EXPERIMENT NO. 4

Title: Use Docker Compose to Link a Web Application with a MySQL Database.

Aim: To demonstrate how Docker Compose is used to connect a web application with a MySQL database in a multi-container environment.

Theory:

Docker Compose is a tool used to define and manage multi-container Docker applications. It uses a YAML file (docker-compose.yml) to configure the services (like a web server and a database), volumes, and networks. This eliminates the need to run containers manually using multiple Docker commands.

Benefits of Docker Compose:

- Easy management of service dependencies.
- Central configuration of services, ports, volumes.
- One-command setup using docker-compose up.

In this experiment:

- We will build a **PHP + Apache web app** that connects to a **MySQL** database.
- Use Docker Compose to manage both services.
- Demonstrate service linking and communication.

Steps of Execution:

1. Create a project directory with required files.
2. Write a simple web app in PHP to connect with MySQL.
3. Write a Dockerfile to containerize the PHP app.
4. Define services using docker-compose.yml.
5. Build and run the application using Docker Compose.
6. Validate the connection to MySQL through browser output.

Stepwise Procedure with Outputs:

Step 1: Create Project Structure

bash

CopyEdit

```
mkdir docker-compose-webapp
cd docker-compose-webapp
mkdir web
```

Output:

CopyEdit

```
docker-compose-webapp/
└── web/
```

Step 2: Create index.php (PHP Web App)

Path: web/index.php

php

CopyEdit

```
<?php
$servername = "db";
$username = "root";
$password = "rootpass";
$dbname = "testdb";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo " Connected successfully to MySQL!";
?>
```

Output (in browser later):

css

CopyEdit

```
Connected successfully to MySQL!
```

Step 3: Create Dockerfile for PHP App

Path: web/Dockerfile

dockerfile

CopyEdit

```
FROM php:8.2-apache
COPY index.php /var/www/html/
```

Output (when built):

ruby

CopyEdit

```
 => [internal] load build context
 => [1/2] FROM docker.io/library/php:8.2-apache
 => [2/2] COPY index.php
```

Step 4: Create docker-compose.yml

Path: docker-compose.yml

yaml

CopyEdit

```
version: '3.8'
services:
  web:
    build: ./web
    ports:
      - "8080:80"
    depends_on:
      - db
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: testdb
    volumes:
      - db-data:/var/lib/mysql
volumes:
  db-data:
```

Output: No direct output, file used by Compose engine.

Step 5: Run Docker Compose

bash

CopyEdit

```
docker-compose up –build
```

Output:

ruby

CopyEdit

```
[+] Building 3.0s (6/6) FINISHED
 => [web internal] load build context
 => [web] => => naming to docker-compose-webapp_web
 => [db] Pull complete
 => Container db ... started
 => Container web ... started
```

Step 6: Visit the Web App

Open your browser and go to:

arduino

CopyEdit

http://localhost:8080

Browser Output:

css

CopyEdit

```
Connected successfully to MySQL!
```

Key Points:

- Docker Compose uses YAML syntax to define multi-container setups.
- Containers can be connected through internal Docker networks automatically.
- depends_on ensures db starts before web, but not health-checked.
- Docker volumes (db-data) persist database content.
- Port 8080 on host maps to port 80 inside the web container.

Some Examples:

Command	Description	Sample Output
docker-compose up --buil	Builds and starts all services	Services start with logs
docker-compose down	Stops and removes containers, network	Cleanup successful
docker-compose logs	Shows logs for all services	Real-time stdout from service
docker-compose ps	Lists running containers	Container names and ports

Conclusion:

This experiment showed how Docker Compose can be used to manage a PHP-based web application and its backend MySQL database. We successfully built and deployed both containers, demonstrated service linking using Compose, and confirmed database connectivity via browser output. This practice is crucial for full-stack development and deployment automation in microservice architectures.