[#cdk-pipeline-validation] == Testing deployments

You can add steps to a CDK Pipeline to validate the deployments that you’re performing. For example, you can use the CDK Pipeline library’s +link:https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines.ShellStep.html[ShellStep]+ to perform tasks such as the following:

* Trying to access a newly deployed Amazon API Gateway backed by a Lambda function
* Checking a setting of a deployed resource by issuing an {aws} CLI command

In its simplest form, adding validation actions looks like this:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // stage was returned by pipeline.addStage

stage.addPost(new ShellStep(“validate”, { commands: [‘../tests/validate.sh’], })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // stage was returned by pipeline.addStage

stage.addPost(new ShellStep(“validate”, { commands: [‘../tests/validate.sh’], })); —

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

= stage was returned by pipeline.add\_stage

stage.add\_post(ShellStep(“validate”, commands=[’‘../tests/validate.sh’’] )) —

Java:: + [source,java,subs=“verbatim,attributes”] — // stage was returned by pipeline.addStage

stage.addPost(ShellStep.Builder.create(“validate”) .commands(Arrays.asList(“‘../tests/validate.sh’”)) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // stage was returned by pipeline.addStage

stage.AddPost(new ShellStep(“validate”, new ShellStepProps { Commands = new string[] { “‘../tests/validate.sh’” } })); — ====

Many {aws} CloudFormation deployments result in the generation of resources with unpredictable names. Because of this, CDK Pipelines provide a way to read {aws} CloudFormation outputs after a deployment. This makes it possible to pass (for example) the generated URL of a load balancer to a test action.

To use outputs, expose the CfnOutput object you’re interested in. Then, pass it in a step’s envFromCfnOutputs property to make it available as an environment variable within that step.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // given a stack lbStack that exposes a load balancer construct as loadBalancer this.loadBalancerAddress = new cdk.CfnOutput(lbStack, ‘LbAddress’, { value: https://${lbStack.loadBalancer.loadBalancerDnsName}/ });

// pass the load balancer address to a shell step stage.addPost(new ShellStep(“lbaddr”, { envFromCfnOutputs: {lb\_addr: lbStack.loadBalancerAddress}, commands: [‘echo $lb\_addr’] })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — // given a stack lbStack that exposes a load balancer construct as loadBalancer this.loadBalancerAddress = new cdk.CfnOutput(lbStack, ‘LbAddress’, { value: https://${lbStack.loadBalancer.loadBalancerDnsName}/ });

// pass the load balancer address to a shell step stage.addPost(new ShellStep(“lbaddr”, { envFromCfnOutputs: {lb\_addr: lbStack.loadBalancerAddress}, commands: [‘echo $lb\_addr’] })); —

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

= given a stack lb\_stack that exposes a load balancer construct as load\_balancer

self.load\_balancer\_address = cdk.CfnOutput(lb\_stack, “LbAddress”, value=f”https://{lb\_stack.load\_balancer.load\_balancer\_dns\_name}/“)

= pass the load balancer address to a shell step

stage.add\_post(ShellStep(“lbaddr”, env\_from\_cfn\_outputs={“lb\_addr”: lb\_stack.load\_balancer\_address} commands=[“echo $lb\_addr”])) —

Java:: + [source,java,subs=“verbatim,attributes”] — // given a stack lbStack that exposes a load balancer construct as loadBalancer loadBalancerAddress = CfnOutput.Builder.create(lbStack, “LbAddress”) .value(String.format(“https://%s/”, lbStack.loadBalancer.loadBalancerDnsName)) .build();

stage.addPost(ShellStep.Builder.create(“lbaddr”) .envFromCfnOutputs( // Map.of requires Java 9 or later java.util.Map.of(“lbAddr”, loadBalancerAddress)) .commands(Arrays.asList(“echo $lbAddr”)) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — // given a stack lbStack that exposes a load balancer construct as loadBalancer loadBalancerAddress = new CfnOutput(lbStack, “LbAddress”, new CfnOutputProps { Value = string.Format(“https://{0}/”, lbStack.loadBalancer.LoadBalancerDnsName) });

stage.AddPost(new ShellStep(“lbaddr”, new ShellStepProps { EnvFromCfnOutputs = new Dictionary<string, CfnOutput> { { “lbAddr”, loadBalancerAddress } }, Commands = new string[] { “echo $lbAddr” } })); — ====

You can write simple validation tests right in the ShellStep, but this approach becomes unwieldy when the test is more than a few lines. For more complex tests, you can bring additional files (such as complete shell scripts, or programs in other languages) into the ShellStep via the inputs property. The inputs can be any step that has an output, including a source (such as a GitHub repo) or another ShellStep.

Bringing in files from the source repository is appropriate if the files are directly usable in the test (for example, if they are themselves executable). In this example, we declare our GitHub repo as source (rather than instantiating it inline as part of the CodePipeline). Then, we pass this fileset to both the pipeline and the validation test.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const source = CodePipelineSource.gitHub(‘OWNER/REPO’, ‘main’);

const pipeline = new CodePipeline(this, ‘Pipeline’, { pipelineName: ‘MyPipeline’, synth: new ShellStep(‘Synth’, { input: source, commands: [‘npm ci’, ‘npm run build’, ‘npx cdk synth’] }) });

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

stage.addPost(new ShellStep(‘validate’, { input: source, commands: [‘sh ../tests/validate.sh’] })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const source = CodePipelineSource.gitHub(‘OWNER/REPO’, ‘main’);

const pipeline = new CodePipeline(this, ‘Pipeline’, { pipelineName: ‘MyPipeline’, synth: new ShellStep(‘Synth’, { input: source, commands: [‘npm ci’, ‘npm run build’, ‘npx cdk synth’] }) });

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

stage.addPost(new ShellStep(‘validate’, { input: source, commands: [‘sh ../tests/validate.sh’] })); —

Python:: + [source,python,subs=“verbatim,attributes”] — source = CodePipelineSource.git\_hub(“OWNER/REPO”, “main”)

pipeline = CodePipeline(self, “Pipeline”, pipeline\_name=“MyPipeline”, synth=ShellStep(“Synth”, input=source, commands=[“npm install -g aws-cdk”, “python -m pip install -r requirements.txt”, “cdk synth”]))

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

stage.add\_post(ShellStep(“validate”, input=source, commands=[“sh ../tests/validate.sh”], )) —

Java:: + [source,java,subs=“verbatim,attributes”] — final CodePipelineSource source = CodePipelineSource.gitHub(“OWNER/REPO”, “main”);

final CodePipeline pipeline = CodePipeline.Builder.create(this, “pipeline”) .pipelineName(“MyPipeline”) .synth(ShellStep.Builder.create(“Synth”) .input(source) .commands(Arrays.asList(“npm install -g aws-cdk”, “cdk synth”)) .build()) .build();

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

stage.addPost(ShellStep.Builder.create(“validate”) .input(source) .commands(Arrays.asList(“sh ../tests/validate.sh”)) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var source = CodePipelineSource.GitHub(“OWNER/REPO”, “main”);

var pipeline = new CodePipeline(this, “pipeline”, new CodePipelineProps { PipelineName = “MyPipeline”, Synth = new ShellStep(“Synth”, new ShellStepProps { Input = source, Commands = new string[] { “npm install -g aws-cdk”, “cdk synth” } }) });

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

stage.AddPost(new ShellStep(“validate”, new ShellStepProps { Input = source, Commands = new string[] { “sh ../tests/validate.sh” } })); — ====

Getting the additional files from the synth step is appropriate if your tests need to be compiled, which is done as part of synthesis.

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — const synthStep = new ShellStep(‘Synth’, { input: CodePipelineSource.gitHub(‘OWNER/REPO’, ‘main’), commands: [‘npm ci’, ‘npm run build’, ‘npx cdk synth’], });

const pipeline = new CodePipeline(this, ‘Pipeline’, { pipelineName: ‘MyPipeline’, synth: synthStep });

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

// run a script that was transpiled from TypeScript during synthesis stage.addPost(new ShellStep(‘validate’, { input: synthStep, commands: [‘node tests/validate.js’] })); —

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const synthStep = new ShellStep(‘Synth’, { input: CodePipelineSource.gitHub(‘OWNER/REPO’, ‘main’), commands: [‘npm ci’, ‘npm run build’, ‘npx cdk synth’], });

const pipeline = new CodePipeline(this, ‘Pipeline’, { pipelineName: ‘MyPipeline’, synth: synthStep });

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

// run a script that was transpiled from TypeScript during synthesis stage.addPost(new ShellStep(‘validate’, { input: synthStep, commands: [‘node tests/validate.js’] })); —

Python:: + [source,python,subs=“verbatim,attributes”] — synth\_step = 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 = CodePipeline(self, “Pipeline”, pipeline\_name=“MyPipeline”, synth=synth\_step)

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

= run a script that was compiled during synthesis

stage.add\_post(ShellStep(“validate”, input=synth\_step, commands=[“node test/validate.js”], )) —

Java:: + [source,java,subs=“verbatim,attributes”] — final ShellStep synth = ShellStep.Builder.create(“Synth”) .input(CodePipelineSource.gitHub(“OWNER/REPO”, “main”)) .commands(Arrays.asList(“npm install -g aws-cdk”, “cdk synth”)) .build();

final CodePipeline pipeline = CodePipeline.Builder.create(this, “pipeline”) .pipelineName(“MyPipeline”) .synth(synth) .build();

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

stage.addPost(ShellStep.Builder.create(“validate”) .input(synth) .commands(Arrays.asList(“node ./tests/validate.js”)) .build()); —

C#:: + [source,csharp,subs=“verbatim,attributes”] — var synth = new ShellStep(“Synth”, new ShellStepProps { Input = CodePipelineSource.GitHub(“OWNER/REPO”, “main”), Commands = new string[] { “npm install -g aws-cdk”, “cdk synth” } });

var pipeline = new CodePipeline(this, “pipeline”, new CodePipelineProps { PipelineName = “MyPipeline”, Synth = synth });

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

stage.AddPost(new ShellStep(“validate”, new ShellStepProps { Input = synth, Commands = new string[] { “node ./tests/validate.js” } })); — ====