Ghulam Ishaq Khan Institute (GIKI)

DevOps Assignment # 4

Subject: DevOps	**Course Code:** CS - 328 – Spring - 2025
Class: BS CY **Batch:** Fall – 2022	**Submission Deadline:** 27 April 2025 Monday (11:00 AM)
Course Instructor: Muhammad Ahmad Nawaz	**Total Marks:** 15
Course TA:	

Note (Read notes & instructions first)

- First, read the instructions and statements of each exercise/question carefully then write the solution.
- *For Assignment:*
  - The name of your zip file should contain your assignment number and your roll number as shown in following example, For Example if your roll number is 2022532 and you have done assignment number 1 then the name of file should be as ---> 2022532_1.zip
    - monitor_and_push.sh
    - config.cfg
    - README.md
    - Any additional files used for testing.
  - Then upload that pdf file at Microsoft teams. Remember the sequence of pages should be right.

CHEATING/COPY CASE or LATE SUBMISSION will be graded as STRAIGHT ZERO MARKS.

DevOps Assignment: End-to-End Infrastructure as Code (IaC) with Terraform, AWS, Docker, and Git

Objective

The goal of this assignment is to design, implement, and manage a complete DevOps pipeline utilizing Terraform, AWS, Docker, Git, GitHub, and GitBash. You will establish an AWS VPC, deploy a Dockerized application on an EC2 instance, and automate the entire process using Infrastructure as Code (IaC). This assignment will assess your comprehension of DevOps principles, cloud infrastructure management, containerization, and version control.

Assignment Breakdown

1. Pre-Requisites

Before commencing, ensure you possess the following:

- **AWS Account**: Equipped with permissions to create VPCs, EC2 instances, Security Groups, etc.
- **GitBash Installed**: For executing Git commands and scripts.
- **Terraform Installed**: To define and provision AWS resources.
- **Docker Installed**: To containerize the application.
- **GitHub Account**: To host your code repository.
- **Basic Knowledge**:
  - ✓ AWS Services (VPC, EC2, IAM roles)
  - ✓ Terraform syntax and concepts
  - ✓ Docker basics
  - ✓ Git and GitHub workflows

2. Task Overview

You are required to perform the following tasks:

1. **Set up a GitHub Repository**: Create a repository to store your Terraform and Docker files.

2. **Design AWS Infrastructure with Terraform**:

- Establish a VPC with public and private subnets.
- Launch an EC2 instance within the public subnet.
- Configure security groups to allow SSH and HTTP access.

3. **Containerize a Sample Application**:

- Develop a Dockerfile for a basic web application.
- Build the Docker image and push it to Docker Hub.

4. **Deploy the Docker Container on the EC2 Instance**:

- Employ Terraform to install Docker on the EC2 instance.
- Pull and run the Docker container on the EC2 instance.

5. **Automate Everything Using GitBash**:

- Utilize GitBash to execute Terraform commands and manage the deployment process.

Step-by-Step Instructions

Step 1: Set Up GitHub Repository

1. Log in to your GitHub account.
2. Create a new repository named `devops-assignment`.
3. Clone the repository to your local machine using GitBash:

   Bash

   ```
   git clone https://github.com/<your-username>/devops-assignment.git
   cd devops-assignment
   ```

Step 2: Design AWS Infrastructure with Terraform

1. **Initialize Terraform**:

- Within your `devops-assignment` directory, create a folder named `terraform`.
- Inside `terraform`, create a file named `main.tf`.

2. **Write Terraform Code**: Incorporate the subsequent code into `main.tf` to provision a VPC, subnets, security groups, and an EC2 instance:

Terraform

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "devops-vpc"
  }
}

resource "aws_subnet" "public_subnet" {
  vpc_id          = aws_vpc.my_vpc.id
  cidr_block      = "10.0.1.0/24"
  map_public_ip_on_launch = true
  tags = {
    Name = "public-subnet"
  }
}

resource "aws_security_group" "allow_ssh_http" {
  name        = "allow_ssh_http"
  description = "Allow SSH and HTTP inbound traffic"
  vpc_id      = aws_vpc.my_vpc.id

  ingress {
    from_port   = 22
```

```
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
    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"]
  }
}

resource "aws_instance" "web_server" {
  ami             = "ami-0c02fb55956c7d316" # Amazon Linux 2 AMI
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.public_subnet.id
  key_name        = "<your-key-pair-name>"
  vpc_security_group_ids = [aws_security_group.allow_ssh_http.id]

  user_data = <<-EOF
    #!/bin/bash
    sudo yum update -y
    sudo yum install -y docker
    sudo service docker start
    sudo usermod -a -G docker ec2-user
  EOF

  tags = {
    Name = "web-server"
  }
}
```

3. **Initialize and Apply Terraform**: Open GitBash, navigate to the `terraform` folder, and execute the following commands:

Bash
```
terraform init
terraform apply
```

Confirm the changes by typing `yes` when prompted.

Step 3: Containerize a Sample Application

1. **Create a Simple Web Application**:

- In the root directory of your repository, create a folder named `app`.
- Inside `app`, create an `index.html` file with the following content:

  HTML

  ```
  <h1>Welcome to My DevOps Assignment!</h1>
  <p>This is a sample web application running inside a Docker container.</p>
  ```

2. **Write a Dockerfile**:

- Within the `app` folder, create a `Dockerfile` with the subsequent content:

  Dockerfile

  ```
  FROM nginx:alpine
  COPY index.html /usr/share/nginx/html/index.html
  EXPOSE 80
  CMD ["nginx", "-g", "daemon off;"]
  ```

3. **Build and Push the Docker Image**:

- Build the Docker image:

  Bash

  ```
  docker build -t <your-dockerhub-username>/devops-assignment:latest ./app
  ```

- Log in to Docker Hub:

  Bash

  ```
  docker login
  ```

- Push the image to Docker Hub:

  Bash

  ```
  docker push <your-dockerhub-username>/devops-assignment:latest
  ```

Step 4: Deploy the Docker Container on the EC2 Instance

1. **SSH into the EC2 Instance**: Utilize GitBash to establish an SSH connection to the EC2 instance created by Terraform:

Bash
```
ssh -i <path-to-your-key-pair>.pem ec2-user@<public-ip-of-ec2-instance>
```

2. **Pull and Run the Docker Container**: On the EC2 instance, execute the following commands:

Bash
```
sudo docker pull <your-dockerhub-username>/devops-assignment:latest
sudo docker run -d -p 80:80 <your-dockerhub-username>/devops-assignment:latest
```

3. **Verify Deployment**: Open a web browser and navigate to `http://<public-ip-of-ec2-instance>`. Your web application should be visible.

Step 5: Automate Everything Using GitBash

1. Develop a Bash script (`deploy.sh`) in the root directory of your repository to automate the Terraform and Docker deployment steps:

Bash
```
#!/bin/bash
echo "Initializing Terraform..."
terraform init

echo "Applying Terraform..."
terraform apply -auto-approve

echo "Building Docker Image..."
docker build -t <your-dockerhub-username>/devops-assignment:latest ./app

echo "Pushing Docker Image to Docker Hub..."
docker push <your-dockerhub-username>/devops-assignment:latest

echo "Deployment Complete!"
```

2. Make the script executable:

Bash
```
chmod +x deploy.sh
```

3. Execute the script:

Bash
```
./deploy.sh
```

Deliverables

Submit the following items:

1. **GitHub Repository Link**: Containing all your Terraform files, Dockerfile, and the `deploy.sh` script.
2. **Screenshots**:

- Terraform output demonstrating successful resource creation.
- A browser screenshot displaying your web application running on the EC2 instance.

3. **Explanation Document**:

- A concise explanation of each step in your implementation process.

Grading Criteria

- **Code Quality**: Terraform and Docker files that are clean and well-documented.
- **Functionality**: Successful deployment of the web application.
- **Automation**: Effective use of GitBash scripts to automate tasks.
- **Documentation**: Clear and understandable explanation of your work.

Final Notes

This assignment integrates various DevOps tools and practices, providing you with practical experience in real-world scenarios. Best of luck!