Docker Guide with Practical Examples for a Spring Boot Application

# 1. Introduction to Docker

Docker is a platform designed to help developers build, ship, and run applications within containers. Containers are lightweight, portable units that bundle an application with all its dependencies, making it easy to deploy consistently across different environments.  
  
In this guide, we will explore Docker concepts, build a Docker image, run it as a container, and persist data using Docker volumes, all using a Spring Boot application as an example.

# 2. Docker Images

A Docker image is a lightweight, standalone, and executable package that contains everything needed to run an application (code, libraries, environment variables, etc.).  
  
Images are created using Dockerfiles and act as blueprints for running containers. Each time you create an image, it represents a specific application version. Let's create a Docker image for a Spring Boot application.

## 2.1 Dockerfile Explained

A Dockerfile is a text file that contains a set of instructions for building a Docker image. Here’s a sample Dockerfile for our Spring Boot application:

# Use an official OpenJDK image as the base  
FROM openjdk:17-jdk-alpine  
  
# Set the working directory inside the container  
WORKDIR /dockerApp  
  
# Copy the JAR file from the host into the container  
COPY target/docker-volume-0.0.1-SNAPSHOT.jar myDockerVolume.jar  
  
# Expose the application port (default is 8080; here, we're using 2222)  
EXPOSE 2222  
  
# Command to run the JAR file  
ENTRYPOINT ["java", "-jar", "myDockerVolume.jar"]

* Explanation of each line:

1. FROM: Specifies the base image. Here, we're using OpenJDK, required for Java apps.

2. WORKDIR: Sets the working directory inside the container.

3. COPY: Copies the JAR file from the host into the container.

4. EXPOSE: Defines the port the application listens to.

5. ENTRYPOINT: Specifies the command to start the Spring Boot application.

## 2.2 Building the Docker Image

To build the image from the Dockerfile, use the following command in the Dockerfile's directory:

docker build -t spring-boot-docker-volume-image .

# 3. Docker Containers

A container is a running instance of a Docker image. Containers provide a consistent environment by packaging applications with their dependencies.  
  
Once the image is built, we can run it as a container.

## 3.1 Running a Container

To start a container and map ports, use:

docker run -d -p 2222:2222 --name my-docker-volume-container spring-boot-docker-volume-image

- -d: Run the container in detached mode (background).  
- -p: Map port 2222 on the host to port 2222 in the container.

# 4. Docker Volumes

Volumes allow data to persist even when the container stops or is removed. Volumes are managed by Docker and stored outside of the container's file system.

Example: To create a volume and use it with a container:

docker volume create my-docker-volume

docker run -d -p 2222:2222 --name my-docker-volume-container -v my-docker-volume:/app-data spring-boot-docker-volume-image

Example: Suppose the Spring Boot app writes logs to /app-data/logs. With a volume, these logs remain even after the container is stopped, as they’re stored in my-docker-volume.