {aws} KMS keys, life cycle policies, and anything else that’s relevant about the bucket. If you do not, subsequent update operations may not do what you expect. + You can choose whether or not to include the physical bucket name. We usually recommend to not include resource names into your {aws} CDK resource definitions so that it becomes easier to deploy your resources multiple times. + \* Run cdk import +++<STACKNAME>+++. \* If the resource names are not in your model, the CLI will prompt you to pass in the actual names of the resources you are importing. After this, the import starts. \* When cdk import reports success, the resource is now managed by {aws} CDK and CloudFormation. Any subsequent changes you make to the resource properties in your {aws} CDK app the construct configuration will be applied on the next deployment. \* To confirm that the resource definition in your {aws} CDK app matches the current state of the resource, you can start an https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html[CloudFormation drift detection operation]. This feature currently does not support importing resources into nested stacks. [#cli-config] == Configuration (cdk.json) Default values for many CDK CLI command line flags can be stored in a project’s cdk.json file or in the .cdk.json file in your user directory. Following is an alphabetical reference to the supported configuration settings. [cols=“1,1,1”, options=“header”] |=== | Key | Notes | CDK CLI option |app |The command that executes the CDK application. |--app |assetMetadata |If false, CDK does not add metadata to resources that use assets. |--no-asset-metadata |bootstrapKmsKeyId |Overrides the ID of the {aws} KMS key used to encrypt the Amazon S3 deployment bucket. |--bootstrap-kms-key-id |build |The command that compiles or builds the CDK application before synthesis. Not permitted in ~/.cdk.json. |--build |browser |The command for launching a Web browser for the cdk docs subcommand. |--browser |context |See xref:context[Context values and the {aws} CDK]. Context values in a configuration file will not be erased by cdk context --clear. (The CDK CLI places cached context values in cdk.context.json.) |--context |debug |If true, CDK CLI emits more detailed information useful for debugging. |--debug |language |The language to be used for initializing new projects. |--language |lookups |If false, no context lookups are permitted. Synthesis will fail if any context lookups need to be performed. |--no-lookups |notices |If false, suppresses the display of messages about security vulnerabilities, regressions, and unsupported versions. |--no-notices |output |The name of the directory into which the synthesized cloud assembly will be emitted (default "cdk.out"). |--output |outputsFile |The file to which {aws} CloudFormation outputs from deployed stacks will be written (in JSON format). |--outputs-file |pathMetadata |If false, CDK path metadata is not added to synthesized templates. |--no-path-metadata |plugin |JSON array specifying the package names or local paths of packages that extend the CDK |--plugin |profile |Name of the default {aws} profile used for specifying Region and account credentials. |--profile |progress |If set to "events", the CDK CLI displays all {aws} CloudFormation events during deployment, rather than a progress bar. |--progress |requireApproval |Default approval level for security changes. See xref:cli-security[Approve security-related changes] |--require-approval |rollback |If false, failed deployments are not rolled back. |--no-rollback |staging |If false, assets are not copied to the output directory (use for local debugging of the source files with {aws} SAM). |--no-staging |tags |JSON object containing tags (key-value pairs) for the stack. |--tags |toolkitBucketName |The name of the Amazon S3 bucket used for deploying assets such as Lambda functions and container images (see xref:cli-bootstrap[Bootstrap your {aws} environment]). |--toolkit-bucket-name |toolkitStackName |The name of the bootstrap stack (see xref:cli-bootstrap[Bootstrap your {aws} environment]). |--toolkit-stack-name |versionReporting |If false, opts out of version reporting. |--no-version-reporting |watch |JSON object containing "include" and "exclude" keys that indicate which files should (or should not) trigger a rebuild of the project when changed. See xref:cli-deploy-watch[Watch mode]. |--watch |===++++++++++++++++++++++++

include::attributes.txt[]

// Attributes

[.topic] [#configure-access-sso-example-cli] = Example: Authenticate with IAM Identity Center automatic token refresh for use with the {aws} CDK CLI :info\_titleabbrev: Example: Authenticate with IAM Identity Center automatic token refresh :keywords: {aws}, {aws} CDK, {aws} CDK CLI, Programmatic access, Security credentials, IAM, IAM Identity Center

== [abstract]

## In this example, we configure the {aws} Command Line Interface to authenticate our user with the {aws} IAM Identity Center token provider configuration. The SSO token provider configuration lets the {aws} CLI automatically retrieve refreshed authentication tokens to generate short-term credentials that we can use with the {aws} Cloud Development Kit ({aws} CDK) Command Line Interface ({aws} CDK CLI).

// Content start

In this example, we configure the {aws} Command Line Interface ({aws} CLI) to authenticate our user with the {aws} IAM Identity Center token provider configuration. The SSO token provider configuration lets the {aws} CLI automatically retrieve refreshed authentication tokens to generate short-term credentials that we can use with the {aws} Cloud Development Kit ({aws} CDK) Command Line Interface ({aws} CDK CLI).

[#configure-access-sso-example-cli-prerequisites] == Prerequisites

This example assumes that the following prerequisites have been completed:

* Prerequisites required to get set up with {aws} and install our starting CLI tools. For more information, see xref:configure-access-prerequisites[Prerequisites].
* IAM Identity Center has been set up by our organization as the method of managing users.
* At least one user has been created in IAM Identity Center.

[#configure-access-sso-example-cli-configure] == Step 1: Configure the {aws} CLI

For detailed instructions on this step, see https://docs.aws.amazon.com/cli/latest/userguide/sso-configure-profile-token.html[Configure the {aws} CLI to use IAM Identity Center token provider credentials with automatic authentication refresh] in the *{aws} Command Line Interface User Guide*.

We sign in to the {aws} access portal provided by our organization to gather our IAM Identity Center information. This includes the *SSO start URL* and *SSO Region*.

Next, we use the {aws} CLI aws configure sso command to configure an IAM Identity Center profile and sso-session on our local machine:

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

$ aws configure sso SSO session name (Recommended): ++++++SSO start URL [None]: <https://my-sso-portal.awsapps.com/start> SSO region [None]: ++++++SSO registration scopes [sso:account:access]: ++++++—-++++++++++++++++++

The {aws} CLI attempts to open our default browser to begin the login process for our IAM Identity Center account. If the {aws} CLI is unable to open our browser, instructions are provided to manually start the login process. This process associates the IAM Identity Center session with our current {aws} CLI session.

After establishing our session, the {aws} CLI displays the {aws} accounts available to us:

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

There are 2 {aws} accounts available to you.

DeveloperAccount, developer-account-admin@example.com (<123456789011>) ProductionAccount, production-account-admin@example.com (<123456789022>) — \_\_\_\_

We use the arrow keys to select our *DeveloperAccount*.

Next, the {aws} CLI displays the IAM roles available to us from our selected account:

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

Using the account ID <123456789011> There are 2 roles available to you.

ReadOnly FullAccess — \_\_\_\_

We use the arrow keys to select *FullAccess*.

Next, the {aws} CLI prompts us to complete configuration by specifying a default output format, default {aws} Region, and name for our profile:

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

CLI default client Region [None]: ++++++++++++CLI default output format [None]: ++++++++++++CLI profile name [123456789011\_FullAccess]: ++++++++++++—-++++++++++++++++++++++++++++++++++++

The {aws} CLI displays a final message, showing how to use the named profile with the {aws} CLI:

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

To use this profile, specify the profile name using –profile, as shown:

== aws s3 ls –profile ++++++++++++

After completing this step, our config file will look like the following:

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

[profile ++++++] sso\_session = ++++++sso\_account\_id = <123456789011> sso\_role\_name = ++++++region = ++++++output = ++++++++++++++++++++++++++++++++++++

[sso-session ++++++] sso\_region = ++++++sso\_start\_url = <https://my-sso-portal.awsapps.com/start> sso\_registration\_scopes = —-++++++++++++

We can now use this sso-session and named profile to request security credentials.

[#configure-access-sso-example-cli-credentials] == Step 2: Use the {aws} CLI to generate security credentials

For detailed instructions on this step, see https://docs.aws.amazon.com/cli/latest/userguide/sso-using-profile.html[Use an IAM Identity Center named profile] in the *{aws} Command Line Interface User Guide*.

We use the {aws} CLI aws sso login command to request security credentials for our profile:

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

$ aws sso login –profile ++++++—-++++++

The {aws} CLI attempts to open our default browser and verifies our IAM log in. If we are not currently signed into IAM Identity Center, we will be prompted to complete the sign in process. If the {aws} CLI is unable to open our browser, instructions are provided to manually start the authorization process.

After successfully logging in, the {aws} CLI caches our IAM Identity Center session credentials. These credentials include an expiration timestamp. When they expire, the {aws} CLI will request that we sign in to IAM Identity Center again.

Using valid IAM Identity Center credentials, the {aws} CLI securely retrieves {aws} credentials for the IAM role specified in our profile. From here, we can use the {aws} CDK CLI with our credentials.

[#configure-access-sso-example-cli-cdk] == Step 3: Use the CDK CLI

With any CDK CLI command, we use the xref:ref-cli-cmd-options-profile[--profile] option to specify the named profile that we generated credentials for. If our credentials are valid, the CDK CLI will successfully perform the command. The following is an example:

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

$ cdk diff –profile ++++++Stack CdkAppStack Hold on while we create a read-only change set to get a diff with accurate replacement information (use –no-change-set to use a less accurate but faster template-only diff) Resources [-] {aws}::S3::Bucket amzn-s3-demo-bucket amzn-s3-demo-bucket5AF9C99B destroy++++++

Outputs [-] Output BucketRegion: {“Value”:{“Ref”:“{aws}::Region”}}

== ✨ Number of stacks with differences: 1

When our credentials expire, an error message like the following will display:

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

$ cdk diff –profile ++++++Stack CdkAppStack++++++

== Unable to resolve {aws} account to use. It must be either configured when you define your CDK Stack, or through the environment

To refresh our credentials, we use the {aws} CLI aws sso login command:

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

$ aws sso login –profile ++++++—-++++++

include::attributes.txt[]

// Attributes

[.topic] [#configure-access] = Configure security credentials for the {aws} CDK CLI :info\_titleabbrev: Configure security credentials :keywords: {aws}, {aws} CDK, {aws} CDK CLI, Programmatic access, Security credentials, IAM, IAM Identity Center

== [abstract]

## When you use the {aws} Cloud Development Kit ({aws} CDK) to develop applications in your local environment, you will primarily use the {aws} CDK CLI to interact with {aws} to deploy and manage your CDK stacks. To use the CDK CLI, you must configure security credentials.

// Content start

When you use the {aws} Cloud Development Kit ({aws} CDK) to develop applications in your local environment, you will primarily use the {aws} CDK Command Line Interface ({aws} CDK CLI) to interact with {aws}. For example, you can use the CDK CLI to deploy your application or to delete your resources from your {aws} environment.

To use the CDK CLI to interact with {aws}, you must configure security credentials on your local machine. This lets {aws} know who you are and what permissions you have.

To learn more about security credentials, see https://docs.aws.amazon.com/IAM/latest/UserGuide/security-creds.html[{aws} security credentials] in the *IAM User Guide*.

[#configure-access-prerequisites,] == Prerequisites

Configuring security credentials is part of the *getting started* process. Complete all prerequisites and previous steps at xref:getting-started[Getting started with the {aws} CDK].

[#configure-access-how] == How to configure security credentials

How you configure security credentials depends on how you or your organization manages users. Whether you use {aws} Identity and Access Management (IAM) or {aws} IAM Identity Center, we recommend that you use the {aws} Command Line Interface ({aws} CLI) to configure and manage security credentials for the CDK CLI. This includes using {aws} CLI commands like aws configure to configure security credentials on your local machine. However, you can use alternative methods such as manually updating your config and credentials files, or setting environment variables.

For guidance on configuring security credentials using the {aws} CLI, along with information on configuration and credential precedence when using different methods, see https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-authentication.html[Authentication and access credentials] in the *{aws} Command Line Interface User Guide*. The CDK CLI adheres to the same configuration and credential precedence of the {aws} CLI. The --profile command line option takes precedence over environment variables. If you have both the AWS\_PROFILE and CDK\_DEFAULT\_PROFILE environment variables configured, the AWS\_PROFILE environment variable takes precedence.

If you configure multiple profiles, you can use the CDK CLI xref:ref-cli-cmd-options-profile[--profile] option with any command to specify the profile from your credentials and config files to use for authentication. If you don’t provide --profile, the default profile will be used.

If you prefer to quickly configure basic settings, including security credentials, see https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html[Set up the {aws} CLI] in the *{aws} Command Line Interface User Guide*.

Once you’ve configured security credentials on your local machine, you can use the CDK CLI to interact with {aws}.

[#configure-access-sso] == Configure and manage security credentials for IAM Identity Center users

IAM Identity Center users can authenticate with IAM Identity Center or manually by using short-term credentials.

[#configure-access-sso-auto] *Authenticate with IAM Identity Center to generate short-term credentials*:: + You can configure the {aws} CLI to authenticate with IAM Identity Center. This is the recommended approach of configuring security credentials for IAM Identity Center users. IAM Identity Center users can use the {aws} CLI aws configure sso wizard to configure an IAM Identity Center profile and sso-session, which gets stored in the config file on your local machine. For instructions, see https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-sso.html[Configure the {aws} CLI to use {aws} IAM Identity Center] in the *{aws} Command Line Interface User Guide*. + Next, you can use the {aws} CLI aws sso login command to request refreshed credentials. You can also use this command to switch profiles. For instructions, see https://docs.aws.amazon.com/cli/latest/userguide/sso-using-profile.html[Use an IAM Identity Center named profile] in the *{aws} Command Line Interface User Guide*. + Once authenticated, you can use the CDK CLI to interact with {aws} for the duration of your session. For an example, see xref:configure-access-sso-example-cli[Example: Authenticate with IAM Identity Center automatic token refresh for use with the {aws} CDK CLI].

[#configure-access-sso-manual] *Manually configure short-term credentials*:: + As an alternative to using the {aws} CLI and authenticating with IAM Identity Center, IAM Identity Center users can obtain short-term credentials from the {aws} Management Console and manually configure the credentials and config files on their local machine. Once configured, you can use the CDK CLI to interact with {aws} until your credentials expire. For instructions, see https://docs.aws.amazon.com/cli/latest/userguide/cli-authentication-short-term.html[Authenticate with short-term credentials] in the *{aws} Command Line Interface User Guide*.

[#configure-access-iam] == Configure and manage security credentials for IAM users

IAM users can use an IAM role or IAM user credentials with the CDK CLI.

[#configure-access-iam-role] *Use an IAM role to configure short-term credentials*:: + IAM users can assume IAM roles to gain additional (or different) permissions. For IAM users, this is the recommended approach since it provides short-term credentials. + First, the IAM role and user’s permission to assume the role must be configured. This is typically performed by an administrator using the {aws} Management Console or {aws} CLI. Then, the IAM user can use the {aws} CLI to assume the role and configure short-term credentials on their local machine. For instructions, see https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-role.html[Use an IAM role in the {aws} CLI] in the *{aws} Command Line Interface User Guide*.

[#configure-access-iam-user] *Use IAM user credentials*:: + [WARNING] ====

To avoid security risks, we don’t recommend using IAM user credentials since they provide long-term access. If you must use long-term credentials, we recommend that you link:https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#rotate-credentials[update access keys] as an IAM security best practice.

==== + IAM users can obtain access keys from the {aws} Management Console. You can then use the {aws} CLI to configure long-term credentials on your local machine. For instructions, see https://docs.aws.amazon.com/cli/latest/userguide/cli-authentication-user.html[Authenticate with IAM user credentials] in the *{aws} Command Line Interface User Guide*.

[#configure-access-info,] == Additional information

To learn about the different ways that you can sign in to {aws}, depending on the type of user you are, see https://docs.aws.amazon.com/signin/latest/userguide/what-is-sign-in.html[What is {aws} Sign-In?] in the *{aws} Sign-In User Guide*.

For reference information when using {aws} SDKs and tools, including the {aws} CLI, see the https://docs.aws.amazon.com/sdkref/latest/guide/overview.html[{aws} SDKs and Tools Reference Guide].

include::configure-access-sso-example-cli.adoc[leveloffset=+1]

:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#configure-env] = Configure environments to use with the {aws} CDK :info\_titleabbrev: Configure environments :keywords: {aws} CDK, {aws} environments, environments, {aws} account, {aws} Region, env

== [abstract]

## You can configure {aws} environments in multiple ways to use with the {aws} Cloud Development Kit ({aws} CDK). The best method of managing {aws} environments will vary, based on your specific needs.

// Content start

You can configure {aws} environments in multiple ways to use with the {aws} Cloud Development Kit ({aws} CDK). The best method of managing {aws} environments will vary, based on your specific needs.

Each CDK stack in your application must eventually be associated with an environment to determine where the stack gets deployed to.

For an introduction to {aws} environments, see xref:environments[Environments for the {aws} CDK].

[#configure-env-where] == Where you can specify environments from

You can specify environments in credentials and configuration files, or by using the +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#env[env]+ property of the Stack construct from the {aws} Construct Library.

[#configure-env-where-files] === Credentials and configuration files

You can use the {aws} Command Line Interface ({aws} CLI) to create credentials and config files that store, organize, and manage your {aws} environment information. To learn more about these files, see https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html[Configuration and credential file settings] in the *{aws} Command Line Interface User Guide*.

Values stored in these files are organized by *profiles*. How you name your profiles and the key-value pairs in these files will vary based on your method of configuring programmatic access. To learn more about the different methods, see xref:configure-access[Configure security credentials for the {aws} CDK CLI].

In general, the {aws} CDK resolves {aws} account information from your credentials file and {aws} Region information from your config file.

Once you have your credentials and config files configured, you can specify the environment to use with the {aws} CDK CLI and through environment variables.

[#configure-env-where-env] === env property of the Stack construct

You can specify the environment for each stack by using the +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html#env[env]+ property of the Stack construct. This property defines an account and Region to use. You can pass hard-coded values to this property or pass environment variables that are offered by the CDK.

To pass environment variables, use the AWS\_DEFAULT\_ACCOUNT and AWS\_DEFAULT\_REGION environment variables. These environment variables can pass values from your credentials and config files. You can also use logic within your CDK code to determine the values of these environment variables.

[#configure-env-precedence] == Environment precedence with the {aws} CDK

If you use multiple methods of specifying environments, the {aws} CDK adheres to the following precedence:

. Hard-coded values specified with the env property of the Stack construct. . AWS\_DEFAULT\_ACCOUNT and AWS\_DEFAULT\_REGION environment variables specified with the env property of the Stack construct. . Environment information associated with the profile from your credentials and config files and passed to the CDK CLI using the --profile option. . The default profile from your credentials and config files.

[#configure-env-when] == When to specify environments

When you develop with the CDK, you start by defining CDK stacks, which contain constructs that represent {aws} resources. Next, you synthesize each CDK stack into an {aws} CloudFormation template. You then deploy the CloudFormation template to your environment. How you specify environments determines when your environment information gets applied and can affect CDK behavior and outcomes.

[#configure-env-when-synth] === Specify environments at template synthesis

When you specify environment information using the env property of the Stack construct, your environment information is applied at template synthesis. Running cdk synth or cdk deploy produces an environment-specific CloudFormation template.

If you use environment variables within the env property, you must use the --profile option with CDK CLI commands to pass in the profile containing your environment information from your credentials and configuration files. This information will then be applied at template synthesis to produce an environment-specific template.

Environment information within the CloudFormation template takes precedence over other methods. For example, if you provide a different environment with cdk deploy --profile <profile>, the profile will be ignored.

When you provide environment information in this way, you can use environment-dependent code and logic within your CDK app. This also means that the synthesized template could be different, based on the machine, user, or session that it’s synthesized under. This approach is often acceptable or desirable during development, but is not recommended for production use.

[#configure-env-when-deploy,] === Specify environments at stack deployment

If you don’t specify an environment using the env property of the Stack construct, the CDK CLI will produce an environment-agnostic CloudFormation template at synthesis. You can then specify the environment to deploy to by using cdk deploy --profile <profile>.

If you don’t specify a profile when deploying an environment-agnostic template, the CDK CLI will attempt to use environment values from the default profile of your credentials and config files at deployment.

If environment information is not available at deployment, {aws} CloudFormation will attempt to resolve environment information at deployment through environment-related attributes such as stack.account, stack.region, and stack.availabilityZones.

For environment-agnostic stacks, constructs within the stack cannot use environment information and you cannot use logic that requires environment information. For example, you cannot write code like if (stack.region ==== 'us-east-1') or use construct methods that require environment information such as +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ec2.Vpc.html#static-fromwbrlookupscope-id-options[Vpc.fromLookup]+. To use these features, you must specify an environment with the env property.

For environment-agnostic stacks, any construct that uses Availability Zones will see two Availability Zones, allowing the stack to be deployed to any Region.

[#configure-env-how] == How to specify environments with the {aws} CDK

[#configure-env-how-hard-coded] === Specify hard-coded environments for each stack

Use the env property of the Stack construct to specify {aws} environment values for your stack. The following is an example:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const envEU = { account: ‘2383838383’, region: ‘eu-west-1’ }; const envUSA = { account: ‘8373873873’, region: ‘us-west-2’ };

new MyFirstStack(app, ‘first-stack-us’, { env: envUSA }); new MyFirstStack(app, ‘first-stack-eu’, { env: envEU }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const envEU = { account: ‘2383838383’, region: ‘eu-west-1’ }; const envUSA = { account: ‘8373873873’, region: ‘us-west-2’ };

new MyFirstStack(app, ‘first-stack-us’, { env: envUSA }); new MyFirstStack(app, ‘first-stack-eu’, { env: envEU }); —

Python:: + [source,python,subs=“verbatim,attributes”] — env\_EU = cdk.Environment(account=“8373873873”, region=“eu-west-1”) env\_USA = cdk.Environment(account=“2383838383”, region=“us-west-2”)

MyFirstStack(app, “first-stack-us”, env=env\_USA) MyFirstStack(app, “first-stack-eu”, env=env\_EU) —

Java:: + [source,java,subs=“verbatim,attributes”] — public class MyApp {

…. // Helper method to build an environment static Environment makeEnv(String account, String region) { return Environment.builder() .account(account) .region(region) .build(); }

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

Environment envEU = makeEnv("8373873873", "eu-west-1");  
Environment envUSA = makeEnv("2383838383", "us-west-2");  
  
new MyFirstStack(app, "first-stack-us", StackProps.builder()  
 .env(envUSA).build());  
new MyFirstStack(app, "first-stack-eu", StackProps.builder()  
 .env(envEU).build());  
  
app.synth();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — Amazon.CDK.Environment makeEnv(string account, string region) { return new Amazon.CDK.Environment { Account = account, Region = region }; }

var envEU = makeEnv(account: “8373873873”, region: “eu-west-1”); var envUSA = makeEnv(account: “2383838383”, region: “us-west-2”);

new MyFirstStack(app, “first-stack-us”, new StackProps { Env=envUSA }); new MyFirstStack(app, “first-stack-eu”, new StackProps { Env=envEU }); —

Go:: + [source,go,subs=“verbatim,attributes”] — env\_EU := awscdk.Environment{ Account: jsii.String(“8373873873”), Region: jsii.String(“eu-west-1”), }

env\_USA := awscdk.Environment{ Account: jsii.String(“2383838383”), Region: jsii.String(“us-west-2”), }

MyFirstStack(app, “first-stack-us”, &awscdk.StackProps{ Env: &env\_USA, })

MyFirstStack(app, “first-stack-eu”, &awscdk.StackProps{ Env: &env\_EU, }) — ====

We recommend this approach for production environments. By explicitly specifying the environment in this way, you can ensure that the stack is always deployed to the specific environment.

[#configure-env-how-env] === Specify environments using environment variables

The {aws} CDK provides two environment variables that you can use within your CDK code: CDK\_DEFAULT\_ACCOUNT and CDK\_DEFAULT\_REGION. When you use these environment variables within the env property of your stack instance, you can pass environment information from your credentials and configuration files using the CDK CLI --profile option.

The following is an example of how to specify these environment variables:

==== [role=“tablist”] TypeScript:: Access environment variables via Node’s process object. + [NOTE] – You need the DefinitelyTyped module to use process in TypeScript. cdk init installs this module for you. However, you should install this module manually if you are working with a project created before it was added, or if you didn’t set up your project using cdk init.

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

## npm install @types/node

|  |
| --- |
| + |
| == [source,javascript,subs=“verbatim,attributes”] |
| new MyDevStack(app, ‘dev’, { env: { account: process.env.CDK\_DEFAULT\_ACCOUNT, region: process.env.CDK\_DEFAULT\_REGION }}); |

JavaScript:: Access environment variables via Node’s process object. + [source,javascript,subs=“verbatim,attributes”] — new MyDevStack(app, ‘dev’, { env: { account: process.env.CDK\_DEFAULT\_ACCOUNT, region: process.env.CDK\_DEFAULT\_REGION }}); —

Python:: Use the os module’s environ dictionary to access environment variables. + [source,python,subs=“verbatim,attributes”] — import os MyDevStack(app, “dev”, env=cdk.Environment( account=os.environ[“CDK\_DEFAULT\_ACCOUNT”], region=os.environ[“CDK\_DEFAULT\_REGION”])) —

Java:: Use System.getenv() to get the value of an environment variable. + [source,java,subs=“verbatim,attributes”] — public class MyApp {

…. // Helper method to build an environment static Environment makeEnv(String account, String region) { account = (account == null) ? System.getenv(“CDK\_DEFAULT\_ACCOUNT”) : account; region = (region == null) ? System.getenv(“CDK\_DEFAULT\_REGION”) : region;

return Environment.builder()  
 .account(account)  
 .region(region)  
 .build();

}

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

Environment envEU = makeEnv(null, null);  
Environment envUSA = makeEnv(null, null);  
  
new MyDevStack(app, "first-stack-us", StackProps.builder()  
 .env(envUSA).build());  
new MyDevStack(app, "first-stack-eu", StackProps.builder()  
 .env(envEU).build());  
  
app.synth();

} } —- ….

C#:: Use System.Environment.GetEnvironmentVariable() to get the value of an environment variable. + [source,csharp,subs=“verbatim,attributes”] — Amazon.CDK.Environment makeEnv(string account=null, string region=null) { return new Amazon.CDK.Environment { Account = account ?? System.Environment.GetEnvironmentVariable(“CDK\_DEFAULT\_ACCOUNT”), Region = region ?? System.Environment.GetEnvironmentVariable(“CDK\_DEFAULT\_REGION”) }; }

== new MyDevStack(app, “dev”, new StackProps { Env = makeEnv() });

Go:: + [source,go,subs=“verbatim,attributes”] — import “os”

MyDevStack(app, “dev”, &awscdk.StackProps{ Env: &awscdk.Environment{ Account: jsii.String(os.Getenv(“CDK\_DEFAULT\_ACCOUNT”)), Region: jsii.String(os.Getenv(“CDK\_DEFAULT\_REGION”)), }, }) — ====

By specifying environments using environment variables, you can have the same CDK stack synthesize to {aws} CloudFormation templates for different environments. This means that you can deploy the same CDK stack to different {aws} environments without having to modify your CDK code. You only have to specify the profile to use when running cdk synth.

This approach is great for development environments when deploying the same stack to different environments. However, we do not recommend this approach for production environments since the same CDK code can synthesize different templates, depending on the machine, user, or session that it’s synthesized under.

[#configure-env-how-files] === Specify environments from your credentials and configuration files with the CDK CLI

When deploying an environment-agnostic template, use the --profile option with any CDK CLI command to specify the profile to use. The following is an example that deploys a CDK stack named myStack using the prod profile that is defined in the credentials and config files:

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

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

For more information on the --profile option, along with other CDK CLI commands and options, see xref:ref-cli-cmd[{aws} CDK CLI command reference].

[#configure-env-considerations] == Considerations when configuring environments with the {aws} CDK

Services that you define by using constructs within your stacks must support the Region that you are deploying to. For a list of supported {aws} services per region, see https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/[{aws} Services by Region].

You must have valid {aws} Identity and Access Management (IAM) credentials to perform stack deployments with the {aws} CDK into your specified environments.

[#configure-env-examples] == Examples

[#configure-env-examples-agnostic] === Synthesize an environment–agnostic CloudFormation template from a CDK stack

In this example, we create an environment-agnostic CloudFormation template from our CDK stack. We can then deploy this template to any environment.

The following is our example CDK stack. This stack defines an Amazon S3 bucket and a CloudFormation stack output for the bucket’s Region. For this example, env is not defined:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — export class CdkAppStack extends cdk.Stack { constructor(scope: Construct, id: string, props?: cdk.StackProps) { super(scope, id, props);

…. // Create the S3 bucket const bucket = new s3.Bucket(this, ‘amzn-s3-demo-bucket’, { removalPolicy: cdk.RemovalPolicy.DESTROY, });

// Create an output for the bucket’s Region new cdk.CfnOutput(this, ‘BucketRegion’, { value: bucket.env.region, }); } } —- ….

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — class CdkAppStack extends cdk.Stack { constructor(scope, id, props) { super(scope, id, props);

…. // Create the S3 bucket const bucket = new s3.Bucket(this, ‘amzn-s3-demo-bucket’, { removalPolicy: cdk.RemovalPolicy.DESTROY, });

// Create an output for the bucket’s Region new cdk.CfnOutput(this, ‘BucketRegion’, { value: bucket.env.region, }); } } —- ….

Python:: + [source,python,subs=“verbatim,attributes”] — class CdkAppStack(cdk.Stack):

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

# Create the S3 bucket  
bucket = s3.Bucket(self, 'amzn-s3-demo-bucket',  
 removal\_policy=cdk.RemovalPolicy.DESTROY  
)  
  
# Create an output for the bucket's Region  
cdk.CfnOutput(self, 'BucketRegion',  
 value=bucket.env.region  
) ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — public class CdkAppStack extends Stack {

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

// Create the S3 bucket  
Bucket bucket = Bucket.Builder.create(this, "amzn-s3-demo-bucket")  
 .removalPolicy(RemovalPolicy.DESTROY)  
 .build();  
  
// Create an output for the bucket's Region  
CfnOutput.Builder.create(this, "BucketRegion")  
 .value(this.getRegion())  
 .build();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — namespace MyCdkApp { public class CdkAppStack : Stack { public CdkAppStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Create the S3 bucket var bucket = new Bucket(this, “amzn-s3-demo-bucket”, new BucketProps { RemovalPolicy = RemovalPolicy.DESTROY });

// Create an output for the bucket's Region  
 new CfnOutput(this, "BucketRegion", new CfnOutputProps  
 {  
 Value = this.Region  
 });  
 }

} } —-

Go:: + [source,go,subs=“verbatim,attributes”] — func NewCdkAppStack(scope constructs.Construct, id string, props \*CdkAppStackProps) awscdk.Stack { stack := awscdk.NewStack(scope, &id, &props.StackProps)

…. // Create the S3 bucket bucket := awss3.NewBucket(stack, jsii.String(“amzn-s3-demo-bucket”), &awss3.BucketProps{ RemovalPolicy: awscdk.RemovalPolicy\_DESTROY, })

// Create an output for the bucket’s Region awscdk.NewCfnOutput(stack, jsii.String(“BucketRegion”), &awscdk.CfnOutputProps{ Value: stack.Region(), })

return stack } —- ==== ….

When we run cdk synth, the CDK CLI produces a CloudFormation template with the pseudo parameter +{aws}::Region+ as the output value for the bucket’s Region. This parameter will be resolved at deployment:

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

Outputs: BucketRegion: Value: Ref: {aws}::Region —

To deploy this stack to an environment that is specified in the dev profile of our credentials and configuration files, we run the following:

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

## $ cdk deploy CdkAppStack –profile dev

If we don’t specify a profile, the CDK CLI will attempt to use environment information from the default profile in our credentials and configuration files.

[#configure-env-example-logic] === Use logic to determine environment information at template synthesis

In this example, we configure the env property of our stack instance to use a valid expression. We specify two additional environment variables, CDK\_DEPLOY\_ACCOUNT and CDK\_DEPLOY\_REGION. These environment variables can override defaults at synthesis time if they exist:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyDevStack(app, ‘dev’, { env: { account: process.env.CDK\_DEPLOY\_ACCOUNT || process.env.CDK\_DEFAULT\_ACCOUNT, region: process.env.CDK\_DEPLOY\_REGION || process.env.CDK\_DEFAULT\_REGION }}); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — new MyDevStack(app, ‘dev’, { env: { account: process.env.CDK\_DEPLOY\_ACCOUNT || process.env.CDK\_DEFAULT\_ACCOUNT, region: process.env.CDK\_DEPLOY\_REGION || process.env.CDK\_DEFAULT\_REGION }}); —

Python:: + [source,python,subs=“verbatim,attributes”] — MyDevStack(app, “dev”, env=cdk.Environment( account=os.environ.get(“CDK\_DEPLOY\_ACCOUNT”, os.environ[“CDK\_DEFAULT\_ACCOUNT”]), region=os.environ.get(“CDK\_DEPLOY\_REGION”, os.environ[“CDK\_DEFAULT\_REGION”]) ) ) —

Java:: + [source,java,subs=“verbatim,attributes”] — public class MyApp {

…. // Helper method to build an environment static Environment makeEnv(String account, String region) { account = (account == null) ? System.getenv(“CDK\_DEPLOY\_ACCOUNT”) : account; region = (region == null) ? System.getenv(“CDK\_DEPLOY\_REGION”) : region; account = (account == null) ? System.getenv(“CDK\_DEFAULT\_ACCOUNT”) : account; region = (region == null) ? System.getenv(“CDK\_DEFAULT\_REGION”) : region;

return Environment.builder()  
 .account(account)  
 .region(region)  
 .build();

}

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

Environment envEU = makeEnv(null, null);  
Environment envUSA = makeEnv(null, null);  
  
new MyDevStack(app, "first-stack-us", StackProps.builder()  
 .env(envUSA).build());  
new MyDevStack(app, "first-stack-eu", StackProps.builder()  
 .env(envEU).build());  
  
app.synth();

} } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — Amazon.CDK.Environment makeEnv(string account=null, string region=null) { return new Amazon.CDK.Environment { Account = account ?? System.Environment.GetEnvironmentVariable(“CDK\_DEPLOY\_ACCOUNT”) ?? System.Environment.GetEnvironmentVariable(“CDK\_DEFAULT\_ACCOUNT”), Region = region ?? System.Environment.GetEnvironmentVariable(“CDK\_DEPLOY\_REGION”) ?? System.Environment.GetEnvironmentVariable(“CDK\_DEFAULT\_REGION”) }; }

== new MyDevStack(app, “dev”, new StackProps { Env = makeEnv() });

Go:: + [source,go,subs=“verbatim,attributes”] — var account, region string var b bool

if account, b = os.LookupEnv(“CDK\_DEPLOY\_ACCOUNT”); !b || len(account) == 0 { account = os.Getenv(“CDK\_DEFAULT\_ACCOUNT”) } if region, b = os.LookupEnv(“CDK\_DEPLOY\_REGION”); !b || len(region) == 0 { region = os.Getenv(“CDK\_DEFAULT\_REGION”) }

MyDevStack(app, “dev”, &awscdk.StackProps{ Env: &awscdk.Environment{ Account: &account, Region: &region, }, }) — ====

With our stack's environment declared this way, we can then write a short script or batch file and set variables from command line arguments, then call ```cdk deploy```. The following is an example. Any arguments beyond the first two are passed through tocdk deploy` to specify command line options or arguments:

==== [role=“tablist”] macOS/Linux:: + [source,bash,subs=“verbatim,attributes”] — #!/usr/bin/env bash if [[ $# -ge 2 ]]; then export CDK\_DEPLOY\_ACCOUNT=$1 export CDK\_DEPLOY\_REGION=$2 shift; shift npx cdk deploy "$@” exit $? else echo 1>&2 “Provide account and region as first two args.” echo 1>&2 “Additional args are passed through to cdk deploy.” exit 1 fi — + Save the script as cdk-deploy-to.sh, then execute chmod +x cdk-deploy-to.sh to make it executable.

Windows:: + [source,bat,subs=“verbatim,attributes”] — @findstr /B /V @ %~dpnx0 > %~dpn0.ps1 && powershell -ExecutionPolicy Bypass %~dpn0.ps1 %\* @exit /B %ERRORLEVEL% if ($args.length -ge 2) { $env:CDK\_DEPLOY\_ACCOUNT, $args = $args $env:CDK\_DEPLOY\_REGION, $args = $args npx cdk deploy $args exit $lastExitCode } else { [console]::error.writeline(“Provide account and region as first two args.”) [console]::error.writeline(“Additional args are passed through to cdk deploy.”) exit 1 } — + The Windows version of the script uses PowerShell to provide the same functionality as the macOS/Linux version. It also contains instructions to allow it to be run as a batch file so it can be easily invoked from a command line. It should be saved as cdk-deploy-to.bat. The file cdk-deploy-to.ps1 will be created when the batch file is invoked.

====

We can then write additional scripts that use the cdk-deploy-to script to deploy to specific environments. The following is an example:

==== [role=“tablist”] macOS/Linux:: + [source,bash,subs=“verbatim,attributes”] — #!/usr/bin/env bash

= cdk-deploy-to-test.sh

## ./cdk-deploy-to.sh 123457689 us-east-1 “$@”

Windows:: + [source,bat,subs=“verbatim,attributes”] — @echo off rem cdk-deploy-to-test.bat cdk-deploy-to 135792469 us-east-1 %\* — ====

The following is an example that uses the cdk-deploy-to script to deploy to multiple environments. If the first deployment fails, the process stops:

==== [role=“tablist”] macOS/Linux:: + [source,bash,subs=“verbatim,attributes”] — #!/usr/bin/env bash

= cdk-deploy-to-prod.sh

./cdk-deploy-to.sh 135792468 us-west-1 “$@" || exit ./cdk-deploy-to.sh 246813579 eu-west-1 "$@” —

Windows:: + [source,bat,subs=“verbatim,attributes”] — @echo off rem cdk-deploy-to-prod.bat cdk-deploy-to 135792469 us-west-1 %\* || exit /B cdk-deploy-to 245813579 eu-west-1 %\* — ====

: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]

include::attributes.txt[]

// Attributes

[.topic] [#deploy-troubleshoot] = Troubleshoot {aws} CDK deployments :info\_titleabbrev: Troubleshoot CDK deployments :keywords: {aws} CDK, Deploy, Troubleshoot

== [abstract]

## Troubleshoot common issues when deploying {aws} Cloud Development Kit ({aws} CDK) applications.

// Content start

Troubleshoot common issues when deploying {aws} Cloud Development Kit ({aws} CDK) applications.

[#deploy-troubleshoot-sp] == Incorrect service principals are being created at deployment

When deploying CDK applications that contain {aws} Identity and Access Management (IAM) roles with service principals, you find that incorrect domains for the service principals are being created.

The following is a basic example of creating an IAM role that can be assumed by Amazon CloudWatch Logs using its service principal:

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

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

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

…. // Create an IAM role for CloudWatch Logs to assume const cloudWatchLogsRole = new iam.Role(this, ‘CloudWatchLogsRole’, { assumedBy: new iam.ServicePrincipal(‘logs.amazonaws.com’), // This is for CloudWatch Logs managedPolicies: [ iam.ManagedPolicy.fromAwsManagedPolicyName(‘service-role/AWSCloudWatchLogsFullAccess’) ] });

// You can then use this role in other constructs or configurations where CloudWatch Logs needs to assume a role } } —- ….

When you deploy this stack, a service principal named logs.amazonaws.com should be created. In most cases, {aws} services use the following naming for service principals: <service>.amazonaws.com.

[#deploy-troubleshoot-sp-causes] === Common causes

If you are using a version of the {aws} CDK older than v2.150.0, you may encounter this bug. In older {aws} CDK versions, the naming of service principals were not standardized, which could lead to the creation of service principals with incorrect domains.

In {aws} CDK v2.51.0, a fix was implemented by standardizing all automatically created service principals to use <service>.amazonaws.com when possible. This fix was available by allowing the @aws-cdk/aws-iam:standardizedServicePrincipals feature flag.

Starting in {aws} CDK v2.150.0, this became default behavior.

[#deploy-troubleshoot-sp-resolution] === Resolution

Upgrade to {aws} CDK v2.150.0 or newer.

If you are unable to upgrade to {aws} CDK v2.150.0 or newer, you must upgrade to at least v2.51.0 or newer. Then, allow the following feature flag in your cdk.json file: @aws-cdk/aws-iam:standardizedServicePrincipals.

:doctype: book

include::attributes.txt[]

// Attributes [.topic] :info\_titleabbrev: Permissions :keywords: {aws} CDK, Permissions, IAM

[#permissions] = Permissions and the {aws} CDK

== [abstract]

## The {aws} Construct Library uses a few common, widely implemented idioms to manage access and permissions. The IAM module provides you with the tools you need to use these idioms.

// Content start

The {aws} Construct Library uses a few common, widely implemented idioms to manage access and permissions. The IAM module provides you with the tools you need to use these idioms.

{aws} CDK uses {aws} CloudFormation to deploy changes. Every deployment involves an actor (either a developer, or an automated system) that starts a {aws} CloudFormation deployment. In the course of doing this, the actor will assume one or more IAM Identities (user or roles) and optionally pass a role to {aws} CloudFormation.

If you use {aws} IAM Identity Center to authenticate as a user, then the single sign-on provider supplies short-lived session credentials that authorize you to act as a pre-defined IAM role. To learn how the {aws} CDK obtains {aws} credentials from IAM Identity Center authentication, see https://docs.aws.amazon.com/sdkref/latest/guide/understanding-sso.html[Understand IAM Identity Center authentication] in the *{aws} SDKs and Tools Reference Guide*.

[#permissions-principals] == Principals

An IAM principal is an authenticated {aws} entity representing a user, service, or application that can call {aws} APIs. The {aws} Construct Library supports specifying principals in several flexible ways to grant them access your {aws} resources.

In security contexts, the term “principal” refers specifically to authenticated entities such as users. Objects like groups and roles do not *represent* users (and other authenticated entities) but rather *identify* them indirectly for the purpose of granting permissions.

For example, if you create an IAM group, you can grant the group (and thus its members) write access to an Amazon RDS table. However, the group itself is not a principal because it doesn’t represent a single entity (also, you cannot log in to a group).

In the CDK’s IAM library, classes that directly or indirectly identify principals implement the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.IPrincipal.html[IPrincipal] interface, allowing these objects to be used interchangeably in access policies. However, not all of them are principals in the security sense. These objects include:

. IAM resources such as https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html[Role], https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html[User], and https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html[Group] . Service principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ServicePrincipal.html[ServicePrincipal]('service.amazonaws.com')+) . Federated principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.FederatedPrincipal.html[FederatedPrincipal]('cognito-identity.amazonaws.com')+) . Account principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.AccountPrincipal.html[AccountPrincipal]('0123456789012')+) . Canonical user principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.CanonicalUserPrincipal.html[CanonicalUserPrincipal]('79a59d[...]7ef2be')+) . {aws} Organizations principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.OrganizationPrincipal.html[OrganizationPrincipal]('org-id')+) . Arbitrary ARN principals (+new iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ArnPrincipal.html[ArnPrincipal](res.arn)+) . An +iam.link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.CompositePrincipal.html[CompositePrincipal](principal1, principal2, ...)+ to trust multiple principals

[#permissions-grants] == Grants

Every construct that represents a resource that can be accessed, such as an Amazon S3 bucket or Amazon DynamoDB table, has methods that grant access to another entity. All such methods have names starting with *grant*.

For example, Amazon S3 buckets have the methods link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#grantwbrreadidentity-objectskeypattern[grantRead] and link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html#grantwbrreadwbrwriteidentity-objectskeypattern[grantReadWrite] (Python: grant\_read, grant\_read\_write) to enable read and read/write access, respectively, from an entity to the bucket. The entity doesn’t have to know exactly which Amazon S3 IAM permissions are required to perform these operations.

The first argument of a *grant* method is always of type link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.IGrantable.html[IGrantable]. This interface represents entities that can be granted permissions. That is, it represents resources with roles, such as the IAM objects link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html[Role], link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html[User], and link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html[Group].

Other entities can also be granted permissions. For example, later in this topic, we show how to grant a CodeBuild project access to an Amazon S3 bucket. Generally, the associated role is obtained via a role property on the entity being granted access.

Resources that use execution roles, such as link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda.Function.html[lambda.Function], also implement IGrantable, so you can grant them access directly instead of granting access to their role. For example, if bucket is an Amazon S3 bucket, and function is a Lambda function, the following code grants the function read access to the bucket.

==== [role=“tablist”] TypeScript:: + [source,none,subs=“verbatim,attributes”] — bucket.grantRead(function); —

JavaScript:: + [source,none,subs=“verbatim,attributes”] — bucket.grantRead(function); —

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

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

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

Sometimes permissions must be applied while your stack is being deployed. One such case is when you grant an {aws} CloudFormation custom resource access to some other resource. The custom resource will be invoked during deployment, so it must have the specified permissions at deployment time.

Another case is when a service verifies that the role you pass to it has the right policies applied. (A number of {aws} services do this to make sure that you didn’t forget to set the policies.) In those cases, the deployment might fail if the permissions are applied too late.

To force the grant’s permissions to be applied before another resource is created, you can add a dependency on the grant itself, as shown here. Though the return value of grant methods is commonly discarded, every grant method in fact returns an iam.Grant object.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const grant = bucket.grantRead(lambda); const custom = new CustomResource(…); custom.node.addDependency(grant); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const grant = bucket.grantRead(lambda); const custom = new CustomResource(…); custom.node.addDependency(grant); —

Python:: + [source,python,subs=“verbatim,attributes”] — grant = bucket.grant\_read(function) custom = CustomResource(…) custom.node.add\_dependency(grant) —

Java:: + [source,java,subs=“verbatim,attributes”] — Grant grant = bucket.grantRead(function); CustomResource custom = new CustomResource(…); custom.node.addDependency(grant); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var grant = bucket.GrantRead(function); var custom = new CustomResource(…); custom.node.AddDependency(grant); — ====

[#permissions-roles] == Roles

The IAM package contains a link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html[Role] construct that represents IAM roles. The following code creates a new role, trusting the Amazon EC2 service.

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

const role = new iam.Role(this, ‘Role’, { assumedBy: new iam.ServicePrincipal(‘ec2.amazonaws.com’), // required }); —

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

const role = new iam.Role(this, ‘Role’, { assumedBy: new iam.ServicePrincipal(‘ec2.amazonaws.com’) // required }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_iam as iam

role = iam.Role(self, “Role”, assumed\_by=iam.ServicePrincipal(“ec2.amazonaws.com”)) # required —

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

Role role = Role.Builder.create(this, “Role”) .assumedBy(new ServicePrincipal(“ec2.amazonaws.com”)).build(); —

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

var role = new Role(this, “Role”, new RoleProps { AssumedBy = new ServicePrincipal(“ec2.amazonaws.com”), // required }); — ====

You can add permissions to a role by calling the role’s link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#addwbrtowbrpolicystatement[addToPolicy] method (Python: add\_to\_policy), passing in a link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] that defines the rule to be added. The statement is added to the role’s default policy; if it has none, one is created.

The following example adds a Deny policy statement to the role for the actions ec2:SomeAction and s3:AnotherAction on the resources bucket and otherRole (Python: other\_role), under the condition that the authorized service is {aws} CodeBuild.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — role.addToPolicy(new iam.PolicyStatement({ effect: iam.Effect.DENY, resources: [bucket.bucketArn, otherRole.roleArn], actions: [‘ec2:SomeAction’, ‘s3:AnotherAction’], conditions: {StringEquals: { ‘ec2:AuthorizedService’: ‘codebuild.amazonaws.com’, }}})); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — role.addToPolicy(new iam.PolicyStatement({ effect: iam.Effect.DENY, resources: [bucket.bucketArn, otherRole.roleArn], actions: [‘ec2:SomeAction’, ‘s3:AnotherAction’], conditions: {StringEquals: { ‘ec2:AuthorizedService’: ‘codebuild.amazonaws.com’ }}})); —

Python:: + [source,python,subs=“verbatim,attributes”] — role.add\_to\_policy(iam.PolicyStatement( effect=iam.Effect.DENY, resources=[bucket.bucket\_arn, other\_role.role\_arn], actions=[“ec2:SomeAction”, “s3:AnotherAction”], conditions={“StringEquals”: { “ec2:AuthorizedService”: “codebuild.amazonaws.com”}} )) —

Java:: + [source,java,subs=“verbatim,attributes”] — role.addToPolicy(PolicyStatement.Builder.create() .effect(Effect.DENY) .resources(Arrays.asList(bucket.getBucketArn(), otherRole.getRoleArn())) .actions(Arrays.asList(“ec2:SomeAction”, “s3:AnotherAction”)) .conditions(java.util.Map.of( // Map.of requires Java 9 or later “StringEquals”, java.util.Map.of( “ec2:AuthorizedService”, “codebuild.amazonaws.com”))) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — role.AddToPolicy(new PolicyStatement(new PolicyStatementProps { Effect = Effect.DENY, Resources = new string[] { bucket.BucketArn, otherRole.RoleArn }, Actions = new string[] { “ec2:SomeAction”, “s3:AnotherAction” }, Conditions = new Dictionary<string, object> { [“StringEquals”] = new Dictionary<string, string> { [“ec2:AuthorizedService”] = “codebuild.amazonaws.com” } } })); — ====

In the preceding example, we’ve created a new link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] inline with the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#addwbrtowbrpolicystatement[addToPolicy] (Python: add\_to\_policy) call. You can also pass in an existing policy statement or one you’ve modified. The link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] object has link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html#methods[numerous methods] for adding principals, resources, conditions, and actions.

If you’re using a construct that requires a role to function correctly, you can do one of the following:

* Pass in an existing role when instantiating the construct object.
* Let the construct create a new role for you, trusting the appropriate service principal. The following example uses such a construct: a CodeBuild project.

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

// imagine roleOrUndefined is a function that might return a Role object // under some conditions, and undefined under other conditions const someRole: iam.IRole | undefined = roleOrUndefined();

const project = new codebuild.Project(this, ‘Project’, { // if someRole is undefined, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal role: someRole, }); —

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

// imagine roleOrUndefined is a function that might return a Role object // under some conditions, and undefined under other conditions const someRole = roleOrUndefined();

const project = new codebuild.Project(this, ‘Project’, { // if someRole is undefined, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal role: someRole }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_codebuild as codebuild

= imagine role\_or\_none is a function that might return a Role object

= under some conditions, and None under other conditions

some\_role = role\_or\_none();

project = codebuild.Project(self, “Project”,

= if role is None, the Project creates a new default role,

= trusting the codebuild.amazonaws.com service principal

## role=some\_role)

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.services.iam.Role; import software.amazon.awscdk.services.codebuild.Project;

// imagine roleOrNull is a function that might return a Role object // under some conditions, and null under other conditions Role someRole = roleOrNull();

// if someRole is null, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal Project project = Project.Builder.create(this, “Project”) .role(someRole).build(); —

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

// imagine roleOrNull is a function that might return a Role object // under some conditions, and null under other conditions var someRole = roleOrNull();

// if someRole is null, the Project creates a new default role, // trusting the codebuild.amazonaws.com service principal var project = new Project(this, “Project”, new ProjectProps { Role = someRole }); — ====

Once the object is created, the role (whether the role passed in or the default one created by the construct) is available as the property role. However, this property is not available on external resources. Therefore, these constructs have an addToRolePolicy (Python: add\_to\_role\_policy) method.

The method does nothing if the construct is an external resource, and it calls the addToPolicy (Python: add\_to\_policy) method of the role property otherwise. This saves you the trouble of handling the undefined case explicitly.

The following example demonstrates:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // project is imported into the CDK application const project = codebuild.Project.fromProjectName(this, ‘Project’, ‘ProjectName’);

// project is imported, so project.role is undefined, and this call has no effect project.addToRolePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW, // … and so on defining the policy })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // project is imported into the CDK application const project = codebuild.Project.fromProjectName(this, ‘Project’, ‘ProjectName’);

// project is imported, so project.role is undefined, and this call has no effect project.addToRolePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW // … and so on defining the policy })); —

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

= project is imported into the CDK application

project = codebuild.Project.from\_project\_name(self, ‘Project’, ‘ProjectName’)

= project is imported, so project.role is undefined, and this call has no effect

project.add\_to\_role\_policy(iam.PolicyStatement( effect=iam.Effect.ALLOW, # … and so on defining the policy ) ) —

Java:: + [source,java,subs=“verbatim,attributes”] — // project is imported into the CDK application Project project = Project.fromProjectName(this, “Project”, “ProjectName”);

// project is imported, so project.getRole() is null, and this call has no effect project.addToRolePolicy(PolicyStatement.Builder.create() .effect(Effect.ALLOW) // .. and so on defining the policy .build(); ) —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // project is imported into the CDK application var project = Project.FromProjectName(this, “Project”, “ProjectName”);

// project is imported, so project.role is null, and this call has no effect project.AddToRolePolicy(new PolicyStatement(new PolicyStatementProps { Effect = Effect.ALLOW, // … and so on defining the policy })); — ====

[#permissions-resource-policies] == Resource policies

A few resources in {aws}, such as Amazon S3 buckets and IAM roles, also have a resource policy. These constructs have an addToResourcePolicy method (Python: add\_to\_resource\_policy), which takes a link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.PolicyStatement.html[PolicyStatement] as its argument. Every policy statement added to a resource policy must specify at least one principal.

In the following example, the link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3.Bucket.html[Amazon S3 bucket] bucket grants a role with the s3:SomeAction permission to itself.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — bucket.addToResourcePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW, actions: [‘s3:SomeAction’], resources: [bucket.bucketArn], principals: [role] })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — bucket.addToResourcePolicy(new iam.PolicyStatement({ effect: iam.Effect.ALLOW, actions: [‘s3:SomeAction’], resources: [bucket.bucketArn], principals: [role] })); —

Python:: + [source,python,subs=“verbatim,attributes”] — bucket.add\_to\_resource\_policy(iam.PolicyStatement( effect=iam.Effect.ALLOW, actions=[“s3:SomeAction”], resources=[bucket.bucket\_arn], principals=role)) —

Java:: + [source,java,subs=“verbatim,attributes”] — bucket.addToResourcePolicy(PolicyStatement.Builder.create() .effect(Effect.ALLOW) .actions(Arrays.asList(“s3:SomeAction”)) .resources(Arrays.asList(bucket.getBucketArn())) .principals(Arrays.asList(role)) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — bucket.AddToResourcePolicy(new PolicyStatement(new PolicyStatementProps { Effect = Effect.ALLOW, Actions = new string[] { “s3:SomeAction” }, Resources = new string[] { bucket.BucketArn }, Principals = new IPrincipal[] { role } })); — ====

[#permissions-existing] == Using external IAM objects

If you have defined an IAM user, principal, group, or role outside your {aws} CDK app, you can use that IAM object in your {aws} CDK app. To do so, create a reference to it using its ARN or its name. (Use the name for users, groups, and roles.) The returned reference can then be used to grant permissions or to construct policy statements as explained previously.

* For users, call link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html#static-fromwbruserwbrarnscope-id-userarn[User.fromUserArn()] or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.User.html#static-fromwbruserwbrnamescope-id-username[User.fromUserName()]. User.fromUserAttributes() is also available, but currently provides the same functionality as User.fromUserArn().
* For principals, instantiate an link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ArnPrincipal.html[ArnPrincipal] object.
* For groups, call link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html#static-fromwbrgroupwbrarnscope-id-grouparn[Group.fromGroupArn()] or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Group.html#static-fromwbrgroupwbrnamescope-id-groupname[Group.fromGroupName()].
* For roles, call link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#static-fromwbrrolewbrarnscope-id-rolearn-options[Role.fromRoleArn()] or link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Role.html#static-fromwbrrolewbrnamescope-id-rolename[Role.fromRoleName()].

Policies (including managed policies) can be used in similar fashion using the following methods. You can use references to these objects anywhere an IAM policy is required.

* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.Policy.html#static-fromwbrpolicywbrnamescope-id-policyname[Policy.fromPolicyName]
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ManagedPolicy.html#static-fromwbrmanagedwbrpolicywbrarnscope-id-managedpolicyarn[ManagedPolicy.fromManagedPolicyArn]
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ManagedPolicy.html#static-fromwbrmanagedwbrpolicywbrnamescope-id-managedpolicyname[ManagedPolicy.fromManagedPolicyName]
* link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_iam.ManagedPolicy.html#static-fromwbrawswbrmanagedwbrpolicywbrnamemanagedpolicyname[ManagedPolicy.fromAwsManagedPolicyName]

= [NOTE]

As with all references to external {aws} resources, you cannot modify external IAM objects in your CDK app.

====

:doctype: book

include::attributes.txt[]

// Attributes [.topic] [[usage-data,usage-data.title]] = Configure {aws} CDK usage data reporting :info\_titleabbrev: Configure usage data reporting :keywords: {aws} CDK, Reporting, Usage data, Version reporting

== [abstract]

## The {aws} Cloud Development Kit ({aws} CDK) collects usage data from your CDK applications to gain insight into how the {aws} CDK is being used. The CDK team uses this data to do the following:

// Content start

[#usage-data-intro] == What is CDK usage data reporting?

{aws} Cloud Development Kit ({aws} CDK) applications are configured to collect and report on usage data to gain insight into how the {aws} CDK is being used. The CDK team uses this data to do the following:

* *Communicate with customers* – Identify stacks using a construct with known security or reliability issues and send out communication on customer-impacting topics.
* *Inform CDK development* – Gain insight on CDK usage to better inform CDK development.
* *Investigate CDK issues* – When bugs are reported, usage data provides valuable insight to the CDK team when troubleshooting.

[#usage-data-categories] == What usage data is collected?

There are two categories of usage data collected by the CDK:

* General usage data
* Additional usage data

[#usage-data-categories-general] === General usage data collection

The CDK collects the following types of general usage data from your CDK applications:

* Versions of CDK libraries used.
* Names of constructs used from the following NPM modules:

|  |
| --- |
| \*\* {aws} CDK core modules |
| \*\* {aws} Construct Library modules |
| \*\* {aws} Solutions Constructs module |
| \*\* {aws} Render Farm Deployment Kit module |

= [NOTE]

Before version 1.93.0, the {aws} CDK reported the names and versions of the modules loaded during synthesis, instead of the constructs used in the stack.

====

[#usage-data-categories-additional] === Additional usage data collection

Starting with CDK version 2.178.0, usage data collection expanded to include the following additional usage data:

* *CDK–defined property keys* – When you use a built-in property of an L2 construct, the property key will be collected. This includes built-in property keys nested in dictionary objects.
* *Boolean and enum property values from CDK–defined property keys* – For CDK–defined property keys, property values of only Boolean and enum types will be collected. All other types, such as string values or construct references will be redacted.
* *Method name, keys, and property values of Boolean and enum types* – When you use an L2 construct method, we will collect the method name, property keys, and property values of Boolean and enum types.

For property keys and values that you uniquely create, the entire object will be redacted. For example, if you use +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_apigateway.InlineApiDefinition.html[InlineApiDefinition]+ to define an [.noloc]OpenAPI specification and pass it into a +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_apigateway.RestApi.html[RestApi]+ construct, the entire InlineApiDefinition object will be redacted.

For more information on additional usage data collection, including its benefits and potential concerns, see the https://github.com/aws/aws-cdk/discussions/33198[CDK Collecting Additional Metadata (under feature flag) #33198] discussion in the *aws-cdk repository*.

[#usage-data-how] == How the CDK collects usage data

At synthesis, the CDK collects usage data from your application and stores it within the +{aws}::CDK::Metadata+ resource. The following is an example of this on a synthesized {aws} CloudFormation template:

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

CDKMetadata: Type: “{aws}::CDK::Metadata” Properties: Analytics: “v2:deflate64:H4sIAND9SGAAAzXKSw5AMBAA0L1b2PdzBYnEAdio3RglglY60zQi7u6TWL/XKmNUlxeQSOKwaPTBqrNhwEWU3hGHiCzK0dWWfAxoL/Fd8mvk+QkS/0X6BdjnCdgmOOQKWz+AqqLDt2Y3YMnLYWwAAAA=” —

The Analytics property is a gzipped, base64-encoded, prefix-encoded list of the constructs in the stack.

[#usage-data-configure] == How to opt out or opt in to usage data reporting

Your options to opt out or opt in to general usage data reporting and additional usage data reporting depends on the CDK version you used to originally create your app.

By default, CDK applications are configured to automatically opt in to usage data reporting as follows:

* *All CDK apps* – Opted in to general usage data reporting.
* *CDK apps created with a version older than v2.178.0* – Not automatically opted in to additional usage data reporting.
* *CDK apps created with v2.178.0 or newer* – Opted in to additional usage data reporting.

= [WARNING]

If you choose to opt out, the CDK will not be able to identify if you`’ve been impacted by security issues and will not send you notifications for them.

====

[#usage-data-configure-optout-all] === Opt out of all usage data reporting

To opt out of all usage data reporting, you can use the {aws} Cloud Development Kit ({aws} CDK) Command Line Interface (CLI) or configure your project’s cdk.json file.

*To opt out of all usage data reporting using the CDK CLI*:: +

* Use the --no-version-reporting option with any CDK CLI command to opt out for a single command. The following is an example of opting out during template synthesis:

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

## $ cdk synth –no-version-reporting

* Since the CDK synthesizes templates automatically when you run cdk deploy, you should also use --no-version-reporting with the cdk deploy command.

*To opt out of all usage data reporting by configuring the cdk.json file*:: + . Set versionReporting to false in ./cdk.json or ~/.cdk.json. This opts you out by default. The following is an example: + [source,json,subs=“verbatim,attributes”] — { “app”: “…”, “versionReporting”: false } — + . After configuring, you can override this behavior and opt in by specifying --version-reporting on an individual command.

[#usage-data-configure-optout-additiona] === Opt out of only the additional usage data reporting

If your CDK app was created with a CDK version older than 2.178.0, you are automatically opted out of additional usage data reporting, even if you are opted in to general usage data reporting. You don`’t have to do anything to opt out of additional usage data reporting.

If your CDK app was created with CDK version 2.178.0 or newer, you must opt out of all usage data reporting. You can`’t opt out of only the additional usage data reporting.

[#usage-data-configure-optin] === Opt in to usage data reporting

If your CDK app was created with CDK version 2.178.0 or newer, you can opt in to all usage data reporting by setting versionReporting to true. This is the default behavior of CDK apps.

If your CDK app was created with a CDK version older than 2.178.0, you can opt in to general usage data reporting by setting versionReporting to true. To opt in to additional usage data reporting, you must enable a feature flag.

NOTE: These steps are for CDK apps originally created with a version older than 2.178.0

. Verify that you are now using CDK version 2.178.0 or newer. . In your CDK configuration file, specify @aws-cdk/core:enableAdditionalMetadataCollection as true. The following is an example: + [source,json,subs=“verbatim,attributes”] — { “context”: { “@aws-cdk/core:enableAdditionalMetadataCollection”: “true” } } — + . You can also use the ENABLE\_ADDITIONAL\_METADATA\_COLLECTION value with the +https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.FeatureFlags.html[FeatureFlags]+ class. The following is an example: + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘aws-cdk-lib’; import { FeatureFlags } from ‘aws-cdk-lib’; import \* as cx\_api from ‘aws-cdk-lib/cx-api’; import { Construct } from ‘constructs’;

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

…. // Set the feature flag ENABLE\_ADDITIONAL\_METADATA\_COLLECTION to true FeatureFlags.of(this).add(cx\_api.ENABLE\_ADDITIONAL\_METADATA\_COLLECTION, true);

// Your stack resources go here new cdk.aws\_s3.Bucket(this, ‘MyBucket’); } } ….

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

[#usage-data-examples] == Examples

[#usage-data-examples-example1] === General and additional usage data collected from a CDK application

The following is an example of a CDK app:

== [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’;

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

…. // Create an S3 bucket (L2 construct) const myBucket = new s3.Bucket(this, ‘MyBucket’, { bucketName: ‘my-cdk-example-bucket’, // String type versioned: true, // Boolean type removalPolicy: cdk.RemovalPolicy.DESTROY, // ENUM type lifecycleRules: [{ // Array of object type expirationDate: new Date(‘2019-10-01’), objectSizeLessThan: 600, objectSizeGreaterThan: 500, }], });

// Use a method of the L2 construct to define additional properties myBucket.addLifecycleRule({ id: ‘ExpireOldObjects’, enabled: true, // Boolean expiration: cdk.Duration.days(90), // Expire objects after 90 days }); } } ….

// Define the CDK app and stack const app = new cdk.App(); new MyStack(app, ‘MyStack’); app.synth(); —

At synthesis, usage data is collected, compressed, and stored in the +{aws}::CDK::Metadata+ resource.

The following is an example of general usage data collected with a CDK version older than 2.178.0:

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

{ “fqn”: “aws-cdk-lib.aws-s3.Bucket”, “version”: “v2.170.0” } —

The following is an example of usage data collected, including the additional usage data introduced in CDK version 2.178.0:

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

{ “fqn”: “aws-cdk-lib.aws\_s3.Bucket”, “version”: “2.170.0”, “metadata”: [ { “type”: “aws:cdk:analytics:construct”, “data”: { “bucketName”: “*”, ”versioned”: true, ”removalPolicy”: ”cdk.RemovalPolicy.DESTROY”, ”lifecycleRules”: [ { ”expirationDate”: ””, ”objectSizeLessThan”: ””, ”objectSizeGreaterThan”: ”\_” } ] } }, { ”type”: ”aws:cdk:analytics:method”, ”data”: { ”name”: ”addLifecycleRule”, ”prop”: { ”id”: ”*”, “enabled”: true, “expiration”: “\_“, } } } ] } —