# FINAL ASSESSMENT TASK

Using aws terraform code to create an EC2 instance and deploy sample application. - Build docker image for sample nginx/python and push to aws elastic container registry(ECR) using docker cli. - Create VPC with 1 public and private subnet, Security Groups and EC2 instance using terraform. - Deploy Nginx Application as Docker Container using User Data script using terraform. Image should be pulled from ECR. - expected output will be access the nginx website with url http://<public\_ip>:<port>.

### STEP 1:

Using Aws Terraform code to create Ec2 instance and deploy the sample application.

- create an server with using aws cloud

- On the server install docker to create dockerfile.

- and the same server installed the terraform also for creating vpc and subnets and security groups and Ec2 instances.

- In the server install aws cli also connects with aws console.

- write dockerfile in server for creating a nginx image.

#### Vi dockerfile

FROM nginx:latest

EXPOSE 8080

CMD ["nginx", "-g", "daemon off;"]

- After completing the dockerfile give the aws credentials with using ‘aws configure’

Command if you enter this command it will your access\_key and secret\_key and aws region

And then its a successful login.

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

* Now push the nginx image that you created in dockerfile to Aws ECR by using the ECR( Elastic Container Registry) commands.

Here are the steps:

1. Retrieve an authentication token and authenticate your Docker client to your registry.

Use the AWS CLI:

aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/d5z3t2m0

1. Build your Docker image using the following command. For information on building a Docker file from scratch, see the instructions [here](http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html) . You can skip this step if your image has already been built:

docker build -t chaitanya .

1. After the build is completed, tag your image so you can push the image to this repository:

docker tag chaitanya:latest public.ecr.aws/d5z3t2m0/chaitanya:latest

1. Run the following command to push this image to your newly created AWS repository:

docker push public.ecr.aws/d5z3t2m0/chaitanya:latest

Step-2:

* Create VPC with 1 public and private subnet, Security Groups and EC2 instance using terraform
* Before creating vpc first you need terraform. The following commands can help to install.

* sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
* wget -O- https://apt.releases.hashicorp.com/gpg | \
* gpg --dearmor | \
* sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
* gpg --no-default-keyring \
* --keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
* --fingerprint
* echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
* https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | \
* sudo tee /etc/apt/sources.list.d/hashicorp.list
* sudo apt update
* sudo apt-get install terraform
* Now create a directory and go inside the directory and create a file named Main.tf and write sample code.

#### MAIN.TF

provider "aws" {

region = "ap-south-1"

}

resource "aws\_vpc" "myVpc1" {

cidr\_block = "10.0.0.0/24"

}

data "aws\_availability\_zones" "available\_zones" {}

resource "aws\_subnet" "publicSubnet1" {

vpc\_id = aws\_vpc.myVpc1.id

cidr\_block = "10.0.0.0/25"

availability\_zone = data.aws\_availability\_zones.available\_zones.names[0]

tags = {

Name = "publicSubnet1"

}

}

resource "aws\_subnet" "privateSubnet1" {

vpc\_id = aws\_vpc.myVpc1.id

cidr\_block = "10.0.0.128/25"

availability\_zone = data.aws\_availability\_zones.available\_zones.names[1]

tags = {

Name = "privateSubnet1"

}

}

resource "aws\_internet\_gateway" "myIGW1" {

vpc\_id = aws\_vpc.myVpc1.id

tags = {

Name = "myIGW1"

}

}

resource "aws\_route\_table" "myPublicRoute" {

vpc\_id = aws\_vpc.myVpc1.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.myIGW1.id

}

tags = {

Name = "myRoute"

}

}

// associate subnet with route table

resource "aws\_route\_table\_association" "myPublicRouteAssociate" {

subnet\_id = aws\_subnet.publicSubnet1.id

route\_table\_id = aws\_route\_table.myPublicRoute.id

}

resource "aws\_security\_group" "mySecureGrp" {

name = "mySecureGrp"

vpc\_id = aws\_vpc.myVpc1.id

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 8080

to\_port = 8080

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

//ipv6\_cidr\_blocks = ["::/0"]

}

tags = {

Name = "mySecureGrp"

}

}

resource "aws\_instance" "myEc2Public" {

ami = "ami-0f5ee92e2d63afc18"

instance\_type = "t2.micro"

key\_name = "terra"

subnet\_id = aws\_subnet.publicSubnet1.id

vpc\_security\_group\_ids = [aws\_security\_group.mySecureGrp.id]

associate\_public\_ip\_address = true

user\_data = <<-EOF

#! /bin/bash

echo "hello world!" > hello.txt

sudo apt-get update -y

sudo apt install docker.io -y

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/d5z3t2m0

sudo docker pull public.ecr.aws/d5z3t2m0/chaitanya:latest

sudo docker run -d -p 8080:80 public.ecr.aws/d5z3t2m0/chaitanya:latest

EOF

tags = {

Name = "ec2-publicip"

}

}

* After creating the main.tf file save it. Now initialize the code using the command

Terraform init

* After initializing the code the verify the code using the command

Terraform validate

* After validating the code to see the plan what are all created using the command

Terraform plan

* After the plan is completed next, apply the plan it automatically created whatever in the file using the command.

Terraform apply

#### Step -3

#### Deploy Nginx Application as Docker Container using User Data script using terraform. Image should be pulled from ECR.

#### 

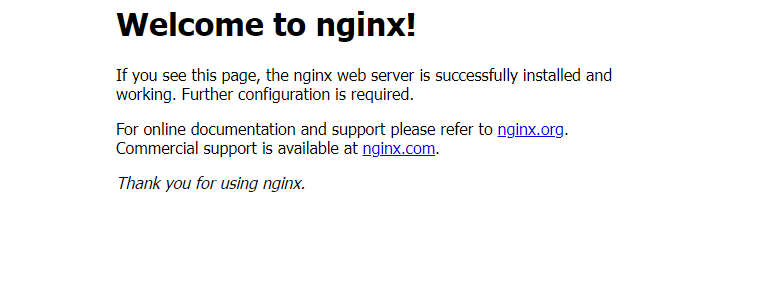
#### The file named main.tf there i have mentioned the user data

* The user data will be installed with docker and aws cli on the instance.and pull nginx image from ECR and create the container using nginx.

#### Step-4

* Now the expected output will be access the nginx website with url

http://13.126.54.157:8080/



#### Step-5

* Now push all the files to github that we created pushing and pulling the nginx image.

https://github.com/chaitanya0715/final-assessent.git