Cloudformation Execution - Building our Stack.

After we finnish all our architecture templates, its now time to build our stack using the cloudformation service.

Lets check first our requirements & considerations for this cloudformation architecture to build successfully:

1.- Taking a look to our main architecture template, parameters section, we are declaring a CodeCommit repository for our future Development Pipeline. You probably notice we are not using this repo in any place, so for this step, we are able to removed, but lets keep it for now, because that parameters are gonna be used in the second part of the course, when we define the deployment pipeline.

2.- Lets now check the VPC resource in the same file. We are declaring a CIDR block for the VPC and their subnets. Its important to know that this cidr block must not in use in our account, remember that the Cloudformation is going to create this VPC and subnets and wont be able to do it if we already have this CIDR in use in our account.

3.- If you see all our TemplateURL: are pointing to an s3 bucket, we need to define this S3 bucket in our account, upload our template files there, give it Versioning and static website hosting capability.

4.- Lets go now to the ECS-Cluster yaml file, remember we defined a Mappings property to have ECS optimized AMI, its always nice to keep this list updated with the latest aws list for this purpose.

5.- In the same file, there is a property called keyname owned by the Autoscaling resource, this one specify the name for an ssh pair key. We need to have a SSH key pair with that name in the same region we are deploying our stack.

6.- Its also important that if we want to access our EC2 instance inside the ECS cluster, we will need a bastion instance in the same VPC with the same Security Group, and of course this bastion instance needs to have access in port 22 from our pc and then from this instance we are going to connect to our EC2 isntance. Remember that our EC2 instance only allows traffic from the ALB in port 80, and from other instances in the same Security Group. and there is no way to access the instances from outside.

With all this in mind, lets now build our Architecture using the clouformation service:

First login to your AWS account and go to the Cloudformation service under the Management Tool Section.

Once you are here, you need to click on the Create Stack button,

Next In the select template screen we click the option Specify an Amazon S3 template URL, and there we type in our S3 bucket url containning our templates and the name of the main architecture file:

https://s3.amazonaws.com/cicdoveraws/main-arch.yaml

You can always use my S3 but I really recommend to build your own to host your architecture templates.

If we want to check all the dependencies and modify in a visual way our architecture, we can click the View/Edit template in Designer link

Here we see a really cool perspective of the Architecture we are building. In this screen we will have our 5 stacks or module to be built and the relation between them.

Following the arrows we notice:

- The load balancer needs outputs from the VPC.the
- The Same happends with the Database Cluster
- The ECS cluster needs outputs from the VPC and from the Load Balancer stack
- The ECS service needs outputs from the Load balancer and of course from the ECS cluster
- Lastly the VPC can be created stand alone.

With this analysis we can figure out an order creation used by the cloudformation service for our architecture template.

We are also able to check our template in both Yaml and json format, and of course, verify its properties, metadata, Deletion Policy, Dependencies and Conditions.

Continuing with our building we click the next button and get into the screen where we define our set up parameters. The first thing to add is a name for our Stack. Lets fill in the Stack name field with: continuous integration and continuous delivery over Amazon Web Services.

Next the parameters we declare in our main architecture template:

Checking our main architecture yaml file we should have the code commit parameters group with the label CodeCommit Repository Configuration and the labels:
- CodeCommit Repository Name
- Code Repository Branch Name (master)

And outside from this parameter group we should have the DB password field.

Going back to our cloduformation create stack screen we can see that everything is perfect.

Lets now add our Code Commit repository name, if you dont have created a codecommit repo, just open the Codecommit AWS service in a new tab,

go to the Create repository button give a name and a description to the repository and click the Create repository button.

Once you have you repo ready, lets type in the repository name we just created in the field CodeCommit Repository name.

The next field is the CodeCommit repository Branch Name, here we just type in the master. Because we want t work with our master branch.

The last parameter is the password for our database we can use a really cool password here. Lets type in a nice passwod. for example $ASD123asdAdmin-$ and click the next button.

I like the idea to add tags to our clouformation resource. You can use the AWS CloudFormation Resource Tags property to apply tags to resources, which can help you identify and categorize those resources. You can tag only resources for which AWS CloudFormation supports tagging.

So the key will be StackName and the key will be cicdoveraws, we are not going to add any IAM Role here, or any advanced option. So we click the next button.

We just got our review screen, here we can check everything we set in the previous screen. If we made a mistake we can click the title for this section and go directly to edit it. We check everything is perfect and just go the Capabilities section, there we just click the check I acknowledge that AWS CloudFormation might create IAM resources with custom names and click the create button to start the magic.

In the next lecture we are going to deep dive in the building magic, the resources created, the order of creation and of course all the service we use in our architecture.