Experiment No. 3

Title:

Design, Deploy, and Manage a microservices architecture on your local machine using Docker and docker-compose

Objective:

To simulate the deployment of a basic microservice application (e.g., frontend + database) using Docker containers, orchestrated with Docker Compose on a local development environment.

Pre-requisites:

- Docker and Docker Compose must be installed on your machine.
- Basic understanding of how microservices work.
- A sample microservice-based project (commonly consists of at least two services, like a frontend and a backend or database).

Step-by-Step Procedure:

Step 1: Start

Begin with a clear understanding of microservices – independent components that interact with each other through APIs, often using containers for isolated deployment.

Step 2: Create Project Structure

```
mkdir microservice-app
cd microservice-app
mkdir app db
```

- Create a root directory to hold your project.
- Inside it, create folders for each microservice (e.g., `app` for the web service and `db` for the database).

Step 3: Write Flask App (Microservice Code)

```
# File: app/app.py
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home():
    return "Hello from Microservice App!"
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- A basic Python web app using Flask.
- Runs on port 5000 and responds with a greeting.

Step 4: Add requirements.txt

```
flask
```

- Lists the Python dependencies to install inside the Docker container.

Step 5: Create Dockerfile

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

- Creates an image with Python installed.
- Copies your code and installs Flask.
- Sets the entry point for container startup.

Step 6: Configure MySQL Database

```
  db:
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: sampledb
      MYSQL_USER: user
      MYSQL_PASSWORD: password
      MYSQL_ROOT_PASSWORD: rootpass
```

- A MySQL container simulating backend storage.
- Uses default environment variables to set DB name and access.

Step 7: Create docker-compose.yml

```
version: '3'
services:
  web:
    build: ./app
    ports:
      - "5000:5000"
    depends_on:
      - db
  db:
    image: mysql:5.7
        restart: always
    environment:
      MYSQL_DATABASE: sampledb
      MYSQL_USER: user
      MYSQL_PASSWORD: password
      MYSQL_ROOT_PASSWORD: rootpass
    ports:
      - "3306:3306"
```

- Describes the architecture: 1 custom-built Flask app + 1 MySQL service.
- Automatically starts and links both containers.

Step 8: Run the Setup

```
docker-compose up --build
```

- Builds the image from the Dockerfile.
- Starts both services as defined.
- Containers run in isolated but networked environments.

Step 9: Access App in Browser

- Visit: http://localhost:5000
- The response `Hello from Microservice App!` confirms the web service is functional.

Step 10: Stop Containers

```
docker-compose down
```

- Gracefully shuts down and removes containers.
- Frees up system resources and ports.

Impact of Each Statement:

Command	Impact
docker-compose up --build	Builds and starts all services defined in the Compose file
docker ps	Lists all running containers (for verification)
docker-compose down	Stops and removes all containers, networks, and volumes created