## Dockerizing A WAR Project with Docker Compose

### 1. What is Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow you to package up an application with all the parts it needs (like libraries and dependencies) and ship it all out as one package.

### 2. Prerequisites

Before you start, ensure you have the following installed on your system:

- **Docker**: You can download and install Docker from Docker's official website. Follow the installation instructions based on your operating system.
- **Docker Compose**: Docker Compose comes installed with Docker Desktop, but you can verify by typing `docker-compose --version` in your terminal.
- **Java and Maven Project**: Your project (`4413FinalProject`) should be a Java-based project that uses Maven for building.

### 3. Understanding Your Dockerfile

Your Dockerfile sets up a multi-stage build for your application:

```
# Build Stage

# Use an official Tomcat base image with JDK 21 for building the application

FROM tomcat:9.0-jdk21 AS builder

# Update package lists and install Maven

RUN apt update && apt install maven -y

# Set the working directory in the container

WORKDIR /app

# Add the Maven project files to the working directory

ADD pom.xml .

ADD src ./src

# Run Maven to build the project

RUN mvn install
```

```
# Runtime Stage

# Use a specific version of Tomcat with JRE 21 for the runtime environment

FROM tomcat:9.0.93-jre21

# Copy the built WAR file from the builder stage to the webapps directory in Tomcat

COPY --from=builder /app/target/4413FinalProject-*.war /usr/local/tomcat/webapps/eStore.war

# Set the working directory in the container

WORKDIR /usr/local/tomcat

# Expose port 8080 to the host

EXPOSE 8080

# Start the Tomcat server

CMD [ "/usr/local/tomcat/bin/catalina.sh", "run" ]
```

**4. Docker Compose Configuration**

- **What is Docker Compose?**

Docker Compose is a tool that allows you to define and manage multi-container Docker applications. It uses a YAML file to configure your application's services, making it easy to start, stop, and rebuild entire environments.

Your `docker-compose.yml` file defines a setup with two services: a MySQL database and the `store-app` application. Here's what each section does:

```
services:

  mysql:

    container_name: mysql

    hostname: mysql

    image: mysql:8.0.39

    ports:
```

```
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: EECS4413
      MYSQL_DATABASE: eStore
    volumes:
      # Mount the folder with the SQL scripts to initialize the
database
      - ./scripts:/docker-entrypoint-initdb.d:ro
      # Mount the folder to be used to store the database data
      - ./data:/var/lib/mysql
      # Mount the configuration file to configure the character and
collation config
      - ./dbcfg/my_conf.cnf:/etc/mysql/conf.d/my_conf.cnf
    healthcheck:
      test: ["CMD", "mysqladmin" ,"ping", "-uroot", "-pEECS4413"]
      interval: 10s
      timeout: 60s
      retries: 5
  store-app:
    image: docker.io/aristostheo/estore:v1.0.0
    container_name: store-app
    ports:
      - "8080:8080"
    environment:
      DB_SERVER: mysql
```

```
      DB_PORT: '3306'
      DB_NAME: eStore
      DB_USERID: shopadmin
      DB_PASSWORD: EECS4413
    restart: always
    depends_on:
      - mysql
```

**Explanation of the `docker-compose.yml` File**:

1. **`mysql` Service**:
   - **Container Name**: Sets the container name to `mysql`.
   - **Hostname**: Sets the hostname within the Docker network to `mysql`.
   - **Image**: Uses MySQL version `8.0.39`.
   - **Ports**: Exposes port `3306` for MySQL communication.
   - **Environment Variables**: Sets the root password, creates a database named `eStore`.
   - **Volumes**:
     - `./scripts:/docker-entrypoint-initdb.d:ro`: Mounts a local directory containing SQL scripts to initialize the database.
     - `./data:/var/lib/mysql`: Mounts a local directory to persist database data.
     - `./dbcfg/my_conf.cnf:/etc/mysql/conf.d/my_conf.cnf`: Mounts a configuration file for MySQL settings.
   - **Healthcheck**: Monitors the health of the MySQL service to ensure it's running correctly.
2. **`store-app` Service**:
   - **Image**: Uses a pre-built Docker image located at `docker.io/aristostheo/estore:v1.0.0`.
   - **Container Name**: Sets the container name to `store-app`.
   - **Ports**: Exposes port `8080` for the application.
   - **Environment Variables**: Configures database connection settings (`DB_SERVER`, `DB_PORT`, `DB_NAME`, `DB_USERID`, `DB_PASSWORD`).
   - **Restart**: Ensures the container restarts automatically in case of failure.
   - **Depends On**: Specifies that `store-app` depends on the `mysql` service, ensuring the database starts first.

**5. Building and Pushing Your Docker Image to Docker Hub**

If you make changes to your application and want to update the Docker image stored in Docker Hub, you can manually build and push the image. This allows others to pull the latest version of your application directly from the registry.

**Building the Docker Image Locally**

Navigate to the directory containing your `Dockerfile` and run the following command to build the image:

```
docker build -t docker.io/aristostheo/estore:v1.0.0 .
```

- `'docker build'`: Command to build a Docker image.
- `'-t docker.io/aristostheo/estore:v1.0.0'`: Tags the image with your Docker Hub username and image name, including a version (`v1.0.0`).
- `'.'`: Specifies the current directory as the build context (where the `Dockerfile` and application code are located).

**Pushing the Image to Docker Hub**

Once the image is built, you can push it to Docker Hub using the following command:

```
docker push docker.io/aristostheo/estore:v1.0.0
```

- `'docker push'`: Command to push a Docker image to a registry.
- `'docker.io/aristostheo/estore:v1.0.0'`: Specifies the image to push, using your Docker Hub username and image tag.

Ensure you are logged into Docker Hub with the command `docker login` before pushing the image. You'll be prompted to enter your Docker Hub credentials.

**6. Running the Docker Compose Setup**

1. Ensure you are in the directory containing your `docker-compose.yml` file.

Run the following command to start all services defined in the Docker Compose file:
`docker-compose up`

2.
    - `'up'`: Builds, creates, starts, and attaches to containers for a service.
    - `(OPTIONAL)'-d'`: Runs the containers in detached mode (in the background).

**7. Accessing Your Application**

Once the services are running, open a web browser and go to:

`http://localhost:8080/eStore`

You should see your application running, which means it has successfully connected to the MySQL database and is operational!

**8. Managing Your Docker Compose Setup**

To **stop** the containers, use:
`docker-compose down`

- This command stops and removes the containers, networks, and volumes created by `docker-compose up`.

To **view logs** from the running containers:
`docker-compose logs -f`

- To **rebuild** the application image and restart containers:
  `docker-compose up -d --build`