Docker and Spring Boot Application

A `Dockerfile` is a script containing instructions on how to build a Docker image. The Docker image is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the application code, runtime, libraries, and dependencies.

Steps:

- In the root of my application, I created a file named `Dockerfile`.
- A typical `Dockerfile` for a Spring Boot application would use a base image (like `openjdk`), copy the application's jar file, and define the command to run the application.

Key Dockerfile Terms: - **FROM**: Specifies the base image. Commonly, for Java applications, we use an image with JDK installed (e.g., `openjdk:11-jre`). - **COPY**: Copies files from your local system into the Docker image. - **WORKDIR**: Sets the working directory inside the container. - **CMD** or **ENTRYPOINT**: Specifies the command to run the application when the container starts. For a Spring Boot app, this would usually involve running the jar file with `java -jar`.

2. Build and run the Docker image

Once the `Dockerfile` is ready, you can build the Docker image and run the container.

Steps:

- Build the Docker image using the command:

  ```
  docker build -t my-application:1.0 .
  ```

- Run the Docker container from the image:

  ```
  docker run -p 8080:8080 my-application:1.0
  ```

Key Docker Commands: - **docker build**: This command builds an image from a Dockerfile. The `-t` flag is used to name and tag the image. - **docker run**: This starts a container from an image. The `-p` flag maps the container port to the host machine's port (e.g., mapping port 8080 inside the container to port 8080 on your machine).

3. Define a multi-container application using Docker Compose (app, Redis, database)

Docker Compose allows you to define and manage multi-container Docker applications using a YAML file. This file describes how the services (containers) work together.

In this step, you'll define a Docker Compose file that includes: - **My application** (Spring Boot app) - **Redis** (an in-memory data structure store, often used as a cache) - **Database** (e.g., PostgreSQL or MySQL)

Steps:

- In the root of my application, I created a file named `docker-compose.yml`.
- Define the services: the Spring Boot app, Redis, and the database.

Key Docker Compose Terms: - **services**: Defines the containers to be run. Each service corresponds to a container (e.g., app, redis, db). - **image**: Specifies the Docker image to use for the service. You can use custom-built images or official images (e.g., `redis`, `postgres`). - **ports**: Exposes container ports to the host machine. - **volumes**: Allows you to persist data by mapping directories from the host to the container. - **depends_on**: Specifies dependencies between services (e.g., the Spring Boot app depends on the database being ready).

4. Deploy the multi-container application using Docker Compose

Once the `docker-compose.yml` file is created and configured, you can deploy the application using Docker Compose.

Steps:

- Run the following command in the root of "my application":

  ```
  docker-compose up
  ```

- Docker Compose will start all the services (Spring Boot app, Redis, and Database), and they will be running in separate containers, but networked together.

Key Docker Compose Commands: - **docker-compose up**: This command starts the application as defined in the `docker-compose.yml` file. - **docker-compose down**: This stops and removes all the containers, networks, and volumes created by `docker-compose up`.