LAB TASK – 11: Dockerizing a MERN Stack Application using Docker Compose

Part 1: Set Up and Dockerize a MERN Stack Application

1. **Clone the Repository**
   Clone the following MERN stack application from GitHub:
   https://github.com/atanu3000/MERN_Notes_App.git
2. **Create Required Configuration Files**
   Inside the project, create the following files:

   - .env in the root folder.
   - .dockerignore for the frontend, backend, root.
   - Dockerfile for both the frontend and backend.

Note:
The frontend Dockerfile uses a **multi-stage build**:

- **Stage 1 (Build Stage):** Uses node:18-alpine to install dependencies and build the Vite React app into static files (dist folder).
- **Stage 2 (Production Stage):** Uses nginx:alpine to serve the static files from dist via NGINX.

This approach ensures the final image is lightweight and production-ready, containing only the compiled frontend assets without Node.js or development dependencies.

3. **Configure Docker Compose**

   - Create a .env file in the **root directory** (where docker-compose.yml is located) that defines **VITE_APP_BASE_URL**. This value will be passed to the React app as a build-time argument and must begin with VITE_APP_ to be used inside the React application.
   - Create a **docker-compose.yml** file in the root directory to define and connect services for MongoDB, Node.js, and React.js.

4. **Run the Application Using Docker Compose**
   Use Docker Compose to build and run all services. Ensure that the application is up and accessible.

5. Test the app in browser and inspect containers in Docker Desktop.

Note: To test the app you need to change the VITE Base URL in frontend Home Page component as it is retrieving notes from cloud.

Example:

const API_URL = import.meta.env.VITE_APP_API_URL; //note the changes

.post(`${API_URL}/addNote`, note)

➔ **USE import.meta.env instead of process.env for vite applications**

Part 2: Docker Networking

Inspect Docker networks:

- After running Compose, use:

  docker network ls

  docker network inspect <network-name>

Then Answer:

- What is the subnet/CIDR block used?
- Which containers are in the same network?

Create custom bridge networks:

- Create two custom networks:

  docker network create --subnet=192.168.100.0/24 net1

  docker network create --subnet=192.168.200.0/24 net2

- Use docker-compose.yml to connect all services to net1/net2 **(Recommended)**

Manually:

Connect your mongo container to both net1 and the default Compose network using:

docker network connect net1 <mongo-container>

Try pinging between containers, for this you need to:

Check base system

cat /etc/os-release

This will tell you whether it's Alpine, Debian, etc

If it's Alpine:

Run this inside the container:

apk update && apk add iputils

If it's Debian or Ubuntu:

Run this inside the container:

apt update && apt install iputils-ping -y

After installing, you can do:

ping mongo2 etc.

Execute the Command: ***docker network inspect net1:*** You can see all services listed under Containers.

Part 3: Explore and Modify an Existing Compose Project

Repository: https://github.com/bezkoder/docker-compose-nodejs-mysql

This project demonstrates how to set up a **Node.js** application with a **MySQL** database using Docker Compose.

1. **Clone the Repository: git clone** https://github.com/bezkoder/docker-compose-nodejs-mysql.git
2. **Explore the docker-compose.yml File:**
    - Open the docker-compose.yml file to understand the service configurations for Node.js and MySQL.
3. **Modify the Compose File:**
    - **Add phpMyAdmin Service:**

      Integrate phpMyAdmin to provide a web interface for managing the MySQL database.
4. **Adjust Environment Variables:**
    - Update the environment variables to match your desired database credentials and settings.
5. **Add Networks and Volumes:**

- Define custom networks and volumes to manage communication between services and persist data.
- Assign the services to the network and mount volumes accordingly.

6. **Build and Run the Application**

Access the Node.js application at http://localhost:3000 and phpMyAdmin at http://localhost:8080

7. **Document the Changes:**

- Record the original and modified docker-compose.yml files.
- Provide explanations for each change made.
- Include screenshots demonstrating the running application and phpMyAdmin interface.