Container Introduction Containers are lightweight, portable, and self-sufficient environments that package applications and their dependencies together. They ensure consistency across multiple development, testi Here's a detailed explanation of each topic with practical **Docker Compose YAML examples** and commands.

---

1. docker-compose.yml

docker-compose.yml is a YAML configuration file that defines multi-container Docker applications. It specifies services, networks, volumes, and environment variables.

Example

A **docker-compose.yml** file for running a **MySQL database**:

```yaml
version: "3.8"

services:

  mysql:

    image: mysql:latest

    container_name: my-mysql

    restart: always

    environment:

      MYSQL_ROOT_PASSWORD: root  # Keep only the root password

    ports:

      - "3310:3306"

    volumes:

      - ./mysql-data:/var/lib/mysql
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql
```

Explanation

- **Services**: Defines MySQL as a service.
- **Image**: Uses the latest MySQL image.
- **Container Name**: Names the container my-mysql.
- **Restart Policy**: Restarts automatically if it crashes.
- **Environment Variables**: Sets MySQL root password, database, user, and password.
- **Ports**: Maps MySQL's port 3306 to the host machine.
- **Volumes**:
    - **Persistent storage**: Maps local folder mysql-data to MySQL's data directory.
    - **Schema initialization**: Runs init.sql to create tables on startup.

---

2. docker-compose up Command

The docker-compose up command starts all services defined in the docker-compose.yml file.

Command to Start the Containers

docker-compose up –d
docker exec -it my-mysql mysql -u root –p
Enter password
Mysql- show databases;

Options

- -d: Runs in detached mode (in the background).

Stopping Containers

docker stop my-mysql

docker rm my-mysql

docker-compose down

This removes containers but **keeps the database data** if using **volumes**.

---

3. Mapping MySQL Data File to Local Folder

By mapping a **local folder** to the MySQL **data directory**, you ensure that the data is persistent even if the container is removed.

Example in docker-compose.yml

```
volumes:
  - ./mysql-data:/var/lib/mysql
```

- ./mysql-data: Local folder on the host machine.
- /var/lib/mysql: MySQL's internal data directory.

Now, even if the container stops or is deleted, the MySQL data remains safe in ./mysql-data.

---

4. Schema Creation Script Execution Definition

You can **automatically execute a SQL script** when MySQL starts by mounting the script to /docker-entrypoint-initdb.d/.

Example in docker-compose.yml

```
volumes:
  - ./init.sql:/docker-entrypoint-initdb.d/init.sql
```

Example init.sql File E:/YML3/init.sql

- -- Select the database where the users table should be created
- USE mydatabase;
-
- -- Create users table
- CREATE TABLE users (

- id INT AUTO_INCREMENT PRIMARY KEY,
- name VARCHAR(100),
- email VARCHAR(100) UNIQUE NOT NULL
-);
-
- -- Insert sample data
- INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
- The script is executed **only on the first run**.
- It **creates a users table** and inserts sample data.

- **docker exec -it my-mysql ls -l /docker-entrypoint-initdb.d/**
- **docker exec -i my-mysql mysql -u root -proot mydatabase < E:/YML3/init.sql**
- **docker exec -it my-mysql mysql -u root -proot -e "USE mydatabase; SHOW TABLES;"**

mysql> show databases;

mysql> use mydatabase;

mysql> show tables;

mysql> select * from users;

---

5. Defining Port in Docker Compose

You need to expose MySQL's **port 3306** so applications can connect.

Example

```
ports:
  - "3306:3306"
```

This maps **container port 3306** to **host port 3306** so that MySQL can be accessed using:

```
mysql -h 127.0.0.1 -P 3306 -u root -p
```

---

6. Password Definition

MySQL's root password, database, and user credentials are set using **environment variables**.

Example

```
environment:
  MYSQL_ROOT_PASSWORD: rootpass
  MYSQL_DATABASE: mydatabase
  MYSQL_USER: myuser
  MYSQL_PASSWORD: mypassword
```

Effect

- **Root Password**: rootpass
- **Database**: mydatabase
- **User**: myuser
- **User Password**: mypassword

These credentials can be used when connecting to MySQL.

---

7. Running docker-compose up

After writing the docker-compose.yml file, start the MySQL container with:

```
docker-compose up -d
```

- Runs MySQL in the background.
- Creates the database, user, and schema (if init.sql is present).
- Maps ports and volumes.

---

8. Executing MySQL Client on the MySQL Server Container

To interact with the MySQL database inside the container, use:

```
docker exec -it my-mysql mysql -u root -p
```

- my-mysql: The container name.
- mysql -u root -p: Logs into MySQL as root.

Example Query

Once inside the MySQL prompt, check the available databases:

```
SHOW DATABASES;
```

Final docker-compose.yml Example

```
version: "3.8"
services:
  mysql:
    image: mysql:latest
    container_name: my-mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root  # Set root password
      MYSQL_DATABASE: mydatabase # Ensure database is created
    ports:
      - "3306:3306"
    volumes:
      - ./mysql-data:/var/lib/mysql
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
```

Summary

Feature	Description
docker-compose.yml	Defines MySQL container settings.
docker-compose up -d	Starts MySQL in detached mode.
Mapping MySQL Data	Binds MySQL data to ./mysql-data.

Schema Execution	Mounts init.sql for automatic table creation.
Port Mapping	Maps MySQL port 3306 to host.
Password Setup	Uses environment variables to set MySQL credentials.
Executing MySQL Client	docker exec -it my-mysql mysql -u root -p

---

Docker-REST

mvn clean package

ls target/

docker-compose down –v

docker-compose build --no-cache

docker-compose up –d

docker ps

docker stop 9a8853a53f8e

docker rm 9a8853a53f8e

docker exec -it mysql-db mysql -uroot -proot -e "SHOW DATABASES;"

http://localhost:8082/actuator/health

http://localhost:8082/products

docker-compose down -v # Stop and remove volumes

docker-compose up -d # Restart fresh

This is a **Dockerfile** for packaging and running a **Spring Boot** application inside a Docker container:

Explanation of Each Line:

1 FROM openjdk:11

- This specifies the **base image** for the container.
- It uses **OpenJDK 11**, which is needed to run the Java application.
- Docker will **pull this image** from Docker Hub if it is not already available locally.

2 WORKDIR /app

- Sets the **working directory** inside the container to /app.
- All subsequent commands will be executed from this directory.
- Helps maintain a structured file system inside the container.

3 COPY target/RESTcrud-0.0.1-SNAPSHOT.jar app.jar

- Copies the **built Spring Boot JAR file** from the local target/ directory into the /app folder inside the container.
- RESTcrud-0.0.1-SNAPSHOT.jar is the **output JAR** generated after building the Spring Boot project (mvn package or gradle build).

4 EXPOSE 8082

- Declares that the application will run on **port 8082** inside the container.
- This doesn't actually publish the port; it just serves as documentation.
- The actual **port binding** is done in docker-compose.yml (8082:8082).

5 ENTRYPOINT ["java", "-jar", "app.jar"]

- This **defines the command** to execute when the container starts.
- It runs the **Spring Boot JAR file** using java -jar app.jar.
- ENTRYPOINT ensures that this command always runs, even if arguments are passed to docker run.

What Happens When You Run This Dockerfile?

1. **Pulls** the OpenJDK 11 image.
2. **Creates** the /app directory inside the container.
3. **Copies** the Spring Boot JAR into /app/app.jar.
4. **Exposes** port 8082 for communication.
5. **Starts** the Spring Boot application inside the container using java -jar app.jar.

Example Usage

After building the Docker image (my-spring-app), you can run it like this:

```
docker build -t my-spring-app .
docker run -p 8082:8082 my-spring-app
```

Then access the application at:

http://localhost:8082/products

---

docker-compose.yml

Your **docker-compose.yml** file defines two services:

1. **mysql** → Runs a MySQL 8.0 database container
2. **app** → Runs a Spring Boot application that connects to MySQL

Breakdown of Each Section

1 Version Declaration

```
version: "3.8"
```

- Specifies the **Compose file format version**.
- "3.8" ensures compatibility with modern Docker versions.

---

2 mysql Service (MySQL Database)

```
services:
  mysql:
    image: mysql:8.0
```

- Uses **MySQL 8.0** official image.

```
    container_name: mysql-db
```

- Names the container **mysql-db** instead of a random name.

```
    restart: always
```

- Ensures MySQL **automatically restarts** if it crashes or Docker is restarted.

```
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: coforge
```

- **MYSQL_ROOT_PASSWORD** → Sets root password to "root".
- **MYSQL_DATABASE** → Creates a default database "coforge" on startup.

```
    ports:
      - "3312:3306"
```

- Maps **MySQL port** inside the container (3306) to host machine **port 3312**.
- This avoids conflicts if another MySQL is running on **port 3306**.

```
networks:
  - mynetwork
```

- Connects **MySQL** to a custom **Docker network (mynetwork)** for inter-container communication.

```
volumes:
  - mysql-data:/var/lib/mysql
```

- **Persistent storage** → Stores MySQL data in a **named volume (mysql-data)** instead of losing it when the container stops.

```
healthcheck:
  test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
  interval: 10s
  retries: 5
```

- **Checks MySQL health every 10 seconds** (tries 5 times).
- Ensures **MySQL is ready** before other services start.

```
command: --default-authentication-plugin=mysql_native_password
```

- **Forces MySQL to use mysql_native_password authentication** (solves login issues in some MySQL versions).

---

3 app Service (Spring Boot Application)

```
app:
  build:
    context: .
    dockerfile: Dockerfile
```

- **Builds the Spring Boot app** from the **Dockerfile** in the current directory (.).

```
container_name: springboot-app
```

- Names the container **springboot-app**.

```
restart: always
```

- **Restarts automatically** if the application crashes.

```
depends_on:
  mysql:
    condition: service_healthy
```

- Ensures **Spring Boot starts only after MySQL is healthy** (checked using healthcheck).

```
ports:
  - "8082:8082"
```

- Exposes **Spring Boot on port 8082** (inside & outside the container).

```
environment:
  SPRING_DATASOURCE_URL: jdbc:mysql://mysql-db:3306/coforge
  SPRING_DATASOURCE_USERNAME: root
  SPRING_DATASOURCE_PASSWORD: root
  SPRING_JPA_HIBERNATE_DDL_AUTO: update
```

- **Database Connection Properties**:
    - **SPRING_DATASOURCE_URL** → Connects Spring Boot to MySQL (mysql-db is the container name).
    - **SPRING_DATASOURCE_USERNAME & SPRING_DATASOURCE_PASSWORD** → Uses root credentials.
    - **SPRING_JPA_HIBERNATE_DDL_AUTO=update** → Auto-creates tables if missing.

```
networks:
  - mynetwork
```

- Connects **Spring Boot** to the **same network** as MySQL (mynetwork).
- Allows **communication without using localhost**.

---

4 Networks

```
networks:
  mynetwork:
    driver: bridge
```

- **Creates a custom bridge network (mynetwork)** for services to communicate internally.

---

5 Volumes (Persistent Storage)

```
volumes:
  mysql-data:
```

- **Named Volume (mysql-data)** → Persists MySQL database data **even if the container is restarted or deleted**.

Summary (How It Works)

1. **MySQL container (mysql-db)** starts on **port 3312**.
2. Spring Boot **waits for MySQL to be healthy** before starting.
3. Spring Boot **connects to MySQL** using jdbc:mysql://mysql-db:3306/coforge.
4. **Data is stored persistently** in mysql-data volume.

Next Steps

Run it:

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

Check running containers:

```
docker ps
```

Access logs:

```
docker logs springboot-app
docker logs mysql-db
```