Placement Empowerment Program

Cloud Computing and DevOps Centre

Build and Deploy a Multi-Tier Application Using Docker Compose: Use Docker Compose to build a web application and connect it to a database.

Name: Saravana Krishnan J Department: IT

Introduction

In modern application development, **containerization** plays a crucial role in simplifying deployment and scaling. **Docker Compose** allows developers to define and manage multi-container applications using a simple YAML file. This Proof of Concept (PoC) demonstrates how to build and deploy a **multi-tier application** using **Docker Compose**, connecting a **Flask web application** to a **PostgreSQL database** in a containerized environment.

Overview

This PoC involves:

1. A Flask Web Application – A Python-based web app that interacts with a database.

2. A PostgreSQL Database – A relational database running as a separate service in a Docker container.

3. Docker & Docker Compose – The web and database services are containerized and orchestrated using `docker-compose.yml`.

Using **Docker Compose**, we can build, run, and manage both services seamlessly, ensuring smooth communication between the application and the database.

Objectives

1. **Containerize a web application and a database** using Docker.

2. **Use Docker Compose** to define multi-container applications.

3. **Deploy and test the application** locally with containerized services.

4. **Ensure communication** between the web app and database inside Docker containers.

5. **Simplify deployment** by automating multi-container setups with a single command.

Importance

1. **Real-world Use Case**: Multi-tier applications are commonly used in production environments (e.g., web apps with databases).

2. **Portability**: With containerization, the app runs identically on any machine with Docker.

3. **Scalability**: Services can be easily scaled by modifying `docker-compose.yml`.

4. **Efficient Deployment**: Eliminates dependency issues by packaging everything in containers.

5. **DevOps Readiness**: This PoC aligns with DevOps practices by automating environment setup and deployment.

Step-by-Step Overview

Step 1:

Check if Docker is installed

Open Command Prompt (CMD) and run:

docker --version

Now, check Docker Compose:

docker-compose --version

If you don’t have Docker installed, download and install it from: □ Docker Desktop for Windows

Once installed, restart your computer and ensure Docker Desktop is running.

```
C:\Users\Hi>docker --version
Docker version 27.5.1, build 9f9e405

C:\Users\Hi>docker-compose --version
Docker Compose version v2.32.4-desktop.1
```

Step 2:

Navigate to your preferred location (e.g., Desktop)

Create a new folder named multi-tier-app:

mkdir multi-tier-app
cd multi-tier-app

```
C:\Users\Hi>cd desktop

C:\Users\Hi\Desktop>mkdir multi-tier-app

C:\Users\Hi\Desktop>cd multi-tier-app
```

Step 3:

Inside multi-tier-app, create another folder named app:

mkdir app
cd app

```
C:\Users\Hi\Desktop\multi-tier-app>mkdir app

C:\Users\Hi\Desktop\multi-tier-app>cd app
```

Step 4:

Inside the app folder, create a file named **app.py**.

notepad app.py

```
C:\Users\Hi\Desktop\multi-tier-app\app>notepad app.py
```

app.py

File Edit View

```
from flask import Flask
import psycopg2

app = Flask(__name__)

def get_db_connection():
    conn = psycopg2.connect(
        dbname="mydatabase",
        user="myuser",
        password="mypassword",
        host="db",  # Refers to PostgreSQL container
        port="5432"
    )
    return conn

@app.route('/')
def index():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT 'Hello, Docker Compose!'")
    message = cur.fetchone()[0]
    cur.close()
    conn.close()
    return message

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 5:

We need a list of dependencies for our Flask app.

1. Run:

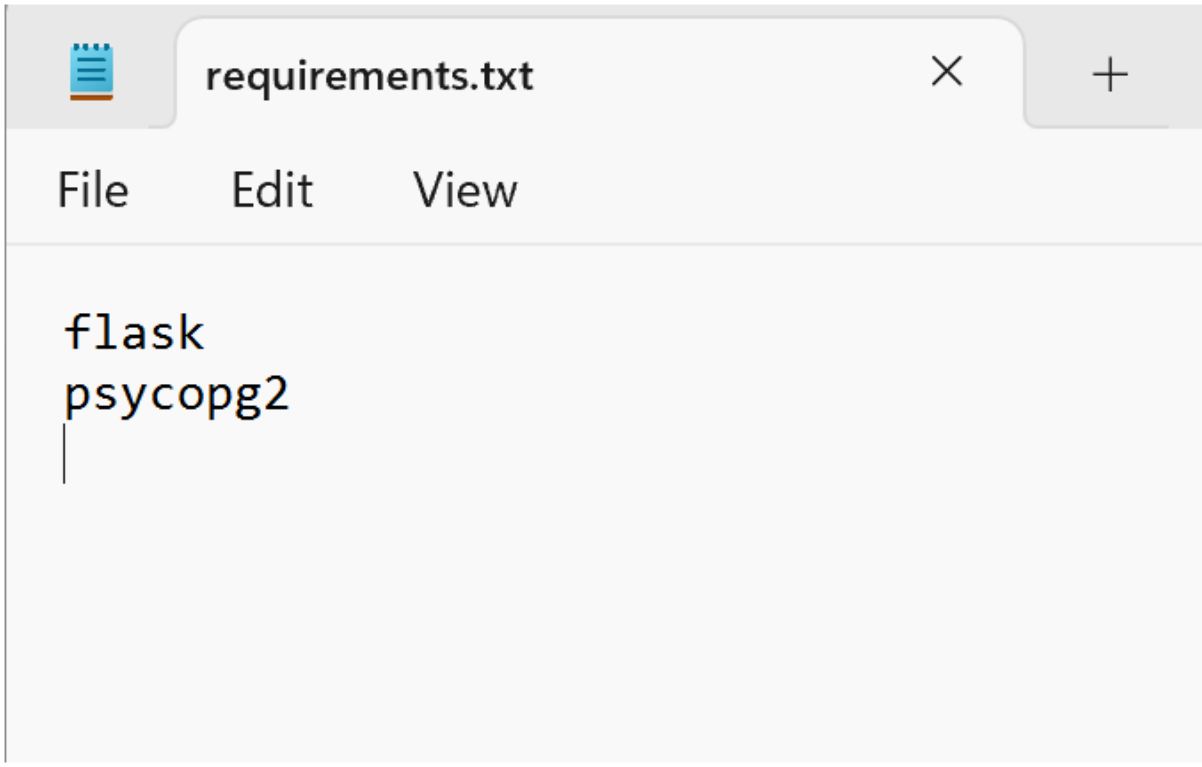
notepad requirements.txt

2. Paste this:

flask
psycopg2

3. Save and close.

```
C:\Users\Hi\Desktop\multi-tier-app\app>notepad requirements.txt
```



Step 6:

Now, create a Dockerfile inside the app folder.

1. Run:

notepad Dockerfile

2. Paste this

3. Save and close.

```
C:\Users\Hi\Desktop\multi-tier-app\app>notepad Dockerfile
```

Dockerfile

File Edit View

```
# Use Python 3.9 as base image
FROM python:3.9

# Set the working directory inside the container
WORKDIR /app

# Copy application files to the container
COPY . .

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

# Define Flask environment variable
ENV FLASK_APP=app.py

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

Step 7:

Now, navigate back to multi-tier-app:

cd ..

1. Run:

notepad docker-compose.yml

2. Paste this content:

3. Save and close.

```
C:\Users\Hi\Desktop\multi-tier-app\app>cd..

C:\Users\Hi\Desktop\multi-tier-app>notepad docker-compose.yml
```

docker-compose.yml

File Edit View

```yml
version: '3.8'

services:
  web:
    build: ./app
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://myuser:mypassword@db:5432/mydatabase

  db:
    image: postgres:13
    restart: always
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydatabase
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Step 8:

Now, **ensure Docker Desktop is running**. Then, from the multi-tier-app folder, run:

docker-compose up --build

 What Happens?

- The **Flask app** is built and runs in a container.
- The **PostgreSQL database** starts as another container.
- Both services are connected.

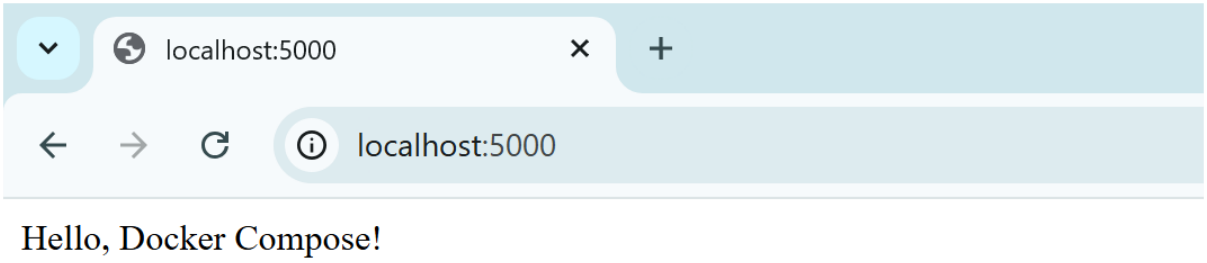
Wait until you see logs indicating that both containers are running.

```
C:\Users\Hi\Desktop\multi-tier-app>docker-compose up --build
```

Step 9:

Once the containers are running, **open a browser** and visit:

http://localhost:5000



Step 10:

Press Ctrl+C two times to cancel the process.

```
Gracefully stopping... (press Ctrl+C again to force)
[+] Stopping 2/2
 ✔Container multi-tier-app-web-1  Stopped
 ✔Container multi-tier-app-db-1   Stopped
canceled
```

Step 11:

To **stop the application**, run:

docker-compose down

```
C:\Users\Hi\Desktop\multi-tier-app>docker-compose down
time="2025-03-01T10:18:12+05:30" level=warning msg="C:\\Users\\Hi\\Desktop\\multi-tier-app\\docker-compose.yml: the attr
ibute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
 ✔Container multi-tier-app-web-1  Removed                                                                    0.0s
 ✔Container multi-tier-app-db-1   Removed                                                                    0.0s
 ✔Network multi-tier-app_default  Removed                                                                    0.7s
```

Summary

1. We **installed** Docker and verified the setup.

2. We **created** a Flask web application.

3. We **defined** a PostgreSQL database.

4. We **built** and **deployed** the application using Docker Compose.

5. We **tested** the setup successfully.

Outcomes

By completing this PoC, you will:

1. **Understand Multi-Tier Application Architecture** – Gain hands-on experience in designing and deploying a **web application with a separate database layer** using containerization.

2. **Use Docker Compose for Multi-Container Deployment** – Learn how to define and manage multiple services (**web app and database**) in a docker-compose.yml file and orchestrate them with a single command.

3. **Deploy and Run a Flask Web Application in Docker** – Build a **Flask-based web application**, containerize it with a Dockerfile, and run it as a service inside a Docker container.

4. **Set Up and Configure a PostgreSQL Database in a Container** – Deploy a **PostgreSQL database** in a separate container, expose necessary ports, and configure authentication and database settings using environment variables.

5. **Establish Communication Between Containers** – Ensure seamless interaction between the **Flask web app** and the **PostgreSQL database** by leveraging Docker's networking capabilities.