Multi-Container Application Deployment with Docker

Manually configure and deploy **multiple containers** (a web server, a backend, and a database) using Docker, managing their communication via a Docker network

📌 You will deploy **three containers** that interact with each other:

1️⃣ **MySQL database**
2️⃣ **Backend in Node.js**
3️⃣ **Nginx web server**

Part 1: Setting Up the Environment

1. Install Docker on your system and verify the installation:

```
docker --version
```

Submission: Screenshot showing the installed Docker version.

Part 2: Creating the Application

You will deploy three containers that interact with each other:
1. MySQL database
2. Backend in Node.js
3. Nginx web server

Step 1: Create a Docker Network
Before running the containers, you need to create a custom network so they can communicate:

```
docker network create my-network
```

Submission: Screenshot showing the created network (docker network ls).

Step 2: Launch the Database (MySQL)
Run the MySQL container with a database named testdb:

```
docker run -d --name db --network my-network \
  -e MYSQL_ROOT_PASSWORD=rootpassword \
  -e MYSQL_DATABASE=testdb \
  -p 3306:3306 \
  mysql:5.7
```

Submission: Screenshot of docker ps showing the running container.

Step 3: Create the Backend in Node.js

Create a server.js file inside a new folder (backend/):

```
const express = require("express");
const mysql = require("mysql2");
const app = express();

const db = mysql.createConnection({
    host: "db",
    user: "root",
    password: "rootpassword",
    database: "testdb"
});

app.get("/", (req, res) => {
    db.query("SELECT 'Hello from MySQL' AS message", (err, result) => {
        if (err) throw err;
        res.json(result[0]);
    });
});

app.listen(3000, () => console.log("Backend running on port 3000"));
```

Create a Dockerfile inside the same folder:

```
FROM node:18
WORKDIR /app
COPY . .
RUN npm install express mysql2
CMD ["node", "server.js"]
```

Build the image and run the container:

```
docker build -t my-backend ./backend
docker run -d --name backend --network my-network -p 3000:3000 my-backend
```

Submission:
- Screenshot of docker images showing the created image.
- Screenshot of docker ps showing the running backend.

Step 4: Configure the Web Server with Nginx
Create a default.conf file inside a new folder (nginx/):

```
server {
    listen 80;
    location / {
        proxy_pass http://backend:3000;
    }
}
```

Create a Dockerfile inside nginx/:

```
FROM nginx:latest
COPY default.conf /etc/nginx/conf.d/default.conf
```

Build the image and run the container:

```
docker build -t my-nginx ./nginx
docker run -d --name web --network my-network -p 8080:80 my-nginx
```

Submission:
- Screenshot of docker ps showing the running web server.
- Screenshot of the browser accessing http://localhost:8080 and displaying the backend response.

Part 3: Management and Verification

Check that all containers are running:

```
docker ps
```

Inspect the network and verify the connections:

```
docker network inspect my-network
```

Submission:
- Screenshot of docker ps showing all three running containers.
- Screenshot of docker network inspect my-network showing the connection between containers.

Part 4: Demonstration

- Architecture diagram of this deployment with draw.io
- Short 2 minute video showcasing the example, using your words to describe it.