# 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

COPY . .

CMD [“echo”, ‘’hello world’’]

- 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.

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 yum install -y yum-utils
* sudo yum-config-manager --add-repo <https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo>
* sudo yum -y install terraform
* Now create a directory and go inside the directory and create file named as Main.tf

#### MAIN.TF

provider "aws" {

region = var.region

access\_key = var.access\_key

secret\_key = var.secret\_key

}

# step-1 creating VPC

resource "aws\_vpc" "my-vpc-1" {

cidr\_block = var.cidr\_block

tags = {

Name = "my-vpc"

}

}

# step-2 creating internet gateway

resource "aws\_internet\_gateway" "my-igw-1" {

vpc\_id = aws\_vpc.my-vpc-1.id

tags = {

Name = "my-igw"

}

}

#step-3 creating route table

resource "aws\_route\_table" "my-rt-1" {

vpc\_id = aws\_vpc.my-vpc-1.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.my-igw-1.id

}

tags = {

Name = "my-rt"

}

}

# step-4 creating subnet

resource "aws\_subnet" "my-subnet" {

vpc\_id = aws\_vpc.my-vpc-1.id

cidr\_block = "10.0.1.0/24"

availability\_zone = var.availability\_zone

tags = {

Name = "my-subnet-1"

}

}

#step-5: associate subnet with route table

resource "aws\_route\_table\_association" "my-rt-association" {

subnet\_id = aws\_subnet.my-subnet.id

route\_table\_id = aws\_route\_table.my-rt-1.id

}

#step-6 : creating security group and allow ports 22,80,443

resource "aws\_security\_group" "my-sg-1" {

vpc\_id = aws\_vpc.my-vpc-1.id

ingress {

description = "SSH"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

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

}

ingress {

description = "HTTP"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

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

}

ingress {

description = "HTTPS"

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

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

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "my-sg"

}

}

#step-7 : create network interface with an IP in subnet created in step-4

resource "aws\_network\_interface" "my-ni" {

subnet\_id = aws\_subnet.my-subnet.id

private\_ips = ["10.0.1.50"]

security\_groups = [aws\_security\_group.my-sg-1.id]

}

#step-8: assigning elastic ip to network interface

resource "aws\_eip" "name" {

vpc = true

network\_interface = aws\_network\_interface.my-ni.id

associate\_with\_private\_ip = "10.0.1.50"

depends\_on = [aws\_internet\_gateway.my-igw-1]

}

#step-9: creating ubuntu instance and downloading nginx

resource "aws\_instance" "ubuntu" {

ami = var.ami

instance\_type = var.instance\_type

availability\_zone = var.availability\_zone

key\_name = var.key\_name

network\_interface {

device\_index = 0

network\_interface\_id = aws\_network\_interface.my-ni.id

}

user\_data = <<-EOF

#! /bin/bash

sudo su -

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 | sudo docker login --username AWS --password-stdin public.ecr.aws/x4l2t1n6

sudo docker pull public.ecr.aws/x4l2t1n6/mkreddy:latest

sudo docker run -d -p 8080:80 public.ecr.aws/x4l2t1n6/mkreddy@sha256:9ac1131ddaab7fded0f61d509642ed09268256c5afe52aad46000b9167c7c683 i

EOF

tags = {

Name = "tf-ubuntu"

}

}

* 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

13.126.64.5:8000/

#### Step-5

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

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