Containerizing applications using Docker and Docker Compose

The project this tutorial is based on includes a frontend Angular service, two Java backends **java_organizer** and **java_grading** performing different tasks, a HiveMQ broker for MQTT communication, and a PostgreSQL database.

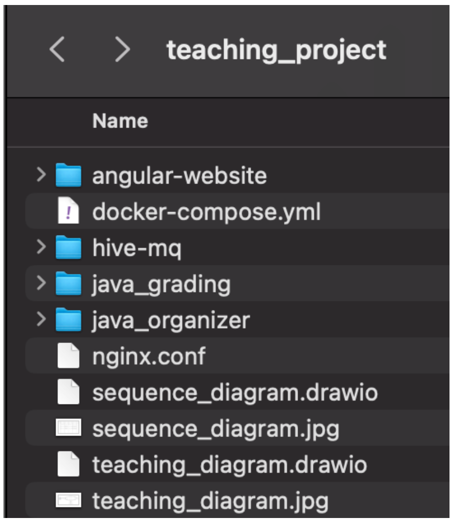
This tutorial assumes you have a basic understanding of Docker, Docker Compose, and the structure of your applications.

1. Prerequisites

- **Docker**: Ensure Docker is installed on your system. Docker Compose is included in Docker Desktop for Mac/Windows.
- **Docker Compose**: For Linux, you might need to install Docker Compose separately.
- **Application Code**: Have your Angular application, Java backend codes, HiveMQ configuration, and any necessary scripts or files ready.

2. Directory Structure

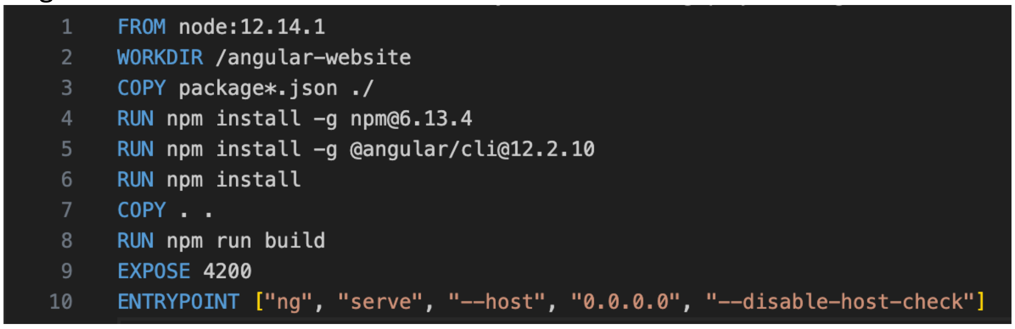
The project this tutorial is based contains directory for each service:



```
angular-website
docker-compose.yml
hive-mq
java_grading
java_organizer
nginx.conf
sequence_diagram.drawio
sequence_diagram.jpg
teaching_diagram.drawio
teaching_diagram.jpg
```

3. Creating Docker files for each service

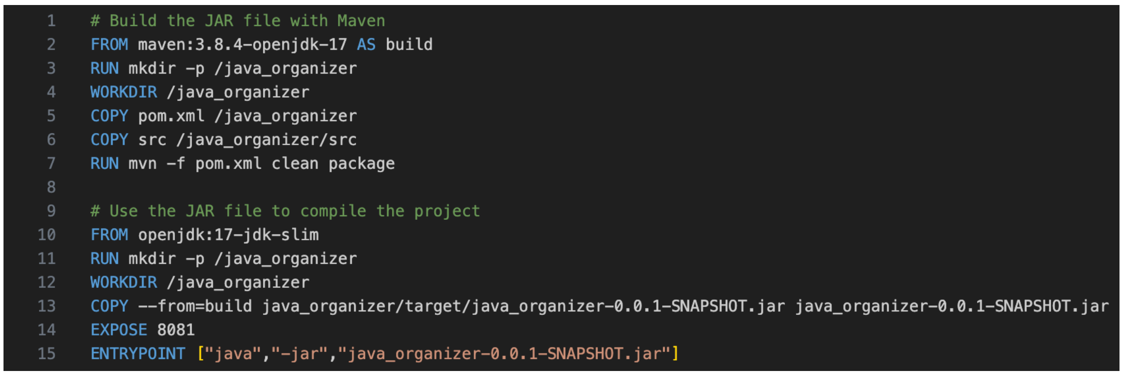
a. Angular Frontend



```
FROM node:12.14.1
WORKDIR /angular-website
COPY package*.json ./
RUN npm install -g npm@6.13.4
RUN npm install -g @angular/cli@12.2.10
RUN npm install
COPY . .
RUN npm run build
EXPOSE 4200
ENTRYPOINT ["ng", "serve", "--host", "0.0.0.0", "--disable-host-check"]
```

b. Java Backends

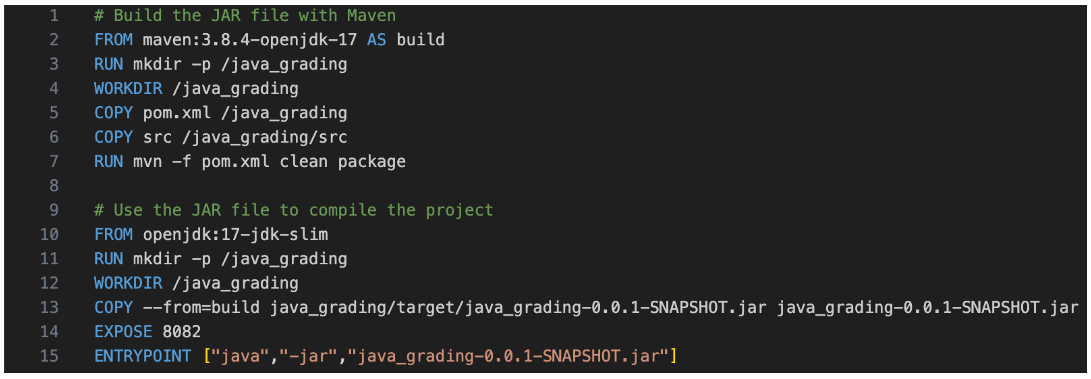
i. java_organizer



```
# Build the JAR file with Maven
FROM maven:3.8.4-openjdk-17 AS build
RUN mkdir -p /java_organizer
WORKDIR /java_organizer
COPY pom.xml /java_organizer
COPY src /java_organizer/src
RUN mvn -f pom.xml clean package

# Use the JAR file to compile the project
FROM openjdk:17-jdk-slim
RUN mkdir -p /java_organizer
WORKDIR /java_organizer
COPY --from=build java_organizer/target/java_organizer-0.0.1-SNAPSHOT.jar java_organizer-0.0.1-SNAPSHOT.jar
EXPOSE 8081
ENTRYPOINT ["java","-jar","java_organizer-0.0.1-SNAPSHOT.jar"]
```

ii. java_grading

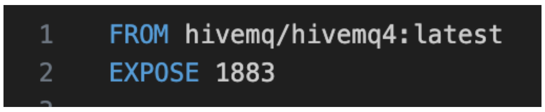


```
# Build the JAR file with Maven
FROM maven:3.8.4-openjdk-17 AS build
RUN mkdir -p /java_grading
WORKDIR /java_grading
COPY pom.xml /java_grading
COPY src /java_grading/src
RUN mvn -f pom.xml clean package

# Use the JAR file to compile the project
FROM openjdk:17-jdk-slim
RUN mkdir -p /java_grading
WORKDIR /java_grading
COPY --from=build java_grading/target/java_grading-0.0.1-SNAPSHOT.jar java_grading-0.0.1-SNAPSHOT.jar
EXPOSE 8082
ENTRYPOINT ["java","-jar","java_grading-0.0.1-SNAPSHOT.jar"]
```

Ensure your Java applications are packaged as a jar file (`mvn package` or `gradle assemble`).

c. HiveMQ Broker



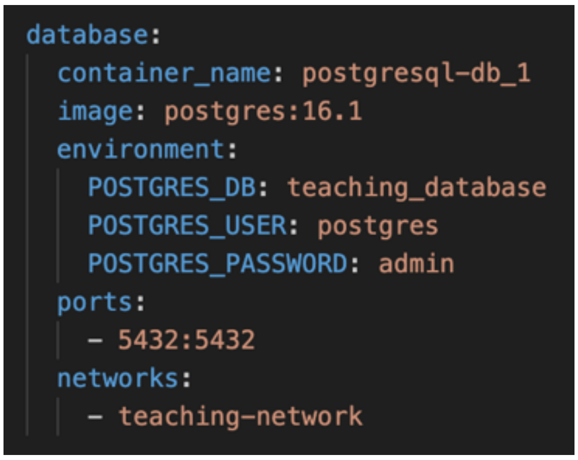
```
FROM hivemq/hivemq4:latest
EXPOSE 1883
```

Minimal configuration for the HiveMQ message broker. We are using the public image fetched from Docker Hub with the **latest** tag.

d. PostgreSQL Database

You can use the official `postgres` image and specify the image:<tag> in your docker-compose.yaml file.

The default image can be leveraged as there is no customization needed to the database service.



```yaml
database:
  container_name: postgresql-db_1
  image: postgres:16.1
  environment:
    POSTGRES_DB: teaching_database
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: admin
  ports:
    - 5432:5432
  networks:
    - teaching-network
```

e. Nginx (Optional)

If you choose to use Nginx as a reverse proxy, you must add the configuration file "nginx.conf" in the `/nginx` directory. This will be mounted in the container and Nginx service will use it at start-up.

4. **Docker Compose**

The docker-compose.yml file defines the services, networks, and volumes needed to run your application.

It specifies dependencies between services, environment variables, and exposed ports.

You can find the Docker compose code in the git repository found at https://github.com/vladfrunzescu/SOA/blob/master/teaching_project/docker-compose.yml .

5. Building and Running

- Build Services: Run `docker-compose build` in the directory containing your `docker-compose.yml`. This will build the Docker images for your services.

- Run Services: Execute `docker-compose up` to start your application. Use the `-d` flag to run in detached mode.

6. Accessing Your Application

- Access the Angular frontend via `http://localhost:4200` - the port should be mapped correctly in the `docker-compose.yml` file.

- Other Java applications and HiveMQ broker will be accessible on their configured ports found in the docker-compose file.

7. Updates

- To update a service, modify its code or configuration, then rebuild and restart the service using Docker Compose.
- Monitor logs and performance, adjusting as necessary.