SPRING-APP-DEPLOYMENT
Written by Hong Le Nguyen last update **17.02.2025**

Docker Compose

In practice, in a microservices deployment we have more than one service, where each service has its own container. Also, there can be cases where one microservice depends on other services like a database. The database will be another container, but still part of the same application.

The Docker Compose helps define and manage such multi-container environments. It's another tool that we have to install apart from the Docker client and the Docker engine (host).

Link
https://www.javaguides.net/2024/05/spring-boot-with-postgresql-using-docker-compose.html
https://github.com/hong1234/Spring-Services-Dockerizing

Use Spring Initializr
https://start.spring.io/
to create a new project with the following configuration:

Project: Maven Project
Language: Java 17
Spring Boot: 3.4.x
Dependencies:
Spring Web, Spring Data JPA, PostgreSQL Driver

Download and unzip the project, then open it in your IDE.

1.6 application.properties Configuration

```
# src/main/resources/application.properties
spring.datasource.url=jdbc:postgresql://localhost:5432/testdb
spring.datasource.username=postgres
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Step 2: Create Docker Compose Configuration

2.1 Create a **Dockerfile**

Create a Dockerfile in the root directory of your project.

```
# Use the official OpenJDK base image
        FROM openjdk:17-jdk-alpine
# Set the working directory inside the container
        WORKDIR /app
# Copy the built jar file into the container
        COPY target/demo-0.0.1-SNAPSHOT.jar app.jar
# Expose port 8080
        EXPOSE 8080
# Run the application
        ENTRYPOINT ["java", "-jar", "app.jar"]
```

2.2 Create **docker-compose.yml**

Docker Compose allows you to define and run multi-container Docker applications.
You will create a docker-compose.yml file to define the services for your Spring Boot application and PostgreSQL database.

```
// docker-compose.yml
version: '3.8'

services:
  postgres:
    image: postgres:14
    environment:
      POSTGRES_DB: testdb
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  app:
    image: demo-app
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/testdb
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: password
    depends_on:
      - postgres

volumes:
  postgres_data:
```

Step 3: Build and Run the Application

3.1 Build the Jar File

```
./mvnw clean package
```

3.2 Build and Run Docker Compose

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

3.3 Verify the Application

Create a new student --

POST URL: http://localhost:8080/students

Body:

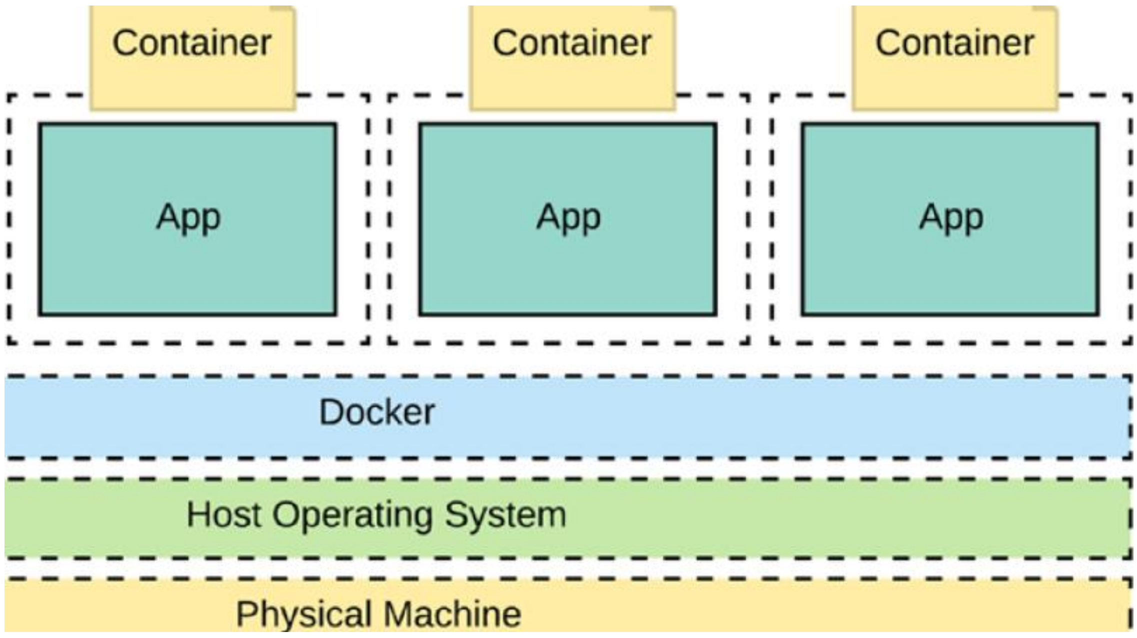
```
{
  "name": "Ramesh Fadatare",
  "email": "ramesh.fadatare@example.com"
}
```

Retrieve all students ---

GET URL: http://localhost:8080/students

Introduction to Docker

Unlike a virtual machine, all the containers deployed in the same host machine share the same operating system kernel. It's in fact an isolated process.

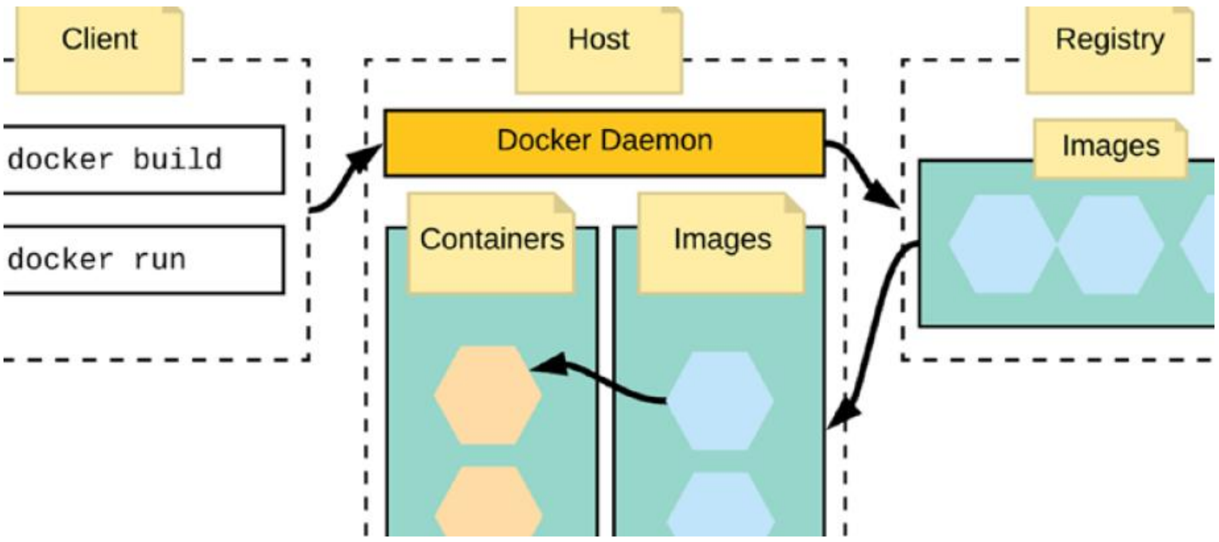


Installing Docker --
Docker follows a client-server architecture model, where you have a Docker client and a Docker host (which is also known as the Docker Engine). When you install Docker locally in your machine, you will get both the client and the host installed.

Docker Architecture --
Figure shows the high-level Docker architecture where we have the Docker client, Docker host, and the registry.

The Docker client is the command-line tool that we are going to interact with. The client can communicate with the daemon using a REST API, over UNIX sockets, or a network interface.

The Docker Daemon running on the Docker host listens to the requests coming from the client and acts accordingly.



Docker Images --
A Docker image is a package that includes your microservice (your application) along with all its dependencies. Your application will see only the dependencies that you pack in your Docker image. You define the filesystem for your image, and it will not have access to the host filesystem.
A Docker image is created using a Dockerfile. The Dockerfile defines all the dependencies to build a Docker image.

The following command will list all the Docker images stored in your local machine. This will include all the images you built locally as well as pulled from a Docker registry.

```
:\> docker image ls
```

The following command will list all the containers available in the host machine and their status.

```
:\> docker ps
```

Containers --
A container is a running instance of a Docker image. In fact, images become containers when they run on a Docker engine.
There can be many images in your local machine, but not all of them are running all the time, unless you explicitly make them to run.
A container is in fact a regular Linux container defined by namespaces and control groups.

We use the following command to start a Docker container from a Docker image.

```
:\> docker run sample01
```

Here, sample01 is the image name. This command will first check whether sample01 image and all the other base images are in the local Docker registry, and if not, it will pull all the missing images from a remote registry. Finally, once all the images are there, the container will start and execute the program or the command, set as the ENTRYPOINT in the corresponding Dockerfile.

Creating a Docker Image with a Microservice --

```
docker build -t sample01 .
```

Running a Microservice as a Docker Container --

```
docker run -p 9000:9000 sample01
```

Other Links ---

https://www.geeksforgeeks.org/how-to-dockerize-a-spring-boot-application-with-maven/
https://www.baeldung.com/dockerizing-spring-boot-application

https://www.ignek.com/blog/dockerized-microservices/
https://igventurelli.io/spring-boot-with-docker-and-kubernetes-containerizing-and-deploying-your-java-applications/