CSYE 6225: Network Structure & Cloud Computing Course

Assignment # 5

Lab Assignment: Virtualization in a Private Cloud using EC2 Instances – VM and Container Creation for Application Deployment

Objective:

The goal of this lab is to simulate a private cloud environment by creating virtual machines (VMs) and containers on EC2 instances, treating the EC2 instances as the underlying infrastructure in place of bare-metal servers. You will deploy containers for modern, lightweight application deployments.

Prerequisites:

- Basic knowledge of Docker and Docker Compose
- Access to a Linux-based system (e.g., Ubuntu) with sudo privileges
- Docker and Docker Compose installed on your system

Lab Environment Setup:

- Active AWS (Use AWS Academy Lab - DO NOT USE YOUR PERSONAL AWS ACCOUNT) account to launch EC2 instances that will serve as the "bare metal" servers in a private cloud setup.
- EC2 instances with support for nested virtualization (requires instance types such as m5, m5d, c5, or c5d). Use [m5.large]
- Install Docker and minikube (step below).

Task 1: Setting up EC2 Instances as Private Cloud Hosts for VMs and Containers

Launch EC2 Instances to Simulate Private Cloud Nodes

1. Log into AWS Console **[AWS ACADEMY NOT YOUR PERSONAL ACCOUNT]**:
    - Navigate to the AWS Management Console.
    - Under Services, search for and select EC2.
2. Launch EC2 Instance for Virtual Machines (Host 1):
    - Click Launch Instance.
    - Select an instance type that supports nested virtualization, such as m5.large or c5.large. **[Select m5.large]**
    - Choose Ubuntu 22.04 LTS (or any preferred Linux AMI) as the operating system.
    - Configure the following:
        - Storage: **50 GB** or more.
        - Network: Use the default VPC or create a new one.
    - Set up a key pair (for SSH access) or use an existing one.
    - Open the following ports in the security group:
        - Port 22 for SSH.
        - Port 80 and 443 for HTTP/HTTPS (if you plan to run web applications).
        - Port 8080 for other services (like Portainer for Docker monitoring).
3. Access the Instance via SSH:

[Screenshot#1: EC2 Instance showing its type and EBS]

Once the instance is running, connect to it using SSH:

```
ssh -i "your-keypair.pem" ubuntu@<instance-public-ip>
```

Install Docker

Update your package index:

```
sudo apt-get update
```

Install required packages:

```
sudo apt-get install -y apt-transport-https ca-certificates curl
software-properties-common
```

Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
```

Set up the stable repository:

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Update your package index again:

```
sudo apt-get update
```

Install Docker CE:

```
sudo apt-get install -y docker-ce
```

Verify Docker installation:

```
sudo docker --version
```

[Screenshot#2: Showing the proper installation of docker]

Step 2: Install Docker Compose

Download the latest version of Docker Compose:

```
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-com
pose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Apply executable permissions to the binary:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Verify Docker Compose installation:

```
docker-compose --version
```

[Screenshot#3: Showing proper installation of docker-compose]

Task 2: Setting Up a Simple Web Application with Nginx and MySQL Using Docker

Create the Project Directory

```
mkdir docker-web-mysql-app
```

```
cd docker-web-mysql-app
```

Create the Docker Compose File

Inside your project directory, create a file named docker-compose.yml:

```
touch docker-compose.yml
```

Open the docker-compose.yml file in a text editor and add the following configuration:

```
version: '3.8'

services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/share/nginx/html  # Bind mount the local html
directory
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_DATABASE: mydb
```

Create the HTML Directory and Index File

Create a directory for your HTML files:

```
mkdir html
```

Create an index.html file inside the html directory and add the following HTML content:

```
apiVersion: v1

kind: ConfigMap
metadata:
```

```
  name: web-content
data:
  index.html: |
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Simple Web App</title>
    </head>
    <body>
        <h1>Welcome to My Simple Web Application!</h1>
        <p>This application is powered by Kubernetes and MySQL.</p>
    </body>
    </html>
```

Note: Add your name in this file under h1 tag

Start the Docker Containers

Run the following command to start your containers using Docker Compose:

```
sudo docker-compose up -d
```

This command will:

- **Pull the Nginx and MySQL images if they are not already available locally.**
- **Start the containers in detached mode.**

Verify that the Containers are Running

Check the status of the running containers:

```
sudo docker ps
```

You should see two containers: one for Nginx (web) and another for MySQL (db).

[Screenshot#4:Showing the two running containers]

Test the Application

Open your web browser and navigate to:

http://<Your-EC2-Public-IP>:8080

Replace <Your-EC2-Public-IP> with the public IP address of your EC2 instance.
Note: You will need to open suitable ports for inbound traffic.

Moreover, you can use curl to test the endpoint:
curl -v http://<Your-EC2-Public-IP>:8080

You should see the following output: The content of the HTML file.

[Screenshot#5: Showing the Website from a browser revealing the IP address of your node. Also, ensure to show your name as part of the HTML - see the note in red above].

Clean Up

When you are done testing, stop the containers with:

```
sudo docker-compose down
```

This command will stop and remove the containers.

Check status again showing no containers are running:

```
sudo docker ps
```

[Screenshot#6: Showing no containers are running].

Terminate your instance.

Notes:

- **Ensure your security group settings in AWS allow inbound traffic on port 8080.**
- **If you want to connect the Nginx application to the MySQL database, you'll need to implement additional application logic, which could involve a backend language like PHP or Node.js. However, this assignment focuses solely on serving a static HTML page with Nginx.**

Grading:

- No late assignments are accepted.
- 5 points for each required screen shot (6 screenshots - max 30 points).
- 10 points for documented screen shots (a couple statements about each screenshot).

End Assignment # 5