CloudFormation

Enabling Continuous Integration on Lambda Functions

What happens when you use CodeBuild to build?

- The source code from the Github repo is zipped up and placed within an auto-generated codepipeline s3 bucket (specifically, within /sourceArti).
- During the build stage, codepipeline looks for this artifact (the zip file with the source code), calls `aws cloudformation package` which does 2 things:

    1. Converts samTemplate.yaml into outputsamtemplate.yaml and adds a zip with {samTemplate.yaml, outputsamtemplate.yml} to /buildArti in the generated S3 bucket. The samTemplate.yaml is basically just a unique version of a cloudformation template that provides some shorthands (a cloudformation template instructs how to create and manage AWS resources—we'll look at one later). In fact, under the hood, the samTemplate.yml gets converted into a cloudformation template, which is executed to create your lambda functions.

        Documentation for SAM templates are here: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification.html

        Note: outputsamtemplate.yml is literally the exact same thing as samTemplate.yaml except it is generated by `aws cloudformation package` in the build stage and it changes CodeUri to a S3 location that is accessible when executing later.

    2. Uploads local artifacts to a separate S3 bucket (ie. lambdapipeline-bucket-1) Before you build, make sure to actually create the S3 bucket which will store a build artifact for the source code of each of your lambda functions.

Note: Make sure for the building stage, the build project has all the permissions to execute `aws cloudformation package` fully.

What happens when you use CloudFormation to deploy?

To add CodeFormation as the deploy stage of your codepipeline:

https://docs.aws.amazon.com/codepipeline/latest/userguide/integrations-action-type.html#integrations-deploy-CloudFormation

To actually use CloudFormation on the deploy stage, former TA Rishabh explains:

> In the deploy stage of your CodePipeline, choose "CloudFormation". For Action, choose "create or update stack". Select BuildArtifact under artifacts and give the output yaml file name that you use in buildspec.yml that I have attached. For capabilities, select IAM and AUTO-EXPAND and use the CloudFormation role you created.

From here, CloudFormation will look for the ouputsamtemplate.yml artifact from the previous step (in /buildArti) to create your lambda functions on the fly. Of course, the outputsamtemplate.yml knows where the source code for your lambda functions reside by looking at the CodeUri, which now points to a S3 location.

Great! Now when you make a change to your Github repo, the CodePipeline will trigger and ultimately re-build your lambda functions using CloudFormation.

Getting started with your functional stack using CloudFormation

Documentation for CloudFormation templates is here:

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html

- Go to CloudFormation and click "Create Stack > With new resources"
- Choose "Upload a template file" and provide your CloudFormation template file.
- Enter your stack name, then click "next" until CloudFormation begins creating the resources you specified in CF_template.json.
- After a while, you'll see that you created a lambda function that the CF_template specified.
- One of the primary challenges of this assignment is actually wading through the CloudFormation documentation and writing your CF_template.json to be able to create the large majority of your functional stack, all with a couple of clicks.
