[#cdk-pipeline-stages] == Application stages

To define a multi-stack {aws} application that can be added to the pipeline all at once, define a subclass of +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stage.html[Stage]+. (This is different from CdkStage in the CDK Pipelines module.)

The stage contains the stacks that make up your application. If there are dependencies between the stacks, the stacks are automatically added to the pipeline in the right order. Stacks that don’t depend on each other are deployed in parallel. You can add a dependency relationship between stacks by calling stack1.addDependency(stack2).

Stages accept a default env argument, which becomes the default environment for the stacks inside it. (Stacks can still have their own environment specified.).

An application is added to the pipeline by calling +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines.CodePipeline.html#addwbrstagestage-optionss[addStage()]+ with instances of https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stage.html[Stage]. A stage can be instantiated and added to the pipeline multiple times to define different stages of your DTAP or multi-Region application pipeline.

We will create a stack containing a simple Lambda function and place that stack in a stage. Then we will add the stage to the pipeline so it can be deployed.

==== [role=“tablist”] TypeScript:: Create the new file lib/my-pipeline-lambda-stack.ts to hold our application stack containing a Lambda function. + [source,javascript,subs=“verbatim,attributes”] — import \* as cdk from ‘aws-cdk-lib’; import { Construct } from ‘constructs’; import { Function, InlineCode, Runtime } from ‘aws-cdk-lib/aws-lambda’;

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

new Function(this, ‘LambdaFunction’, { runtime: Runtime.NODEJS\_18\_X, handler: ‘index.handler’, code: new InlineCode(‘exports.handler = \_ => “Hello, CDK”;’) }); } } —- + Create the new file lib/my-pipeline-app-stage.ts to hold our stage. + [source,javascript,subs=“verbatim,attributes”] —- import \* as cdk from ‘aws-cdk-lib’; import { Construct } from “constructs”; import { MyLambdaStack } from ‘./my-pipeline-lambda-stack’;

export class MyPipelineAppStage extends cdk.Stage {

…. constructor(scope: Construct, id: string, props?: cdk.StageProps) { super(scope, id, props);

const lambdaStack = new MyLambdaStack(this, ‘LambdaStack’); } } —- + Edit lib/my-pipeline-stack.ts to add the stage to our pipeline. + [source,javascript,subs=“verbatim,attributes”] —- import \* as cdk from ‘aws-cdk-lib’; import { Construct } from ‘constructs’; import { CodePipeline, CodePipelineSource, ShellStep } from ‘aws-cdk-lib/pipelines’; import { MyPipelineAppStage } from ‘./my-pipeline-app-stage’; ….

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

…. const pipeline = new CodePipeline(this, ‘Pipeline’, { pipelineName: ‘MyPipeline’, synth: new ShellStep(‘Synth’, { input: CodePipelineSource.gitHub(‘OWNER/REPO’, ‘main’), commands: [‘npm ci’, ‘npm run build’, ‘npx cdk synth’] }) });

pipeline.addStage(new MyPipelineAppStage(this, “test”, { env: { account: “111111111111”, region: “eu-west-1” } })); } } —- ….

JavaScript:: Create the new file lib/my-pipeline-lambda-stack.js to hold our application stack containing a Lambda function. + [source,javascript,subs=“verbatim,attributes”] — const cdk = require(‘aws-cdk-lib’); const { Function, InlineCode, Runtime } = require(‘aws-cdk-lib/aws-lambda’);

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

new Function(this, ‘LambdaFunction’, { runtime: Runtime.NODEJS\_18\_X, handler: ‘index.handler’, code: new InlineCode(‘exports.handler = \_ => “Hello, CDK”;’) }); } }

== module.exports = { MyLambdaStack }

* Create the new file lib/my-pipeline-app-stage.js to hold our stage.

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

* const cdk = require(‘aws-cdk-lib’); const { MyLambdaStack } = require(‘./my-pipeline-lambda-stack’);

class MyPipelineAppStage extends cdk.Stage {

…. constructor(scope, id, props) { super(scope, id, props);

const lambdaStack = new MyLambdaStack(this, ‘LambdaStack’); } } ….

== module.exports = { MyPipelineAppStage };

* Edit lib/my-pipeline-stack.ts to add the stage to our pipeline.

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

* const cdk = require(‘aws-cdk-lib’); const { CodePipeline, CodePipelineSource, ShellStep } = require(‘aws-cdk-lib/pipelines’); const { MyPipelineAppStage } = require(‘./my-pipeline-app-stage’);

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

…. const pipeline = new CodePipeline(this, ‘Pipeline’, { pipelineName: ‘MyPipeline’, synth: new ShellStep(‘Synth’, { input: CodePipelineSource.gitHub(‘OWNER/REPO’, ‘main’), commands: [‘npm ci’, ‘npm run build’, ‘npx cdk synth’] }) });

pipeline.addStage(new MyPipelineAppStage(this, “test”, { env: { account: “111111111111”, region: “eu-west-1” } })); ….

} }

== module.exports = { MyPipelineStack }

Python:: Create the new file my\_pipeline/my\_pipeline\_lambda\_stack.py to hold our application stack containing a Lambda function. + + [source,python,subs=“verbatim,attributes”] — import aws\_cdk as cdk from constructs import Construct from aws\_cdk.aws\_lambda import Function, InlineCode, Runtime

class MyLambdaStack(cdk.Stack): def *init*(self, scope: Construct, construct\_id: str, **kwargs) -> None: super().*init*(scope, construct\_id,** kwargs)

Function(self, "LambdaFunction",  
 runtime=Runtime.NODEJS\_18\_X,  
 handler="index.handler",  
 code=InlineCode("exports.handler = \_ => 'Hello, CDK';")  
 ) ---- + Create the new file `my\_pipeline/my\_pipeline\_app\_stage.py` to hold our stage. + [source,python,subs="verbatim,attributes"] ---- import aws\_cdk as cdk from constructs import Construct from my\_pipeline.my\_pipeline\_lambda\_stack import MyLambdaStack

class MyPipelineAppStage(cdk.Stage): def *init*(self, scope: Construct, construct\_id: str, **kwargs) -> None: super().*init*(scope, construct\_id,** kwargs)

lambdaStack = MyLambdaStack(self, "LambdaStack") ---- + Edit `my\_pipeline/my-pipeline-stack.py` to add the stage to our pipeline. + [source,python,subs="verbatim,attributes"] ---- import aws\_cdk as cdk from constructs import Construct from aws\_cdk.pipelines import CodePipeline, CodePipelineSource, ShellStep from my\_pipeline.my\_pipeline\_app\_stage import MyPipelineAppStage

class MyPipelineStack(cdk.Stack):

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

pipeline = CodePipeline(self, "Pipeline",  
 pipeline\_name="MyPipeline",  
 synth=ShellStep("Synth",  
 input=CodePipelineSource.git\_hub("OWNER/REPO", "main"),  
 commands=["npm install -g aws-cdk",  
 "python -m pip install -r requirements.txt",  
 "cdk synth"]))  
  
pipeline.add\_stage(MyPipelineAppStage(self, "test",  
 env=cdk.Environment(account="111111111111", region="eu-west-1"))) ----

….

Java:: Create the new file src/main/java/com.myorg/MyPipelineLambdaStack.java to hold our application stack containing a Lambda function. + [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.lambda.Function; import software.amazon.awscdk.services.lambda.Runtime; import software.amazon.awscdk.services.lambda.InlineCode;

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

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

Function.Builder.create(this, "LambdaFunction")  
 .runtime(Runtime.NODEJS\_18\_X)  
 .handler("index.handler")  
 .code(new InlineCode("exports.handler = \_ => 'Hello, CDK';"))  
 .build();

} ….

== }

* Create the new file src/main/java/com.myorg/MyPipelineAppStage.java to hold our stage.

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

* package com.myorg;

import software.constructs.Construct; import software.amazon.awscdk.Stack; import software.amazon.awscdk.Stage; import software.amazon.awscdk.StageProps;

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

…. public MyPipelineAppStage(final Construct scope, final String id, final StageProps props) { super(scope, id, props);

Stack lambdaStack = new MyPipelineLambdaStack(this, "LambdaStack");

} ….

== }

* Edit src/main/java/com.myorg/MyPipelineStack.java to add the stage to our pipeline.

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

* package com.myorg;

import java.util.Arrays; import software.constructs.Construct; import software.amazon.awscdk.Environment; import software.amazon.awscdk.Stack; import software.amazon.awscdk.StackProps; import software.amazon.awscdk.StageProps; import software.amazon.awscdk.pipelines.CodePipeline; import software.amazon.awscdk.pipelines.CodePipelineSource; import software.amazon.awscdk.pipelines.ShellStep;

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

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

final CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")  
 .pipelineName("MyPipeline")  
 .synth(ShellStep.Builder.create("Synth")  
 .input(CodePipelineSource.gitHub("OWNER/REPO", "main"))  
 .commands(Arrays.asList("npm install -g aws-cdk", "cdk synth"))  
 .build())  
 .build();  
  
pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()  
 .env(Environment.builder()  
 .account("111111111111")  
 .region("eu-west-1")  
 .build())  
 .build()));

} } —- ….

C#:: Create the new file src/MyPipeline/MyPipelineLambdaStack.cs to hold our application stack containing a Lambda function. + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Constructs; using Amazon.CDK.{aws}.Lambda;

namespace MyPipeline { class MyPipelineLambdaStack : Stack { public MyPipelineLambdaStack(Construct scope, string id, StackProps props=null) : base(scope, id, props) { new Function(this, “LambdaFunction”, new FunctionProps { Runtime = Runtime.NODEJS\_18\_X, Handler = “index.handler”, Code = new InlineCode(“exports.handler = \_ => ‘Hello, CDK’;”) }); } } } — + Create the new file src/MyPipeline/MyPipelineAppStage.cs to hold our stage. + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Constructs;

namespace MyPipeline { class MyPipelineAppStage : Stage { public MyPipelineAppStage(Construct scope, string id, StageProps props=null) : base(scope, id, props) { Stack lambdaStack = new MyPipelineLambdaStack(this, “LambdaStack”); } } } — + Edit src/MyPipeline/MyPipelineStack.cs to add the stage to our pipeline. + + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Constructs; using Amazon.CDK.Pipelines;

namespace MyPipeline { public class MyPipelineStack : Stack { internal MyPipelineStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { var pipeline = new CodePipeline(this, “pipeline”, new CodePipelineProps { PipelineName = “MyPipeline”, Synth = new ShellStep(“Synth”, new ShellStepProps { Input = CodePipelineSource.GitHub(“OWNER/REPO”, “main”), Commands = new string[] { “npm install -g aws-cdk”, “cdk synth” } }) });

pipeline.AddStage(new MyPipelineAppStage(this, "test", new StageProps  
 {  
 Env = new Environment  
 {  
 Account = "111111111111", Region = "eu-west-1"  
 }  
 }));  
 }

} } —- ====

Every application stage added by addStage() results in the addition of a corresponding pipeline stage, represented by a +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines.StageDeployment.html[StageDeployment]+ instance returned by the addStage() call. You can add pre-deployment or post-deployment actions to the stage by calling its addPre() or addPost() method.

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

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, ‘testing’, { env: { account: ‘111111111111’, region: ‘eu-west-1’ } }));

testingStage.addPost(new ManualApprovalStep(‘approval’)); —-

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

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, ‘testing’, { env: { account: ‘111111111111’, region: ‘eu-west-1’ } }));

== testingStage.addPost(new ManualApprovalStep(‘approval’));

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

= from aws\_cdk.pipelines import ManualApprovalStep

testing\_stage = pipeline.add\_stage(MyPipelineAppStage(self, “testing”, env=cdk.Environment(account=“111111111111”, region=“eu-west-1”)))

== testing\_stage.add\_post(ManualApprovalStep(‘approval’))

Java:: + [source,java,subs=“verbatim,attributes”] — // import software.amazon.awscdk.pipelines.StageDeployment; // import software.amazon.awscdk.pipelines.ManualApprovalStep;

StageDeployment testingStage = pipeline.addStage(new MyPipelineAppStage(this, “test”, StageProps.builder() .env(Environment.builder() .account(“111111111111”) .region(“eu-west-1”) .build()) .build()));

== testingStage.addPost(new ManualApprovalStep(“approval”));

C#:: + [source,csharp,subs=“verbatim,attributes”] — var testingStage = pipeline.AddStage(new MyPipelineAppStage(this, “test”, new StageProps { Env = new Environment { Account = “111111111111”, Region = “eu-west-1” } }));

== testingStage.AddPost(new ManualApprovalStep(“approval”));

====

You can add stages to a +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines.Wave.html[Wave]+ to deploy them in parallel, for example when deploying a stage to multiple accounts or Regions.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const wave = pipeline.addWave(‘wave’); wave.addStage(new MyApplicationStage(this, ‘MyAppEU’, { env: { account: ‘111111111111’, region: ‘eu-west-1’ } })); wave.addStage(new MyApplicationStage(this, ‘MyAppUS’, { env: { account: ‘111111111111’, region: ‘us-west-1’ } })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const wave = pipeline.addWave(‘wave’); wave.addStage(new MyApplicationStage(this, ‘MyAppEU’, { env: { account: ‘111111111111’, region: ‘eu-west-1’ } })); wave.addStage(new MyApplicationStage(this, ‘MyAppUS’, { env: { account: ‘111111111111’, region: ‘us-west-1’ } })); —

Python:: + [source,python,subs=“verbatim,attributes”] — wave = pipeline.add\_wave(“wave”) wave.add\_stage(MyApplicationStage(self, “MyAppEU”, env=cdk.Environment(account=“111111111111”, region=“eu-west-1”))) wave.add\_stage(MyApplicationStage(self, “MyAppUS”, env=cdk.Environment(account=“111111111111”, region=“us-west-1”))) —

Java:: + [source,java,subs=“verbatim,attributes”] — // import software.amazon.awscdk.pipelines.Wave; final Wave wave = pipeline.addWave(“wave”); wave.addStage(new MyPipelineAppStage(this, “MyAppEU”, StageProps.builder() .env(Environment.builder() .account(“111111111111”) .region(“eu-west-1”) .build()) .build())); wave.addStage(new MyPipelineAppStage(this, “MyAppUS”, StageProps.builder() .env(Environment.builder() .account(“111111111111”) .region(“us-west-1”) .build()) .build())); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var wave = pipeline.AddWave(“wave”); wave.AddStage(new MyPipelineAppStage(this, “MyAppEU”, new StageProps { Env = new Environment { Account = “111111111111”, Region = “eu-west-1” } })); wave.AddStage(new MyPipelineAppStage(this, “MyAppUS”, new StageProps { Env = new Environment { Account = “111111111111”, Region = “us-west-1” } })); — ====