Deployment Pipeline Template file. Part 2.

Lets continue with our Codepipeline template file. In the past lecture we define our description and parameter section and started with the resources section.

In resources we defined the 3 roles we are going to use in our Codepipeline implementation:

CloudFormationExecutionRole: needed in the Deploy stage.
CodeBuildServiceRole: needed in the Codebuild project execution.
CodePipelineServiceRole: needed in the pipeline definition.

Lets continue with our Codepipeline template file.

Its time now to define our Pipeline with the 3 stages we need in this course:

We are going to name the resource Pipeline And give it a AWS::CodePipeline::Pipeline type this one creates a pipeline that describes how software changes go through a release process.

Next the Properties:
Rolearn is the first property to define. Here we are going to reference the role we created above and extract the ARN attribute !GetAtt CodePipelineServiceRole.Arn

Next the ArtifactStore:
As we explained in the last lecture this one specifies The S3 bucket location where AWS CodePipeline stores pipeline artifacts.

First we define the type S3 and reference the Artifact Location

Next we are going to define the core of the pipeline, the stages

As we talked, we are going to implement here 3 stages: Source, Build and Deploy

Lets start with the Stage Source

We are going to use the Name Source. The first thing to do are the Actions In AWS CodePipeline, an action is part of the sequence in a stage of a pipeline. It is a task performed on the artifact in that stage.

Pipeline actions occur in a specified order, in sequence or in parallel, as determined in the configuration of the stage.

Lets give the action a Name: CodeCommitRepoSource

Next we will need the ActionTypeId property this one specifies the action type and provider for an AWS CodePipeline action.

Now we define the Actiontypeid Category: A category specify which action type is going to be performed, and of course constraint the provider for this action.

In our case will be Source, so the provider must be a Source Provider, CodeCommit and the owner of the Provider is AWS.

Next we specify the Version 1

Lets now go with the Configuration for our source Actiontypeid you probably figured out we will need here.

The repository name and the branch name both will be referenced from our parameters section

Then the OutputArtifacts property this specifies an artifact that is the result of an AWS CodePipeline action.

We need to give a name to the OutputArtifacts So we type in Name CodeCommitRepoSource

Next we want to specify a running order with the RunOrder property The default runOrder value for an action is 1. The value must be a positive integer. To specify a serial sequence of actions, use the smallest number for the first action and larger numbers for each of the rest of the actions in sequence.

To specify parallel actions, use the same integer for each action you want to run in parallel.

At this Source stage we want to also specify a S3 bucket containing our template files.

This is going be used in the deploy stage.

First lets give it a name, we can call it Architecture-Template.
So we define in the same way we just did the ActionTypeId, the Category, the Owner and the version but in the Provider we will need to use the S3 value, of course, our OutputArtifact will be Architecture-Template. We want this to run in parallel with the Source action so we will have RunOrder 1, And the configuration will be S3Bucket: with a reference to the TemplateBucket parameter and S3ObjectKey with the templates.zip

We are going to be uploading this file containing all our templates in the templates folder.

Next stage will be the Build stage

The properties will be really similar to the ones we specified in the source stage

The Name will be Build the ActionTypeId property has:
Category: Build
Owner: AWS
Version: 1
Our Provider will be CodeBuild

In the configuration we will reference the already defined CodeBuild project resource. This stage will have an InputArtifacts precisely the one we just output before
Name: CodeCommitRepoSource

The OutputArtifacts for this stage will be BuildOutput

This will be the only one running so we will have RunOrder 1

We are going to define a Buildtest here this one is not going to be used in our course but its nice to have it done in case you need it in your projects.

We will use almost the same configuration we used in the Build action we declared before the only difference will be the ParameterOverrides specifying the TestSemaphore.

The last stage will be our Deploy stage as we did on the other two stages

we will have:
Category: Deploy
Owner: AWS
Version: 1
Provider: CloudFormation

Lets check the Configuration details.

We are going to use ChangeSetName property here this property let us to give a name of an existing change set or a new change set that you want to create for the specified stack.

With this we are giving a name to the changeset performed by our deploy stage

We want our deploy stage to be able to creates the stack if the stack does not exist, or in our case, the stack exists and AWS CloudFormation updates the stack.

ActionMode: CREATE_UPDATE knowing this, the next property is the stackname affected for our deploy stage as usual we use the reference to our stackname StackName: !Ref StackName

Next we want to deal with the permissions and use the IAM resources or roles we created for this we use the Capabilities property with the value CAPABILITY_IAM.

Now we need to specify out template file location affected for this we use the property TemplatePath: with the value Architecture-Template::templates/service.yaml

Next we need the role to assume RoleArn: !GetAtt CloudFormationExecutionRole.Arn this is the one we declare first in our resources

We want our deploy stage be able to change parameters in our template, I’m going to include an example here able to change the Desired Count variable we specify in our service template file and we are leaving the Cluster and the targetgroup variables the same.

Lastly the inputartifacts for this stage will be
- Name: Architecture-Template
- Name: BuildOutput

With the run order of 1.

The final section in our template is the Outputs section, here we are going to have the Pipeline url created, as we did in other template files we are going to use the Sub function to concatenate string and build our URL

Our deployment and architecture is finished and ready to use In the next lecture we are going to build our Dockerfie with a simple PHP app and see how everything works together