

# Getting Started with Docker and Spring Boot

**Linkedin:** <https://www.linkedin.com/in/shivasrivastava1/>

**Github:** [hacker123shiva/docker-springboot-project: Getting Started with Docker and Spring Boot \(github.com\)](https://github.com/hacker123shiva/docker-springboot-project)

**Docker image :** docker push hacker123shiva/docker-project

In today's fast-paced development environment, containerization has become essential. Docker allows developers to package applications into containers, ensuring they run consistently across various environments. This blog will guide you through the prerequisites and steps to set up a simple Spring Boot project and deploy it using Docker.

## Prerequisites

Before we dive into the setup, ensure you have the following:

### 1. Account on Docker Hub:

- Sign up for an account at [Docker Hub](https://hub.docker.com/). Docker Hub is a cloud-based repository where you can store and share Docker images. Example of username for Docker Hub is **hacker123shiva**.

### 2. Basic Knowledge of Docker:

- Understand the fundamentals of Docker, including:
  - **Containers:** Lightweight, standalone, executable packages of software that include everything needed to run an application.
  - **Images:** Read-only templates used to create containers.
  - **Dockerfile:** A script that contains a series of instructions on how to build a Docker image.

### 3. Maven Basic Command and Setup Knowledge:

- Maven is a build automation tool primarily used for Java projects. Familiarize yourself with basic Maven commands such as:
  - **mvn clean:** Cleans the target directory.
  - **mvn package:** Packages the compiled code into a JAR file.

### 4. How to Work with a Spring Boot Project:

- Understand the basics of Spring Boot and how to create a simple RESTful web service. Familiarity with annotations like **@RestController** and **@GetMapping** is beneficial.

# Project Setup

## Step 1: Create a Spring Boot Project

**Directory Structure:** Your project directory should resemble the following structure:

```
docker-project/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── com/
│   │   │   │   ├── telusko/
│   │   │   │   │   ├── controller/
│   │   │   │   │   │   └── DemoController.java
│   │   │   └── resources/
│   │   │       └── application.properties
│   └── test/
├── pom.xml
└── target/
```

1. **Create the Controller:** Create a new Java class `DemoController.java` in the `com.telusko.controller` package:

```
package com.telusko.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class DemoController {
    @GetMapping("/hello")
    public String hello() {
        return "<h1>Welcome Java Developers!<br>In world of DevOps</h1>";
    }
}
```

2. **Configure Application Properties:** In `src/main/resources/application.properties`, set the server port:


```
server.port=9090
```














3. **Create the `pom.xml`:** Below is the `pom.xml` configuration for your Spring Boot project. It includes necessary dependencies and builds settings:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.10</version>
    <relativePath/>
  </parent>
  <groupId>com.telusko</groupId>
  <artifactId>shiva</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>docker-project</name>
  <description>docker project</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
```

```
        <finalName>demo-docker-project</finalName> <!-- Specify the name of
jar by this tag-->
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

## Step 2: Build the Project

▼  > docker-project [boot] [devtools] [Blog-application main]

- >  > src/main/java
- >  > src/main/resources
- >  src/test/java
- >  JRE System Library [JavaSE-17]
- >  Maven Dependencies
- >  > src
- target
-  Dockerfile
-  HELP.md
-  hs\_err\_pid19716.log
-  hs\_err\_pid23816.log
-  mvnw
-  mvnw.cmd
-  pom.xml

## Method1:

Open your terminal, navigate to the project directory, and run the following command:

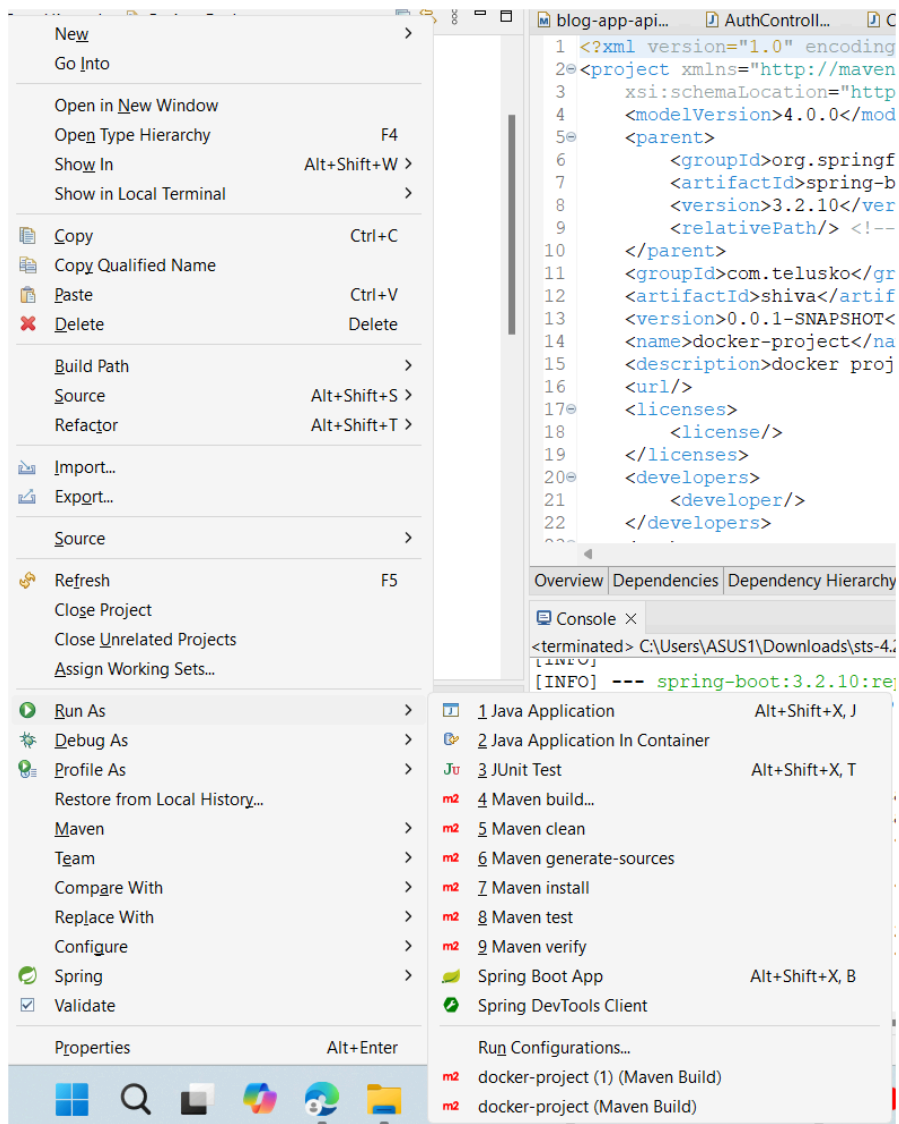
```
mvn clean package
```

```
[INFO] --- spring-boot-maven-plugin:3.2.10:repackage (repackage) @ shiva ---
[INFO] Replacing main artifact D:\Blog-application\Blog-project\docker-project\target\demo-docke:
with repackaged archive, adding nested dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to D:\Blog-application\Blog-project\docker-project\
docker-project.jar.original
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.840 s
[INFO] Finished at: 2024-10-04T23:00:09+05:30
[INFO] -----

D:\Blog-application\Blog-project\docker-project>
```

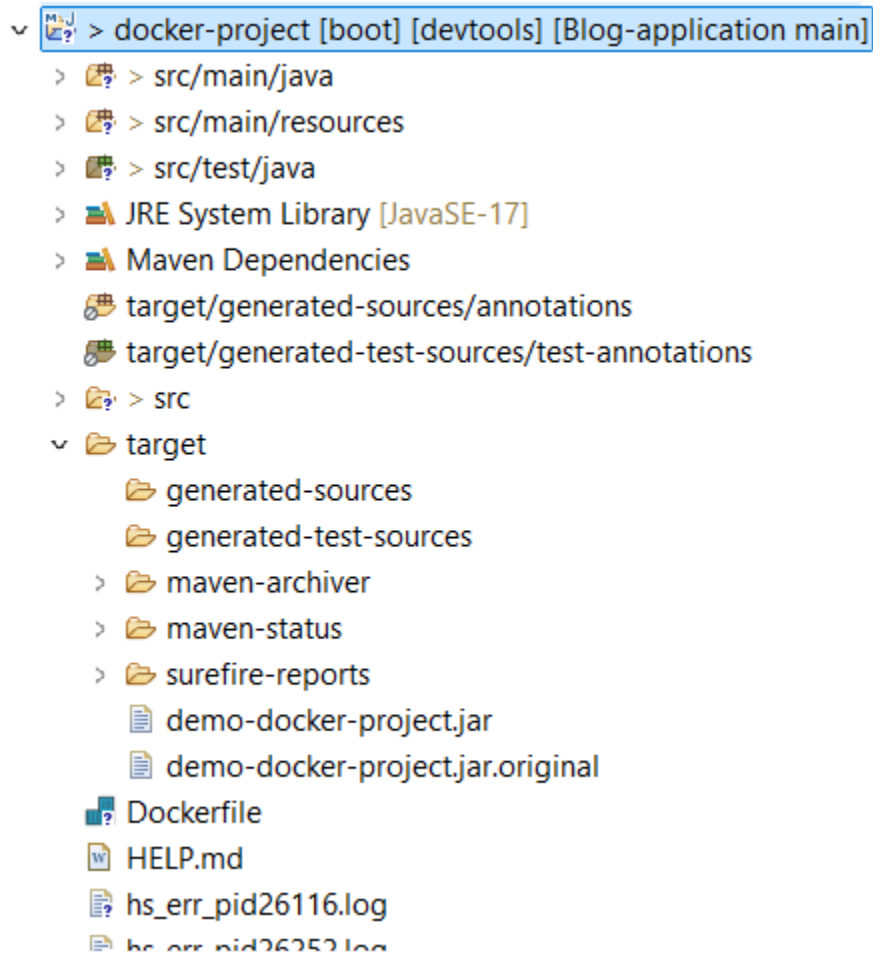
Or

## Method2:



**Click on maven clean then maven install**

This command compiles your application and packages it into a JAR file located in the **target** directory.



### Step 3: Create a Dockerfile

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

```
# Use the official OpenJDK 17 image as the base image
FROM openjdk:17-jdk-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the JAR file from the target directory to the container
COPY target/demo-docker-project.jar demo-docker-project.jar

# Expose the port the application runs on
```

EXPOSE 9090

*# Command to run the application*

ENTRYPOINT ["java", "-jar", "demo-docker-project.jar"]

## Step 4: Build the Docker Image

1. **Open your terminal** and navigate to the project directory.
2. **Open Docker desktop** before running the command.
3. **Build the Docker image** using the following command.

```
docker build -t hacker123shiva/docker-project .
```

```
D:\Blog-application\Blog-project\docker-project>docker build -t hacker123shiva/docker-project .
[+] Building 2.5s (9/9) FINISHED                                docker:default
=> [internal] load .dockerignore                                0.1s
=> => transferring context: 2B                                   0.0s
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 448B                               0.0s
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-alpine 2.2s
=> [auth] library/openjdk:pull token for registry-1.docker.io   0.0s
=> [1/3] FROM docker.io/library/openjdk:17-jdk-alpine@sha256:4b6abae565492dbe9e7a894137c966a748515423 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 80B                                    0.0s
=> CACHED [2/3] WORKDIR /app                                    0.0s
=> CACHED [3/3] COPY target/demo-docker-project.jar /app/docker-project.jar 0.0s
```

## Step 5: Run the Docker Container

To run the Docker container from the image you just built, use the following command:

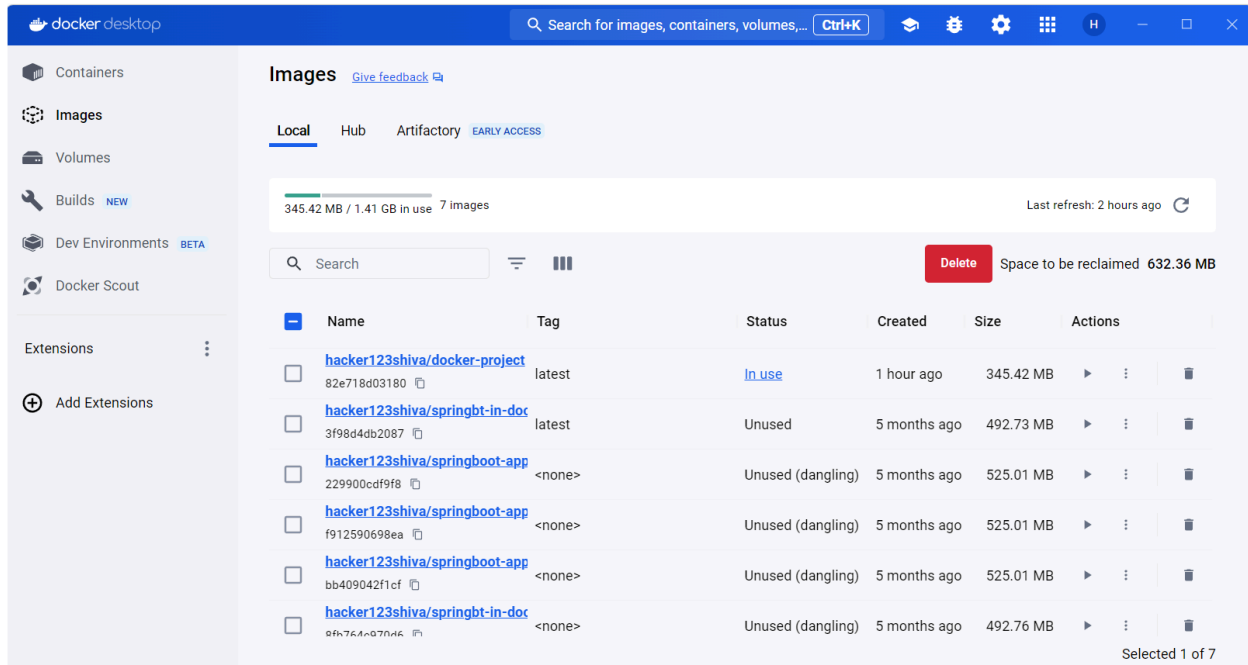
```
docker run -d -p 9090:9090 hacker123shiva/docker-project
```

View build details:

```
D:\Blog-application\Blog-project\docker-project>docker run -d -p 9090:9090 hacker123shiva
/docker-project
df942e6a89eb2b4db73601ce49d5b20435e8bde83b3959a2e6852bb69edc867c
```

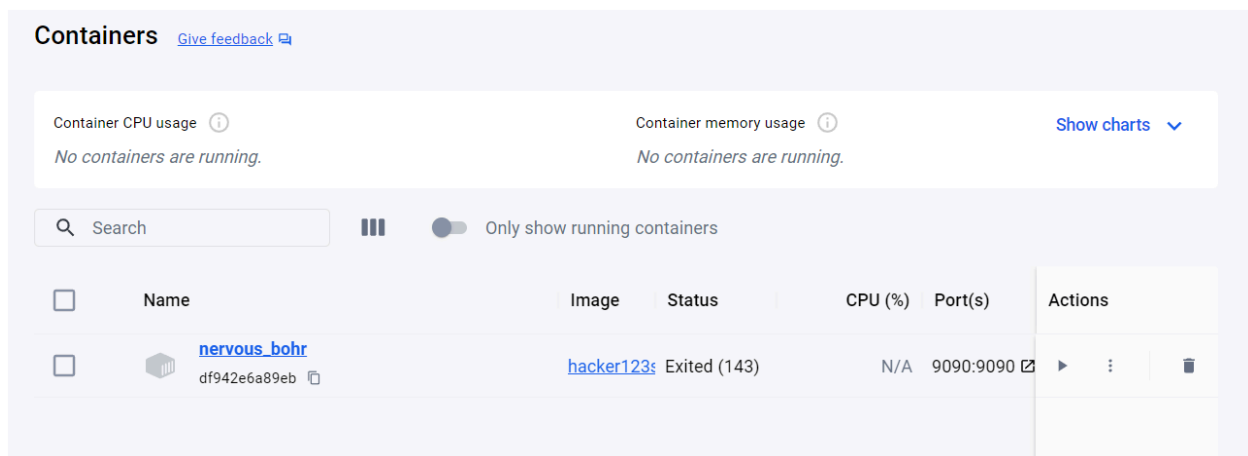
## Check images:

```
D:\Blog-application\Blog-project\docker-project>docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
hacker123shiva/docker-project   latest      82e718d03180     About an hour ago  345MB
hacker123shiva/springbt-in-docker latest      3f98d4db2087     4 months ago    493MB
hacker123shiva/springboot-app    <none>      229900cdf9f8     5 months ago    525MB
hacker123shiva/springboot-app    <none>      f912590698ea     5 months ago    525MB
hacker123shiva/springboot-app    <none>      bb409042f1cf     5 months ago    525MB
hacker123shiva/springbt-in-docker <none>      8fb764c970d6     5 months ago    493MB
mysql                    latest      6f343283ab56     6 months ago    632MB
```



## Check containers:

```
D:\Blog-application\Blog-project\docker-project>docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
PORTS         NAMES
df942e6a89eb   hacker123shiva/docker-project       "java -jar docker-pr..." About a minute ago Up About a minut
e 0.0.0.0:9090->9090/tcp   nervous_bohr
```



## Step 6: Push the Docker Image to Docker Hub

### Log in to Docker Hub:

```
docker login
```

1. Enter your Docker Hub credentials when prompted.

Or



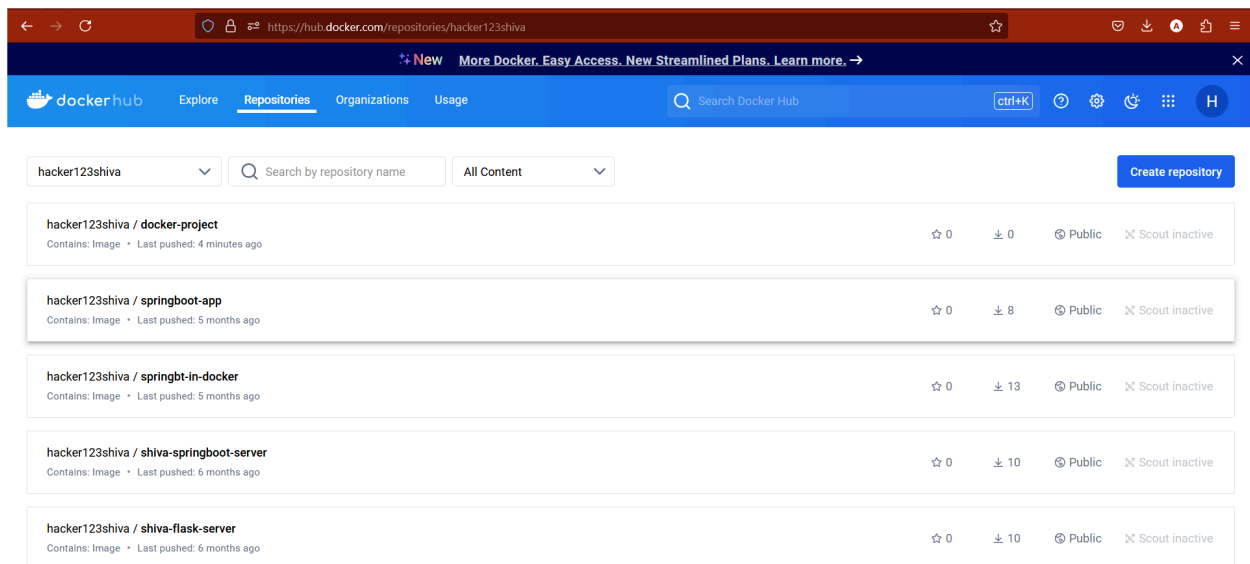
Direct login through browser (<https://hub.docker.com>)

## Push the image to Docker Hub:

```
docker push hacker123shiva/docker-project
```

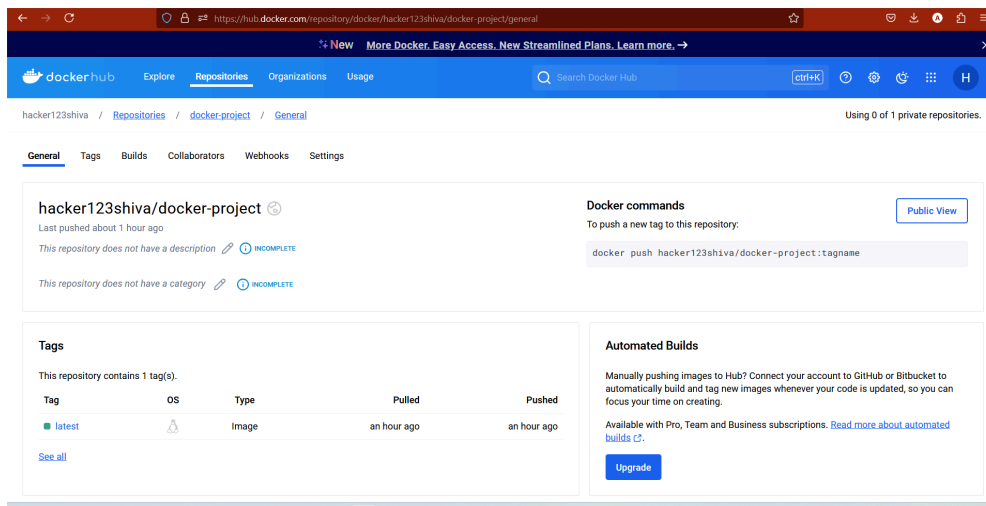
```
D:\Blog-application\Blog-project\docker-project>docker push hacker123shiva/docker-project
Using default tag: latest
The push refers to repository [docker.io/hacker123shiva/docker-project]
22e6716f9139: Pushed
960ee189585f: Pushed
34f7184834b2: Mounted from library/openjdk
5836ece05bfd: Mounted from library/openjdk
72e830a4dff5: Mounted from library/openjdk
latest: digest: sha256:15f6c5f458f06544d7bf93927a8cbc95c187419d4a1312f5349d06c84cd3ffa7 size: 1369

D:\Blog-application\Blog-project\docker-project>
```



The screenshot shows the Docker Hub interface for the user 'hacker123shiva'. The page lists several repositories:

- hacker123shiva / docker-project**: Contains: Image • Last pushed: 4 minutes ago. 0 stars, 0 pulls, Public, Scout inactive.
- hacker123shiva / springboot-app**: Contains: Image • Last pushed: 5 months ago. 0 stars, 8 pulls, Public, Scout inactive.
- hacker123shiva / springbt-in-docker**: Contains: Image • Last pushed: 5 months ago. 0 stars, 13 pulls, Public, Scout inactive.
- hacker123shiva / shiva-springboot-server**: Contains: Image • Last pushed: 6 months ago. 0 stars, 10 pulls, Public, Scout inactive.
- hacker123shiva / shiva-flask-server**: Contains: Image • Last pushed: 6 months ago. 0 stars, 10 pulls, Public, Scout inactive.



The screenshot shows the details page for the repository 'hacker123shiva/docker-project'. The page includes the following sections:

- General**: Last pushed about 1 hour ago. This repository does not have a description (incomplete). This repository does not have a category (incomplete).
- Docker commands**: To push a new tag to this repository: `docker push hacker123shiva/docker-project:tagname`. A 'Public View' button is available.
- Tags**: This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest	linux	Image	an hour ago	an hour ago

[See all](#)
- Automated Builds**: Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating. Available with Pro, Team and Business subscriptions. [Read more about automated builds](#). An 'Upgrade' button is present.

## Conclusion

Congratulations! You have successfully set up a Spring Boot project and deployed it using Docker. You can now access your application at <http://localhost:9090/hello>.

To pull and run the Docker image on another system, follow these steps:

**Pull the Image from Docker Hub:** Use the following command to pull your image from Docker Hub:

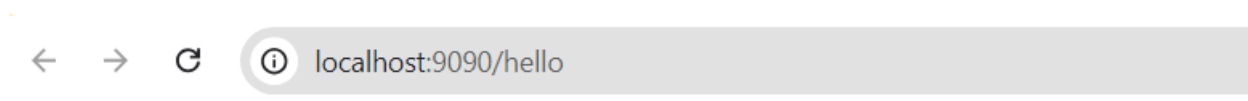
```
docker pull hacker123shiva/docker-project:latest
```

**Run the Docker Image:** After pulling the image, run it using the following command:

```
docker run -d -p 9090:9090 hacker123shiva/docker-project:latest
```

1. This command runs the container in detached mode (`-d`) and maps port 9090 on the container to port 9090 on the host (`-p 9090:9090`).
2. **Access the Application:** Open a web browser and go to <http://localhost:9090/hello> to see the application running.

Output:



# Welcome Java Developers!

## In world of Devops

**Note:** If you don't know Docker and Maven you can start learning from this page for complete Overview.

## Introduction to Containers

### What is Docker?

**Docker** is a platform that enables developers to **containerize applications**. Containers package an application and all its dependencies, ensuring that it runs consistently across different environments (e.g., development, testing, production).

### Why Containerization?

Without containers, developers may face the problem of "it works on my machine," where applications behave differently across environments due to differing dependencies. Containers eliminate this by bundling the application with everything it needs to run.

### Difference Between Containers and Virtual Machines (VMs)

Containers	Virtual Machines (VMs)
Lightweight, shares the host OS kernel	Full OS inside the VM
Starts in milliseconds	Takes seconds or minutes to start
Lower overhead (less CPU, memory, storage)	Higher overhead (more CPU, memory, storage)
Multiple containers can share the same OS	Each VM needs a separate OS

Suitable for microservices and modern apps

Suitable for running multiple OS environments

Docker is lightweight because it shares the host operating system's kernel instead of booting a full OS like virtual machines.

## Installing Docker

- **Windows:** Install Docker Desktop for Windows.
- **macOS:** Install Docker Desktop for Mac.
- **Linux:** Use the following commands for most distributions:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

---

## Basic Docker Concepts

### Images vs. Containers

- **Docker Image:** A read-only blueprint of an application that contains everything it needs to run.
- **Docker Container:** A running instance of a Docker image. Containers are isolated environments where the application runs.

### Dockerfile

A **Dockerfile** is a script containing a set of instructions for building a Docker image. It's a declarative way to create a custom Docker image.

### Basic Commands

```
# Pull an image from Docker Hub
docker pull nginx
# Run a container from the image
docker run -d -p 8080:80 nginx
# List running containers
```

```

docker ps
# Stop a running container
docker stop <container-id>
# Remove a stopped container
docker rm <container-id>

```

## Container Lifecycle

- **run container:** `docker run -d -p 9090:9090 hacker123shiva/docker-project`

```

View build details:
D:\Blog-application\Blog-project\docker-project>docker run -d -p 9090:9090 hacker123shiva
/docker-project
df942e6a89eb2b4db73601ce49d5b20435e8bde83b3959a2e6852bb69edc867c

```

- **Start:** `docker start <container-id>`
- **See all running container:** `docker ps`

```

D:\Blog-application\Blog-project\docker-project>docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
5504433af63b	hacker123shiva/docker-project	"java -jar docker-pr..."	9 minutes ago	Up 9 minutes	0.0.0.	inspiring_wozniak

- **Stop:** `docker stop <container-id>`

```
docker stop 55044331f63b
```

```

D:\Blog-application\Blog-project\docker-project>docker stop 5504433af63b
5504433af63b

```

```

D:\Blog-application\Blog-project\docker-project>docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

- **Check images in docker:** `docker images`

```

D:\Blog-application\Blog-project\docker-project>docker images

```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hacker123shiva/docker-project	latest	82e718d03180	2 hours ago	345MB
hacker123shiva/springbt-in-docker	latest	3f98d4db2087	4 months ago	493MB
hacker123shiva/springboot-app	<none>	229900cdf9f8	5 months ago	525MB
hacker123shiva/springboot-app	<none>	f912590698ea	5 months ago	525MB
hacker123shiva/springboot-app	<none>	bb409042f1cf	5 months ago	525MB
hacker123shiva/springbt-in-docker	<none>	8fb764c970d6	5 months ago	493MB
mysql	latest	6f343283ab56	6 months ago	632MB

- **Remove container:** `docker rm <container-id>`

```
docker rm df942e6a89eb
```

```
D:\Blog-application\Blog-project\docker-project>docker rm df942e6a89eb
df942e6a89eb
```

- **Remove image:** `docker rmi <image-id>`

e.g .

```
docker rmi hacker123shiva/docker-project
```

or

```
docker rmi 82e718d0380
```

```
D:\Blog-application\Blog-project\docker-project>docker images
REPOSITORY                                TAG          IMAGE ID      CREATED        SIZE
hacker123shiva/docker-project             latest       82e718d03180  2 hours ago   345MB
hacker123shiva/springbt-in-docker          latest       3f98d4db2087  4 months ago  493MB
hacker123shiva/springboot-app              <none>       229900cdf9f8  5 months ago  525MB
hacker123shiva/springboot-app              <none>       f912590698ea  5 months ago  525MB
hacker123shiva/springboot-app              <none>       bb409042f1cf  5 months ago  525MB
hacker123shiva/springbt-in-docker          <none>       8fb764c970d6  5 months ago  493MB
mysql                                       latest       6f343283ab56  6 months ago  632MB

D:\Blog-application\Blog-project\docker-project>docker rmi 82e718d03180
Untagged: hacker123shiva/docker-project:latest
Deleted: sha256:82e718d03180f47478de703a66c2d9aceb031c845538a98b90400d9cd8b88e65
```

- **Force remove image:** `docker rmi -f <image-id>`
- **Restart:** `docker restart <container-id>`

## Docker Hub

Docker Hub is the central repository for Docker images. You can **pull** images to your local system and **push** your images to Docker Hub.

```
# Pull an image from Docker Hub
```

```
docker pull ubuntu
```

```
# Push an image to Docker Hub
```

```
docker push <your-username>/<image-name>
```

---

## Building Docker Images

### Dockerfile Syntax

A basic **Dockerfile** structure:

```
# Base image
FROM openjdk:17-jdk-alpine
# Set the working directory
WORKDIR /app
# Copy the application JAR file into the container
COPY target/demo.jar /app/demo.jar
# Run the application
CMD ["java", "-jar", "demo.jar"]
```

### Key commands:

- **FROM**: Defines the base image.
- **WORKDIR**: Sets the working directory inside the container.
- **COPY**: Copies files from the local filesystem into the container.
- **CMD**: Specifies the default command to run in the container.

### Building Images

Use the **docker build** command to create an image from a Dockerfile.

```
docker build -t my-app .
```

### Layers in Docker Images

Each instruction in the Dockerfile creates a layer in the image. Docker caches these layers to improve build performance.

---

## Managing Docker Containers

### Ports and Networking

When you run a container, you may want to expose its internal ports to the host machine.

```
docker run -d -p 8080:80 nginx
```

This maps port **80** inside the container to port **8080** on the host machine.

### Volume Management

To persist data outside of the container, use **volumes**.

```
# Create a volume  
docker volume create my-volume  
# Run a container with a volume mounted  
docker run -d -v my-volume:/app/data my-app
```

---

## Docker Compose

### Introduction to Docker Compose

**Docker Compose** allows you to define and run multi-container applications using a **docker-compose.yml** file.

**docker-compose.yml**

#### Example

```
version: '3'  
services:  
  app:  
    image: my-app  
    ports:
```



```
- "8080:8080"
volumes:
  - my-data:/app/data
volumes:
  my-data:
```

## Basic Commands

```
# Start the services defined in docker-compose.yml
docker-compose up
# Stop the services
docker-compose down
# View logs of the running services
docker-compose logs
```

---

## Advanced Docker Networking

### Bridge Network

The **bridge network** is Docker's default network. Containers can communicate with each other using the container name.

```
docker run -d --name container1 my-app
docker run -d --name container2 --network container1 my-app
```

### Host Network

Running containers on the host network gives the container access to the host's network stack.

```
docker run --network host my-app
```

## Custom Networks

You can create custom networks for containers to communicate securely.

```
# Create a custom network  
docker network create my-network  
# Run containers on the custom network  
docker run --network my-network my-app
```

---

## Conclusion

This guide covers the essentials of Docker, from basic concepts to advanced networking. Each section gives you a foundation in understanding containers, managing Docker images and containers, and deploying multi-container applications using Docker Compose.

## Steps to Work with Docker with example

---

### 1. Create the **Dockerfile**

Ensure you have a **Dockerfile** in the root directory of your Spring Boot project (next to the **pom.xml** file).

**Dockerfile:**

```
# Use an official OpenJDK runtime as a parent image  
FROM openjdk:17-jdk-alpine
```

```
# Set the working directory inside the container
WORKDIR /app

# Copy the jar file from the target folder into the container
COPY target/docker-project-0.0.1-SNAPSHOT.jar /app/docker-project.jar

# Expose the port that Spring Boot will run on
EXPOSE 9090

# Run the jar file
ENTRYPOINT ["java", "-jar", "docker-project.jar"]
```

---

## 2. Build the Project JAR

Package the Spring Boot project into a `.jar` file using Maven.

```
# Clean and build the Maven project
mvn clean package
```

This will create a `demo-docker-project.jar` file in the `target` directory.

---

## 3. Build the Docker Image

Now, use Docker to build the image from your `Dockerfile`.

Open Docker Desktop before running below command.

```
# Build the Docker image and tag it as hacker123shiva/docker-project
docker build -t hacker123shiva/docker-project .
```

This command will:

- Build a Docker image using the Dockerfile.
- Tag the image with the name `hacker123shiva/docker-project`.

---

## 4. Run the Docker Container

Now that the Docker image is built, run a container based on it. You'll map the host's port (`9090`) to the container's port (`9090`).

```
# Run the container, exposing port 9090 on both host and container  
docker run -p 9090:9090 hacker123shiva/docker-project
```

This will:

- Map port `9090` from your host machine to port `9090` inside the container.
- Run the container, making your Spring Boot application accessible on `http://localhost:9090/hello`.

---

## 5. Push the Image to Docker Hub

If everything works fine, you can push the image to Docker Hub.

### 1. Tag the Docker image:

```
docker tag hacker123shiva/docker-project hacker123shiva/docker-project:v1
```

### 2. Login to Docker Hub:

```
docker login
```

### 3. Push the image:

```
docker push hacker123shiva/docker-project:v1
```

Your image will now be available on Docker Hub at

<https://hub.docker.com/r/hacker123shiva/docker-project>.

**Note:** If you have issues with Maven, then what should we do? *There are two methods to solve this issue.*

# Maven Issues resolution:

## Method 1: Download and Set Up Maven in Environment Variables

1. **Download Maven:**
  - Go to the [Maven Official Website](#).
  - Download the **Binary zip archive** (e.g., `apache-maven-3.9.4-bin.zip`).
2. **Extract the Zip File:**
  - Extract the downloaded file to a directory, such as `C:\Program Files\Apache\Maven` (or any other location you prefer).
3. **Set Up Environment Variables on Windows:**

**Steps:**

  - Right-click on **This PC** (or **My Computer**) and select **Properties**.
  - Click on **Advanced system settings**.
  - In the **System Properties** window, click on the **Environment Variables** button.
  - Under **System variables**, find the `Path` variable, and select **Edit**.

Click **New**, and add the path to the `bin` directory of Maven:

`C:\Program Files\Apache\Maven\bin`

- **Create a new `MAVEN_HOME` environment variable** (if not created):
  - Click on **New** under **System variables**.
  - Set **Variable name** to `MAVEN_HOME`.

Set **Variable value** to the directory where Maven was extracted:

`C:\Program Files\Apache\Maven`

4. **Verify the Maven Installation:**
  - Open **Command Prompt** or **Terminal**.

Run the following command to verify if Maven is set up correctly:

```
mvn -version
```

5. If correctly set up, it will display Maven's version and environment details.

---

## Method 2: Run Maven Directly Through Spring Tool Suite (STS)

If Maven is not set up in your environment variables, you can still run Maven directly from the **Spring Tool Suite (STS)** IDE.

1. **Import or Create the Spring Boot Project:**
  - If you haven't done so, create or import your Spring Boot project in STS.
2. **Use Maven Wrapper** (If your project has `mvnw`):
  - Most Spring Boot projects come with a **Maven Wrapper** (`mvnw` for Linux/macOS, or `mvnw.cmd` for Windows).
  - **Run Maven commands via Maven Wrapper** directly from your terminal or IDE without needing to install Maven globally.
3. **Example:**

In **STS Terminal** or the project's root directory:

```
./mvnw clean package # For Linux/macOS  
mvnw.cmd clean package # For Windows
```

4. **Run Maven Goals from STS:**  
**Steps:**
  - **Right-click** on your project in the **Project Explorer**.
  - Select **Run As > Maven Build...**
  - In the **Goals** field, enter the Maven command, such as `clean package`, and click **Run**.
5. This will execute Maven goals using the Maven installation configured in STS.
6. **View Maven Console:**
  - STS will use the built-in Maven support to run the build, and you can view the output in the **Console** tab of the IDE.

---

## Conclusion:

- **Method 1:** Download and configure Maven in your system environment, allowing you to run Maven from any terminal or command line globally.
- **Method 2:** Use **Spring Tool Suite (STS)** to run Maven commands directly through the IDE or using the Maven Wrapper (`mvnw`).