**1. What is Docker?**

Docker is a platform that allows you to automate the deployment, scaling, and management of applications in lightweight containers. Containers are isolated environments that package an application and all its dependencies, ensuring it runs consistently across different computing environments.

**Key Concepts**

* **Image**: A lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, and dependencies.
* **Container**: A runnable instance of an image. You can think of it as a lightweight, standalone, and executable package of software.
* **Dockerfile**: A script that contains instructions on how to build a Docker image.
* **Docker Hub**: A public registry that contains a large number of Docker images shared by the community.

**2. Setting Up Docker**

1. **Installation**:  
   Download and install Docker Desktop from the official Docker website.
2. **Basic Commands**:
   * docker --version: Check the Docker version installed.
   * docker ps: List running containers.
   * docker images: List downloaded images.
   * docker build: Build an image from a Dockerfile.
   * docker run: Run a container from an image.

**3. Creating a Java Spring Boot Application**

Let's create a simple Spring Boot application and then containerize it using Docker.

**Step 1: Create a Spring Boot Project**

You can use [Spring Initializr](https://start.spring.io/) to bootstrap your project:

1. **Project Metadata**:
   * Project: Maven Project
   * Language: Java
   * Spring Boot: 3.0.0 (or latest stable version)
   * Group: com.example
   * Artifact: demo
   * Name: demo
   * Packaging: Jar
   * Java: 17 (or your preferred version)
2. **Dependencies**:
   * Spring Web

Download the generated project and unzip it.

**Step 2: Writing a Simple Controller**

Create a simple REST controller in your application:

java

Copy code

package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HelloController {

@GetMapping("/")

public String hello() {

return "Hello, Docker!";

}

}

**Step 3: Build the Application**

Build the application using Maven:

bash

Copy code

./mvnw clean package

This will create a JAR file in the target directory.

**4. Dockerizing the Spring Boot Application**

**Step 1: Create a Dockerfile**

In the root of your project directory, create a file named Dockerfile with the following content:

dockerfile

Copy code

# Start with a base image containing Java runtime

FROM openjdk:17-jdk-alpine

# Add Maintainer info

LABEL maintainer="your-email@example.com"

# Add a volume pointing to /tmp

VOLUME /tmp

# Make port 8080 available to the world outside this container

EXPOSE 8080

# The application's jar file

ARG JAR\_FILE=target/demo-0.0.1-SNAPSHOT.jar

# Add the application's jar to the container

ADD ${JAR\_FILE} app.jar

# Run the jar file

ENTRYPOINT ["java","-jar","/app.jar"]

**Step 2: Build the Docker Image**

Navigate to your project directory and build the Docker image using the following command:

bash

Copy code

docker build -t spring-boot-demo .

This command will create an image named spring-boot-demo based on the instructions in your Dockerfile.

**Step 3: Run the Docker Container**

To run the application inside a Docker container, use the following command:

bash

Copy code

docker run -p 8080:8080 spring-boot-demo

This command will run the container, mapping the container's port 8080 to your local machine's port 8080. You can access the application at http://localhost:8080.

**5. Key Docker Commands for Managing Containers**

* **List running containers**:

bash

Copy code

docker ps

* **Stop a container**:

bash

Copy code

docker stop <container\_id>

* **Remove a container**:

bash

Copy code

docker rm <container\_id>

* **Remove an image**:

bash

Copy code

docker rmi <image\_id>

**6. Next Steps**

Now that you've containerized a simple Spring Boot application, you can explore more advanced topics like:

* **Docker Compose**: Orchestrating multi-container applications.
* **Docker Networking**: Understanding Docker's networking model.
* **Docker Volumes**: Persisting data in Docker.

# Docker Image automation with single Maven command

* Spring boot application
* Add Dockerfile
  1. FROM openjdk
  2. EXPOSE 8080
  3. ADD target/demo-0.0.1-SNAPSHOT.jar
  4. ENTRYPOINT ["java","-jar","/app.jar"]

Add tag <finalname>DockerSpringbootDemo</finalname>

### Auto Push Spring Boot App Image on Docker Hub Repository# 9561826040

Pom.xml plugin

<plugin>

<groupId>com.spotify</groupId>

<artifactId>dockerfile-maven-plugin</artifactId>

<version>1.4.13</version>

<executions>

<execution>

<id>default</id>

<goals>

<goal>build</goal>

<goal>push</goal>

</goals>

</execution>

</executions>

<configuration>

<repository>gmantrig/bootapp1</repository>

<tag>11.8</tag>

<useMavenSettingsForAuth>true</useMavenSettingsForAuth>

</configuration>

</plugin>

</plugins>

============================================================================================================================================

settings.xml m2 folder dockerhiub username password

 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">

    <servers>

<server>

<id>docker.io</id>

<username>gmantrig</username>

<password>Perfect@12345</password>

</server>

 </servers>

<pluginGroups>

        <pluginGroup>com.spotify</pluginGroup>

    </pluginGroups>

    </settings>