Contents

[Setup: 2](#_Toc101983397)

[Installation Instructions: 2](#_Toc101983398)

[1. Develop REST API Application. 2](#_Toc101983399)

[1.1. Test the application locally. 3](#_Toc101983400)

[2. Create a Pipeline 4](#_Toc101983401)

[3. Attach repo with the Pipeline 4](#_Toc101983402)

[4. Place Pipeline file in GitHub 5](#_Toc101983403)

[5. Created Credentials in the Jenkins for AWS 5](#_Toc101983404)

[6. Create Terraform Script 5](#_Toc101983405)

[6.1. Prepare Infrastructure, build docker image and deploy REST API app in ECS. 5](#_Toc101983406)

[6.2. Canary/Blue-Green Deployment. 6](#_Toc101983407)

[7. Architecture diagram 7](#_Toc101983408)

[8. Future Work for Production Environment 7](#_Toc101983409)

# Setup:

I have built a simple restful application, which is responding to http://thebazarpoint.com/version endpoint and display its current version (2.0.0).

# Installation Instructions:

# Develop REST API Application.

Created a [mashed-potato/project/demo.py](https://github.com/At112/meshed-potato) file for the REST API. I exposed 5001 port for the application and building this app in flask using flask\_restfull.

Text

Description automatically generated

Create a [mashed-potato/project/Dockerfile](https://github.com/At112/meshed-potato) in the same directory and add the exposed port, entry point, python and set requirement.txt file.

Text

Description automatically generated

I have created a [**mashed-potato**](https://github.com/At112/meshed-potato)/**project**/**requirement.txt** file to add the required modules to run the app.

Text

Description automatically generated with low confidence

Run the below command to create the docker image.

*“docker build --tag flask-docker-demo-app .”*

*Text

Description automatically generated*

Now docker image is ready for the application to deploy on any container platform. But I have automated the docker image build and push to the ECR using the terraform script described below.

## Test the application locally.

I run the application locally “*python3 demo.py”* then use **postman** to run the API call.

Graphical user interface, text, application, email

Description automatically generated

# Create a Jenkins Pipeline

I have created a Jenkins Pipeline, which is able to pull the deployment code from the GitHub repo and if there are any changes committed to the “main” branch then it will trigger the pipeline.

It has two stages which will run the init and plan in first stage and second stage will run the apply.

# Attach repo with the Jenkins Pipeline

I have added GitHub repository in the Jenkins pipeline shown below.

Graphical user interface, text, application, email

Description automatically generated

# Place Pipeline file in GitHub

I have created a pipeline file to run the terraform in the below location in Git Repo.

“[mashed-potato](https://github.com/At112/meshed-potato)/pipeline”

Text

Description automatically generated

# Created Credentials in the Jenkins for AWS

I have created Access key and Secret Key in Jenkins credentials and used the secret name in the pipeline file showing above.

A picture containing text

Description automatically generated

# Create Terraform Script

## Prepare Infrastructure, build docker image and deploy app in ECS.

I have prepared the infrastructure script and pushed to the GitHub. From where my Jenkins pipeline is pulling the code and running it automatically.

[**mashed-potato**](https://github.com/At112/meshed-potato)/

I have created below files for terraform.

1. File **iam.tf** is creating IAM role for creation of the Fargate ECS resources.
2. File **main.tf** is the main terraform file to create most of the resources.
   1. ECS Cluster using default subnet and VPC specify in network.tf
   2. Security group for the LB to allow traffic on port 80/http.
   3. ALB for the application
   4. Listener at port 80/http
   5. Target group for the loadbalancer
   6. ECR Repository for the docker image placement
   7. ECR Lifecycle policy
   8. Task definition to deploy on the ECS
   9. ECS Service and attached it to the target group
   10. Cloud Watch target for logging
   11. Setup Route53 zone and create CNAME for the ALB
3. File **network.tf** is to specify the VPC and subnets of the infrastructure.
4. File **provider.tf** is to specify the aws provider for terraform.
5. File **push.sh** is used to build docker image and push it to the repo.
6. File **app.json.tpl** is used to create the container definition.
7. File **variables.tf** is used to set the prefix, port, region, app source path and tags.

## Canary/Blue-Green Deployment.

I have setup variable for deploying the Blue-Green Deployment or setting up different environment with the same Terraform code. To achieve the canary deployment we need to create another set of container attach it to the new target group which will be part of the existing ALB. Once we will confirm the new release deployment is ready then we can switch the container target group to the already running one and get zero downtime during the deployment of new release version.

# Architecture diagram

I have created the flow diagram of the build architecture which is being used in the current challenge task.

Graphical user interface, application

Description automatically generated

# Future Work for Production Environment

1. We can separate pipelines for the Image build and deploying the new release version.
2. We can create additional listener to the ALB and point the new version target group to the listener to achieve more seamless deployment.

Additional questions:

o What would you monitor on the app?

We are

o Which metrics would you set alerts on?

o Can we deploy the whole setup in new account? (In terms of disaster

recovery)

o How would you protect the API from public access?