# FINAL EVALUATION DOCUMENTATION

## 1. 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>. - Keep the code in the git/bb repository.

Document all the execution steps.

## Here are the steps 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.**

1. **Install Docker:** Docker is installed on your local machine. You can download and install Docker from the official Docker website.
2. **Create a Dockerfile:** Create a file named Dockerfile (without any file extension) in your project directory. Open the Dockerfile with a text editor and add the following content:

**Dockerfile:**

```
FROM nginx:latest
EXPOSE 8080
CMD ["nginx", "-g", "daemon off;"]
```

**FROM nginx:latest**

This line specifies the base image for the Docker image being built. In this case, it uses the latest version of the NGINX image available on Docker Hub. The FROM instruction sets the base image that will be used as the starting point for building the final image.

**EXPOSE 8080**

The EXPOSE instruction informs Docker that the container will listen on the specified port, in this case, port 8080. It does not actually publish the port or make it accessible from the host machine. This instruction is used to document the intended network ports that the container should listen on.

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

The CMD instruction is used to provide the default command to be executed when a container is run from the Docker image.

- nginx: The first element in the array specifies the executable or command to be run, which is nginx. It refers to the NGINX web server executable.
- -g: -g is an option flag used by NGINX to specify a global configuration directive.
- daemon off;: The third element is the value for the -g flag. Here, it sets the NGINX daemon directive to off, which means NGINX will not run in the background as a daemon. The daemon off; directive is used to keep NGINX running in the foreground when the container is started.

**3. Build the Docker image:** Open a terminal or command prompt, navigate to your project directory containing the Dockerfile, and run the following command to build the Docker image:

```
docker build -t <image name>
docker run -it -d <image name>
Docker run -it -p 8080:80 -d <image name>
```

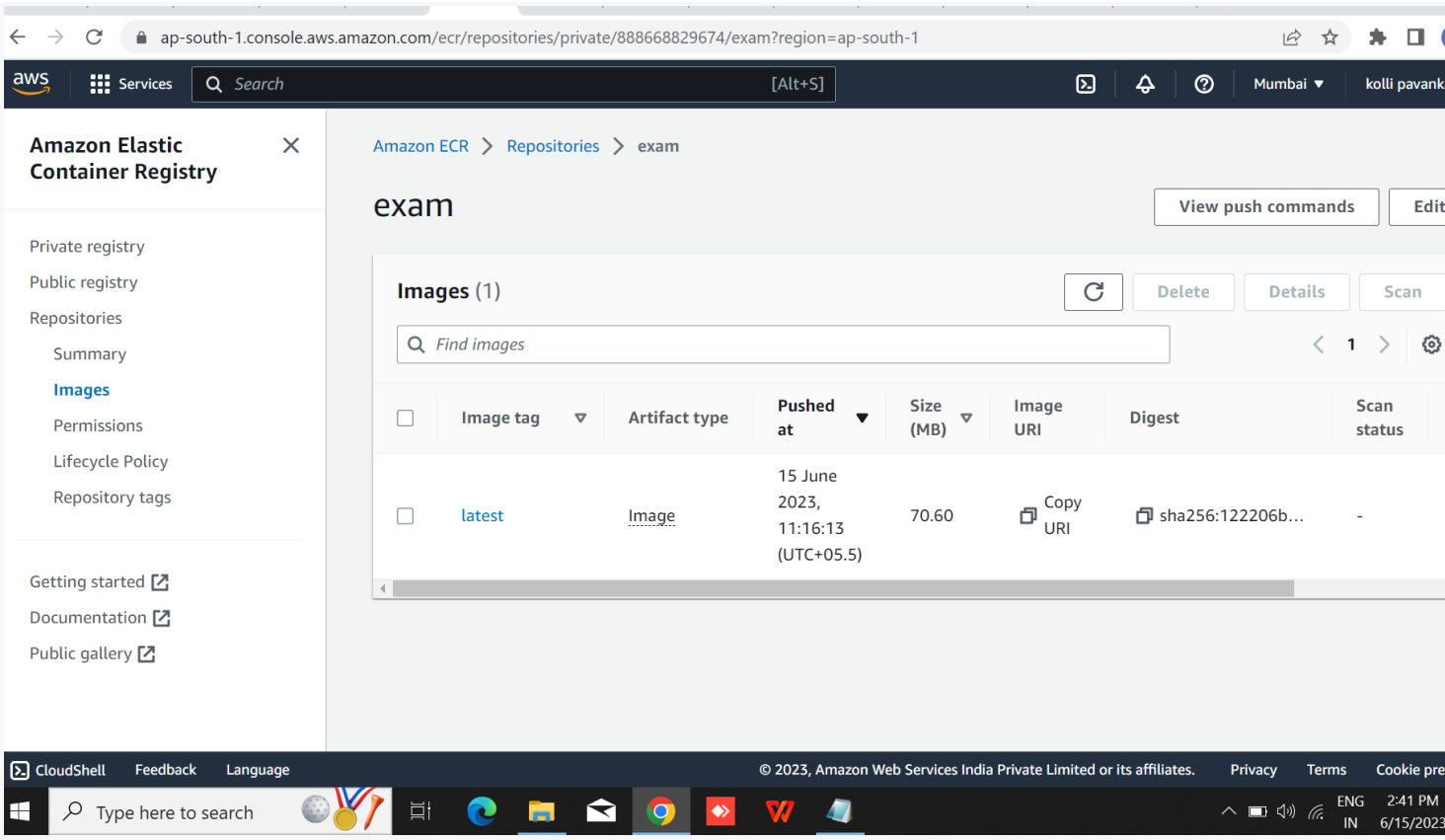
**4.Authenticate Docker with AWS ECR:** Before pushing the image, you need to authenticate Docker with your AWS ECR repository. Using following command,

aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 696303204139.dkr.ecr.ap-south-1.amazonaws.com

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

docker tag exam:latest 696303204139.dkr.ecr.ap-south-1.amazonaws.com/exam:latest

**6.Push the Docker image to ECR:** Finally, push the Docker image to your ECR repository using the following command:
docker push 696303204139.dkr.ecr.ap-south-1.amazonaws.com/exam:latest




## ➔ Create VPC with 1 public and private subnet, Security Groups and EC2 instance using terraform.

1. **Install Terraform:** download and install Terraform from the official Terraform website
2. **Create a directory:** Create a new directory where you'll store your Terraform configuration files.
3. **Create a Terraform configuration file:** Inside the directory, create a file named main.tf and open it with a text editor. Add the following code:

```
# Configure AWS provider

provider "aws" {

  region = "ap-south-1"  # Replace with your desired region
```

```
}

# Create VPC

resource "aws_vpc" "my_vpc" {

  cidr_block = "10.0.0.0/16"

  tags = {

    Name = "vpc-exam0001"

  }

}

# Create public subnet

resource "aws_subnet" "public_subnet" {

  vpc_id                  = aws_vpc.my_vpc.id

  cidr_block              = "10.0.1.0/24"

  availability_zone       = "ap-south-1a"  # Replace with your desired availability zone

  map_public_ip_on_launch = true

  tags = {

    Name = "public-subnet0001"

  }

}



# Create private subnet

resource "aws_subnet" "private_subnet" {
```

```
  vpc_id                  = aws_vpc.my_vpc.id

  cidr_block              = "10.0.2.0/24"

  availability_zone       = "ap-south-1a"  # Replace with your desired availability zone

  tags = {

    Name = "private-subnet0002"

  }

}
```

```
# Create security group for EC2 instance
```

```
resource "aws_security_group" "instance_sg" {

  name        = "instance-sg"

  description = "Security group for EC2 instance"

  vpc_id      = aws_vpc.my_vpc.id



  ingress {

    from_port   = 22

    to_port     = 22

    protocol    = "tcp"

    cidr_blocks = ["0.0.0.0/0"]  # Replace with your desired source IP range

  }



  egress {
```

```
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "instance-sg"
}
}

# Create EC2 instance
resource "aws_instance" "my_instance" {
  ami           = "ami-057752b3f1d6c4d6c"  # Replace with your desired AMI ID
  instance_type = "t2.micro"  # Replace with your desired instance type
  subnet_id              = aws_subnet.public_subnet.id
  vpc_security_group_ids = [aws_security_group.instance_sg.id]
  user_data = <<-EOF
    #!/bin/bash
    echo "Hello, World!" > index.html
    nohup python -m SimpleHTTPServer 80 &
```

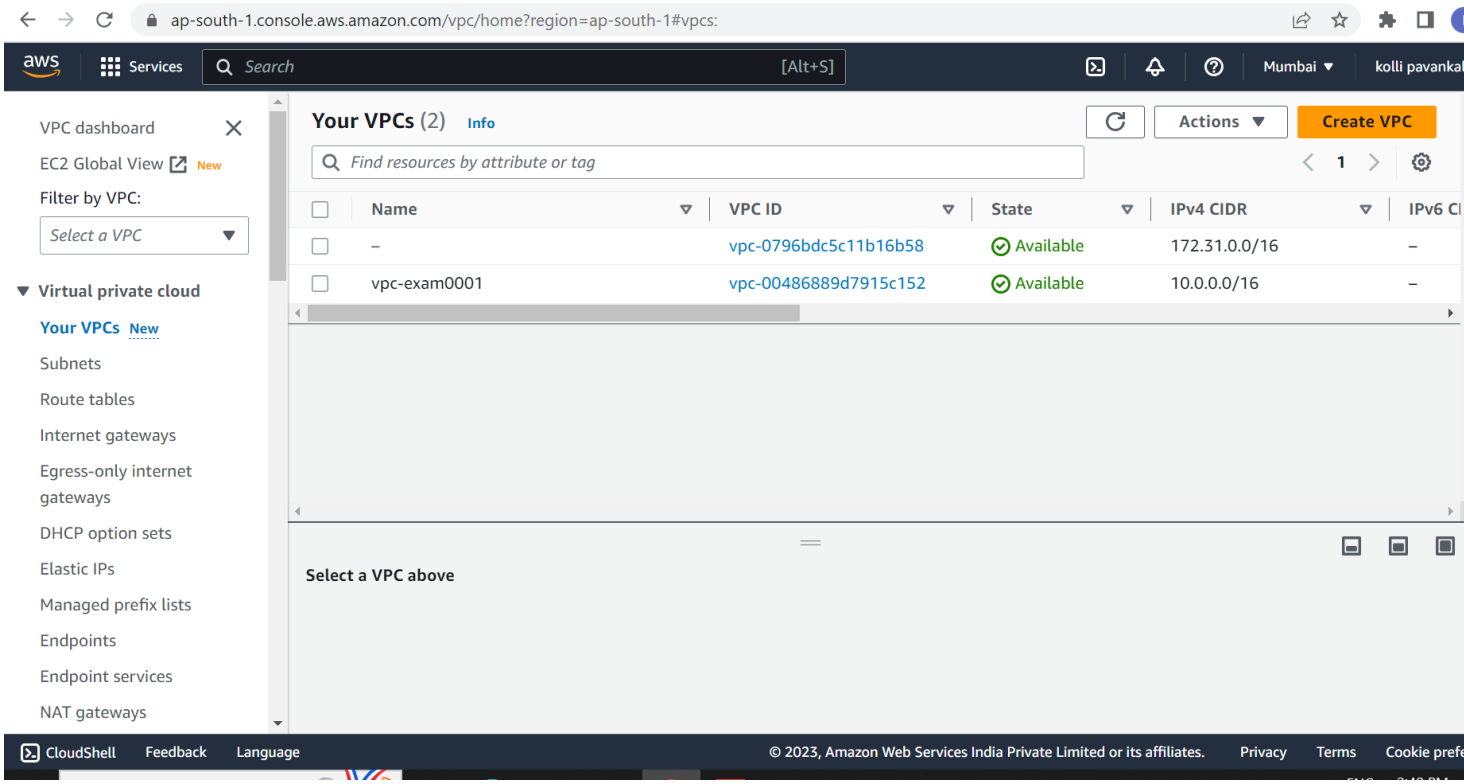
```
  EOF



tags = {

  Name = "ec2-exam-instance"

}

}
```

## Used Commands:

**terraform init:** command initializes a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

**terraform plan:** Plan lets you preview any changes before you apply them.

**terraform apply:** Apply executes the changes defined by your Terraform configuration to create, update, or destroy resources**.**



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

1. **Install Terraform:** download and install Terraform from the official Terraform website.
2. **Create a directory:** Create a new directory where you'll store your Terraform configuration files.
3. **Create a Terraform configuration file:** Inside the directory, create a file named main.tf and open it with a text editor. Add the following code:

```
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"

  }

}
```

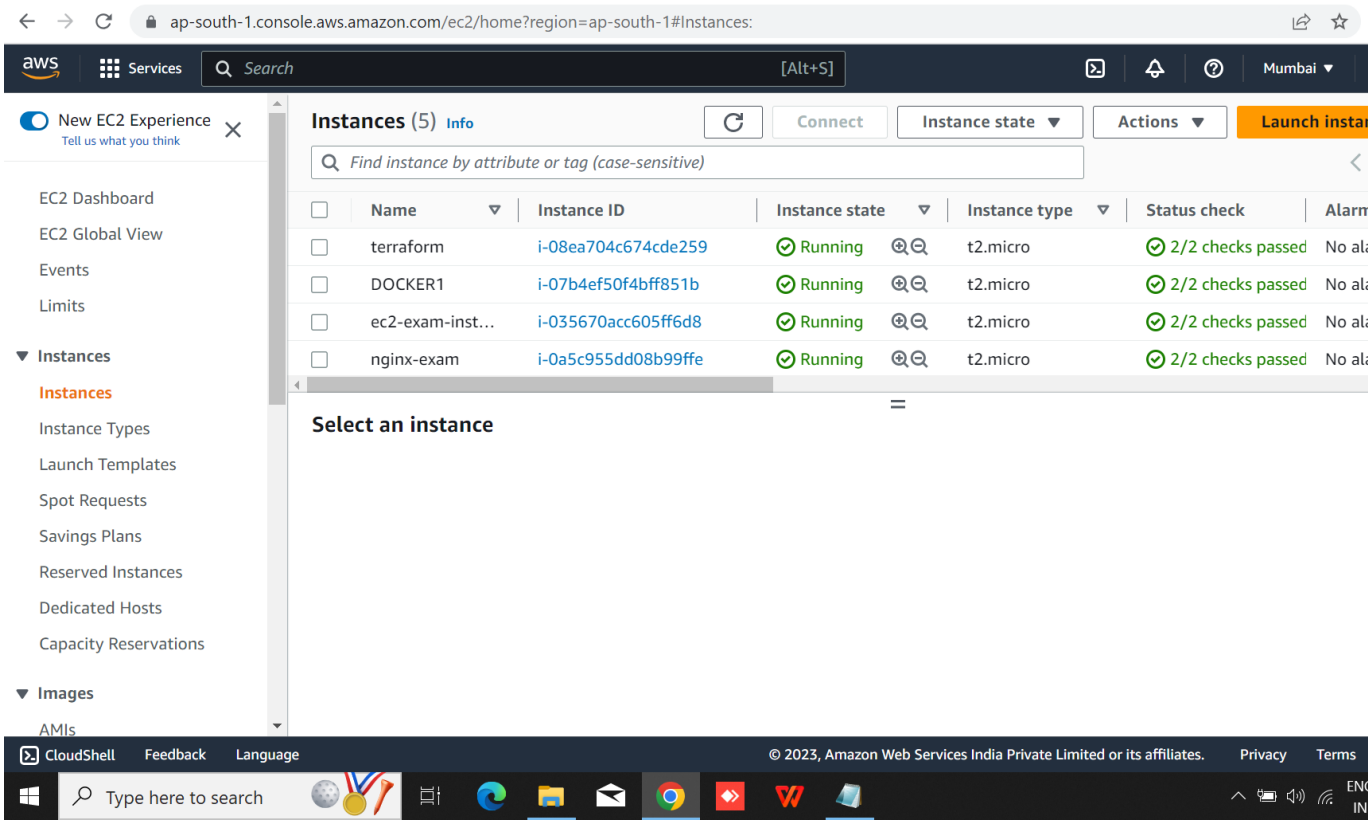
```
resource "aws_subnet" "privateSubnet1" {

  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-057752b3f1d6c4d6c"

  instance_type          = "t2.micro"

  key_name               = "terraform"

  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 | sudo docker login --username AWS
--password-stdin public.ecr.aws/b9c2h9h8
```

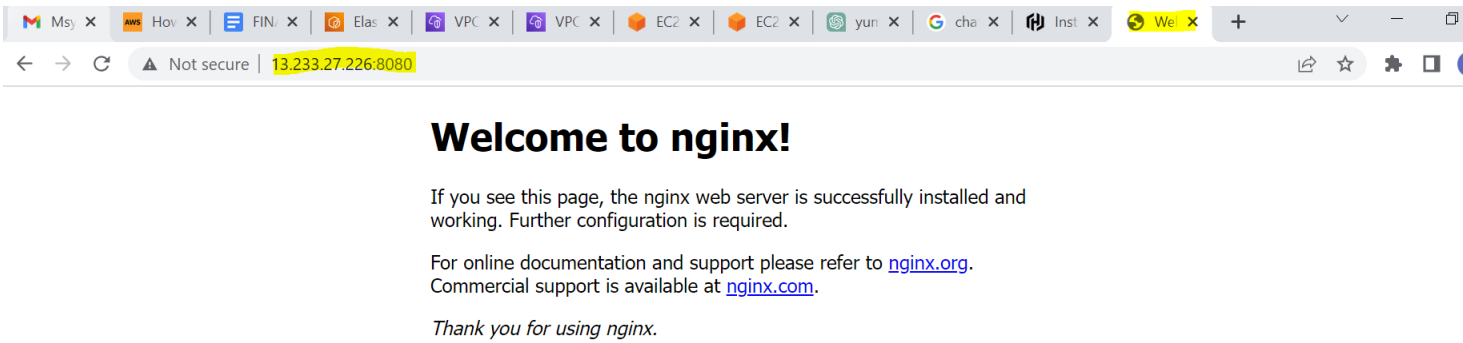
```
        sudo docker pull public.ecr.aws/b9c2h9h8/gayu_repo1:latest

        sudo docker run -d -p 8080:80 public.ecr.aws/b9c2h9h8/gayu_repo1:latest

        EOF



 tags = {

   Name = "ec2-publicip"

 }

}
```



➔ **Expected output will be access the nginx website with url http://<public_ip>:<port>. - Keep the code in the git/bb repository.**

1. Deploy Nginx on EC2 instance: Deploy an EC2 instance using the Terraform ,This will set up an EC2 instance with Nginx running inside a Docker container.
2. Obtain the public IP address: Once the EC2 instance is successfully deployed, you can obtain its public IP address from the AWS Management Console.

Public ip address:port




**Git commands:**

**Git init:-** The git init command is used to initialize a new Git repository in your current directory

**Git add:-** The git add command is used to stage changes in your working directory for the next commit. You can specify specific files or directories to add, or use options like . or * to add all changes or modifications and deletions, respectively.

**Git remote add <name> <url>:-** The git remote <name> <url> command is used to add a new remote repository to your local Git repository. The <name> parameter represents a name for the remote repository, and the <url> parameter specifies the URL of the remote repository.

**Git commit -m "comment":-**
The git commit -m "comment" command is used to create a new commit with a specified comment. The comment should describe the changes made in the commit, providing a brief and meaningful description of the modifications

**Git push -u <name> <branch>:-** This command is used to push your local branch to a remote repository specified by <name>. The -u flag sets the upstream branch for your local branch, enabling you to use git pull and git push without specifying the remote and branch names.