Deployment 5

11/29/22

Mike Major

Deployment 5

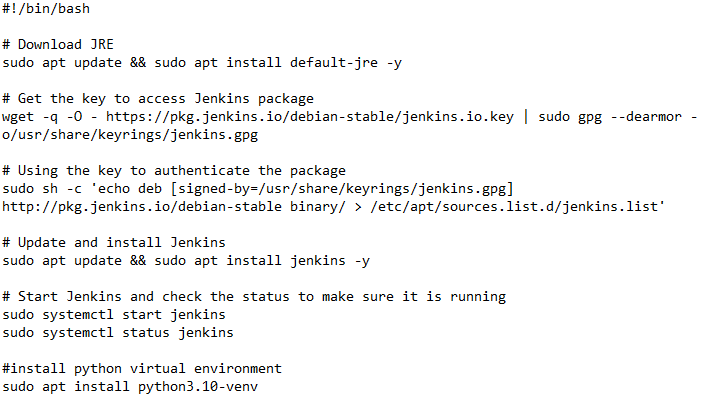
Deploying Containerized Application

Deployment Focus

The primary focus on this deployment is to deploy a containerized application utilizing Jenkins, Terraform, Docker and Amazon Elastic Container Server (ECS). Below you will see a walk through of the steps alongside diagrams and screenshots.

Create EC2’s to host Jenkins, Docker, and Terraform

1. Create EC2’s within AWS Console
   1. Utilize userdata to build a script that will install all the necessary components to automate the installation. Please see an example below. For this example, Ubuntu Linux distribution’s were used.



* 1. Repeat the steps to create your EC2’s for Docker and Terraform
     1. After creating your EC2’s for Docker -- install Docker. You can utilize this page for steps to install docker using the convenience script -- <https://docs.docker.com/engine/install/ubuntu/>
     2. Then, after creating an EC2 for Terraform, use the steps below to install Terraform. <https://developer.hashicorp.com/terraform/downloads>

Create Dockerfile

The dockerfile is a set of guidelines or instructions that Docker utilizes to create an image. From that image, a container can be created and eventually deployed. As we will be utilizing ECS later on in our steps, the container that will be deployed in ECS must be up to date. The steps below will show how to create a dockerfile with the latest up to date files.

1. First install git
   1. apt -y install git
   2. git clone (repo that is hosting your containerized application)

Create Jenkinsfile

The Jenkinsfile will provide Jenkins the necessary steps to build, test, and deploy your application. For this step, we will need to build the Jenkinsfile from scratch. Unlike previous deployments, we will now need to incorporate stages for Jenkins, Terraform, and Docker

1. Build
   1. Creates the application environment
2. Test
   1. Unit test is performed on specific portions
3. Build and Docker Push
4. Terraform Init
   1. Terraform is initialized
5. Terraform Plan
   1. Terraform checks the deltas for any changes vs requested changes in config file
6. Terraform Apply
   1. Terraform applies requested changes in the AWS environment as specified
7. Terraform Destroy
   1. Terraform destroys all resources that is has created

Create Terraform File

Terraform will be instrumental in creating your infrastructure that will support your container.

Here is a general layout as to how you should create your Terraform files.

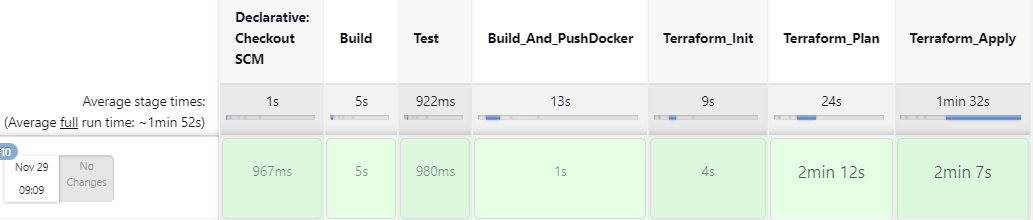
Include the following resources:

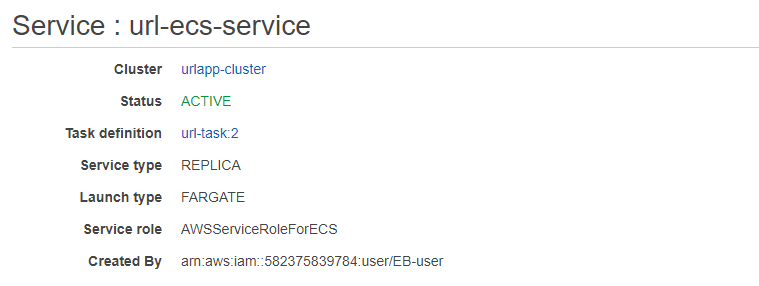
1. Subnets
2. VPC’s
3. Internet Gateways
4. Security Groups
5. Elastic IP’s
6. Route Tables
7. Load Balancers
8. Clusters
9. ECS services
10. Task Definitions

Configure Pipeline

Here is a layout of the overall pipeline from start to finish

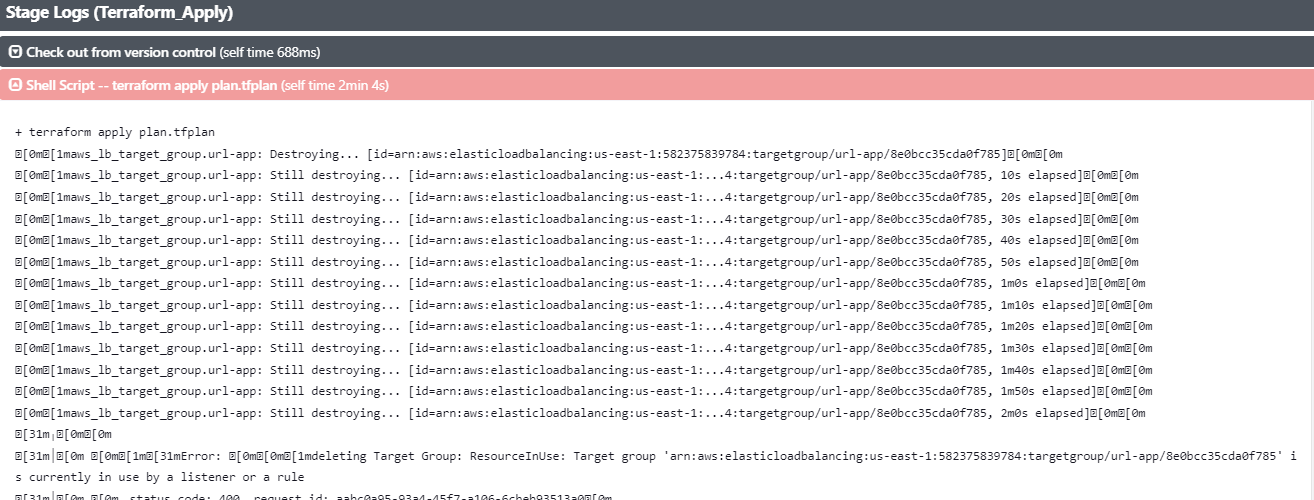
1. The GitHub files for containerized application is attached to Jenkins via webhook
2. Docker pull the docker file from the repository and creates a new image
3. The dockerfile, now equipped with the latest files, is cloned.
4. The docker image is now pushed to the DockerHub, online repository.
5. Terraform completes the process by building the infrastructure around the application





Potential Issues Discovered

During the deployment I ran into an issue during my Terraform Apply stage.



Terraform was attempting to destroy my load balancer target group and for some reason, could not. This caused a significant wait time that impacted the ability for my build to fail. After logging into my AWS environment, I was able to manually delete all the resources in question. This includes my load balancer and corresponding target group. After deleting all the resources, I re-ran my Terraform init and plan stages and was able to complete my build.