Department of Computer Science and Engineering (Data Science)

AY: 2025-26

Semester: V

Subject: DevOps Laboratory (DJS23OLOE501)

Experiment 1b

(Virtualization and Containerization with Docker)

Aim: To Build a Custom Docker Image and Use Docker Compose for Multi-Container Applications.

Theory:

Introduction to Docker

Before Docker, there was a big issue faced by most developers whenever they created any code that code was working on that developer computer, but when they try to run that particular code on the server, that code was not working. This happened because apps need the right environment to run (like the right OS, libraries, and settings). If something was different on your computer vs. the server, things would break.
To solve this issue, Docker container comes. Docker is a popular open-source containerization platform. It enables developers to package, ship, and run applications in isolated environments called containers. Docker is an open-source containerization platform by which you can pack your application and all its dependencies into a standardized unit called a container. Containers are light in weight, which makes them portable, and they are isolated from the underlying infrastructure and from each other's container. You can run the docker image as a docker container in any machine where docker is installed without depending on the operating system. There are two big pieces to Docker: The Docker Engine, which is the Docker binary that's running on your local machine and servers and does the work to run your software. The Docker Hub is a website and cloud service that makes it easy for everyone to share their docker images.

Key Components of Docker

The following are the some of the key components of Docker:

- Docker Engine: Docker Engine is a core part of docker, that handles the creation and management of containers.
- Docker Image: Docker Image is a read-only template that is used for creating containers, containing the application code and dependencies.
- Docker Hub: It is a cloud based repository that is used for finding and sharing the container images.
- Dockerfile: It is a file that describes the steps to create an image quickly.
- Docker Registry : It is a storage distribution system for docker images, where you can store the images in both public and private modes.

Department of Computer Science and Engineering (Data Science)

What is a Dockerfile?

The Dockerfile uses DSL (Domain Specific Language) and contains instructions for generating a Docker image. Dockerfile will define the processes to quickly produce an image. While creating your application, you should create a Dockerfile in order since the Docker daemon runs all of the instructions from top to bottom.
The Dockerfile is the source code of the image.
(The Docker daemon, often referred to simply as "Docker," is a background service that manages Docker containers on a system.)

- It is a text document that contains necessary commands which on execution help assemble a Docker Image.
- Docker image is created using a Dockerfile.

What is Docker Image?

It is a file, comprised of multiple layers, used to execute code in a Docker container. They are a set of instructions used to create docker containers. Docker Image is an executable package of software that includes everything needed to run an application.
This image informs how a container should instantiate, determining which software components will run and how. Docker Container is a virtual environment that bundles application code with all the dependencies required to run the application. The application runs quickly and reliably from one computing environment to another.

What is Docker Container?

Docker container is a runtime instance of an image. Allows developers to package applications with all parts needed such as libraries and other dependencies. Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way.

What is Docker Compose?

Docker Compose is a tool that makes it easier to define and manage multi-container Docker applications. It simplifies running interconnected services, such as a frontend, backend API, and database, by allowing them to be launched and controlled together.
Using a YAML configuration file (typically docker-compose.yml), you can describe each service and its dependencies as code. This setup can be committed to your source repository for consistent deployments. Once defined, all services can be started with a single docker-compose command, making it easier to coordinate development or testing environments.

Department of Computer Science and Engineering (Data Science)

Docker Compose File (YAML Format)

Docker Compose configurations are mainly stored in a file named docker-compose.yml, which uses YAML format to define an application's environment. This file includes all the necessary details to set up and run your application, such as services, networks, and volumes. To use Docker Compose in an effective way you have to know the structure of this file.

Key elements of the YAML Configuration:
Version: It defines the format of the Compose file, by ensuring compatibility with specific Docker Compose features and syntax.
Services: The services section lists each containerized service required for the application. Each service can have its configuration options, such as which image to use, environment variables, and resource limits.
Networks: In network section, you can define custom networks that enable communication between containers. Additionally, it allows you to specify network drivers and custom settings for organizing container interactions.
Volumes: Volumes allow for data persistence across container restarts and can be shared between containers if needed. They enable you to store data outside the container's lifecycle, making it useful for shared storage or preserving application state.

1. **Build a custom Docker image for a Python web application.**
   Steps:

1. **Install Nano**

sudo apt update
sudo apt install nano -y

2. **Create App Files on the VM**

Run the following commands in your SSH terminal:
mkdir myflaskapp
cd myflaskapp

Then create the Python app:
nano app.py

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello from Docker in GCP VM!"

if __name__ == '__main__':
```

app.run(host='0.0.0.0')

Save and exit (press Ctrl+O, then Enter, then Ctrl+X).

3. **Create Dockerfile**

nano Dockerfile

```
FROM python:3.9-slim

WORKDIR /app
COPY app.py /app

RUN pip install flask

CMD ["python", "app.py"]
```

Save and exit (Ctrl+O, Enter, Ctrl+X).

4. **Build the Docker Image**

sudo docker build -t my-python-app .

Confirm if image was created

sudo docker images

5. **Run the Docker Container**

sudo docker run -d -p 5000:5000 my-python-app

Check if the container is running

sudo docker ps

6. **Allow Traffic on Port 5000 (GCP Firewall)**

- Create Firewall Rule for Port 5000 (if not done)
- Go to the GCP console and do this:
- VPC Network → Firewall Rules → Create Firewall Rule
- Name: allow-port-5000
- Direction: Ingress
- Targets: All instances
- Source IP ranges: 0.0.0.0/0
- Protocols and ports:
- Check ☑ Specified protocols and ports
- TCP: 5000
- Click Create

7. **Access Your Flask App**

Open a browser and go to:

http://<your-external-ip>:5000
(Find external IP on VM page in GCP)

8. **Output on browser:**

Hello from Docker in GCP VM!

2. **Use Docker Compose to Run Multi-Container Application**

- Run your Flask app
- Alongside a Redis container (as an example backend service)

1. **Update your app to use Redis**

nano app.py

```
from flask import Flask
import redis

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = cache.incr('hits')
    return f'Hello from Docker! This page has been viewed {count} times.'

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

2. **Update your Dockerfile**

nano Dockerfile

```
FROM python:3.9-slim

WORKDIR /app
COPY . /app

RUN pip install flask redis

CMD ["python", "app.py"]
```

3. **Create docker-compose.yml**

nano docker-compose.yml

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

4. Build and Start with Docker Compose

sudo docker compose up --build

5. Visit your app

http://<your-external-ip>:5000

6. You should now see a counter:

Hello from Docker! This page has been viewed 1 times.

Refresh the page — the number will increase.

Lab experiment to be performed in this session:

1. Build a Docker image that runs a simple Node.js HTTP server returning **"Hello from Node!".**

```
walamsoham@exp1:~/node-server$ curl http://34.71.213.139:3000
Hello from Node!
```

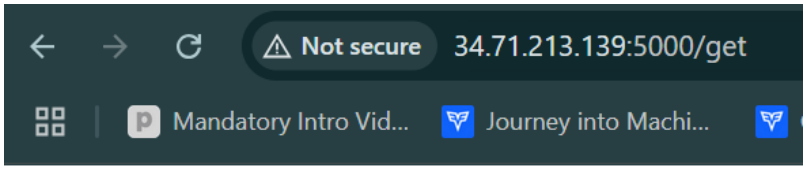
2. Build a simple web service using Docker Compose that allows a user to store and retrieve their name using Flask and Redis.

   Perform the following tasks:

   a) Create a Python Flask application with the following two routes:

   - /set/<name>: Stores the provided name in Redis.
   - /get: Retrieves and displays the stored name from Redis.

```
walamsoham@exp1:~/flask-app$ curl http://34.71.213.139:5000/set/Soham
curl http://34.71.213.139:5000/get
Name 'Soham' stored in Redis!
Stored name: Soham
```



Stored name: Soham

Name 'Soham' stored in Redis!

b) Write a Dockerfile to containerize the Flask app.

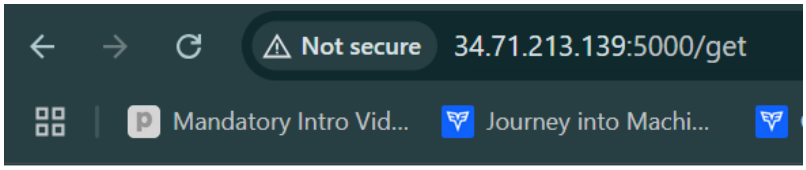
```
# Use Python base image
FROM python:3.10-slim

# Set working directory
WORKDIR /app

# Copy requirements file (optional if you have one)
COPY requirements.txt req1.txt

# Install dependencies
RUN pip install --no-cache-dir -r req1.txt

# Copy app files
COPY . .

# Expose Flask port
EXPOSE 5000

# Run Flask app
CMD ["python", "app.py"]
```

c) Write a docker-compose.yml file to:

- Build and run the Flask container.
- Run a Redis service.
- Expose the Flask app on port 5000.

d) Use a web browser to:

- Set your name using the /set/<name> route.



Stored name: Soham

- Verify the stored name using the /get route.

Name 'Soham' stored in Redis!