:doctype: book

include::attributes.txt[]

// Attributes [.topic] :info\_titleabbrev: Assets :keywords: {aws} CDK, Assets, Amazon S3, Amazon ECR

[#assets] = Assets and the {aws} CDK

== [abstract]

## Assets are local files, directories, or Docker images.

// Content start

Assets are local files, directories, or Docker images that can be bundled into {aws} CDK libraries and apps. For example, an asset might be a directory that contains the handler code for an {aws} Lambda function. Assets can represent any artifact that the app needs to operate.

The following tutorial video provides a comprehensive overview of CDK assets, and explains how you can use them in your infrastructure as code (IaC).

video::jHNtXQmkKfw[youtube,align = center,height = 390,fileref = https://www.youtube.com/embed/jHNtXQmkKfw,width = 480]

You add assets through APIs that are exposed by specific {aws} constructs. For example, when you define a https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda.Function.html[lambda.Function] construct, the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda.Function.html#code[code] property lets you pass an https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda.Code.html#static-fromwbrassetpath-options[asset] (directory). Function uses assets to bundle the contents of the directory and use it for the function’s code. Similarly, https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecs.ContainerImage.html#static-fromwbrassetdirectory-props[ecs.ContainerImage.fromAsset] uses a Docker image built from a local directory when defining an Amazon ECS task definition.

[#assets-details] == Assets in detail

When you refer to an asset in your app, the xref:deploy-how-synth-assemblies[cloud assembly] that’s synthesized from your application includes metadata information with instructions for the {aws} CDK CLI. The instructions include where to find the asset on the local disk and what type of bundling to perform based on the asset type, such as a directory to compress (zip) or a Docker image to build.

The {aws} CDK generates a source hash for assets. This can be used at construction time to determine whether the contents of an asset have changed.

By default, the {aws} CDK creates a copy of the asset in the cloud assembly directory, which defaults to cdk.out, under the source hash. This way, the cloud assembly is self-contained, so if it moved over to a different host for deployment, it can still be deployed. See xref:deploy-how-synth-assemblies[Cloud assemblies] for details.

When the {aws} CDK deploys an app that references assets (either directly by the app code or through a library), the {aws} CDK CLI first prepares and publishes the assets to an Amazon S3 bucket or Amazon ECR repository. (The S3 bucket or repository is created during bootstrapping.) Only then are the resources defined in the stack deployed.

This section describes the low-level APIs available in the framework.

[#assets-types] == Asset types

The {aws} CDK supports the following types of assets:

Amazon S3 assets:: + These are local files and directories that the {aws} CDK uploads to Amazon S3.

Docker Image:: + These are Docker images that the {aws} CDK uploads to Amazon ECR.

These asset types are explained in the following sections.

[#assets-types-s3] == Amazon S3 assets

You can define local files and directories as assets, and the {aws} CDK packages and uploads them to Amazon S3 through the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3\_assets-readme.html[aws-s3-assets] module.

The following example defines a local directory asset and a file asset.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import { Asset } from ‘aws-cdk-lib/aws-s3-assets’;

// Archived and uploaded to Amazon S3 as a .zip file const directoryAsset = new Asset(this, “SampleZippedDirAsset”, { path: path.join(\_\_dirname, “sample-asset-directory”) });

// Uploaded to Amazon S3 as-is const fileAsset = new Asset(this, ‘SampleSingleFileAsset’, { path: path.join(\_\_dirname, ‘file-asset.txt’) }); —

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

// Archived and uploaded to Amazon S3 as a .zip file const directoryAsset = new Asset(this, “SampleZippedDirAsset”, { path: path.join(\_\_dirname, “sample-asset-directory”) });

// Uploaded to Amazon S3 as-is const fileAsset = new Asset(this, ‘SampleSingleFileAsset’, { path: path.join(\_\_dirname, ‘file-asset.txt’) }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import os.path dirname = os.path.dirname(*file*)

from aws\_cdk.aws\_s3\_assets import Asset

= Archived and uploaded to Amazon S3 as a .zip file

directory\_asset = Asset(self, “SampleZippedDirAsset”, path=os.path.join(dirname, “sample-asset-directory”) )

= Uploaded to Amazon S3 as-is

file\_asset = Asset(self, ‘SampleSingleFileAsset’, path=os.path.join(dirname, ‘file-asset.txt’) ) —

Java:: + [source,java,subs=“verbatim,attributes”] — import java.io.File;

import software.amazon.awscdk.services.s3.assets.Asset;

// Directory where app was started File startDir = new File(System.getProperty(“user.dir”));

// Archived and uploaded to Amazon S3 as a .zip file Asset directoryAsset = Asset.Builder.create(this, “SampleZippedDirAsset”) .path(new File(startDir, “sample-asset-directory”).toString()).build();

// Uploaded to Amazon S3 as-is Asset fileAsset = Asset.Builder.create(this, “SampleSingleFileAsset”) .path(new File(startDir, “file-asset.txt”).toString()).build(); —

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

// Archived and uploaded to Amazon S3 as a .zip file var directoryAsset = new Asset(this, “SampleZippedDirAsset”, new AssetProps { Path = Path.Combine(Directory.GetCurrentDirectory(), “sample-asset-directory”) });

// Uploaded to Amazon S3 as-is var fileAsset = new Asset(this, “SampleSingleFileAsset”, new AssetProps { Path = Path.Combine(Directory.GetCurrentDirectory(), “file-asset.txt”) }); —

Go:: + [source,go,subs=“verbatim,attributes”] — dirName, err := os.Getwd() if err != nil { panic(err) }

awss3assets.NewAsset(stack, jsii.String(“SampleZippedDirAsset”), &awss3assets.AssetProps{ Path: jsii.String(path.Join(dirName, “sample-asset-directory”)), })

awss3assets.NewAsset(stack, jsii.String(“SampleSingleFileAsset”), &awss3assets.AssetProps{ Path: jsii.String(path.Join(dirName, “file-asset.txt”)), }) — ====

In most cases, you don’t need to directly use the APIs in the aws-s3-assets module. Modules that support assets, such as aws-lambda, have convenience methods so that you can use assets. For Lambda functions, the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_lambda.Code.html#static-fromwbrassetpath-options[fromAsset()] static method enables you to specify a directory or a .zip file in the local file system.

[#assets-types-s3-lambda] === Lambda function example

A common use case is creating Lambda functions with the handler code as an Amazon S3 asset.

The following example uses an Amazon S3 asset to define a Python handler in the local directory handler. It also creates a Lambda function with the local directory asset as the code property. Following is the Python code for the handler.

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

def lambda\_handler(event, context): message = ‘Hello World!’ return { ‘message’: message } —

The code for the main {aws} CDK app should look like the following.

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

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

new lambda.Function(this, ‘myLambdaFunction’, { code: lambda.Code.fromAsset(path.join(\_\_dirname, ‘handler’)), runtime: lambda.Runtime.PYTHON\_3\_6, handler: ‘index.lambda\_handler’ }); } } —-

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

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

new lambda.Function(this, ‘myLambdaFunction’, { code: lambda.Code.fromAsset(path.join(\_\_dirname, ‘handler’)), runtime: lambda.Runtime.PYTHON\_3\_6, handler: ‘index.lambda\_handler’ }); } }

== module.exports = { HelloAssetStack }

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

import os.path dirname = os.path.dirname(*file*)

class HelloAssetStack(Stack): def *init*(self, scope: Construct, id: str, **kwargs): super().*init*(scope, id,** kwargs)

lambda\_.Function(self, 'myLambdaFunction',  
 code=lambda\_.Code.from\_asset(os.path.join(dirname, 'handler')),  
 runtime=lambda\_.Runtime.PYTHON\_3\_6,  
 handler="index.lambda\_handler") ----

Java:: + [source,java,subs=“verbatim,attributes”] — import java.io.File;

import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.services.lambda.Function; import software.amazon.awscdk.services.lambda.Runtime;

public class HelloAssetStack extends Stack {

…. public HelloAssetStack(final App scope, final String id) { this(scope, id, null); }

public HelloAssetStack(final App scope, final String id, final StackProps props) { super(scope, id, props);

File startDir = new File(System.getProperty("user.dir"));  
  
Function.Builder.create(this, "myLambdaFunction")  
 .code(Code.fromAsset(new File(startDir, "handler").toString()))  
 .runtime(Runtime.PYTHON\_3\_6)  
 .handler("index.lambda\_handler").build();

} } —- ….

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

public class HelloAssetStack : Stack { public HelloAssetStack(Construct scope, string id, StackProps props) : base(scope, id, props) { new Function(this, “myLambdaFunction”, new FunctionProps { Code = Code.FromAsset(Path.Combine(Directory.GetCurrentDirectory(), “handler”)), Runtime = Runtime.PYTHON\_3\_6, Handler = “index.lambda\_handler” }); } } —

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

“github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awslambda” “github.com/aws/aws-cdk-go/awscdk/v2/awss3assets” “github.com/aws/constructs-go/constructs/v10” “github.com/aws/jsii-runtime-go” )

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

dirName, err := os.Getwd() if err != nil { panic(err) }

awslambda.NewFunction(stack, jsii.String(“myLambdaFunction”), &awslambda.FunctionProps{ Code: awslambda.AssetCode\_FromAsset(jsii.String(path.Join(dirName, “handler”)), &awss3assets.AssetOptions{}), Runtime: awslambda.Runtime\_PYTHON\_3\_6(), Handler: jsii.String(“index.lambda\_handler”), })

return stack } — ====

The Function method uses assets to bundle the contents of the directory and use it for the function’s code.

= [TIP]

Java .jar files are ZIP files with a different extension. These are uploaded as-is to Amazon S3, but when deployed as a Lambda function, the files they contain are extracted, which you might not want. To avoid this, place the .jar file in a directory and specify that directory as the asset.

====

[#assets-types-s3-deploy] === Deploy-time attributes example

Amazon S3 asset types also expose xref:resources-attributes[deploy-time attributes] that can be referenced in {aws} CDK libraries and apps. The {aws} CDK CLI command cdk synth displays asset properties as {aws} CloudFormation parameters.

The following example uses deploy-time attributes to pass the location of an image asset into a Lambda function as environment variables. (The kind of file doesn’t matter; the PNG image used here is only an example.)

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import { Asset } from ‘aws-cdk-lib/aws-s3-assets’; import \* as path from ‘path’;

const imageAsset = new Asset(this, “SampleAsset”, { path: path.join(\_\_dirname, “images/my-image.png”) });

new lambda.Function(this, “myLambdaFunction”, { code: lambda.Code.asset(path.join(\_\_dirname, “handler”)), runtime: lambda.Runtime.PYTHON\_3\_6, handler: “index.lambda\_handler”, environment: { ‘S3\_BUCKET\_NAME’: imageAsset.s3BucketName, ‘S3\_OBJECT\_KEY’: imageAsset.s3ObjectKey, ‘S3\_OBJECT\_URL’: imageAsset.s3ObjectUrl } }); —

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

const imageAsset = new Asset(this, “SampleAsset”, { path: path.join(\_\_dirname, “images/my-image.png”) });

new lambda.Function(this, “myLambdaFunction”, { code: lambda.Code.asset(path.join(\_\_dirname, “handler”)), runtime: lambda.Runtime.PYTHON\_3\_6, handler: “index.lambda\_handler”, environment: { ‘S3\_BUCKET\_NAME’: imageAsset.s3BucketName, ‘S3\_OBJECT\_KEY’: imageAsset.s3ObjectKey, ‘S3\_OBJECT\_URL’: imageAsset.s3ObjectUrl } }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import os.path

import aws\_cdk.aws\_lambda as lambda\_ from aws\_cdk.aws\_s3\_assets import Asset

dirname = os.path.dirname(*file*)

image\_asset = Asset(self, “SampleAsset”, path=os.path.join(dirname, “images/my-image.png”))

lambda\_.Function(self, “myLambdaFunction”, code=lambda\_.Code.asset(os.path.join(dirname, “handler”)), runtime=lambda\_.Runtime.PYTHON\_3\_6, handler=“index.lambda\_handler”, environment=dict( S3\_BUCKET\_NAME=image\_asset.s3\_bucket\_name, S3\_OBJECT\_KEY=image\_asset.s3\_object\_key, S3\_OBJECT\_URL=image\_asset.s3\_object\_url)) —

Java:: + [source,java,subs=“verbatim,attributes”] — import java.io.File;

import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.services.lambda.Function; import software.amazon.awscdk.services.lambda.Runtime; import software.amazon.awscdk.services.s3.assets.Asset;

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

…. File startDir = new File(System.getProperty(“user.dir”));

Asset imageAsset = Asset.Builder.create(this, "SampleAsset")  
 .path(new File(startDir, "images/my-image.png").toString()).build())  
  
Function.Builder.create(this, "myLambdaFunction")  
 .code(Code.fromAsset(new File(startDir, "handler").toString()))  
 .runtime(Runtime.PYTHON\_3\_6)  
 .handler("index.lambda\_handler")  
 .environment(java.util.Map.of( // Java 9 or later  
 "S3\_BUCKET\_NAME", imageAsset.getS3BucketName(),  
 "S3\_OBJECT\_KEY", imageAsset.getS3ObjectKey(),  
 "S3\_OBJECT\_URL", imageAsset.getS3ObjectUrl()))  
 .build();

} } —- ….

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

var imageAsset = new Asset(this, “SampleAsset”, new AssetProps { Path = Path.Combine(Directory.GetCurrentDirectory(), @“images-image.png”) });

new Function(this, “myLambdaFunction”, new FunctionProps { Code = Code.FromAsset(Path.Combine(Directory.GetCurrentDirectory(), “handler”)), Runtime = Runtime.PYTHON\_3\_6, Handler = “index.lambda\_handler”, Environment = new Dictionary<string, string> { [“S3\_BUCKET\_NAME”] = imageAsset.S3BucketName, [“S3\_OBJECT\_KEY”] = imageAsset.S3ObjectKey, [“S3\_OBJECT\_URL”] = imageAsset.S3ObjectUrl } }); —

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

“github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awslambda” “github.com/aws/aws-cdk-go/awscdk/v2/awss3assets” )

dirName, err := os.Getwd() if err != nil { panic(err) }

imageAsset := awss3assets.NewAsset(stack, jsii.String(“SampleAsset”), &awss3assets.AssetProps{ Path: jsii.String(path.Join(dirName, “images/my-image.png”)), })

awslambda.NewFunction(stack, jsii.String(“myLambdaFunction”), &awslambda.FunctionProps{ Code: awslambda.AssetCode\_FromAsset(jsii.String(path.Join(dirName, “handler”))), Runtime: awslambda.Runtime\_PYTHON\_3\_6(), Handler: jsii.String(“index.lambda\_handler”), Environment: &map[string]\*string{ “S3\_BUCKET\_NAME”: imageAsset.S3BucketName(), “S3\_OBJECT\_KEY”: imageAsset.S3ObjectKey(), “S3\_URL”: imageAsset.S3ObjectUrl(), }, }) — ====

[#assets-types-s3-permissions] === Permissions

If you use Amazon S3 assets directly through the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3\_assets-readme.html[aws-s3-assets] module, IAM roles, users, or groups, and you need to read assets in runtime, then grant those assets IAM permissions through the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_s3\_assets.Asset.html#grantwbrreadgrantee[asset.grantRead] method.

The following example grants an IAM group read permissions on a file asset.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import { Asset } from ‘aws-cdk-lib/aws-s3-assets’; import \* as path from ‘path’;

const asset = new Asset(this, ‘MyFile’, { path: path.join(\_\_dirname, ‘my-image.png’) });

const group = new iam.Group(this, ‘MyUserGroup’); asset.grantRead(group); —

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

const asset = new Asset(this, ‘MyFile’, { path: path.join(\_\_dirname, ‘my-image.png’) });

const group = new iam.Group(this, ‘MyUserGroup’); asset.grantRead(group); —

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk.aws\_s3\_assets import Asset import aws\_cdk.aws\_iam as iam

import os.path dirname = os.path.dirname(*file*)

…. asset = Asset(self, “MyFile”, path=os.path.join(dirname, “my-image.png”))

group = iam.Group(self, "MyUserGroup")  
asset.grant\_read(group) ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — import java.io.File;

import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.services.iam.Group; import software.amazon.awscdk.services.s3.assets.Asset;

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

…. File startDir = new File(System.getProperty(“user.dir”));

Asset asset = Asset.Builder.create(this, "SampleAsset")  
 .path(new File(startDir, "images/my-image.png").toString()).build();  
  
Group group = new Group(this, "MyUserGroup");  
asset.grantRead(group); } } ----

….

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

var asset = new Asset(this, “MyFile”, new AssetProps { Path = Path.Combine(Path.Combine(Directory.GetCurrentDirectory(), @“images-image.png”)) });

var group = new Group(this, “MyUserGroup”); asset.GrantRead(group); —

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

“github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awsiam” “github.com/aws/aws-cdk-go/awscdk/v2/awss3assets” )

dirName, err := os.Getwd() if err != nil { panic(err) }

asset := awss3assets.NewAsset(stack, jsii.String(“MyFile”), &awss3assets.AssetProps{ Path: jsii.String(path.Join(dirName, “my-image.png”)), })

group := awsiam.NewGroup(stack, jsii.String(“MyUserGroup”), &awsiam.GroupProps{})

== asset.GrantRead(group)

====

[#assets-types-docker] == Docker image assets

The {aws} CDK supports bundling local Docker images as assets through the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecr\_assets-readme.html[aws-ecr-assets] module.

The following example defines a Docker image that is built locally and pushed to Amazon ECR. Images are built from a local Docker context directory (with a Dockerfile) and uploaded to Amazon ECR by the {aws} CDK CLI or your app’s CI/CD pipeline. The images can be naturally referenced in your {aws} CDK app.

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

const asset = new DockerImageAsset(this, ‘MyBuildImage’, { directory: path.join(\_\_dirname, ‘my-image’) }); —

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

const asset = new DockerImageAsset(this, ‘MyBuildImage’, { directory: path.join(\_\_dirname, ‘my-image’) }); —

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk.aws\_ecr\_assets import DockerImageAsset

import os.path dirname = os.path.dirname(*file*)

asset = DockerImageAsset(self, ‘MyBuildImage’, directory=os.path.join(dirname, ‘my-image’)) —

Java:: + [source,java,subs=“verbatim,attributes”] — import software.amazon.awscdk.services.ecr.assets.DockerImageAsset;

File startDir = new File(System.getProperty(“user.dir”));

DockerImageAsset asset = DockerImageAsset.Builder.create(this, “MyBuildImage”) .directory(new File(startDir, “my-image”).toString()).build(); —

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

var asset = new DockerImageAsset(this, “MyBuildImage”, new DockerImageAssetProps { Directory = Path.Combine(Directory.GetCurrentDirectory(), “my-image”) }); —

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

“github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awsecrassets” )

dirName, err := os.Getwd() if err != nil { panic(err) }

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String(“MyBuildImage”), &awsecrassets.DockerImageAssetProps{ Directory: jsii.String(path.Join(dirName, “my-image”)), }) — ====

The my-image directory must include a Dockerfile. The {aws} CDK CLI builds a Docker image from my-image, pushes it to an Amazon ECR repository, and specifies the name of the repository as an {aws} CloudFormation parameter to your stack. Docker image asset types expose xref:resources-attributes[deploy-time attributes] that can be referenced in {aws} CDK libraries and apps. The {aws} CDK CLI command cdk synth displays asset properties as {aws} CloudFormation parameters.

[#assets-types-docker-ecs] === Amazon ECS task definition example

A common use case is to create an Amazon ECS https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecs.TaskDefinition.html[TaskDefinition] to run Docker containers. The following example specifies the location of a Docker image asset that the {aws} CDK builds locally and pushes to Amazon ECR.

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

const taskDefinition = new ecs.FargateTaskDefinition(this, “TaskDef”, { memoryLimitMiB: 1024, cpu: 512 });

const asset = new ecr\_assets.DockerImageAsset(this, ‘MyBuildImage’, { directory: path.join(\_\_dirname, ‘my-image’) });

taskDefinition.addContainer(“my-other-container”, { image: ecs.ContainerImage.fromDockerImageAsset(asset) }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const ecs = require(‘aws-cdk-lib/aws-ecs’); const ecr\_assets = require(‘aws-cdk-lib/aws-ecr-assets’); const path = require(‘path’);

const taskDefinition = new ecs.FargateTaskDefinition(this, “TaskDef”, { memoryLimitMiB: 1024, cpu: 512 });

const asset = new ecr\_assets.DockerImageAsset(this, ‘MyBuildImage’, { directory: path.join(\_\_dirname, ‘my-image’) });

taskDefinition.addContainer(“my-other-container”, { image: ecs.ContainerImage.fromDockerImageAsset(asset) }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_ecs as ecs import aws\_cdk.aws\_ecr\_assets as ecr\_assets

import os.path dirname = os.path.dirname(*file*)

task\_definition = ecs.FargateTaskDefinition(self, “TaskDef”, memory\_limit\_mib=1024, cpu=512)

asset = ecr\_assets.DockerImageAsset(self, ‘MyBuildImage’, directory=os.path.join(dirname, ‘my-image’))

task\_definition.add\_container(“my-other-container”, image=ecs.ContainerImage.from\_docker\_image\_asset(asset)) —

Java:: + [source,java,subs=“verbatim,attributes”] — import java.io.File;

import software.amazon.awscdk.services.ecs.FargateTaskDefinition; import software.amazon.awscdk.services.ecs.ContainerDefinitionOptions; import software.amazon.awscdk.services.ecs.ContainerImage;

import software.amazon.awscdk.services.ecr.assets.DockerImageAsset;

File startDir = new File(System.getProperty(“user.dir”));

FargateTaskDefinition taskDefinition = FargateTaskDefinition.Builder.create( this, “TaskDef”).memoryLimitMiB(1024).cpu(512).build();

DockerImageAsset asset = DockerImageAsset.Builder.create(this, “MyBuildImage”) .directory(new File(startDir, “my-image”).toString()).build();

taskDefinition.addContainer(“my-other-container”, ContainerDefinitionOptions.builder() .image(ContainerImage.fromDockerImageAsset(asset)) .build(); ) —

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

var taskDefinition = new FargateTaskDefinition(this, “TaskDef”, new FargateTaskDefinitionProps { MemoryLimitMiB = 1024, Cpu = 512 });

var asset = new DockerImageAsset(this, “MyBuildImage”, new DockerImageAssetProps { Directory = Path.Combine(Directory.GetCurrentDirectory(), “my-image”) });

taskDefinition.AddContainer(“my-other-container”, new ContainerDefinitionOptions { Image = ContainerImage.FromDockerImageAsset(asset) }); —

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

“github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awsecrassets” “github.com/aws/aws-cdk-go/awscdk/v2/awsecs” )

dirName, err := os.Getwd() if err != nil { panic(err) }

taskDefinition := awsecs.NewTaskDefinition(stack, jsii.String(“TaskDef”), &awsecs.TaskDefinitionProps{ MemoryMiB: jsii.String(“1024”), Cpu: jsii.String(“512”), })

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String(“MyBuildImage”), &awsecrassets.DockerImageAssetProps{ Directory: jsii.String(path.Join(dirName, “my-image”)), })

taskDefinition.AddContainer(jsii.String(“MyOtherContainer”), &awsecs.ContainerDefinitionOptions{ Image: awsecs.ContainerImage\_FromDockerImageAsset(asset), }) — ====

[#assets-types-docker-deploy] === Deploy-time attributes example

The following example shows how to use the deploy-time attributes repository and imageUri to create an Amazon ECS task definition with the {aws} Fargate launch type. Note that the Amazon ECR repo lookup requires the image’s tag, not its URI, so we snip it from the end of the asset’s URI.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — import \* as ecs from ‘aws-cdk-lib/aws-ecs’; import \* as path from ‘path’; import { DockerImageAsset } from ‘aws-cdk-lib/aws-ecr-assets’;

const asset = new DockerImageAsset(this, ‘my-image’, { directory: path.join(\_\_dirname, “..”, “demo-image”) });

const taskDefinition = new ecs.FargateTaskDefinition(this, “TaskDef”, { memoryLimitMiB: 1024, cpu: 512 });

taskDefinition.addContainer(“my-other-container”, { image: ecs.ContainerImage.fromEcrRepository(asset.repository, asset.imageUri.split(“:”).pop()) }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const ecs = require(‘aws-cdk-lib/aws-ecs’); const path = require(‘path’); const { DockerImageAsset } = require(‘aws-cdk-lib/aws-ecr-assets’);

const asset = new DockerImageAsset(this, ‘my-image’, { directory: path.join(\_\_dirname, “..”, “demo-image”) });

const taskDefinition = new ecs.FargateTaskDefinition(this, “TaskDef”, { memoryLimitMiB: 1024, cpu: 512 });

taskDefinition.addContainer(“my-other-container”, { image: ecs.ContainerImage.fromEcrRepository(asset.repository, asset.imageUri.split(“:”).pop()) }); —

Python:: + [source,python,subs=“verbatim,attributes”] — import aws\_cdk.aws\_ecs as ecs from aws\_cdk.aws\_ecr\_assets import DockerImageAsset

import os.path dirname = os.path.dirname(*file*)

asset = DockerImageAsset(self, ‘my-image’, directory=os.path.join(dirname, “..”, “demo-image”))

task\_definition = ecs.FargateTaskDefinition(self, “TaskDef”, memory\_limit\_mib=1024, cpu=512)

task\_definition.add\_container(“my-other-container”, image=ecs.ContainerImage.from\_ecr\_repository( asset.repository, asset.image\_uri.rpartition(“:”)[-1])) —

Java:: + [source,java,subs=“verbatim,attributes”] — import java.io.File;

import software.amazon.awscdk.services.ecr.assets.DockerImageAsset;

import software.amazon.awscdk.services.ecs.FargateTaskDefinition; import software.amazon.awscdk.services.ecs.ContainerDefinitionOptions; import software.amazon.awscdk.services.ecs.ContainerImage;

File startDir = new File(System.getProperty(“user.dir”));

DockerImageAsset asset = DockerImageAsset.Builder.create(this, “my-image”) .directory(new File(startDir, “demo-image”).toString()).build();

FargateTaskDefinition taskDefinition = FargateTaskDefinition.Builder.create( this, “TaskDef”).memoryLimitMiB(1024).cpu(512).build();

// extract the tag from the asset’s image URI for use in ECR repo lookup String imageUri = asset.getImageUri(); String imageTag = imageUri.substring(imageUri.lastIndexOf(“:”) + 1);

taskDefinition.addContainer(“my-other-container”, ContainerDefinitionOptions.builder().image(ContainerImage.fromEcrRepository( asset.getRepository(), imageTag)).build()); —

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

var asset = new DockerImageAsset(this, “my-image”, new DockerImageAssetProps { Directory = Path.Combine(Directory.GetCurrentDirectory(), “demo-image”) });

var taskDefinition = new FargateTaskDefinition(this, “TaskDef”, new FargateTaskDefinitionProps { MemoryLimitMiB = 1024, Cpu = 512 });

taskDefinition.AddContainer(“my-other-container”, new ContainerDefinitionOptions { Image = ContainerImage.FromEcrRepository(asset.Repository, asset.ImageUri.Split(“:”).Last()) }); —

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

“github.com/aws/aws-cdk-go/awscdk/v2” “github.com/aws/aws-cdk-go/awscdk/v2/awsecrassets” “github.com/aws/aws-cdk-go/awscdk/v2/awsecs” )

dirName, err := os.Getwd() if err != nil { panic(err) }

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String(“MyImage”), &awsecrassets.DockerImageAssetProps{ Directory: jsii.String(path.Join(dirName, “demo-image”)), })

taskDefinition := awsecs.NewFargateTaskDefinition(stack, jsii.String(“TaskDef”), &awsecs.FargateTaskDefinitionProps{ MemoryLimitMiB: jsii.Number(1024), Cpu: jsii.Number(512), })

taskDefinition.AddContainer(jsii.String(“MyOtherContainer”), &awsecs.ContainerDefinitionOptions{ Image: awsecs.ContainerImage\_FromEcrRepository(asset.Repository(), asset.ImageTag()), }) — ====

[#assets-types-docker-build] === Build arguments example

You can provide customized build arguments for the Docker build step through the buildArgs (Python: build\_args) property option when the {aws} CDK CLI builds the image during deployment.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const asset = new DockerImageAsset(this, ‘MyBuildImage’, { directory: path.join(\_\_dirname, ‘my-image’), buildArgs: { HTTP\_PROXY: ‘http://10.20.30.2:1234’ } }); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const asset = new DockerImageAsset(this, ‘MyBuildImage’, { directory: path.join(\_\_dirname, ‘my-image’), buildArgs: { HTTP\_PROXY: ‘http://10.20.30.2:1234’ } }); —

Python:: + [source,python,subs=“verbatim,attributes”] — asset = DockerImageAsset(self, “MyBuildImage”, directory=os.path.join(dirname, “my-image”), build\_args=dict(HTTP\_PROXY=“http://10.20.30.2:1234”)) —

Java:: + [source,java,subs=“verbatim,attributes”] — DockerImageAsset asset = DockerImageAsset.Builder.create(this, “my-image”), .directory(new File(startDir, “my-image”).toString()) .buildArgs(java.util.Map.of( // Java 9 or later “HTTP\_PROXY”, “http://10.20.30.2:1234”)) .build(); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var asset = new DockerImageAsset(this, “MyBuildImage”, new DockerImageAssetProps { Directory = Path.Combine(Directory.GetCurrentDirectory(), “my-image”), BuildArgs = new Dictionary<string, string> { [“HTTP\_PROXY”] = “http://10.20.30.2:1234” } }); —

Go:: + [source,go,subs=“verbatim,attributes”] — dirName, err := os.Getwd() if err != nil { panic(err) }

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String(“MyBuildImage”), &awsecrassets.DockerImageAssetProps{ Directory: jsii.String(path.Join(dirName, “my-image”)), BuildArgs: &map[string]\*string{ “HTTP\_PROXY”: jsii.String(“http://10.20.30.2:1234”), }, }) — ====

[#assets-types-docker-permissions] === Permissions

If you use a module that supports Docker image assets, such as https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecs-readme.html[aws-ecs], the {aws} CDK manages permissions for you when you use assets directly or through https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecs.ContainerImage.html#static-fromwbrecrwbrrepositoryrepository-tag[ContainerImage.fromEcrRepository] (Python: from\_ecr\_repository). If you use Docker image assets directly, make sure that the consuming principal has permissions to pull the image.

In most cases, you should use https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecr.Repository.html#grantwbrpullgrantee[asset.repository.grantPull] method (Python: grant\_pull). This modifies the IAM policy of the principal to enable it to pull images from this repository. If the principal that is pulling the image is not in the same account, or if it’s an {aws} service that doesn’t assume a role in your account (such as {aws} CodeBuild), you must grant pull permissions on the resource policy and not on the principal’s policy. Use the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws\_ecr.Repository.html#addwbrtowbrresourcewbrpolicystatement[asset.repository.addToResourcePolicy] method (Python: add\_to\_resource\_policy) to grant the appropriate principal permissions.

[#assets-cfn] == {aws} CloudFormation resource metadata

= [NOTE]

This section is relevant only for construct authors. In certain situations, tools need to know that a certain CFN resource is using a local asset. For example, you can use the {aws} SAM CLI to invoke Lambda functions locally for debugging purposes. See xref:sam[{aws} SAM integration] for details.

====

To enable such use cases, external tools consult a set of metadata entries on {aws} CloudFormation resources:

* aws:asset:path – Points to the local path of the asset.
* aws:asset:property – The name of the resource property where the asset is used.

Using these two metadata entries, tools can identify that assets are used by a certain resource, and enable advanced local experiences.

To add these metadata entries to a resource, use the asset.addResourceMetadata (Python: add\_resource\_metadata) method.