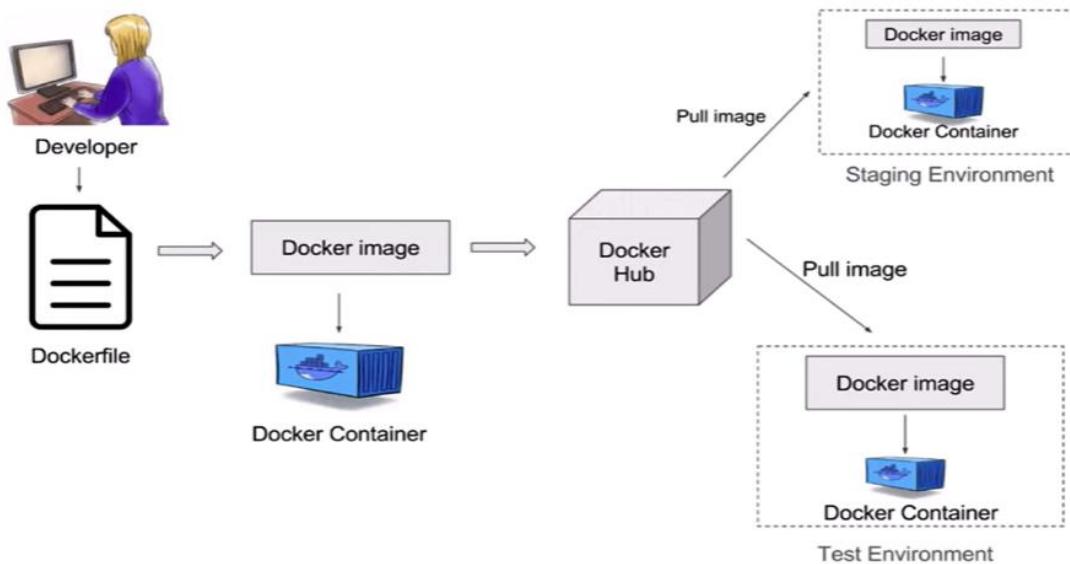


Dockerizing a Spring Boot Application using IntelliJ Environment

Dockerizing a Spring Boot Application - In this tutorial, you will learn how to use Docker to deploy the Spring Boot application in a container. This Spring Boot Docker tutorial is designed for beginners to learn step-by-step how to use Docker with the Spring Boot application.

Docker Workflow



The developer will create a **Dockerfile**, where all the instructions and commands to build the docker image. Dockerfile is a text file, it contains all instructions or commands to build the docker image. Once you create a Dockerfile, you can use the Docker build command to build the docker image. Docker image is an executable package and we can run the docker image in a container. Docker container is just a running instance of Docker image. The process of writing the Dockerfile and building the docker image is called **Dockerization**.

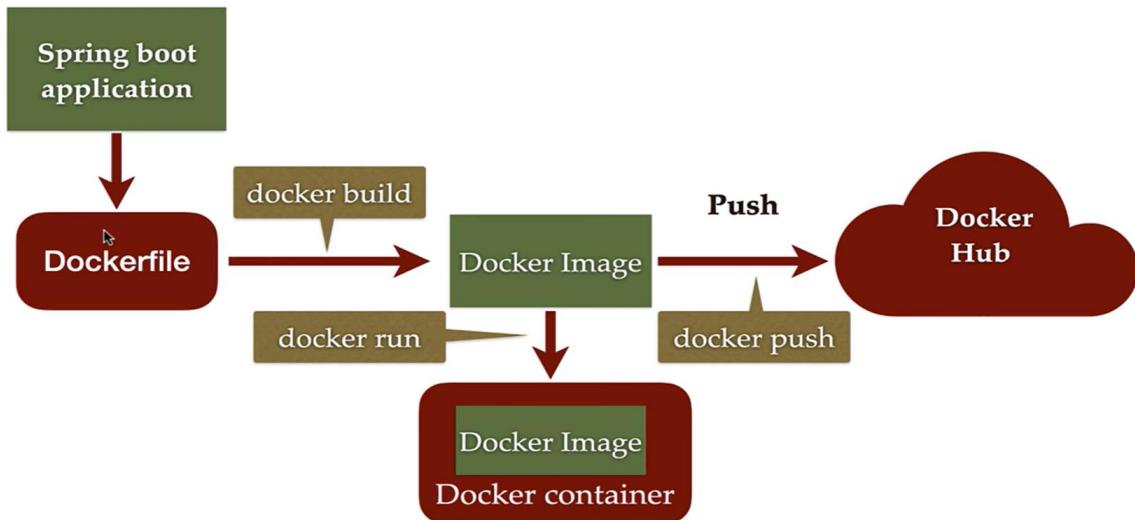
Once we have the Docker image in our local machine, we can push the image to an online Cloud repository called Docker hub. Once we push the image to the Docker hub, we can pull that image on any environment and we can start deploying or running this Docker image in a Docker container. For example, let us say we have different environments like staging, test, production or UAT environment. This is how a typical Docker Workflow looks like.

Architecture

Consider we want to Dockerize the Spring Boot application. We have to first create a Dockerfile where we will define all the instructions or commands to build the Docker Image. Once the Dockerfile is ready, we can use the Docker Build Command to build the Docker image from the Dockerfile. The Docker Image is just an executable package.

Next, we can use the Docker Run Command to run this Docker Image in a container. Docker container is just a running instance or runtime instance of the Docker Image. This complete process is called **Dockerization**.

Once we have a Docker Image on our local machine, then we can go ahead and push this Docker Image on a Docker Hub using Docker Push Command. Once we push the Docker Image to the Docker hub, and if we make this Docker Image as a public on Docker Hub. Then anybody can go ahead and pull this Docker Image from the Docker hub and they can run that Docker image in a Docker Image.



IMPLEMENTATION

Now, let us see how to Dockerize Spring Boot application step by step. We have to implement these steps now.

Dockerizing Spring Boot Application:

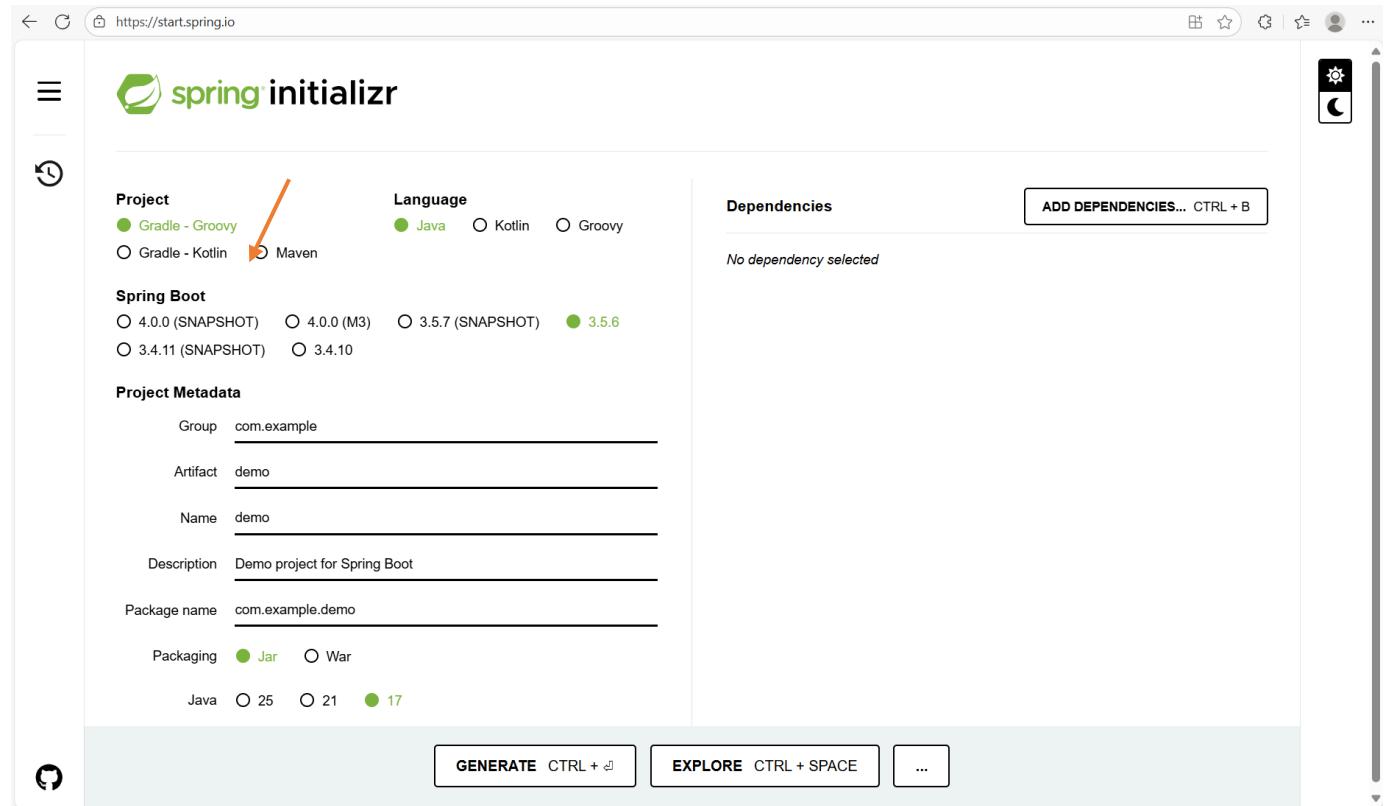
1. Create a Spring Boot Project
2. Build a Simple REST API
3. Build and Package Maven application
4. Create a Dockerfile to Build a Docker Image
5. Build Docker Image from Dockerfile
6. Run Docker Image in a Docker Container
7. Run Docker Image in a Docker Container in Detached Mode
8. Push Docker Image from Local Machine to DockerHub
9. Pull the Docker Image from a Docker Hub

Pull Docker Image from DockerHub

STEP 1: Create a Spring Boot Project

First, we will create a Spring Boot application and we will build some REST API. Let us create a Spring Boot application and we will import that Spring Boot application.

Let us start by generating a simple Spring boot project. Go to <http://start.spring.io>



The screenshot shows the Spring Initializr web interface at <https://start.spring.io>. The 'Project' section has 'Maven' selected (indicated by an orange arrow). Other options like 'Gradle - Groovy' and 'Gradle - Kotlin' are shown with radio buttons. The 'Language' section has 'Java' selected (indicated by an orange arrow). Other options like 'Kotlin' and 'Groovy' are shown with radio buttons. The 'Dependencies' section is empty, showing 'No dependency selected'. There is a button to 'ADD DEPENDENCIES... CTRL + B'. The 'Project Metadata' section includes fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), Package name (com.example.demo), and Packaging (Jar). Below these, Java version options (25, 21, 17) are shown with radio buttons, where 17 is selected (indicated by an orange arrow). At the bottom, there are 'GENERATE CTRL + D' and 'EXPLORE CTRL + SPACE' buttons, along with a '...' button.

On "Project", we will choose "Maven"

The screenshot shows the Spring Initializr web application at <https://start.spring.io>. The user has selected "Maven" as the project type and "Java" as the language. In the "Spring Boot" section, the "3.5.6" version is highlighted with a green dot and an orange arrow points to it from the text above. The "Project Metadata" section includes fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), Package name (com.example.demo), and Packaging (Jar). Below these, Java version statistics are shown: 25, 21, and 17. At the bottom, there are "GENERATE" and "EXPLORE" buttons.

On “Spring Boot”, we will use version “3.5.6”

This screenshot shows the same Spring Initializr interface as the previous one, but with a horizontal orange arrow pointing to the "Group" field in the "Project Metadata" section. The field currently contains "com.example". The rest of the interface remains the same, including the selection of Spring Boot version 3.5.6.

Let us change the Group to “net.javaguides”

The screenshot shows the Spring Initializr interface. In the 'Project' section, 'Gradle - Groovy' and 'Gradle - Kotlin' are listed, with 'Gradle - Groovy' selected. Under 'Language', 'Java' is selected. In the 'Spring Boot' section, '3.5.6' is selected. The 'Project Metadata' section includes fields for Group (net.javaguides), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), Package name (net.javaguides.springboot), and Packaging (Jar). Below these, Java version 17 is selected. At the bottom, there are 'GENERATE' and 'EXPLORE' buttons.

We have to give the project a name, I will call it “**springboot-docker-demo**”

This screenshot is identical to the previous one, but the 'Package name' field has been changed to 'net.javaguides.springboot.docker-demo'. An orange arrow points from the text 'For “Package Name”, we will use “net.javaguides.springboot”' to this field.

For “**Package Name**”, we will use “**net.javaguides.springboot**”

The screenshot shows the Spring Initializr web application. In the 'Project' section, 'Maven' is selected. Under 'Spring Boot', '3.5.6' is selected. In the 'Project Metadata' section, the group is set to 'net.javaguides', artifact to 'springboot-docker-demo', name to 'springboot-docker-demo', description to 'Demo project for Spring Boot', and package name to 'net.javaguides.springboot'. The 'Packaging' section shows 'Jar' selected. An orange arrow points from the text 'On “Packaging”, we will use “Jar”' to the 'Jar' button. Below the packaging options, Java versions 25, 21, and 17 are listed, with 17 being highlighted. At the bottom, there are 'GENERATE' and 'EXPLORE' buttons.

On “Packaging”, we will use “Jar”

This screenshot is identical to the one above, but Java version 21 is now highlighted instead of 17. An orange arrow points from the text 'We will use Java version “17” or above. So, we will just use Java Version 17' to the '21' button.

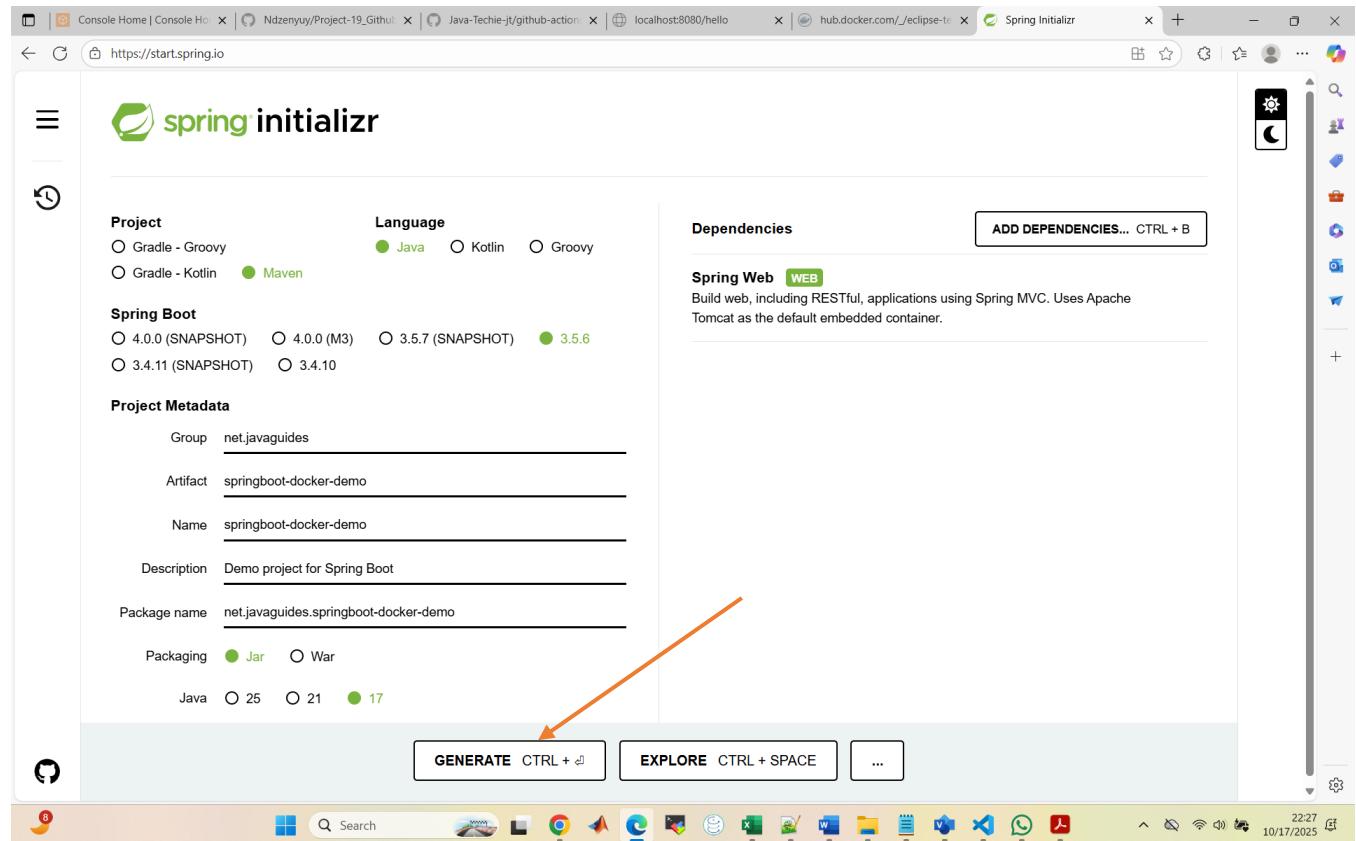
We will use Java version “17” or above. So, we will just use Java Version 17

The screenshot shows the Spring Initializr web application at <https://start.spring.io>. The interface includes sections for Project (Gradle - Groovy, Gradle - Kotlin, Maven), Language (Java, Kotlin, Groovy), and Spring Boot (4.0.0 (SNAPSHOT), 4.0.0 (M3), 3.5.7 (SNAPSHOT), 3.5.6, 3.4.11 (SNAPSHOT), 3.4.10). A 'Project Metadata' section contains fields for Group (net.javaguides), Artifact (springboot-docker-demo), Name (springboot-docker-demo), Description (Demo project for Spring Boot), Package name (net.javaguides.springboot-docker-demo), Packaging (Jar), Java version (25), and Java minor version (17). On the right, a 'Dependencies' section says 'No dependency selected' and features a prominent 'ADD DEPENDENCIES... CTRL + B' button, which is highlighted with an orange arrow. Below the dependencies section are 'GENERATE' and 'EXPLORE' buttons.

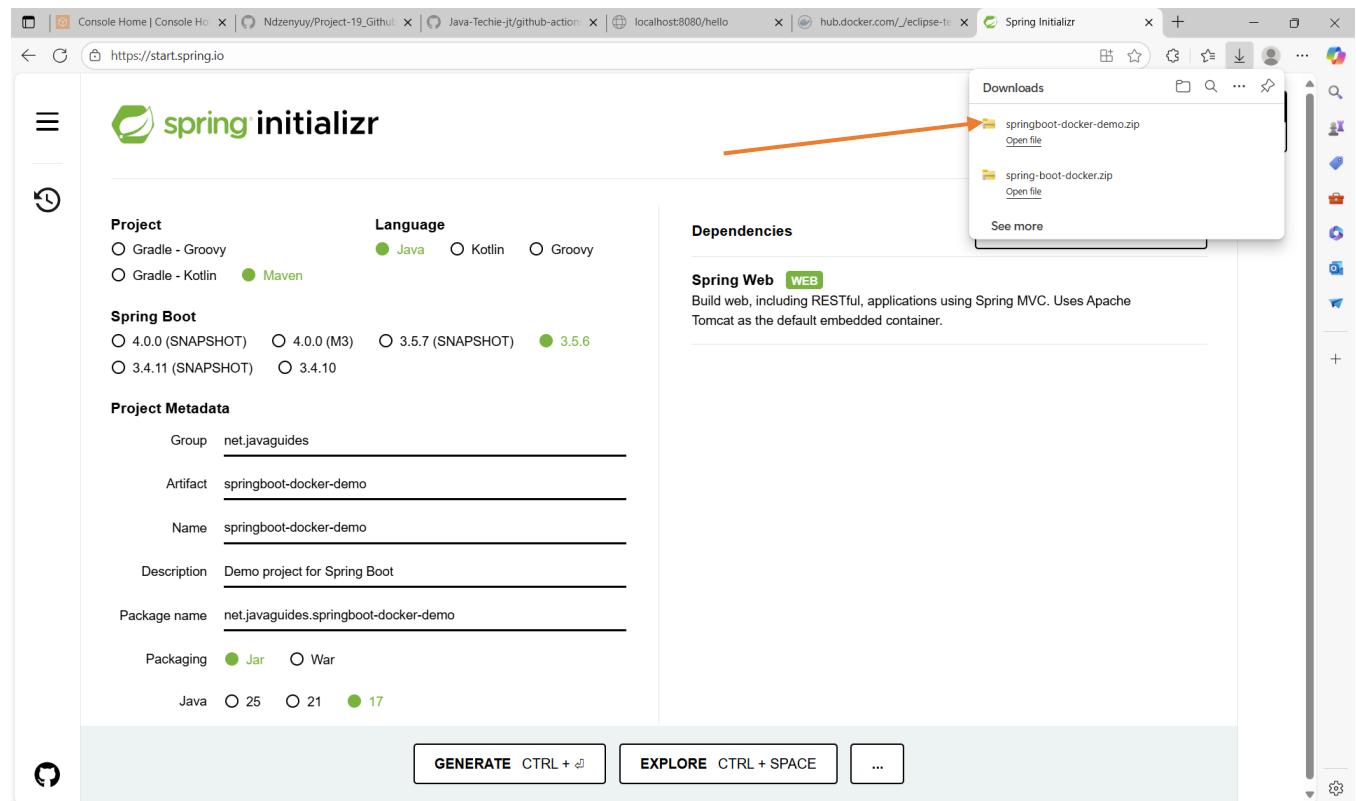
Click on “Add Dependencies”

The screenshot shows the Spring Initializr interface after clicking the 'Add Dependencies' button. The 'Dependencies' section now lists several modules under the 'WEB' tab. The 'Spring Web' section is expanded, showing the requirement: 'Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.' The 'Spring Reactive Web' section is also visible, requiring 'Spring Boot >= 4.0.0-M1'. Other sections like 'HTTP Client' and 'Reactive HTTP Client' are listed below.

Click on “Spring Web”

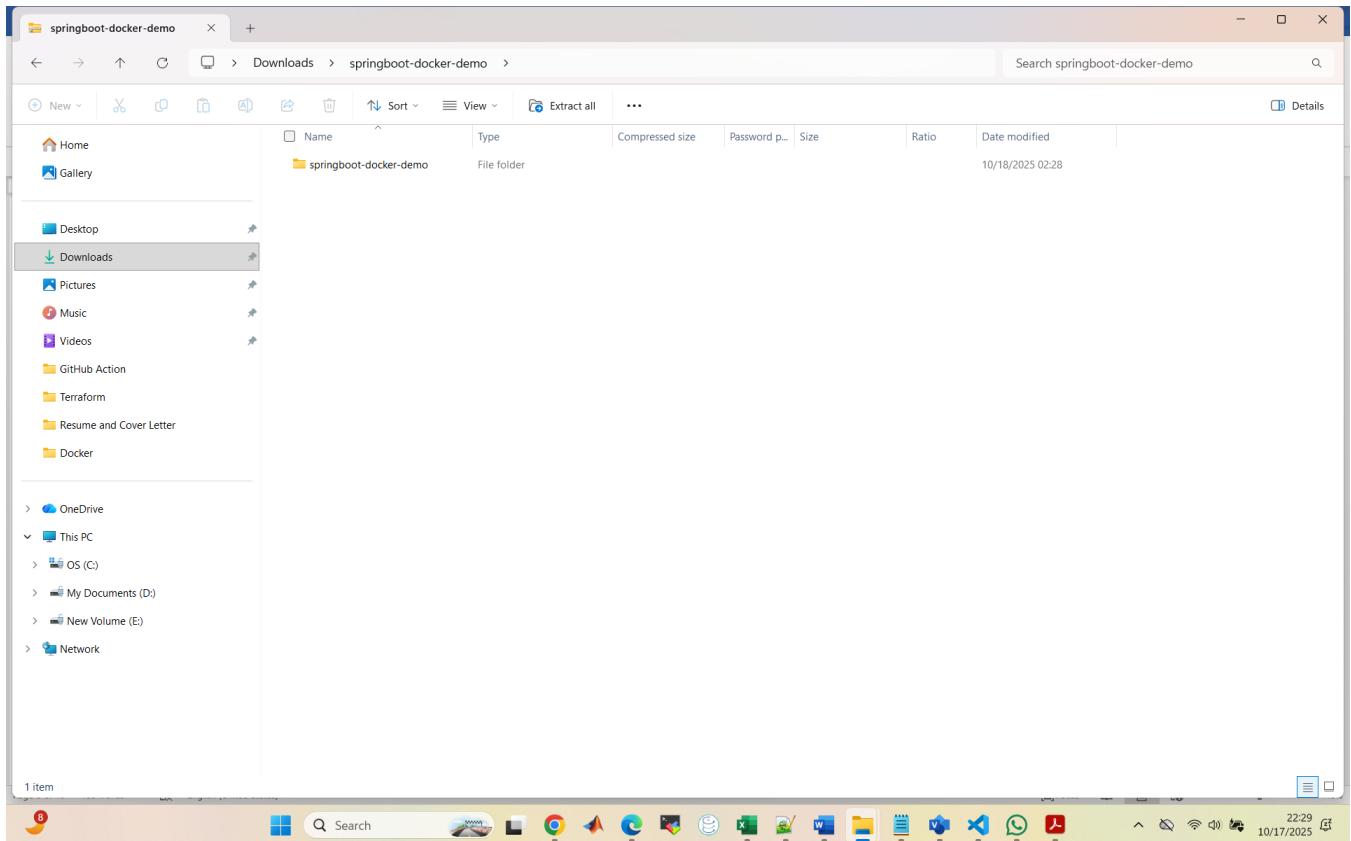


Then click on “Generate CTRL”



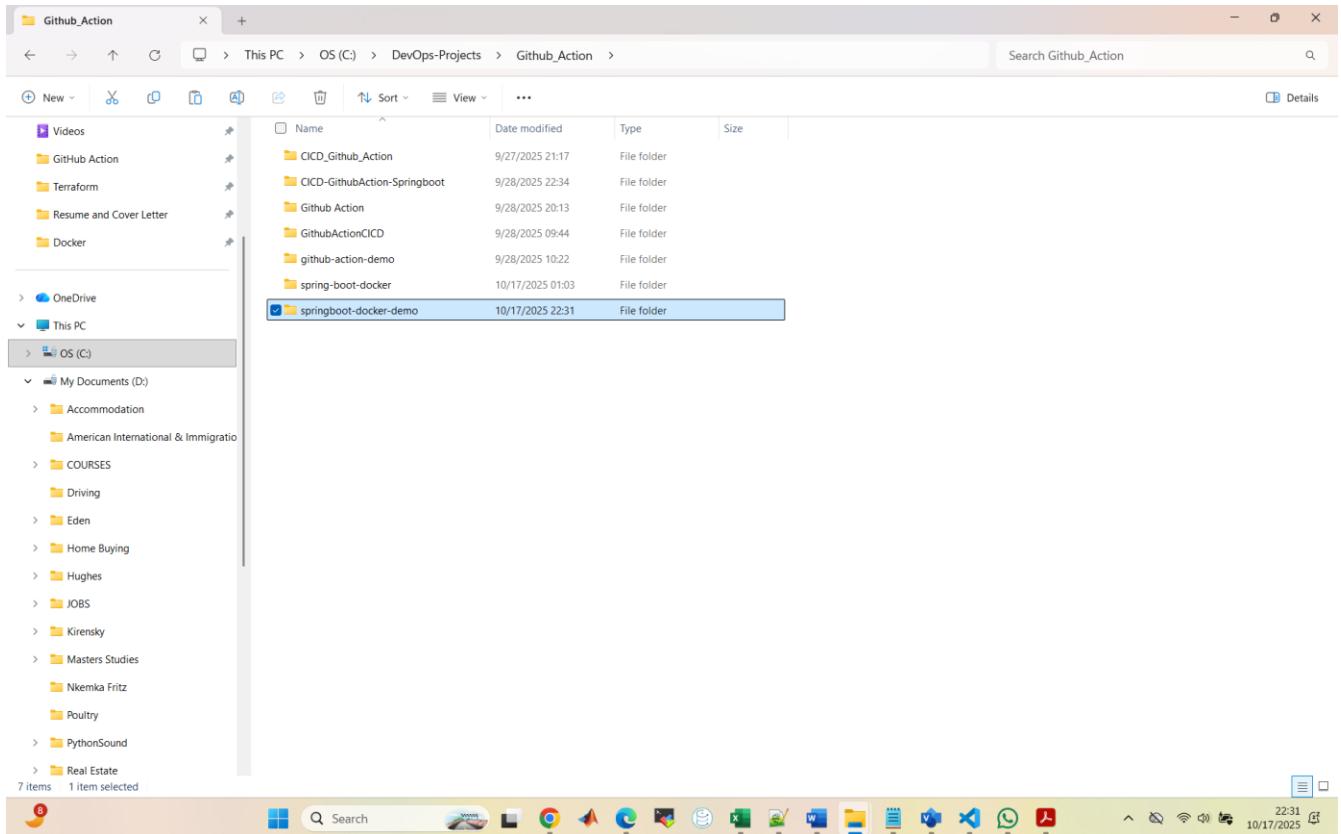
The project has been created and the project folder has been downloaded to the “Downloads” folder.

Prepared by Sidney Smith Ebots



Copy the project folder and save in a suitable location. I will save it in this location:

C:\DevOps-Projects\Github_Action\springboot-docker-demo

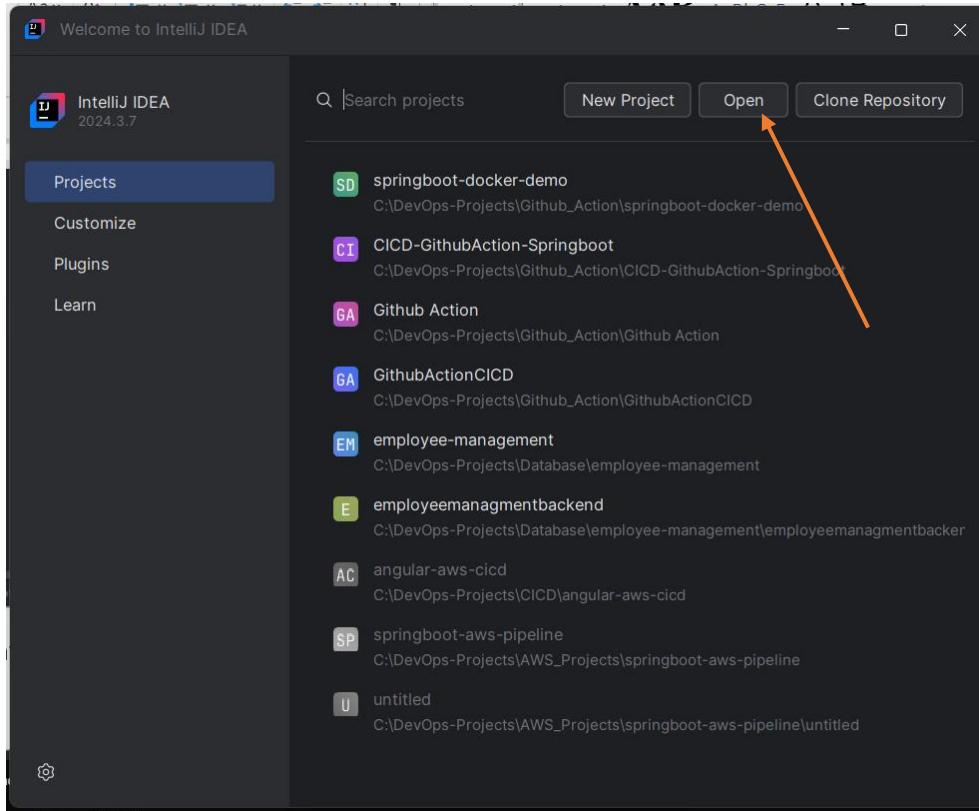


STEP 2: Build a Simple REST API

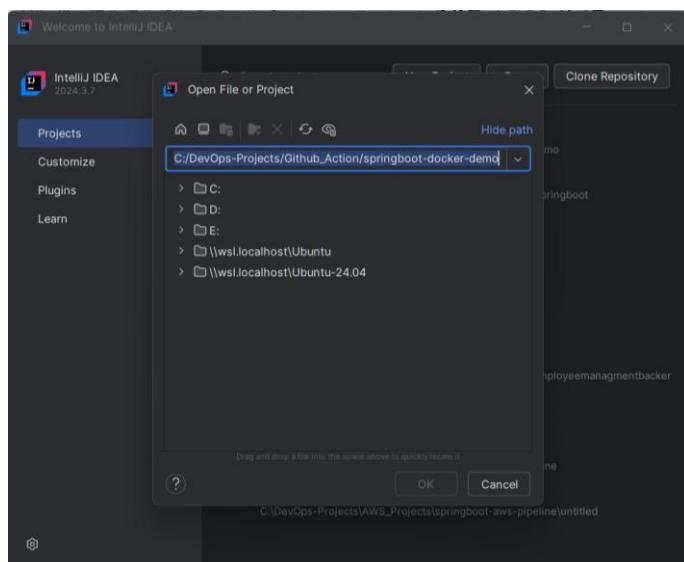
We are going to open the project in IntelliJ. This is done as follows

Part 1: Open IntelliJ and Navigate to Project Folder

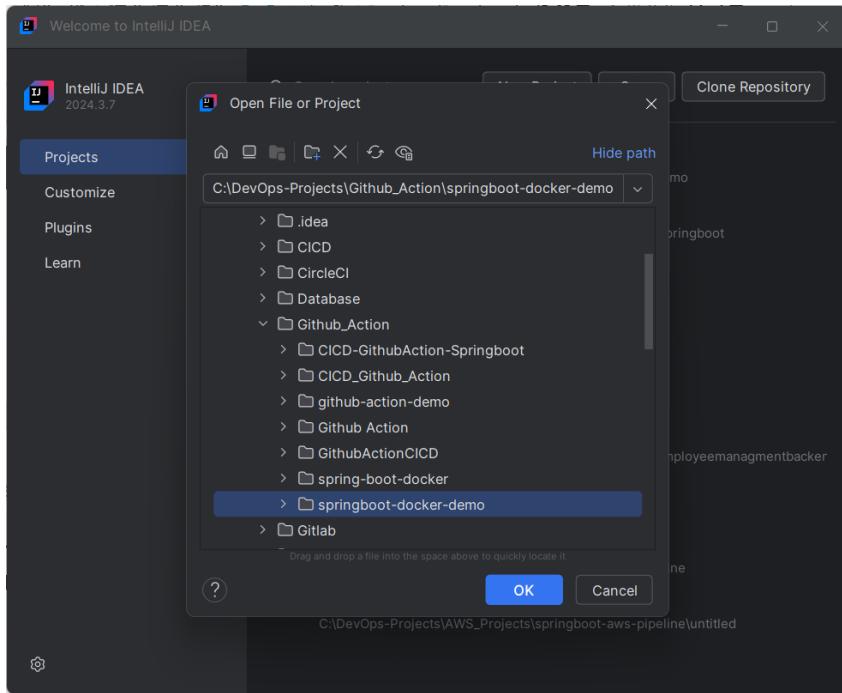
Open IntelliJ



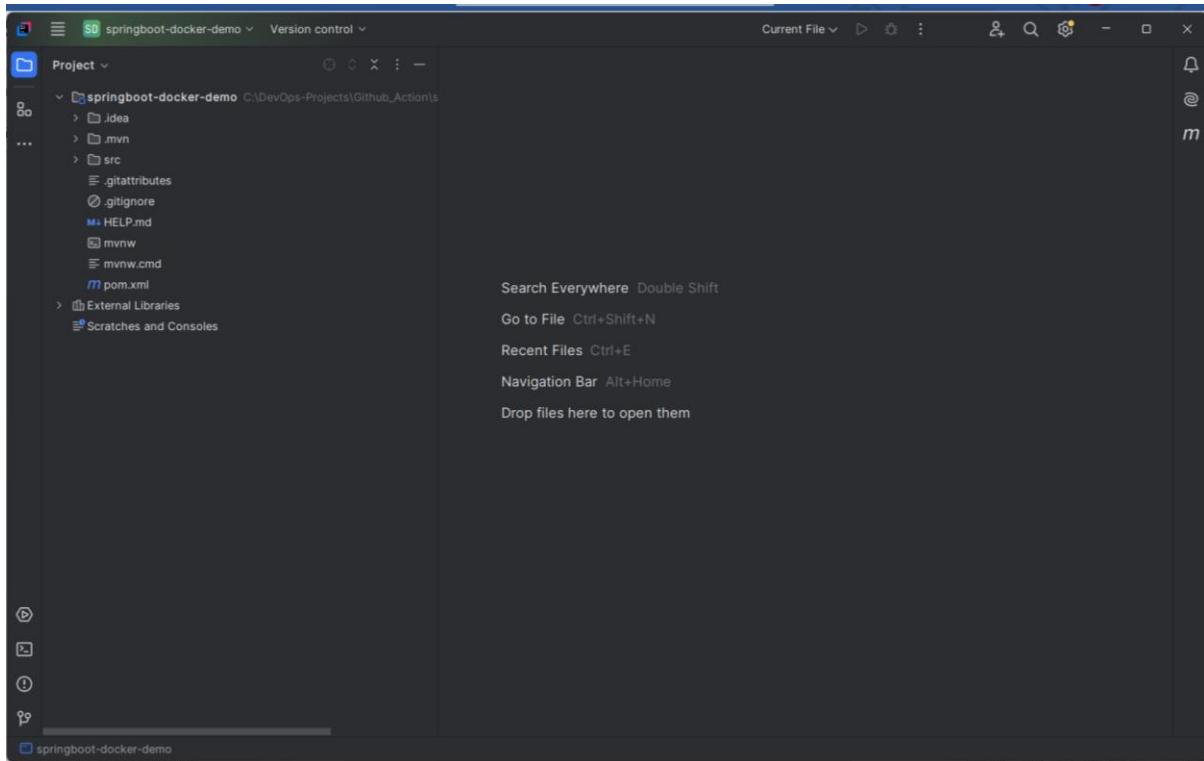
Navigate to the project folder. Click on “Open”



Our project file is located in C:\DevOps-Projects\Github_Action\springboot-docker-demo. So, we navigate to the project folder.

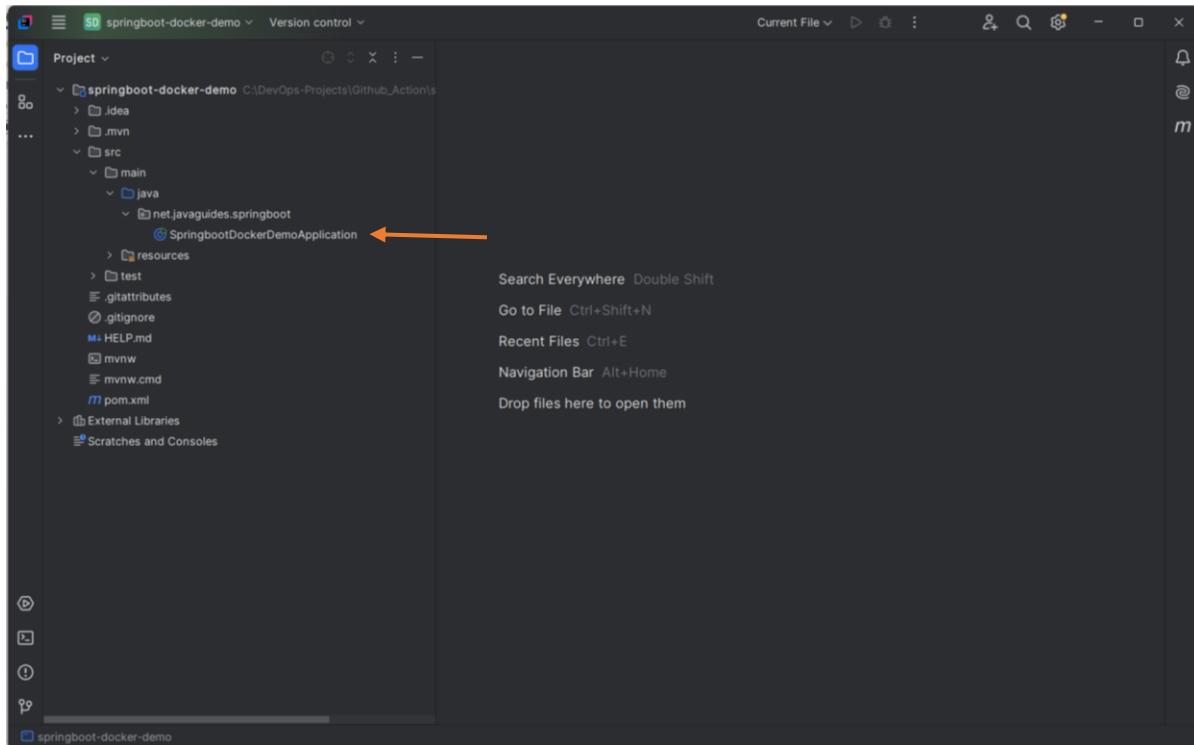


Click on “OK”

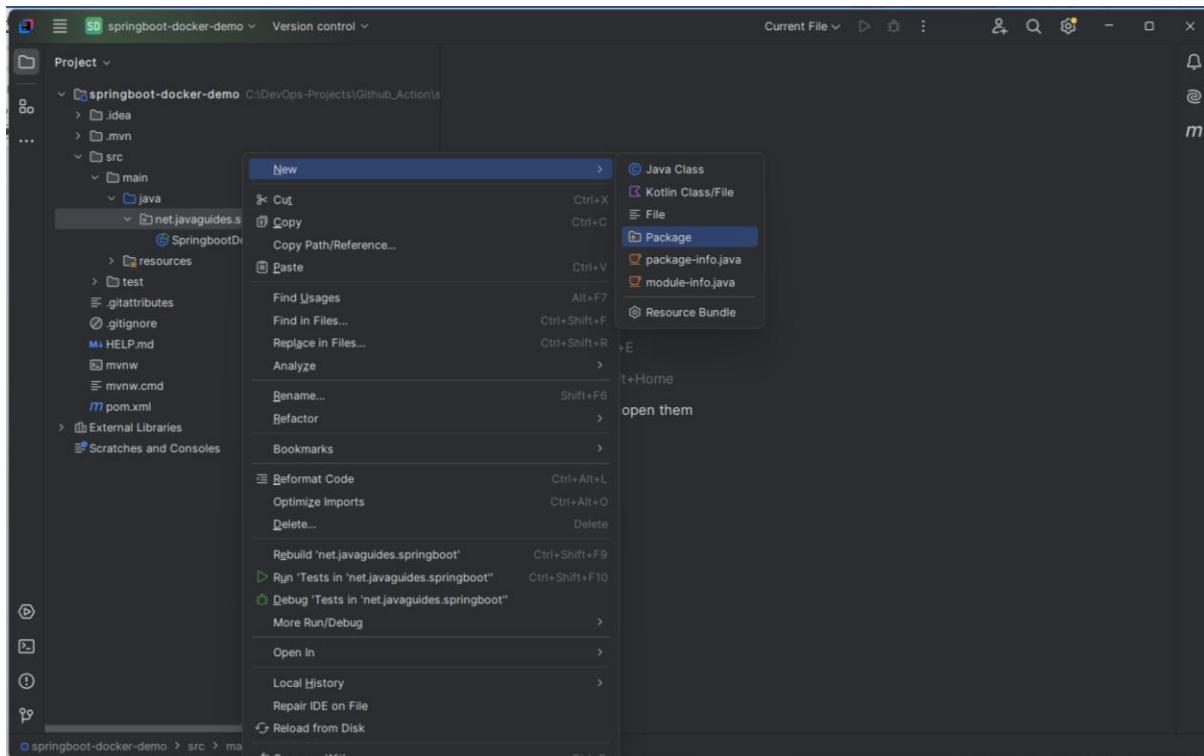


Part 2: Create a Java Package

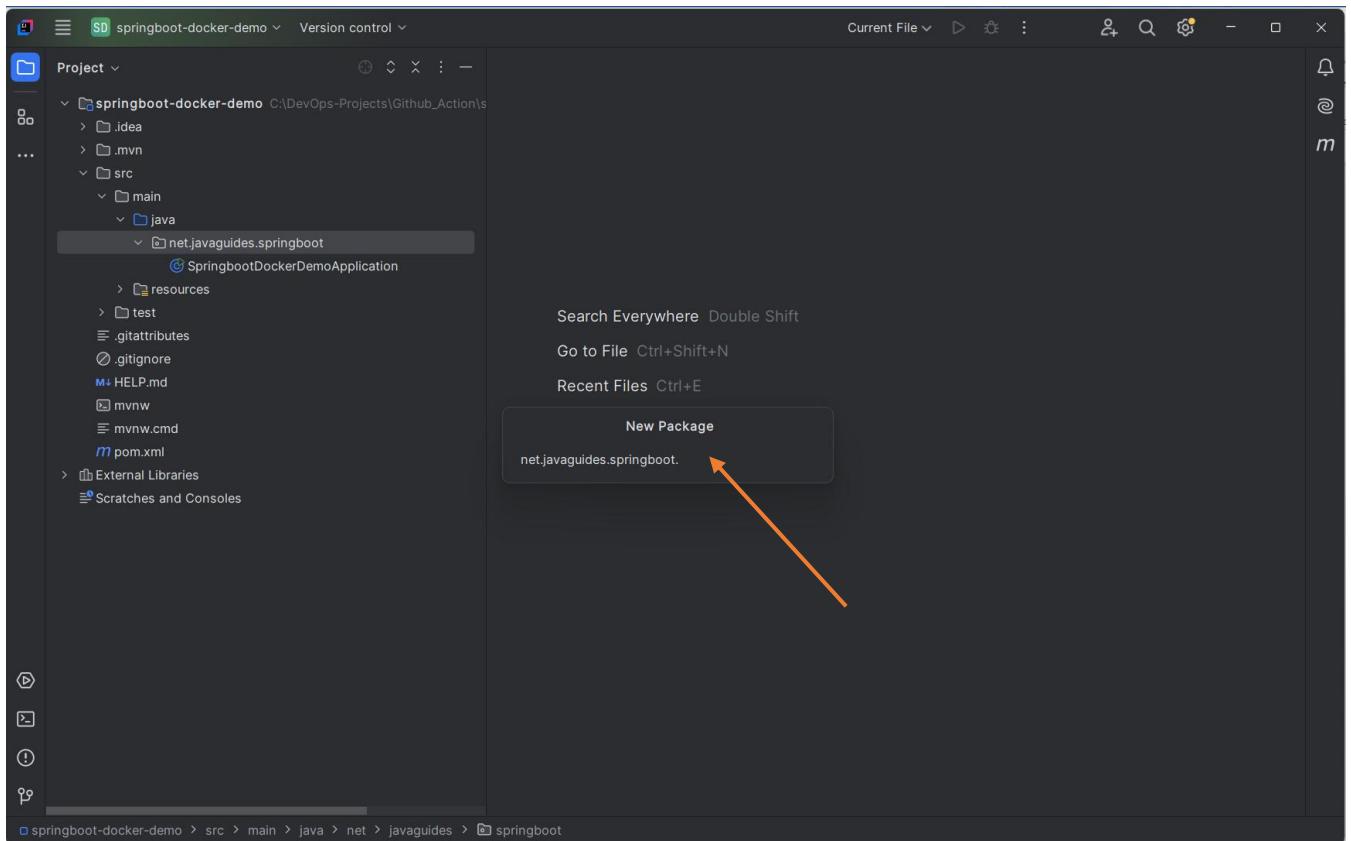
Let us create a new folder in “src” -> “main” -> “Java”



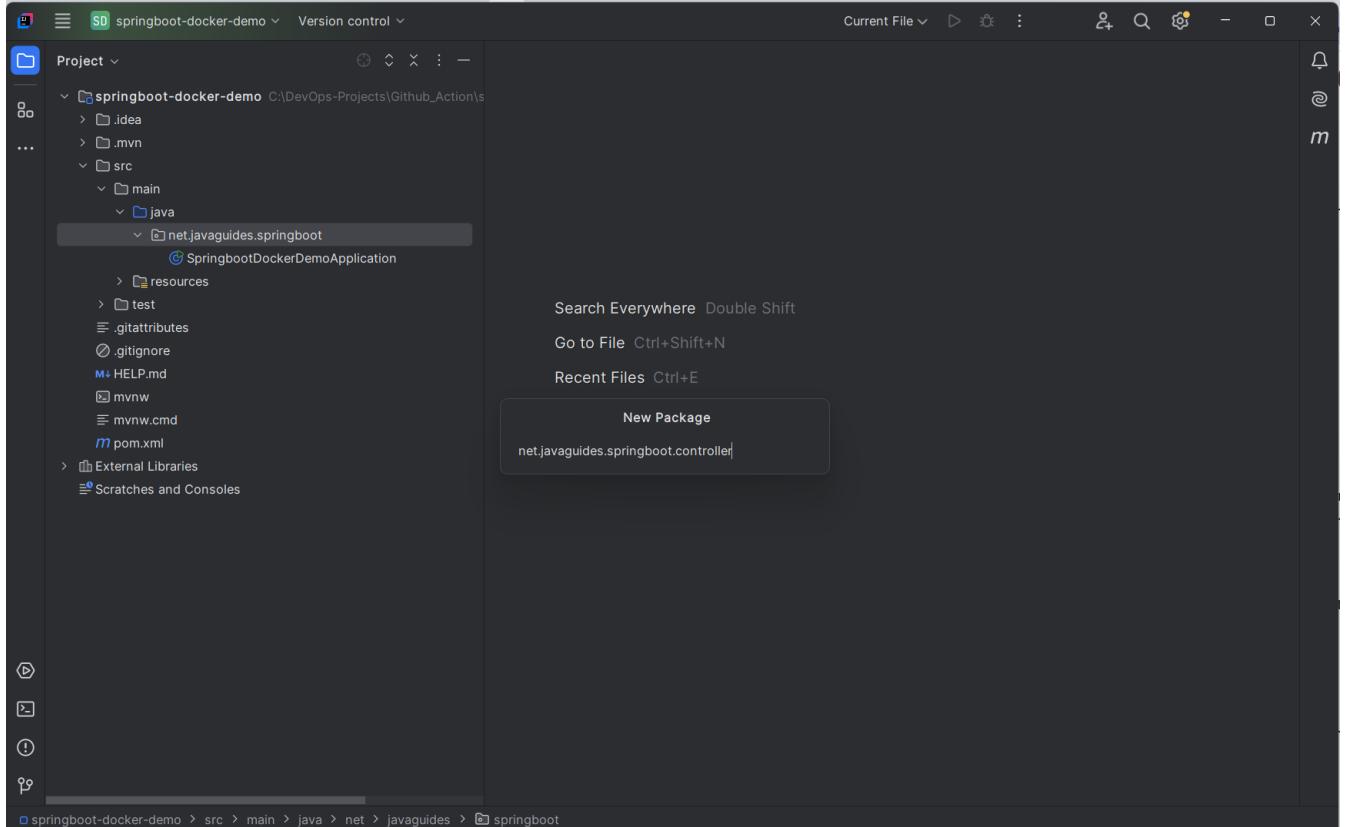
We will create a package. Right-click on “net.javaguides.springboot”



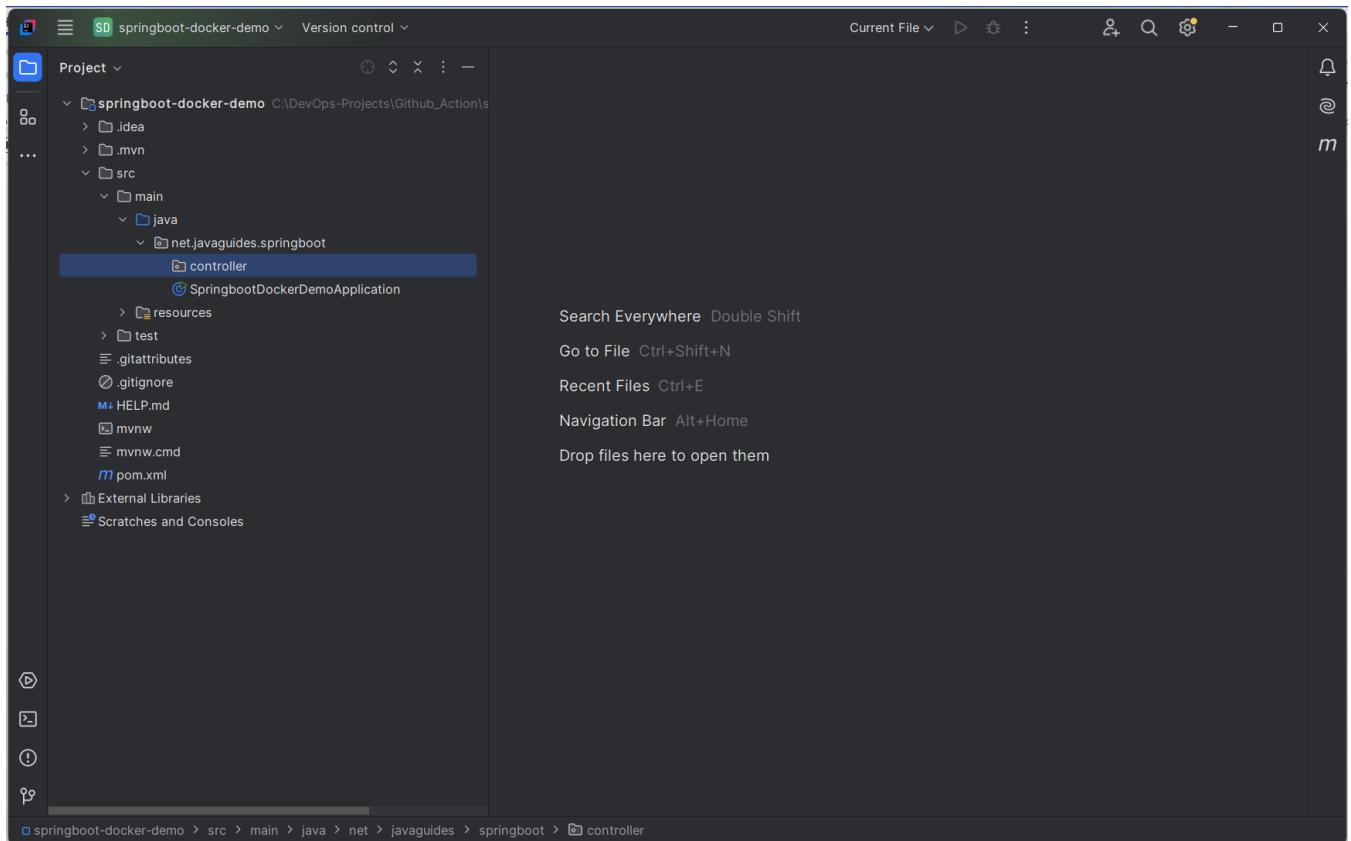
Select “Package”



Name it “controller”

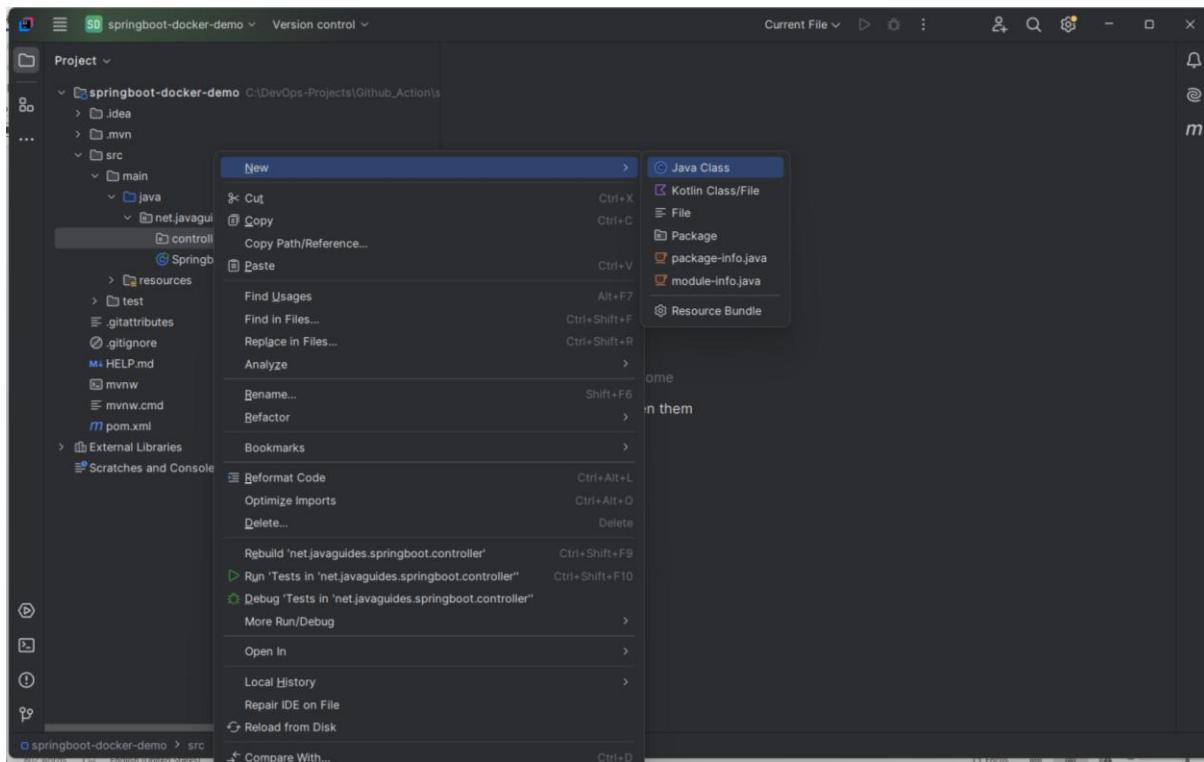


And press “Enter”

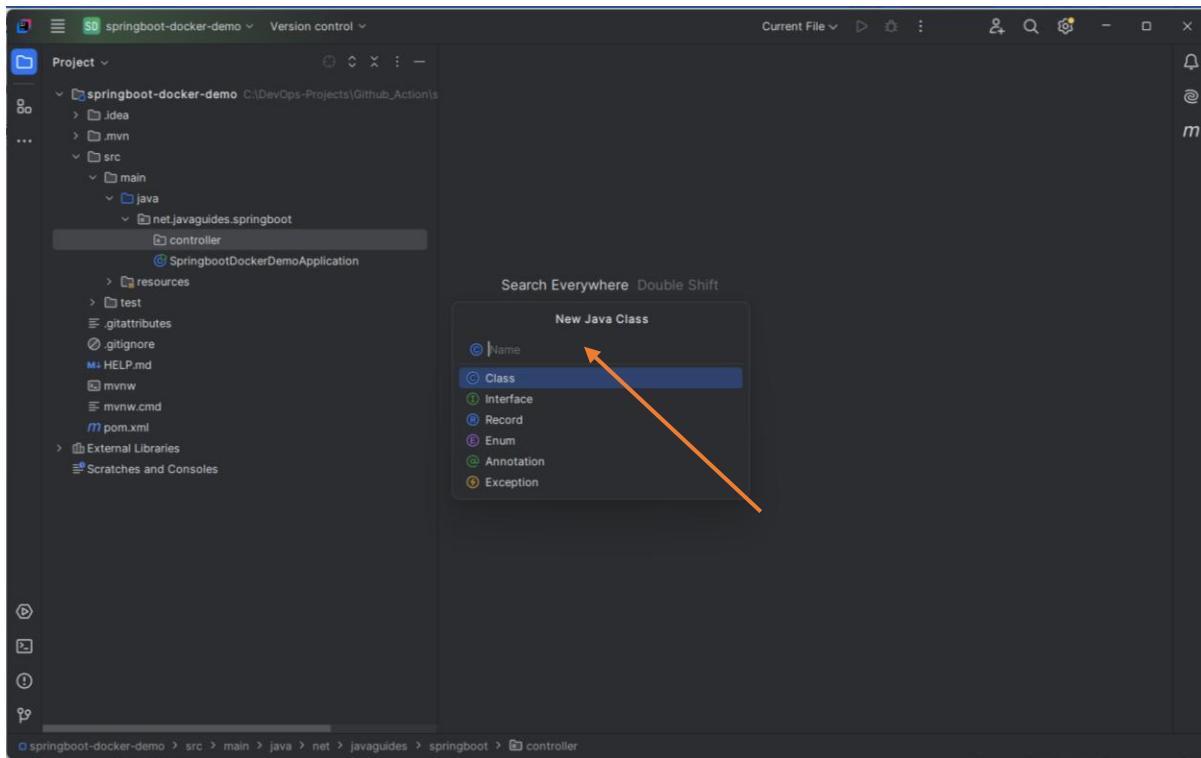


Part 3: Create Java Class file

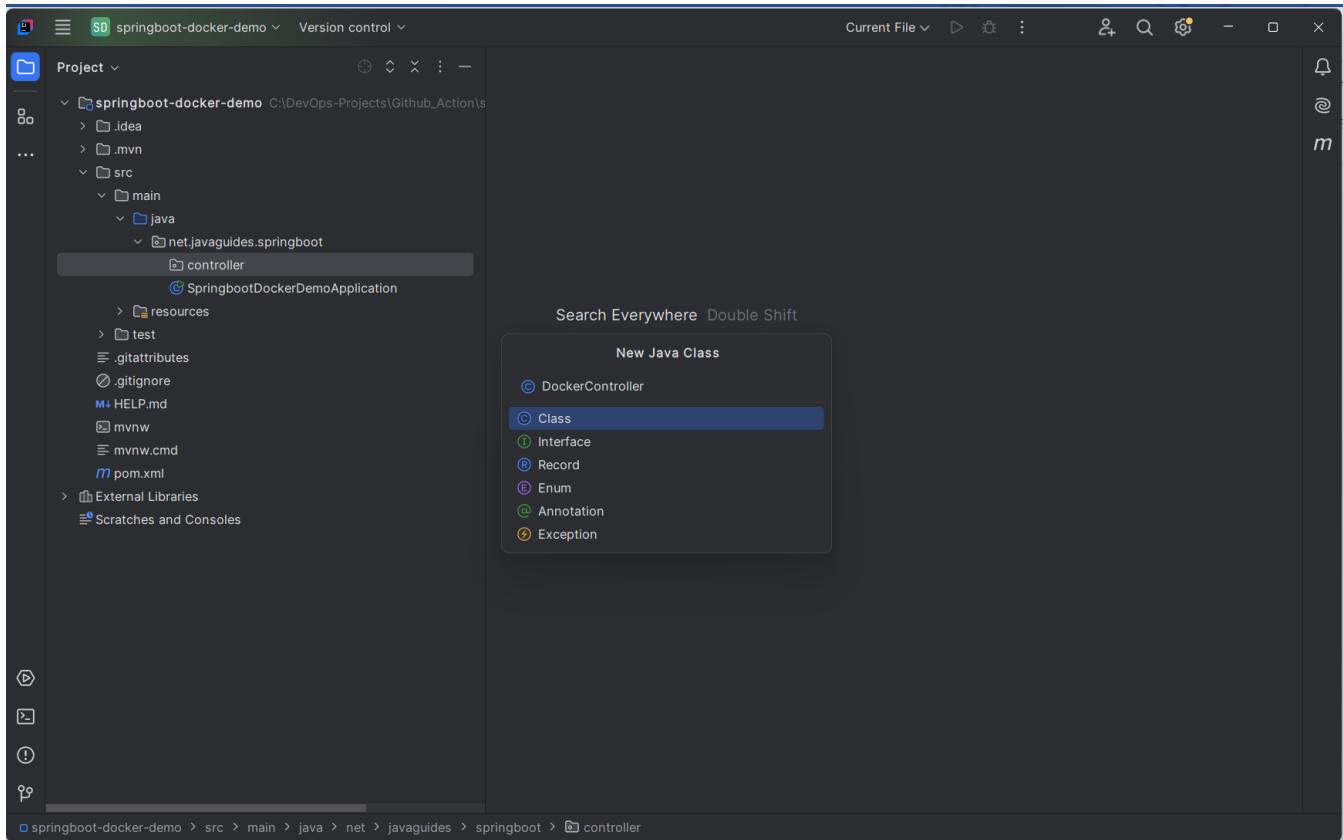
Let us create a new Java class file in the “controller” package. Right-click on “controller”



Select “Java Class”



We will create a simple endpoint called “**docker**”. So, type “**DockerController**”



And press “**Enter**”

The screenshot shows the IntelliJ IDEA interface with the project 'springboot-docker-demo' open. The left sidebar displays the project structure, including the 'src' directory containing 'main', 'java', and 'controller' packages. The 'DockerController.java' file is selected in the editor. The code area contains the following:

```
1 package net.javaguides.springboot.controller;
2
3 public class DockerController { no usages
4 }
5
```

The status bar at the bottom right shows the time as 3:31, encoding as CRLF, character set as UTF-8, and code style as 4 spaces.

Let us go ahead and annotate this controller class to make this class a REST API controller. We will add this line of code.

@RestController

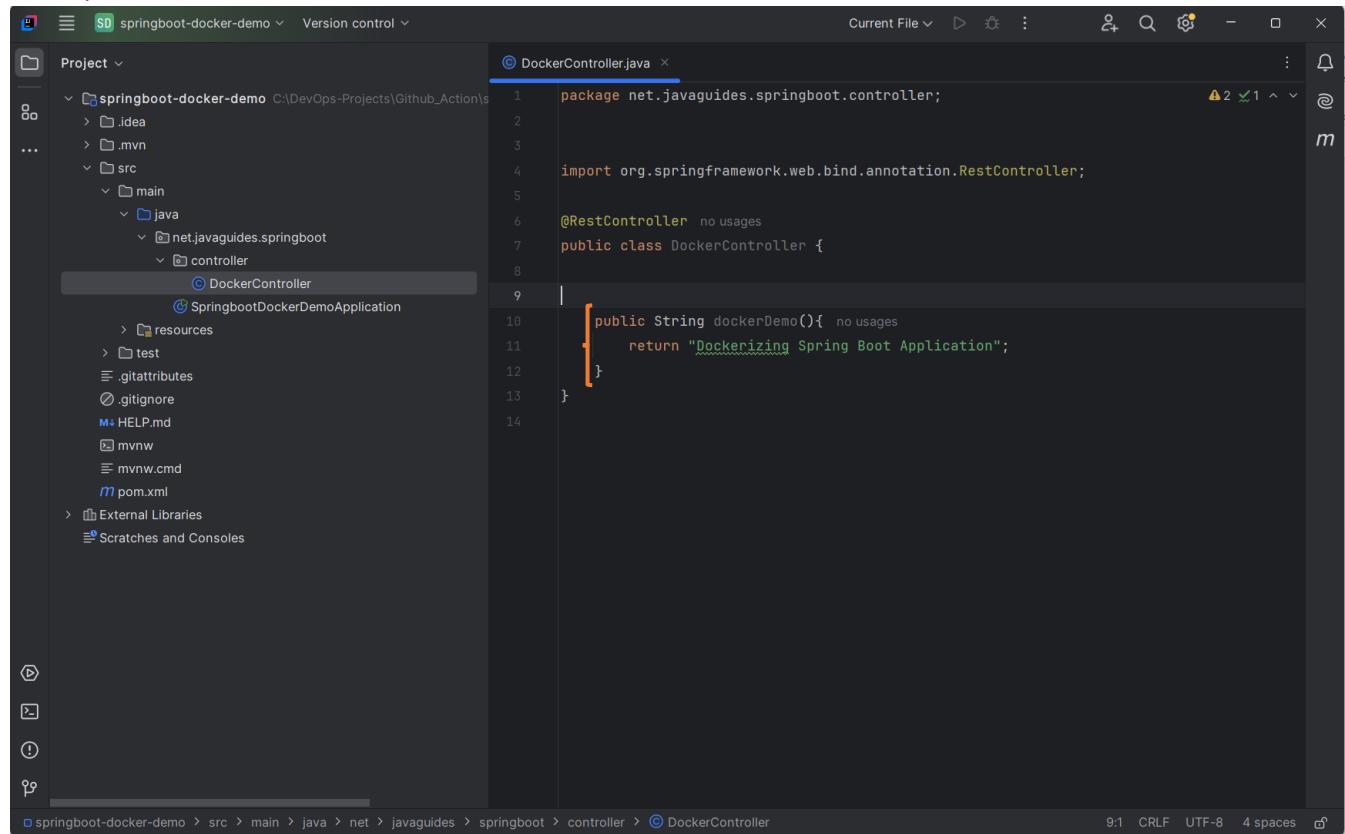
The screenshot shows the IntelliJ IDEA interface with the project 'springboot-docker-demo' open. The left sidebar displays the project structure, including the 'src' directory containing 'main', 'java', and 'controller' packages. The 'DockerController.java' file is selected in the editor. The code area now includes the '@RestController' annotation:

```
1 package net.javaguides.springboot.controller;
2
3 import org.springframework.web.bind.annotation.RestController;
4
5 @RestController no usages
6 public class DockerController {
7 }
8
```

The status bar at the bottom right shows the time as 6:16, encoding as CRLF, character set as UTF-8, and code style as 4 spaces.

Within this controller, we can define a REST API by adding a method called “**dockerDemo**” that will return a string “**Dockerizing Spring Boot Application**”

```
public String dockerDemo(){  
    return "Dockerizing Spring Boot Application";  
}
```



Let us annotate this method with by adding the line

```
@GetMapping("/docker")
```

In this line, we have added an annotation called “`GetMapping`” with API URL called “`docker`”.

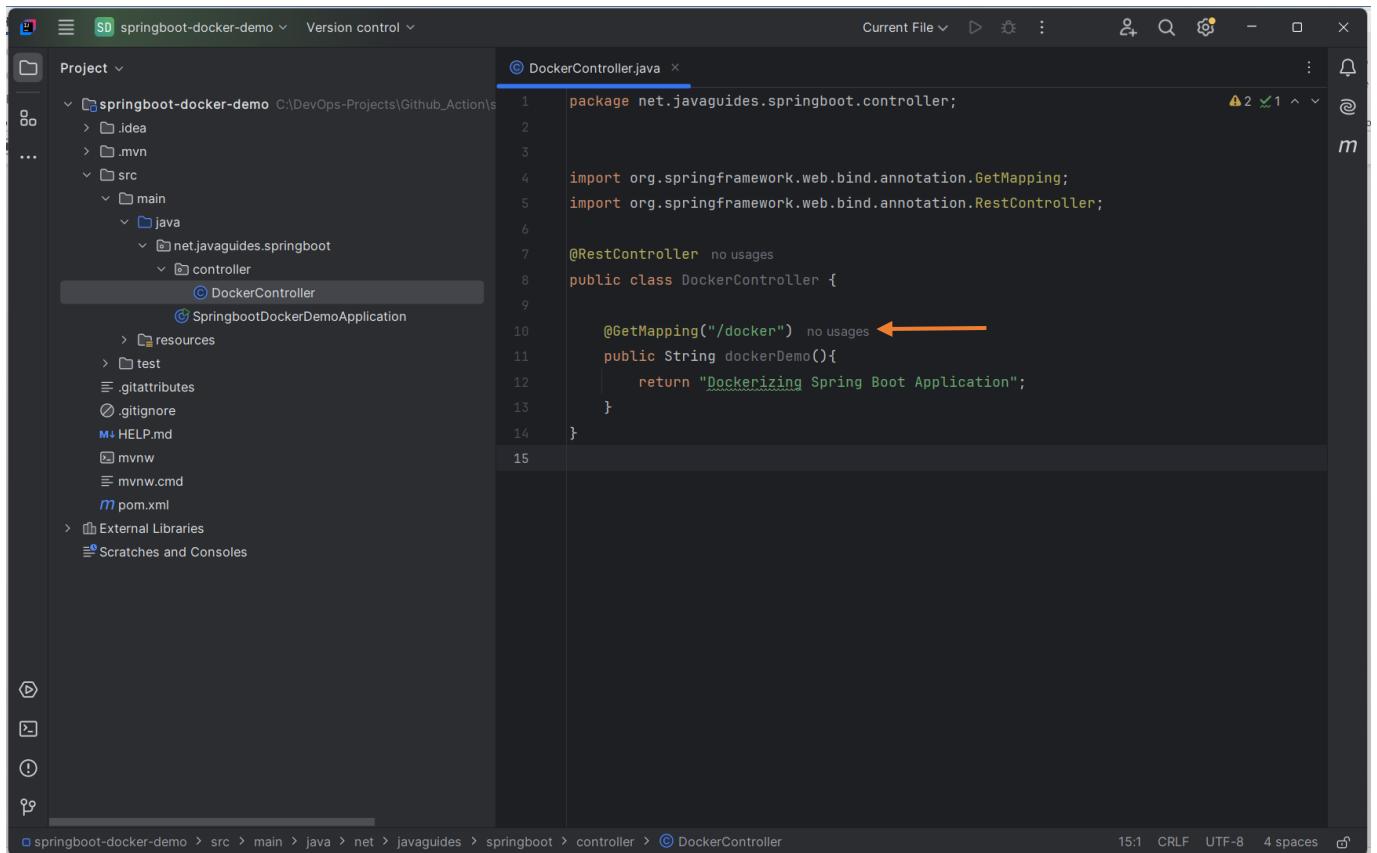
```
package net.javaquides.controller;
```

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

@RestController
public class DockerController {

    @GetMapping("/docker")
    public String dockerDemo() {
        return "Dockerizing Spring Boot Application";
    }

}
```



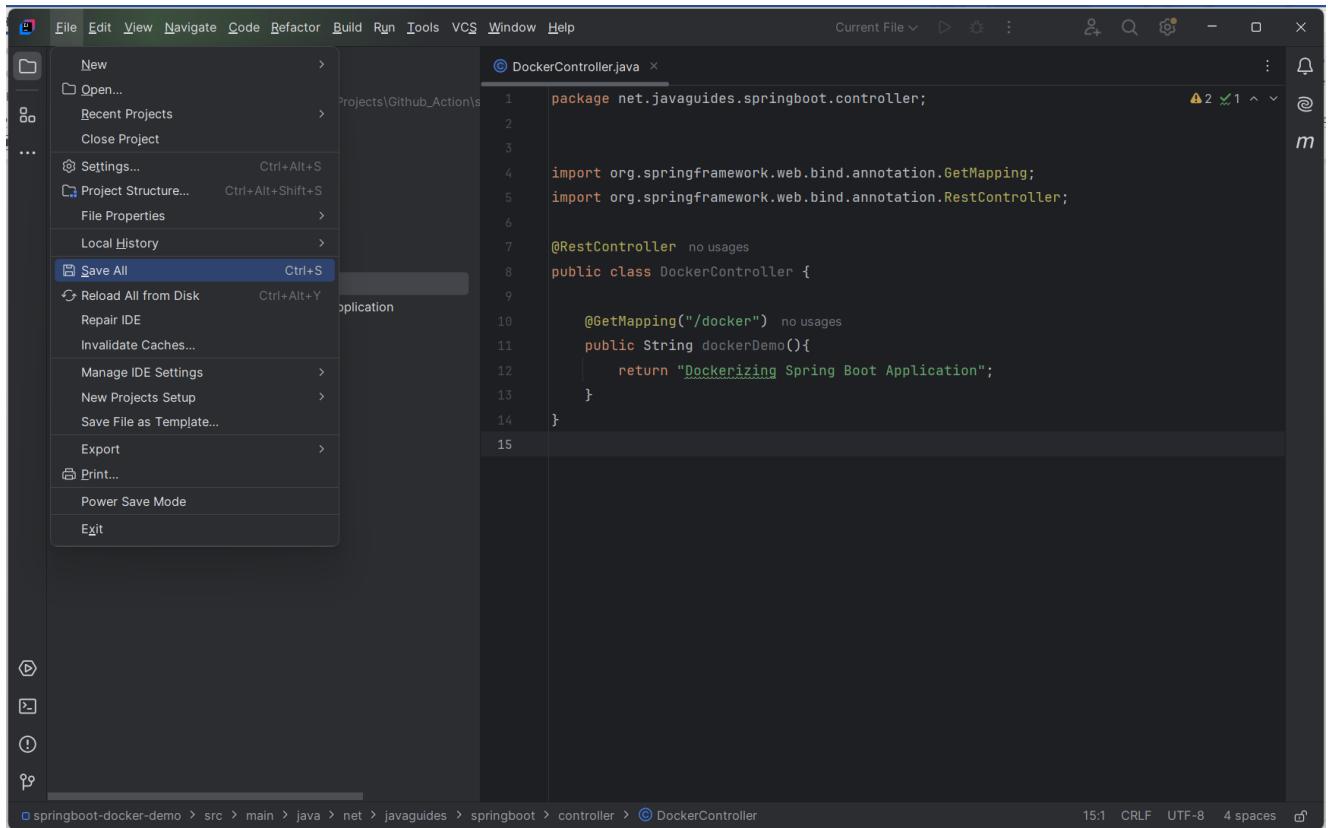
```
package net.javaguides.springboot.controller;

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

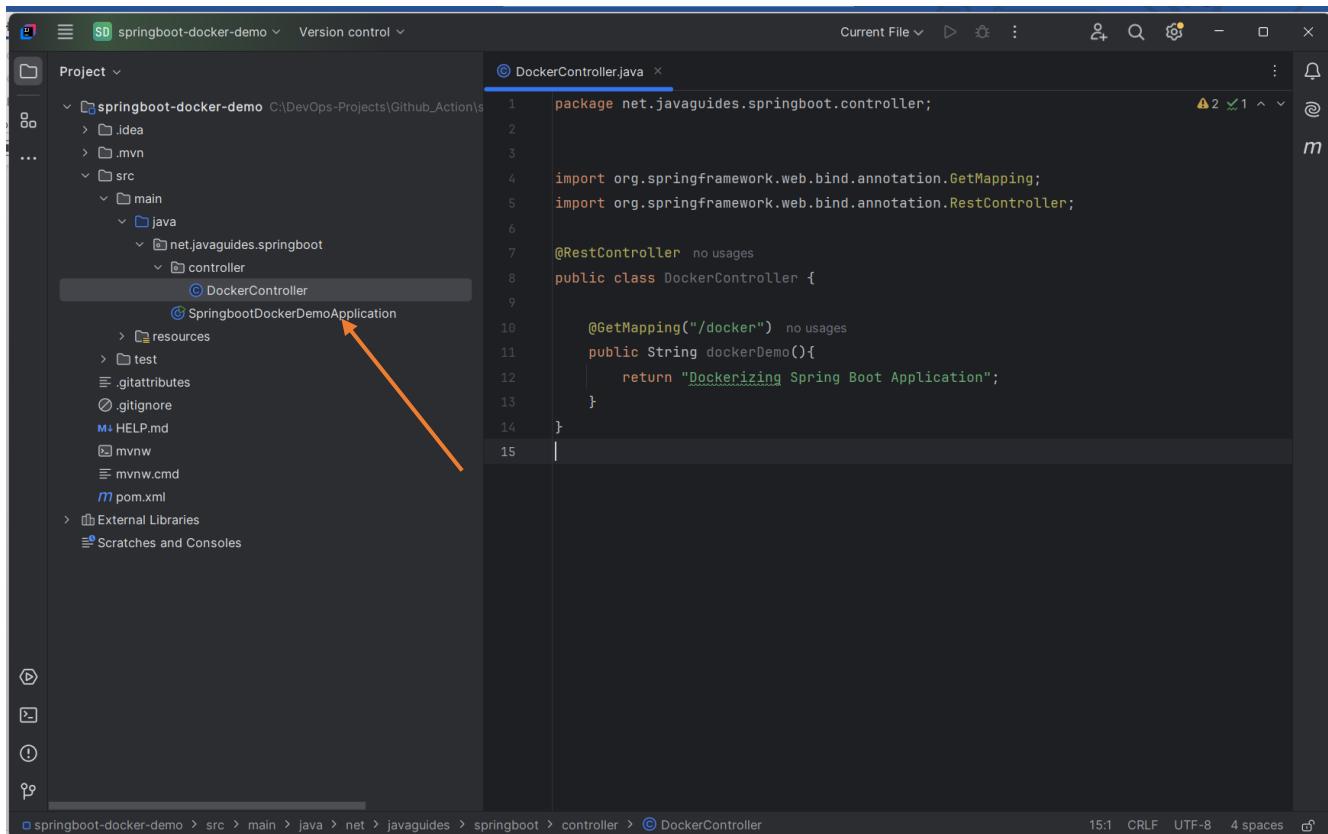
@RestController no usages
public class DockerController {

    @GetMapping("/docker") no usages ←
    public String dockerDemo(){
        return "Dockerizing Spring Boot Application";
    }
}
```

Now, we have built a simple REST API. Save the project by clicking on “File”



Then select “Save All”



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "springboot-docker-demo". The "src" folder contains "main", which has "java", "net", "javaguides", "springboot", and "controller" packages. Inside "controller", there are "DockerController" and "SpringbootDockerDemoApplication" files.
- Code Editor:** Displays the content of `DockerController.java`. The code defines a REST controller with a single endpoint mapping to "/docker" that returns the string "Dockerizing Spring Boot Application".
- Status Bar:** Shows the full file path: "springboot-docker-demo > src > main > java > net > javaguides > springboot > controller > DockerController".
- Bottom Right:** Includes status information: "15:1 CRLF UTF-8 4 spaces".

Part 4: Test the REST API on Browser

Now, let us test the application by running it.

To do this, double-click on "**SpringbootDockerDemoApplication**"

The screenshot shows the IntelliJ IDEA interface with the project 'springboot-docker-demo' open. The left sidebar displays the project structure, including '.idea', '.mvn', 'src' (containing 'main' and 'java' folders), 'resources', 'test', '.gitattributes', '.gitignore', 'HELP.md', 'mvnw', 'mvnw.cmd', 'pom.xml', 'External Libraries', and 'Scratches and Consoles'. The right panel shows the code editor with 'SpringbootDockerDemoApplication.java' selected. The code is as follows:

```
1 package net.javaguides.springboot;
2
3 > import ...
4
5 @SpringBootApplication
6 public class SpringbootDockerDemoApplication {
7 >     public static void main(String[] args) { SpringApplication.run(SpringbootDockerDemoA
8 }
9 > }
```

An orange arrow points from the text 'Click on "Run"' to the 'Run' icon in the gutter of the code editor.

Click on “Run”

The screenshot shows the IntelliJ IDEA interface with the same project structure. The code editor now highlights the 'main()' method in the 'SpringbootDockerDemoApplication' class. A context menu is open over this method, listing options: 'Run 'SpringbootDocker...main()'' (highlighted with an orange arrow), 'Debug 'SpringbootDocker...main()'' (with a green icon), 'Run 'SpringbootDocker...main()' with Coverage' (with a blue icon), and 'Modify Run Configuration...' (with a grey icon). The status bar at the bottom indicates the current time as 14:11 and file encoding as LF.

Select “Run ‘SpringbootDocker...main()’”

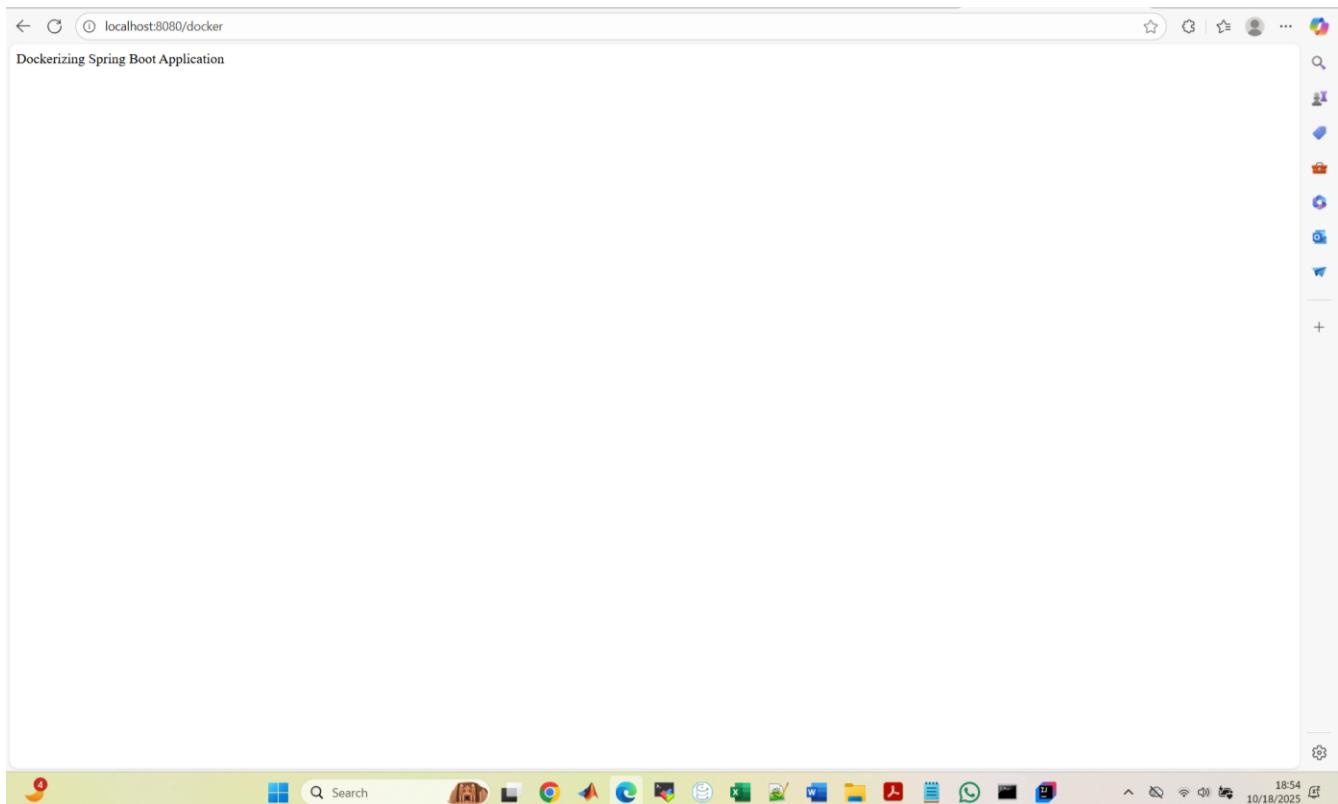
The screenshot shows the VS Code interface with the following details:

- Project Explorer:** Shows the project structure under "springboot-docker-demo". The "target" folder is selected.
- Editor:** Displays the code for `SpringbootDockerDemoApplication.java`. The code is as follows:

```
1 package net.javaguides.springboot;
2
3 import ...
4
5 @SpringBootApplication
6 public class SpringbootDockerDemoApplication {
7
8     public static void main(String[] args) { SpringApplication.run(SpringbootDockerDemoA
9 }
10
11 }
12
13 }
14
```

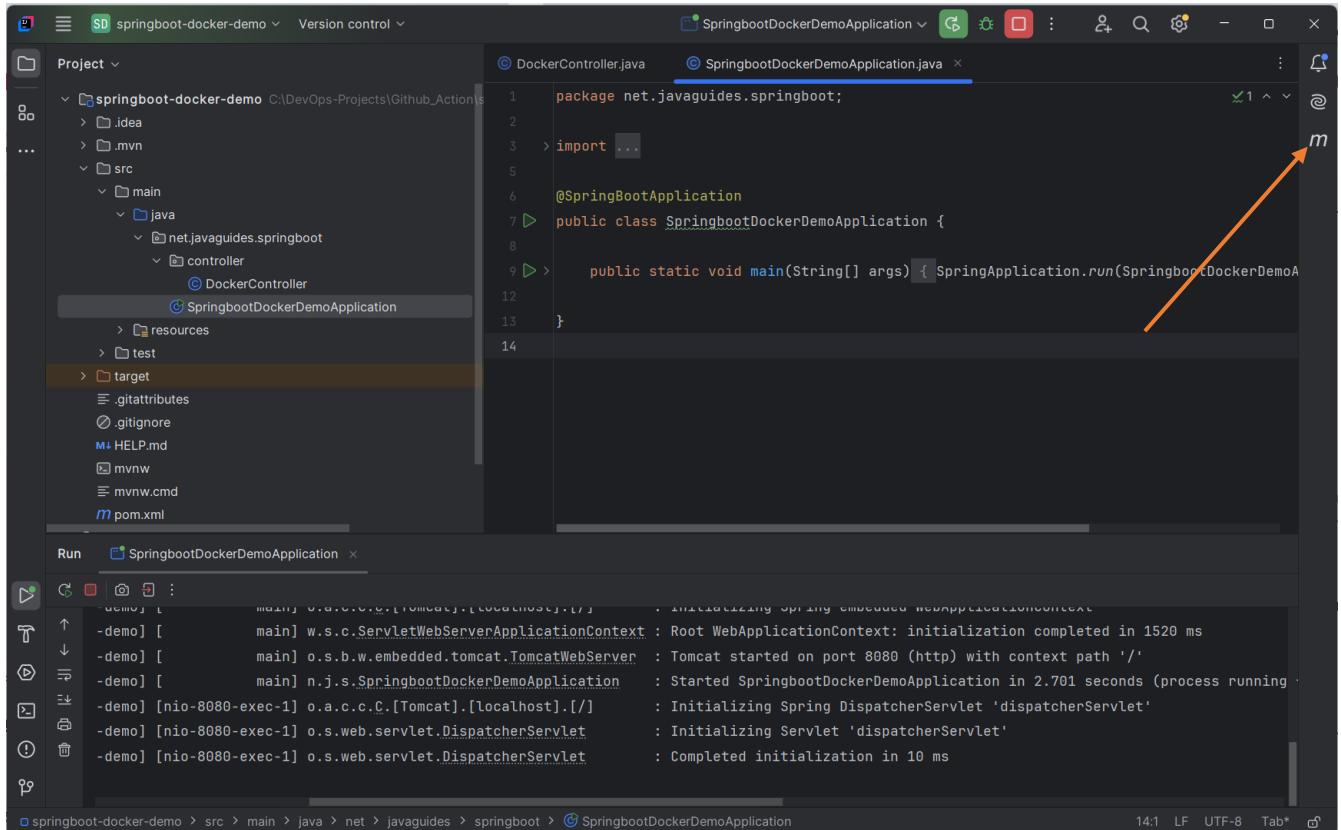
- Terminal:** Shows the application's startup logs. An orange arrow points to the line: `: Tomcat started on port 8080 (http) with context path '/'`.
- Bottom Status Bar:** Shows the path `springboot-docker-demo > src > main > java > net > javaguides > springboot > SpringbootDockerDemoApplication`, the time `14:11`, and the encoding `UTF-8`.

Now, verify on your browser by typing: **localhost:8080/docker**



The application is working. We can now proceed to Dockerize the application.

STEP 3: Build and Package Maven Application



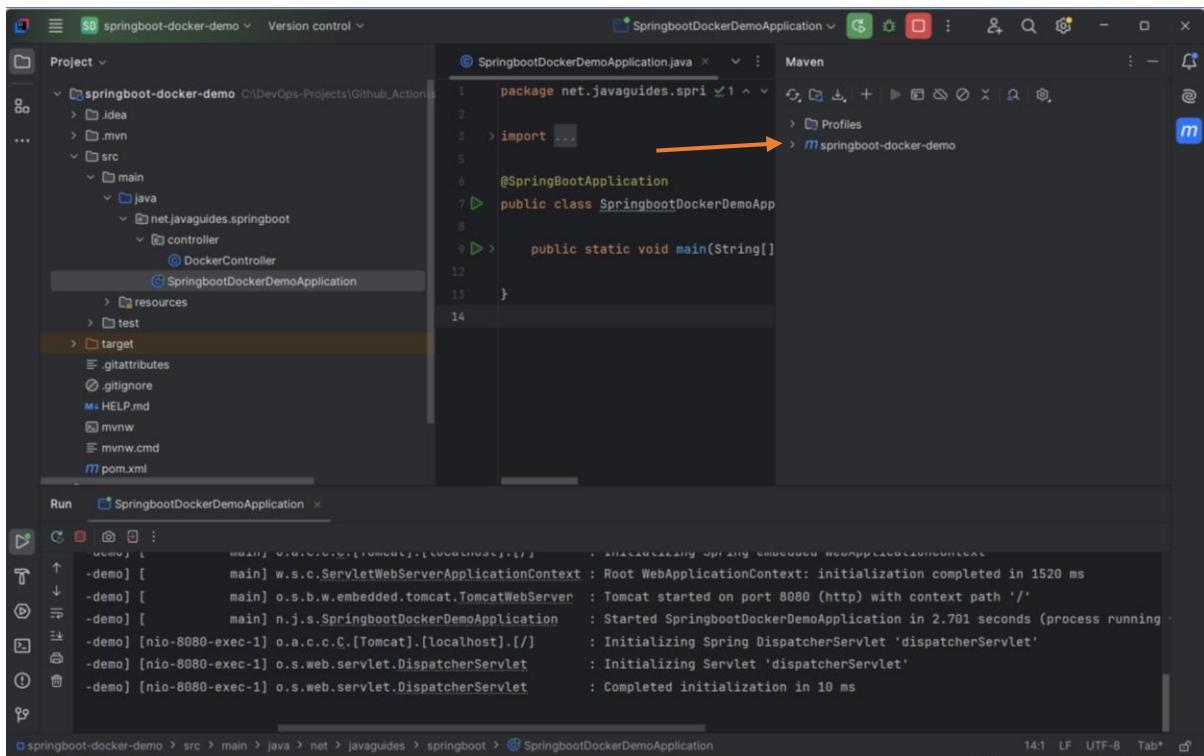
The screenshot shows the IntelliJ IDEA interface. The top bar displays the project name "springboot-docker-demo" and the file "SpringbootDockerDemoApplication.java". The left sidebar shows the project structure with a "target" folder selected. The right pane shows the code for "SpringbootDockerDemoApplication.java". An orange arrow points to the "Maven" icon in the top right corner of the code editor. The bottom pane shows the run output, which includes logs for starting Tomcat and initializing the Spring application.

```

package net.javaguides.springboot;
import ...;
@SpringBootApplication
public class SpringbootDockerDemoApplication {
    public static void main(String[] args) { SpringApplication.run(SpringbootDockerDemoA... }
}

```

Click on “Maven” icon



The screenshot shows the IntelliJ IDEA interface with the Maven tool window open. The top bar displays the project name "springboot-docker-demo" and the file "SpringbootDockerDemoApplication.java". The left sidebar shows the project structure with a "target" folder selected. The right pane shows the code for "SpringbootDockerDemoApplication.java". An orange arrow points to the "Maven" icon in the top right corner of the code editor. The bottom pane shows the run output, which includes logs for starting Tomcat and initializing the Spring application.

```

package net.javaguides.spri...
import ...;
@SpringBootApplication
public class SpringbootDockerDemoApp...
    public static void main(String[])
}

```

Click on “springboot-docker-demo”

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot-docker-demo".
- Code Editor:** Displays the `SpringbootDockerDemoApplication.java` file with the following code:

```

1 package net.javaguides.springboot;
2
3 import ...
4
5 @SpringBootApplication
6 public class SpringbootDockerDemoApp {
7     public static void main(String[] args) {
8     }
9 }

```

- Maven Tool Window:** Shows the Maven project tree with the "Lifecycle" node expanded, displaying goals like clean, validate, compile, test, package, verify, install, site, and deploy.
- Run Tool Window:** Shows the application output log with the following logs:

```

[INFO] demo [main] o.s.w.c.ServletWebServerApplicationContext : Initializing Spring embedded WebApplicationContext
[INFO] -demo [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1520 ms
[INFO] -demo [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
[INFO] -demo [main] n.j.s.SpringbootDockerDemoApplication : Started SpringbootDockerDemoApplication in 2.701 seconds (process running)
[INFO] -demo [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
[INFO] -demo [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
[INFO] -demo [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 10 ms

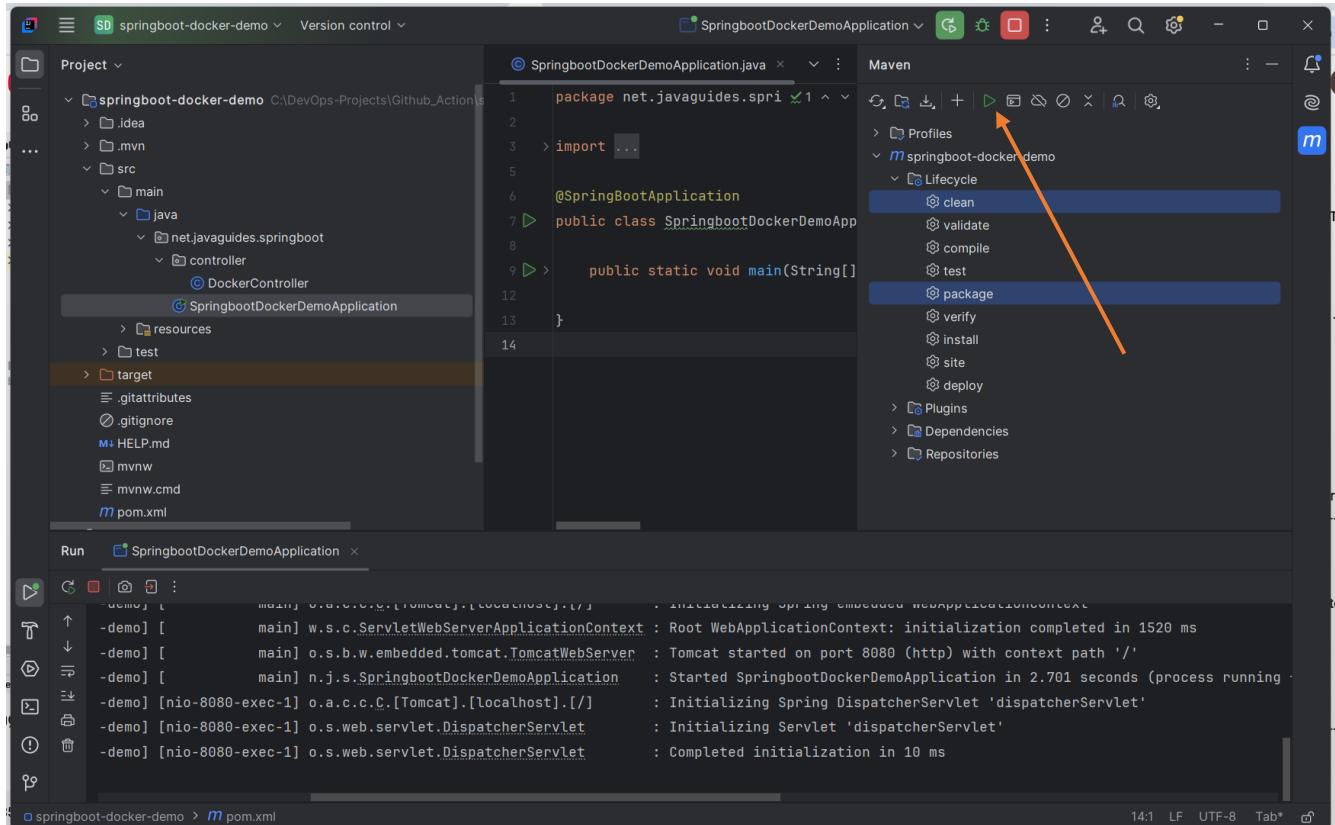
```

Select “Lifecycle”

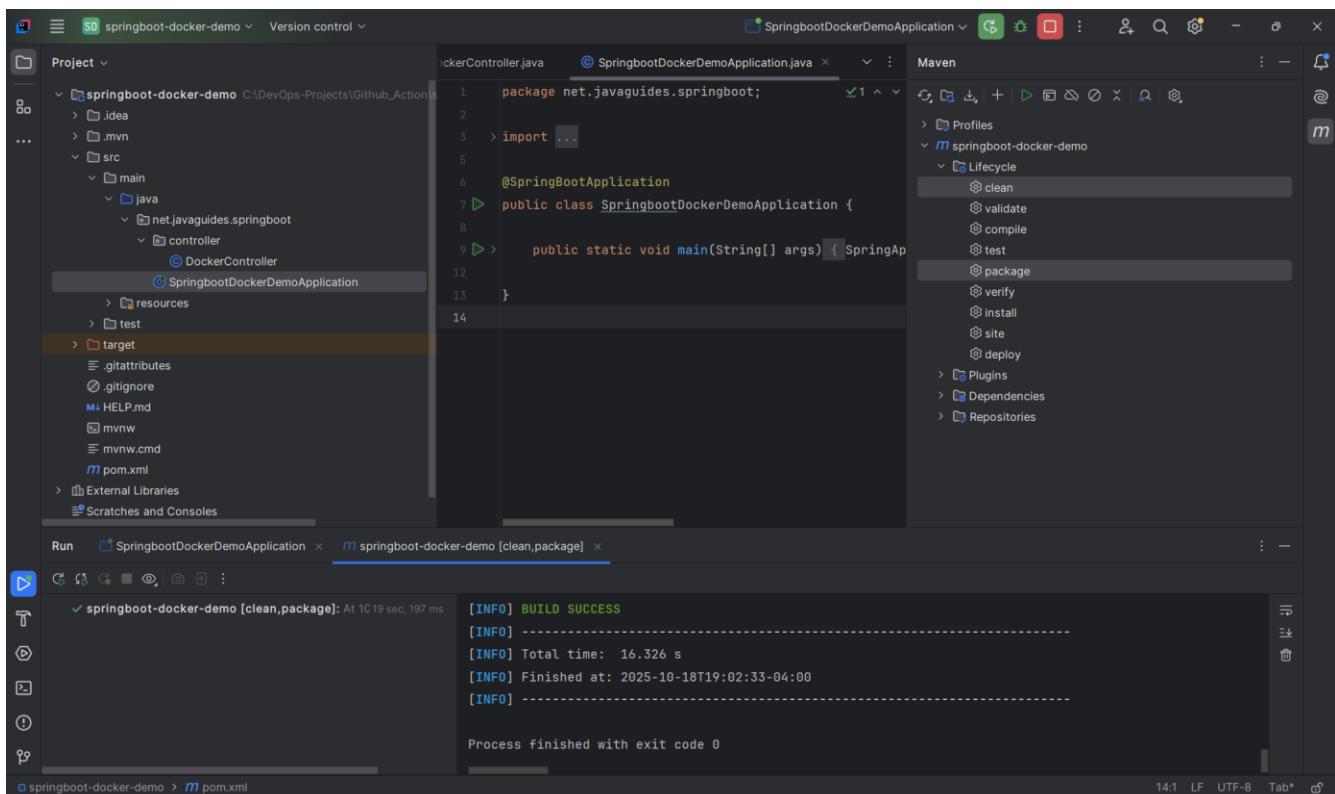
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot-docker-demo".
- Code Editor:** Displays the `SpringbootDockerDemoApplication.java` file with the same code as before.
- Maven Tool Window:** Shows the Maven project tree with the "Lifecycle" node selected, displaying its sub-goals: clean, validate, compile, test, package, verify, install, site, and deploy.
- Run Tool Window:** Shows the application output log with the same logs as the previous screenshot.

Select “clean” and “package”

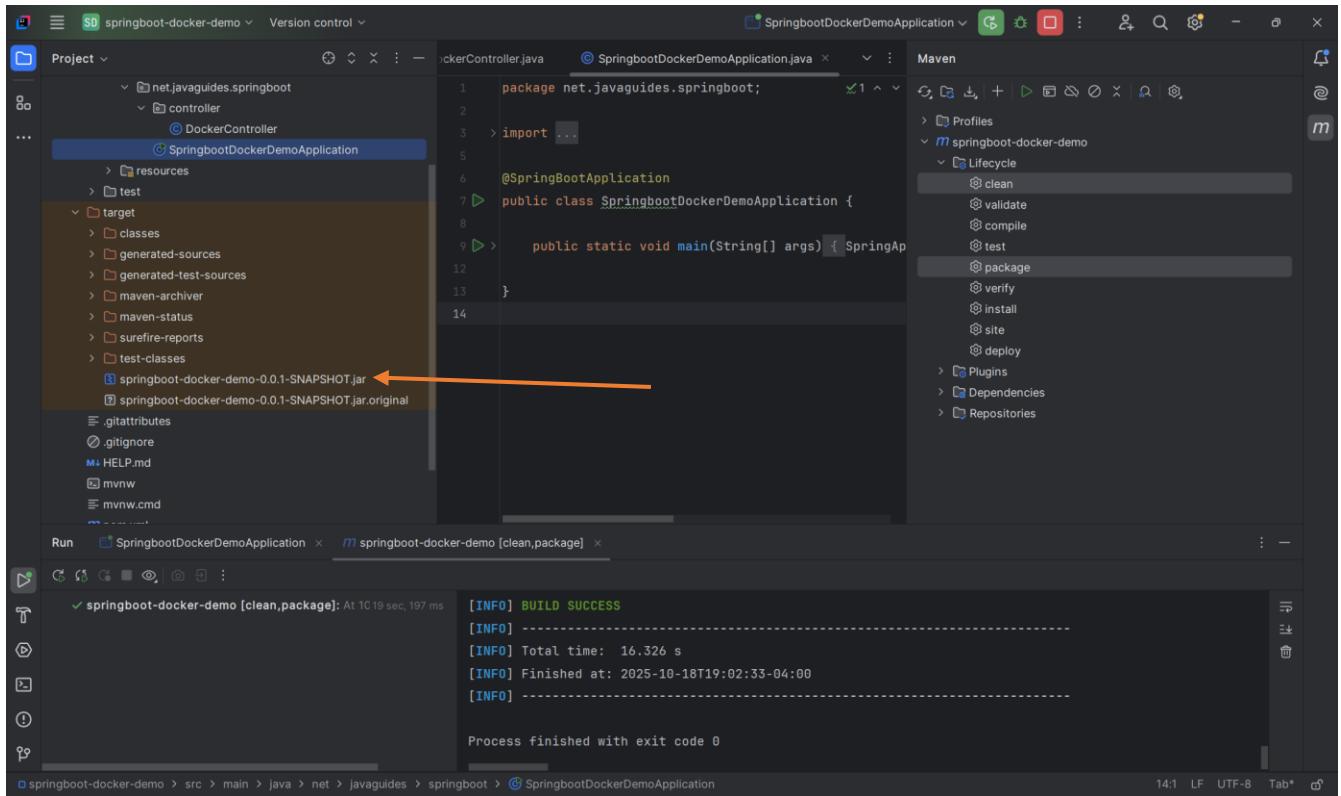


Click on “Run Maven Build”



The build is successful

To verify the Maven build is successful. Click on the “target” folder

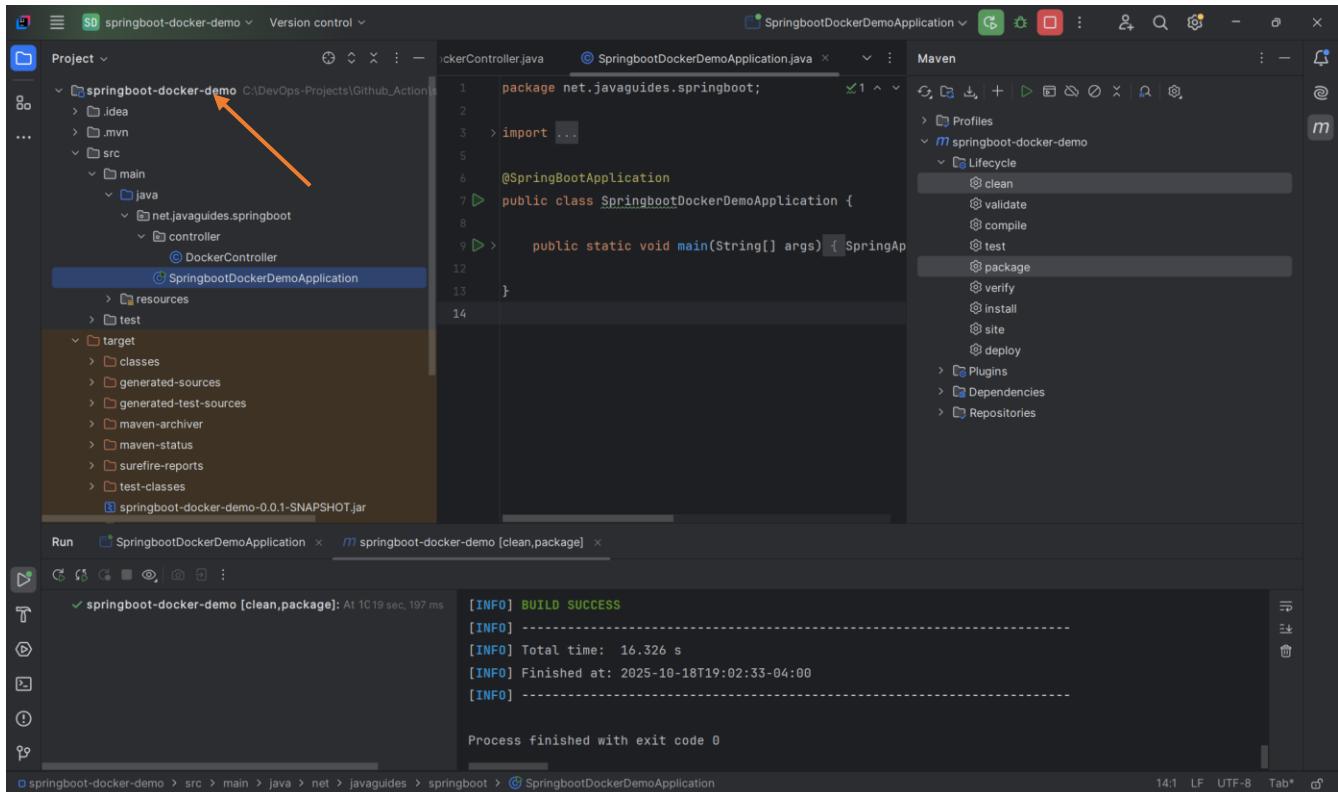


You can see that “**Springboot-docker-demo-0.0.1-SNAPSHOT.jar**” has been created.

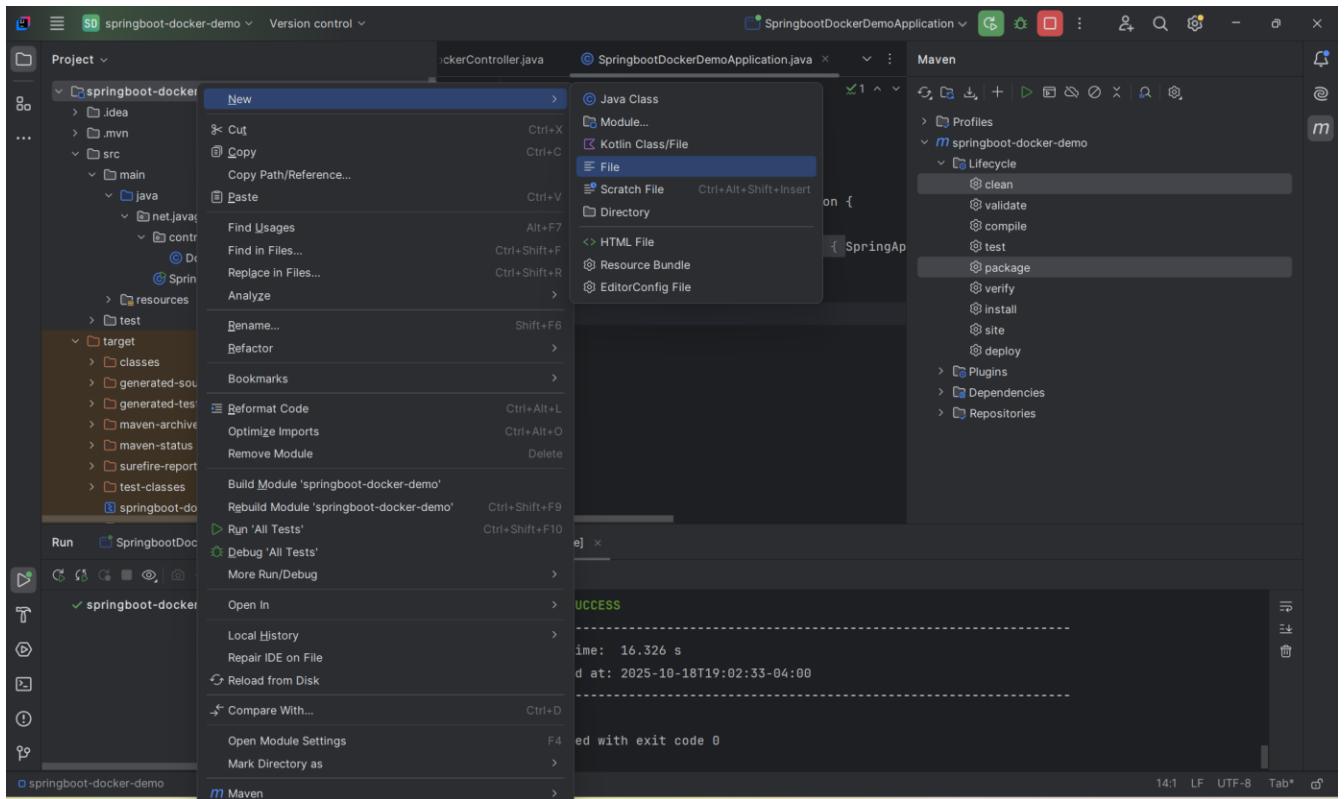
STEP 4: Create a Dockerfile

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. This page describes the commands you can use in a Dockerfile.

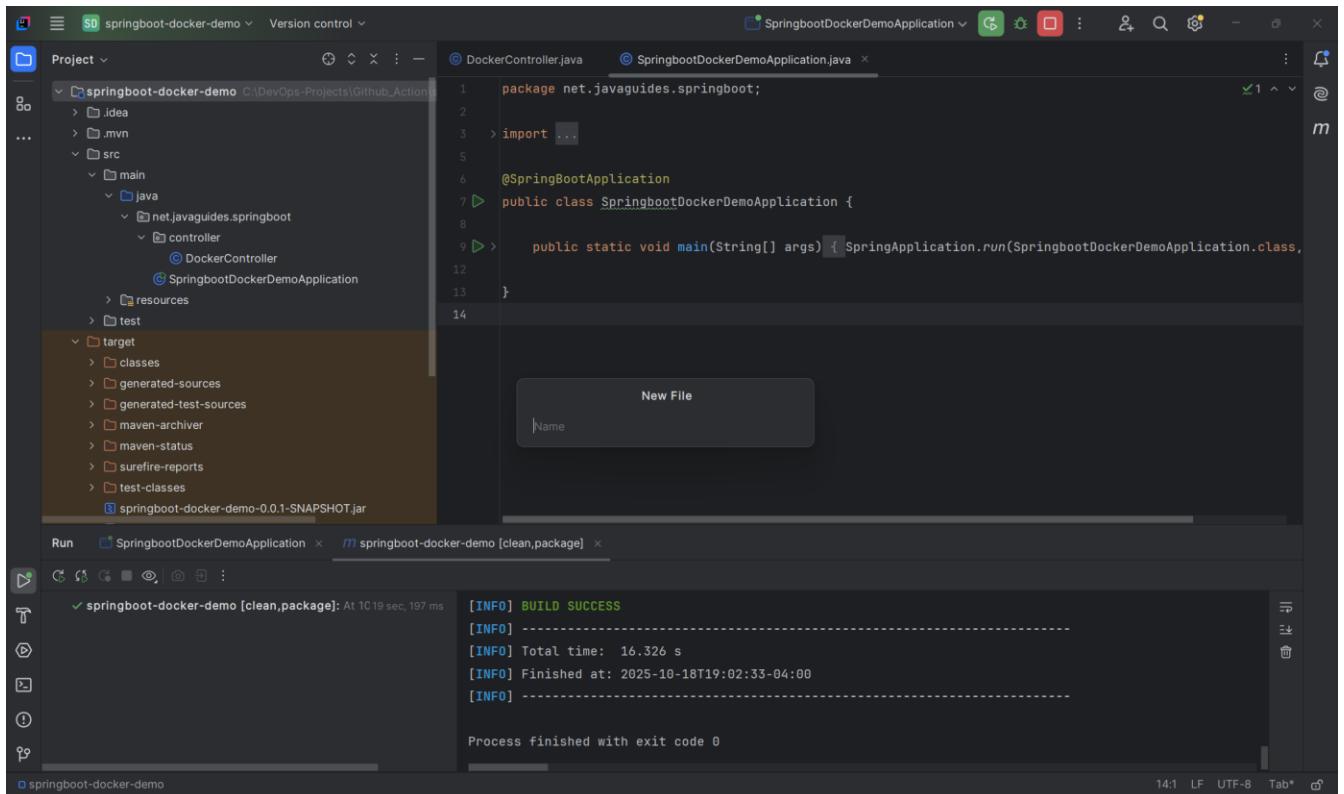
We will create a Dockerfile where we will define all the instructions and commands to build the Docker Image.



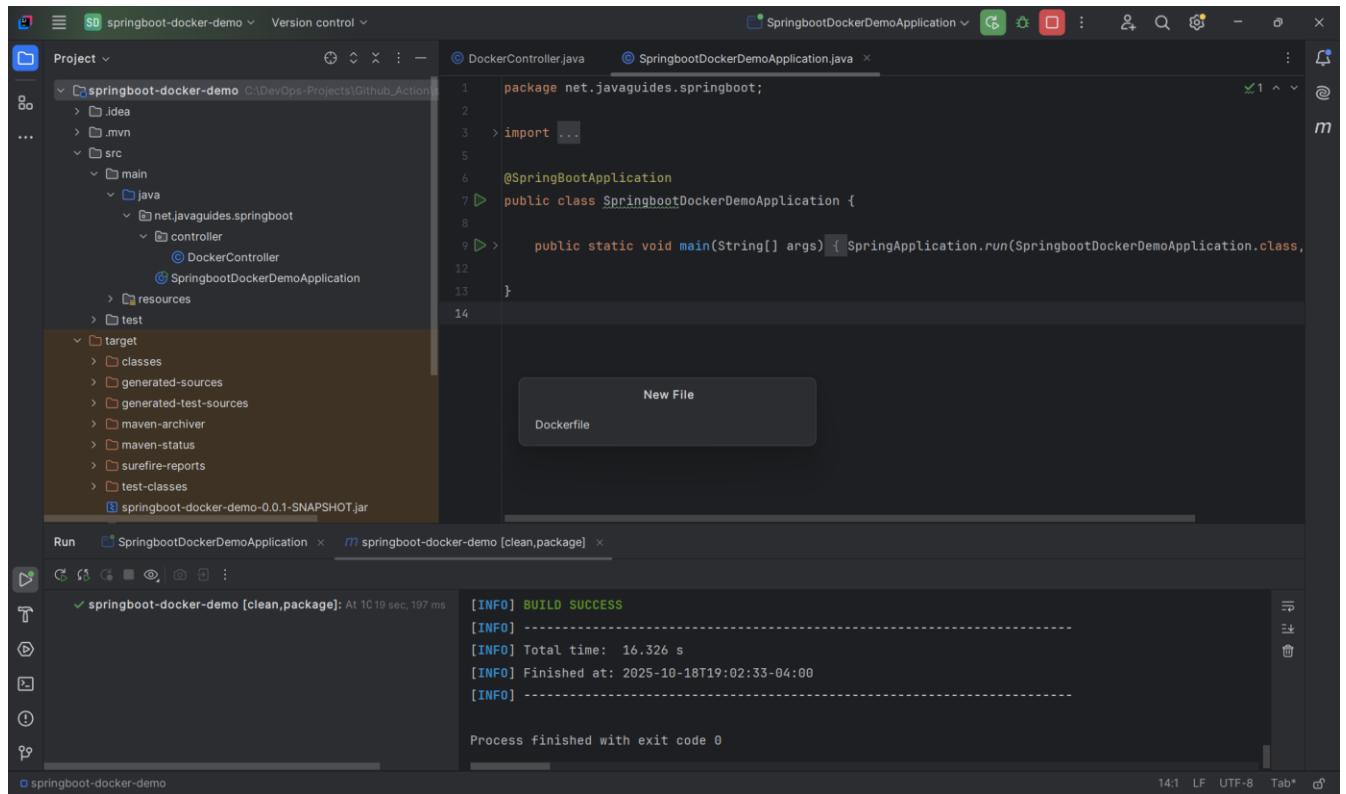
Right-click on the project name “**springboot-docker-demo**”



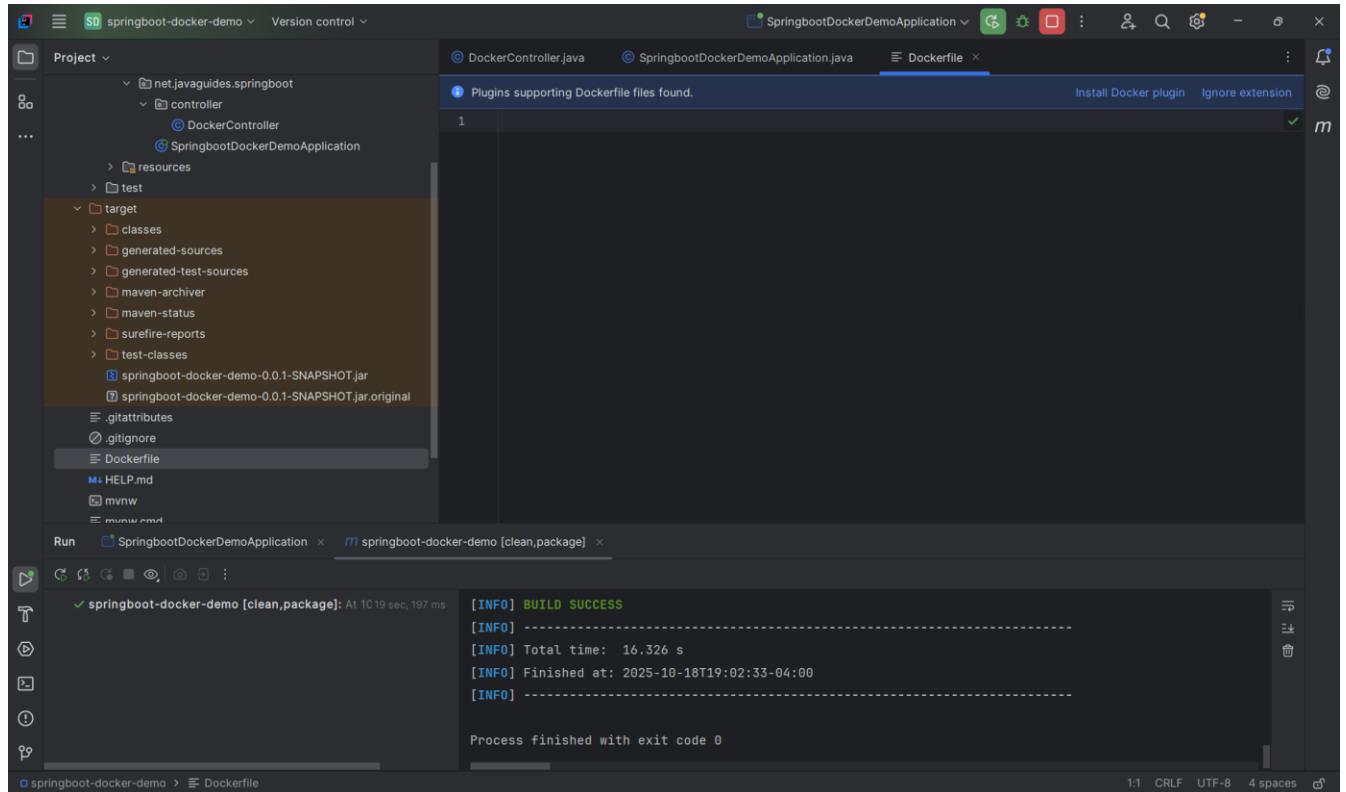
Select “File”



Enter the name of the file as “Dockerfile”



Press “Enter”



Let us now write the commands on our Dockerfile

```
# Use an official Eclipse Temurin base image
```

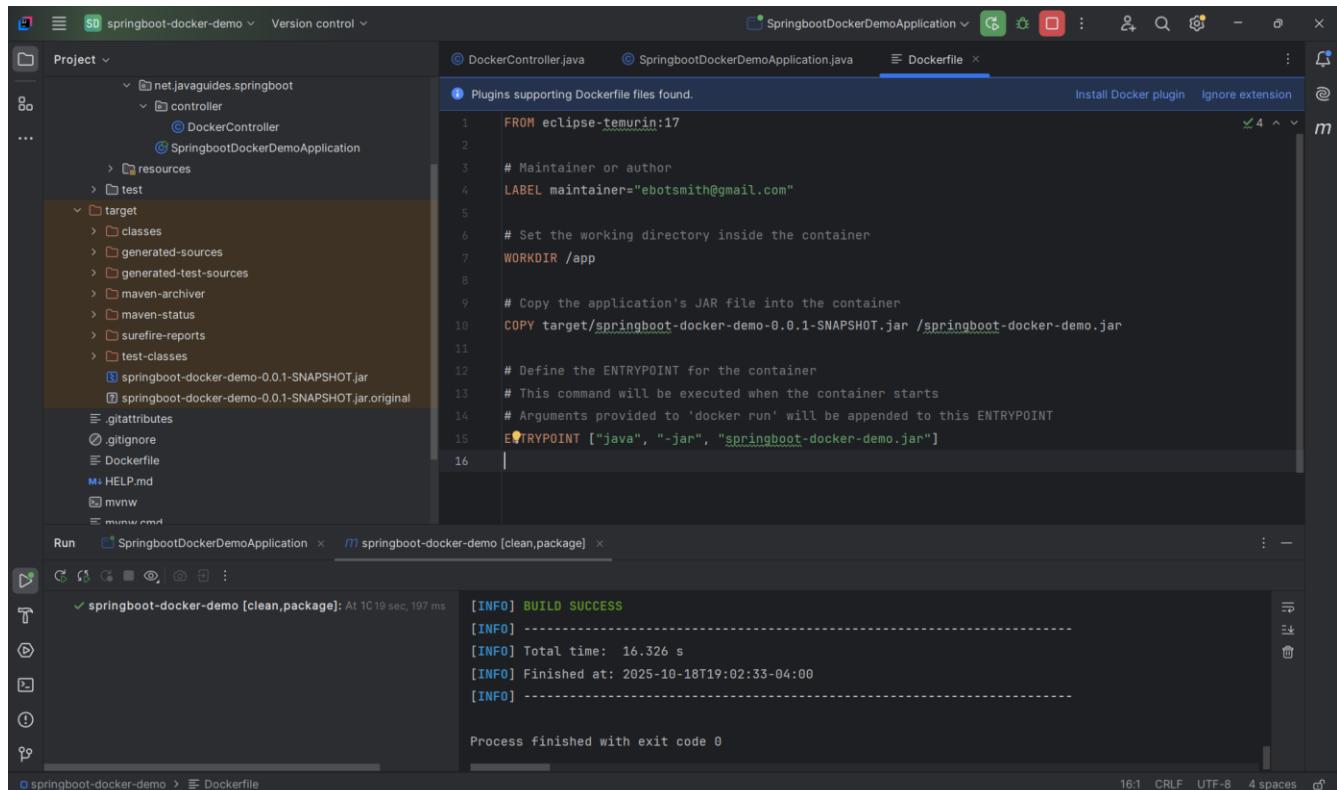
```
FROM eclipse-temurin:17

# Maintainer or author
LABEL maintainer="ebotsmith@gmail.com"

# Set the working directory inside the container
WORKDIR /app

# Copy the application's JAR file into the container
COPY /target/springboot-docker-demo-0.0.1-SNAPSHOT.jar /app/springboot-docker-
demo.jar

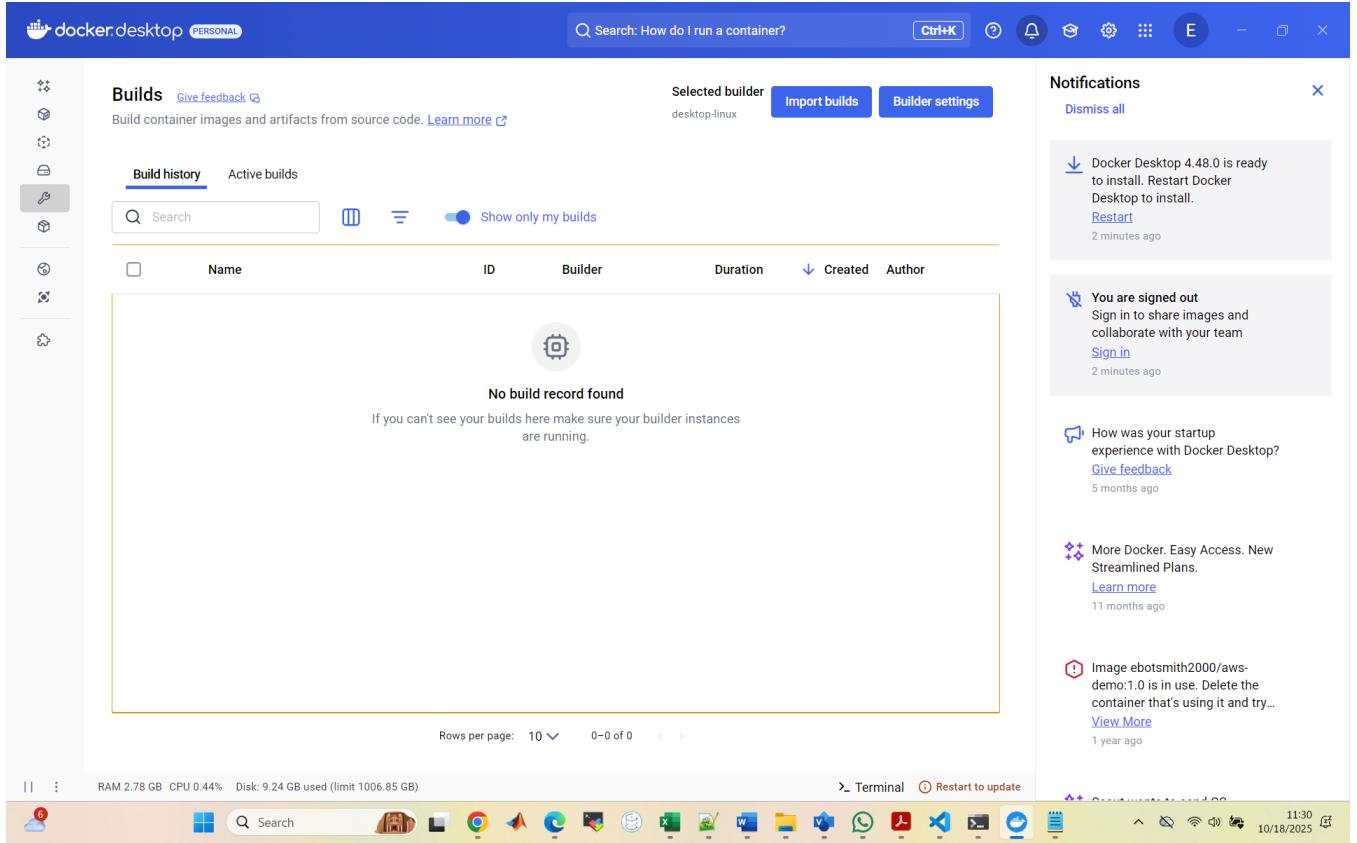
# Define the ENTRYPOINT for the container. This command will be executed when the
container starts
# Arguments provided to 'docker run' will be appended to this ENTRYPOINT
ENTRYPOINT ["java", "-jar", "/app/springboot-docker-demo.jar"]
```



Now, let us save the file.

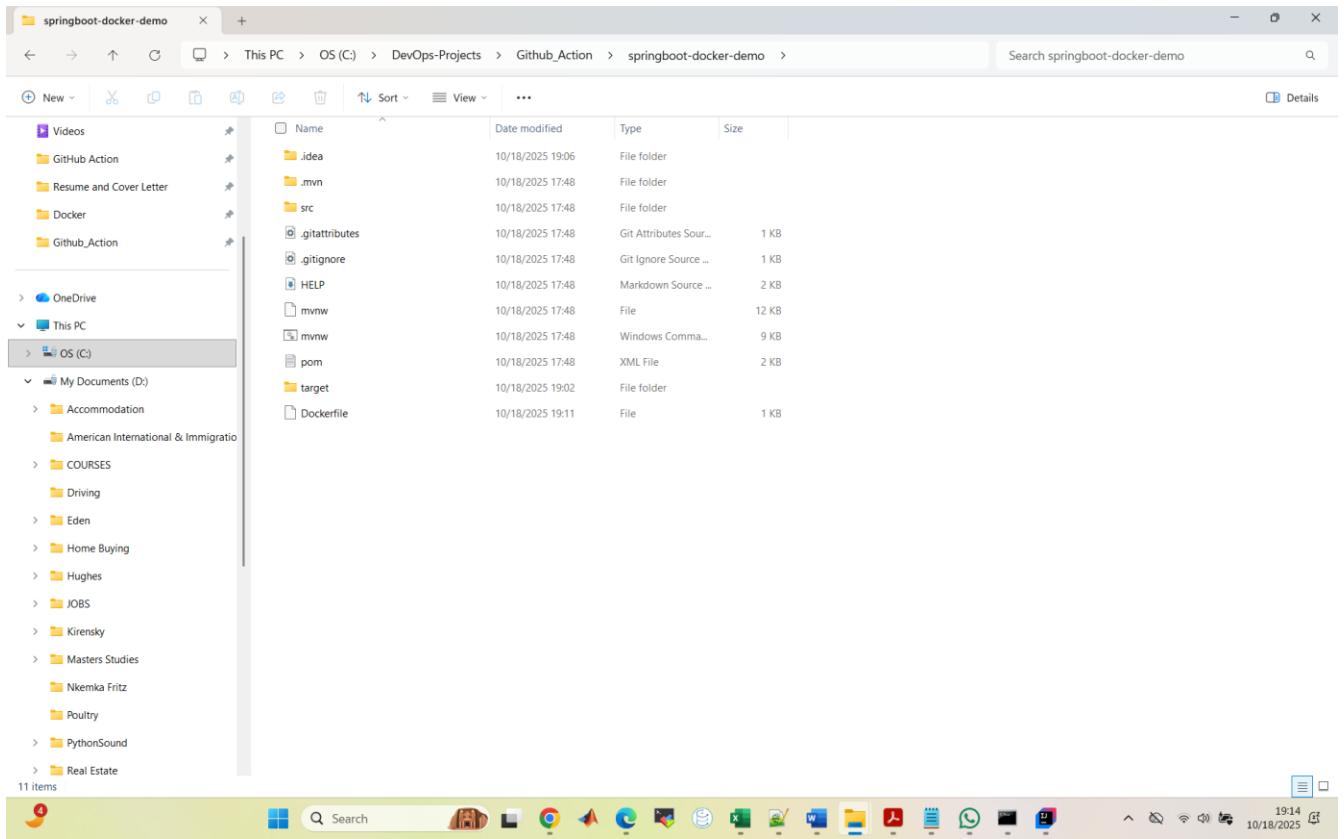
STEP 5: Build Docker Image from Dockerfile

We will now build a Docker Image from the docker file. Before doing this, make sure your Docker desktop is up and running on your local machine.

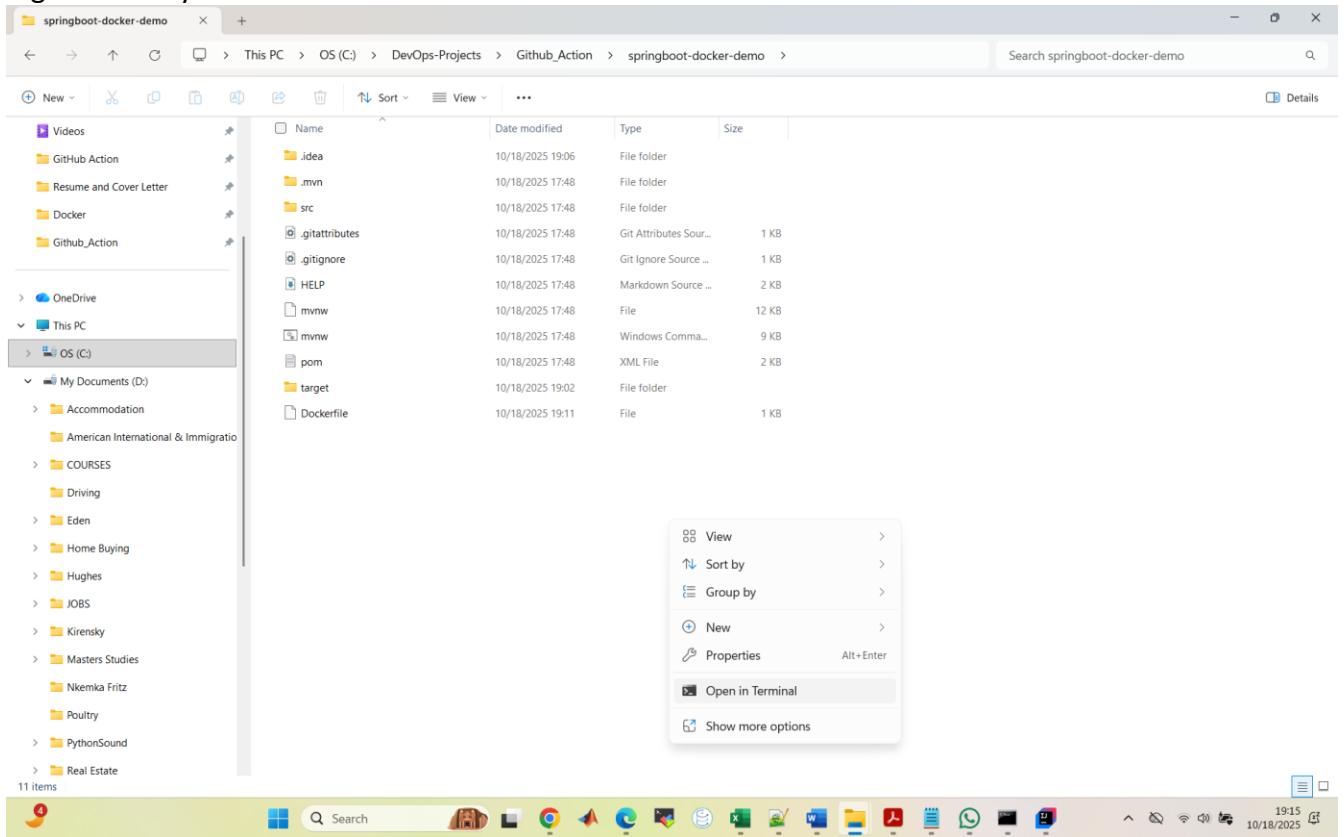


Head back to the project folder in your local machine

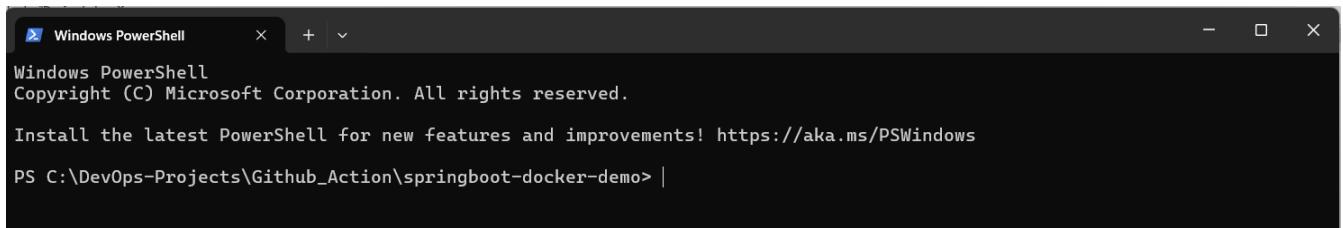
Prepared by Sidney Smith Ebot



Right-click anywhere in this folder



Select “Open in Terminal”



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

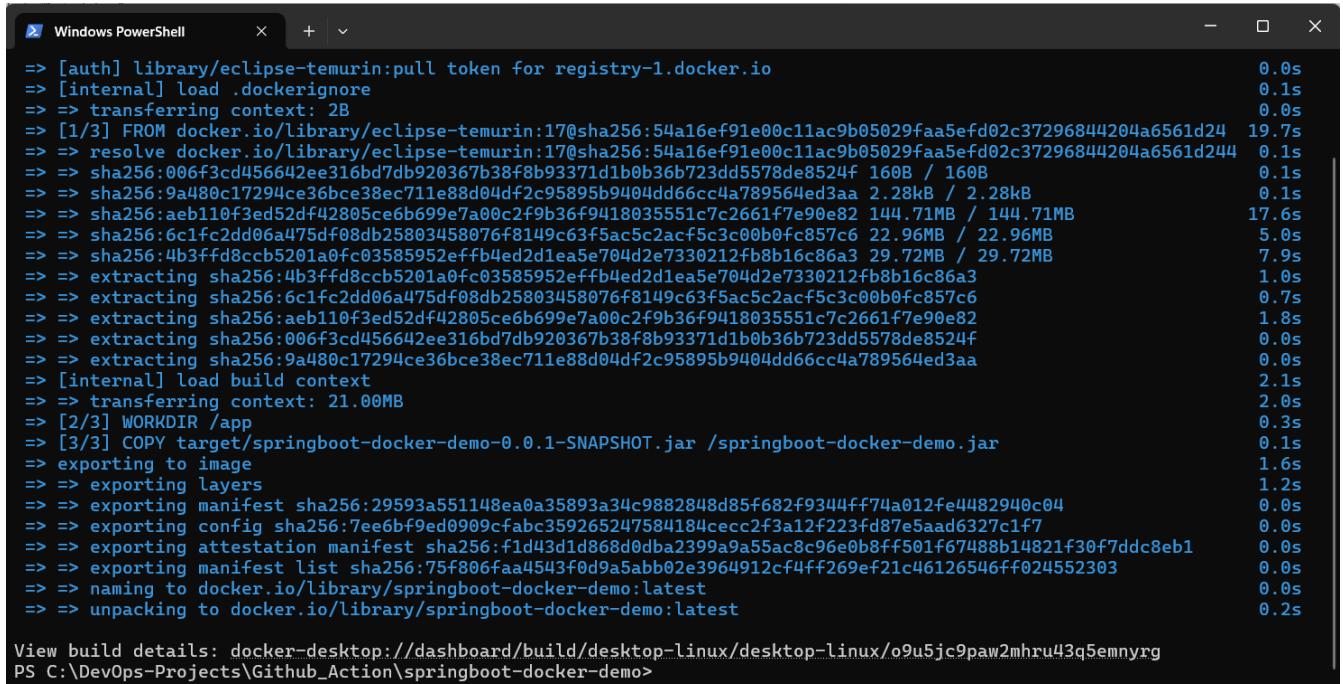
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |

```

Run the command to build the Docker image with the name “**springboot-docker-demo**”

```
docker build -t springboot-docker-demo .
```



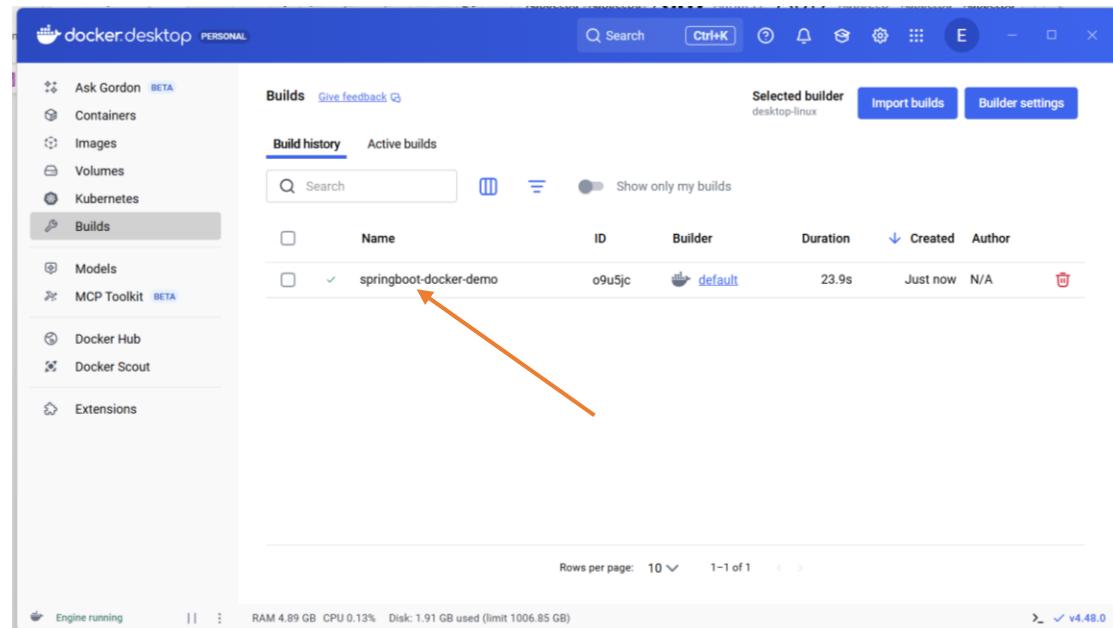
```

=> [auth] library/eclipse-temurin:pull token for registry-1.docker.io          0.0s
=> [internal] load .dockerrcignore                                         0.1s
=> => transferring context: 2B                                              0.0s
=> [1/3] FROM docker.io/library/eclipse-temurin:17@sha256:54a16ef91e00c11ac9b05029faa5efd02c37296844204a6561d24 19.7s
=> => resolve docker.io/library/eclipse-temurin:17@sha256:54a16ef91e00c11ac9b05029faa5efd02c37296844204a6561d244 0.1s
=> => sha256:006f3cd456642ee316bd7db920367b38f8b93371d1b0b36b723dd5578de8524f 160B / 160B   0.1s
=> => sha256:9a480c17294ce36bce38ec711e88d04df2c95895b9404dd66cc4a789564ed3aa 2.28kB / 2.28kB   0.1s
=> => sha256:aeb110f3ed52df42805ce6b699e7a00c2f9b36f9418035551c7c2661f7e90e82 144.71MB / 144.71MB  17.6s
=> => sha256:6c1fc2dd06a475df08db25803458076f8149c63f5ac5c2acf5c3c00b0fc857c6 22.96MB / 22.96MB  5.0s
=> => sha256:4b3ffd8ccb5201a0fc03585952efffb4ed2d1ea5e704d2e7330212fb8b16c86a3 29.72MB / 29.72MB  7.9s
=> => extracting sha256:4b3ffd8ccb5201a0fc03585952efffb4ed2d1ea5e704d2e7330212fb8b16c86a3 1.0s
=> => extracting sha256:6c1fc2dd06a475df08db25803458076f8149c63f5ac5c2acf5c3c00b0fc857c6 0.7s
=> => extracting sha256:aeb110f3ed52df42805ce6b699e7a00c2f9b36f9418035551c7c2661f7e90e82 1.8s
=> => extracting sha256:006f3cd456642ee316bd7db920367b38f8b93371d1b0b36b723dd5578de8524f 0.0s
=> => extracting sha256:9a480c17294ce36bce38ec711e88d04df2c95895b9404dd66cc4a789564ed3aa 0.0s
=> [internal] load build context                                         2.1s
=> => transferring context: 21.00MB                                     2.0s
=> => WORKDIR /app                                                 0.3s
=> [2/3] COPY target/springboot-docker-demo-0.0.1-SNAPSHOT.jar /springboot-docker-demo.jar 0.1s
=> exporting to image                                               1.6s
=> => exporting layers                                             1.2s
=> => exporting manifest sha256:29593a551148ea0a35893a34c9882848d85f682f9344ff74a012fe4482940c04 0.0s
=> => exporting config sha256:7ee6bf9ed0909cfabc359265247584184cecc2f3a12f223fd87e5aad6327c1f7 0.0s
=> => exporting attestation manifest sha256:f1d43d1d868d0dba2399a9a55ac8c96e0b8ff501f67488b14821f30f7ddc8eb1 0.0s
=> => exporting manifest list sha256:75f806faa4543f0d9a5abb02e3964912cf4ff269ef21c46126546ff024552303 0.0s
=> => naming to docker.io/library/springboot-docker-demo:latest    0.0s
=> => unpacking to docker.io/library/springboot-docker-demo:latest 0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/o9u5jc9paw2mhru43q5emnyrg
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo>

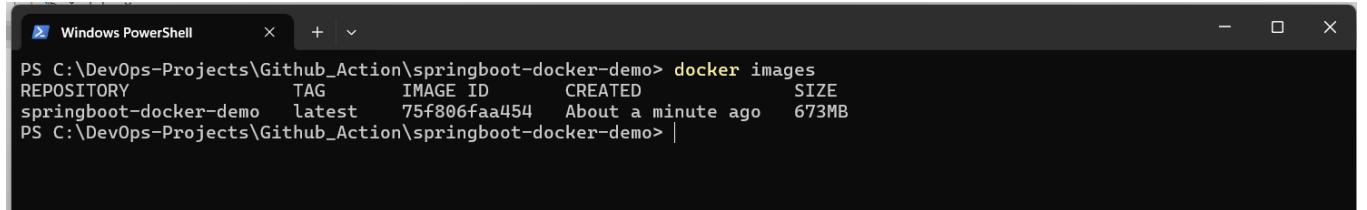
```

Head back to the Docker desktop



You can see the image has been built. Let us verify using the command:

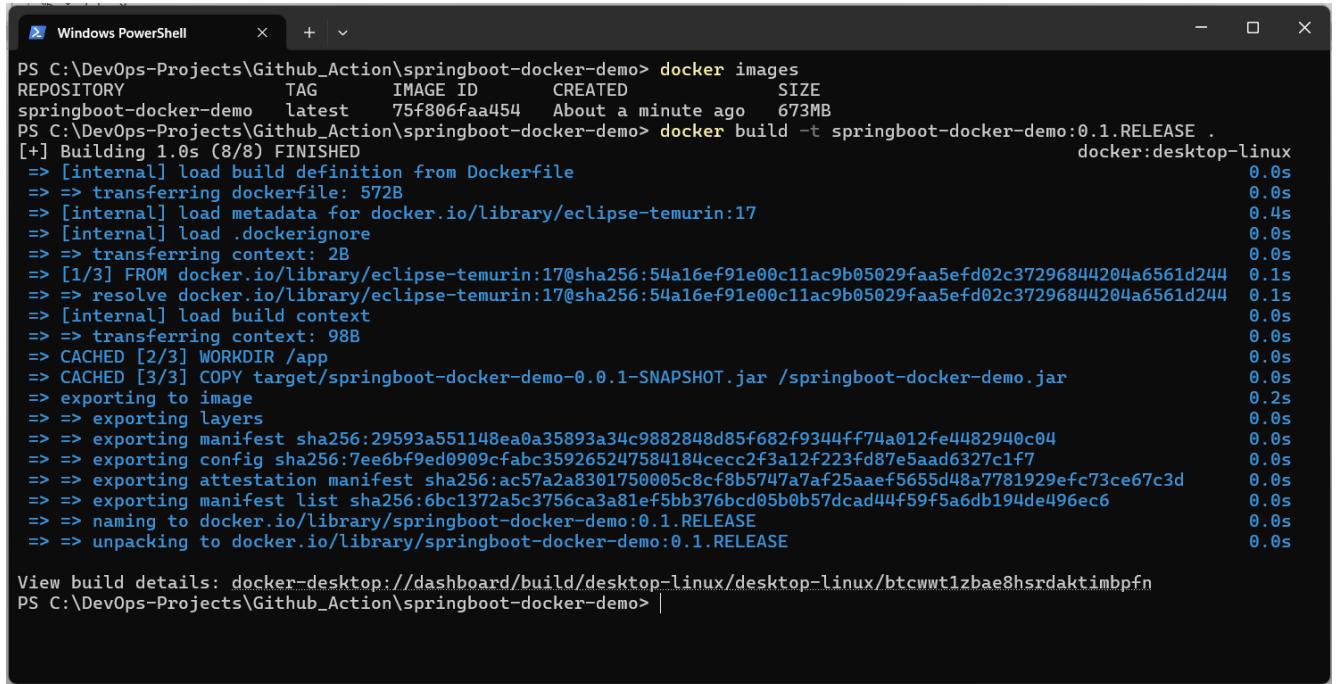
docker images



```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
springboot-docker-demo  latest  75f806faa454  About a minute ago  673MB
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

You can see our image. We can also add a tag to the image. We can add a tag name “0.1.RELEASE” by using the command:

docker build -t springboot-docker-demo:0.1.RELEASE .

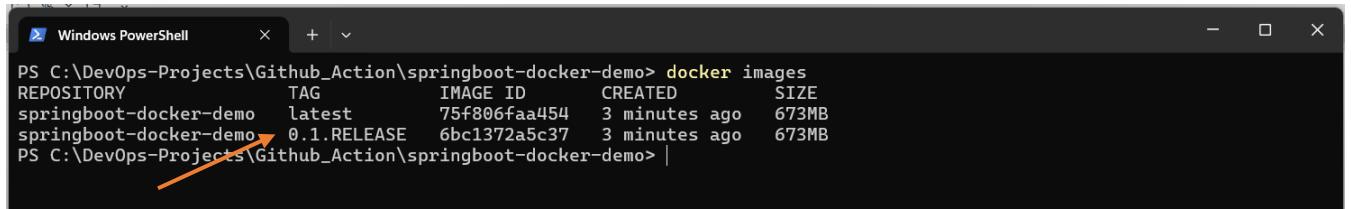


```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
springboot-docker-demo  latest  75f806faa454  About a minute ago  673MB
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker build -t springboot-docker-demo:0.1.RELEASE .
[+] Building 1.0s (8/8) FINISHED
=> [internal] load build definition from Dockerfile               0.0s
=> => transferring dockerfile: 572B                               0.0s
=> [internal] load metadata for docker.io/library/eclipse-temurin:17 0.4s
=> [internal] load .dockignore                                0.0s
=> => transferring context: 2B                                0.0s
=> [1/3] FROM docker.io/library/eclipse-temurin:17@sha256:54a16ef91e00c11ac9b05029faa5efd02c37296844204a6561d244 0.1s
=> => resolve docker.io/library/eclipse-temurin:17@sha256:54a16ef91e00c11ac9b05029faa5efd02c37296844204a6561d244 0.1s
=> [internal] load build context                            0.0s
=> => transferring context: 98B                            0.0s
=> CACHED [2/3] WORKDIR /app                             0.0s
=> CACHED [3/3] COPY target/springboot-docker-demo-0.0.1-SNAPSHOT.jar /springboot-docker-demo.jar 0.0s
=> exporting to image                                     0.2s
=> => exporting layers                                    0.0s
=> => exporting manifest sha256:29593a551148ea0a35893a34c988284d85f682f9344ff74a012fe4482940c04 0.0s
=> => exporting config sha256:7ee6bf9ed0909cfabc359265247584184cecc2f3a12f223fd87e5aad6327c1f7 0.0s
=> => exporting attestation manifest sha256:ac57a2a83017500058cf8b5747a7af25aaef5655d48a7781929efc73ce67c3d 0.0s
=> => exporting manifest list sha256:6bc1372a5c3756ca3a81ef5bb376bcd05b0b57dcad44f59f5a6db194de496ec6 0.0s
=> => naming to docker.io/library/springboot-docker-demo:0.1.RELEASE 0.0s
=> => unpacking to docker.io/library/springboot-docker-demo:0.1.RELEASE 0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/btcwwt1zbae8hsrdaktimbpfn
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

Let us check the list of docker images again using the command:

docker images



```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
springboot-docker-demo  latest  75f806faa454  3 minutes ago  673MB
springboot-docker-demo  0.1.RELEASE  6bc1372a5c37  3 minutes ago  673MB
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

You can see the tag.

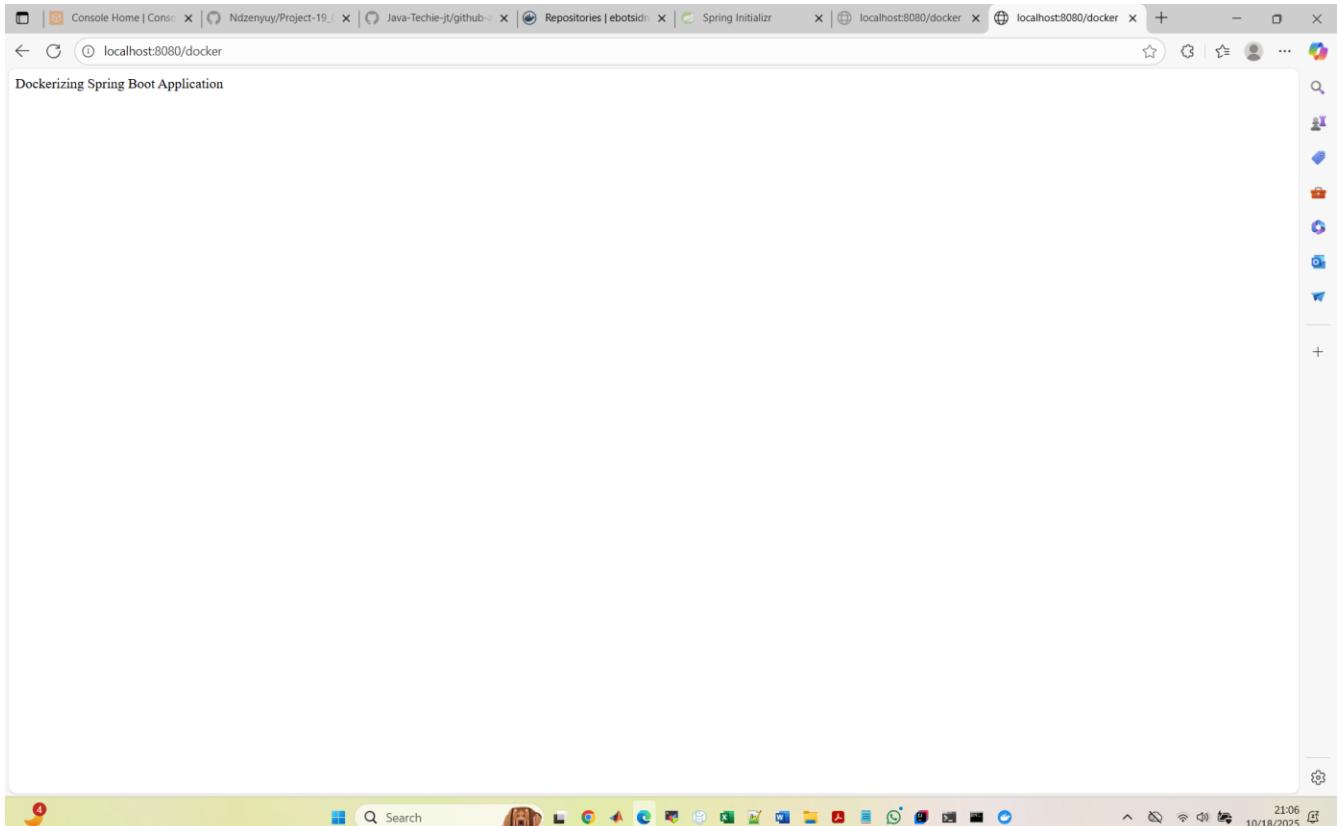
STEP 6: Run Docker Image in a Docker Container

We will run the Docker Image in a Docker container. This will be done using the command:

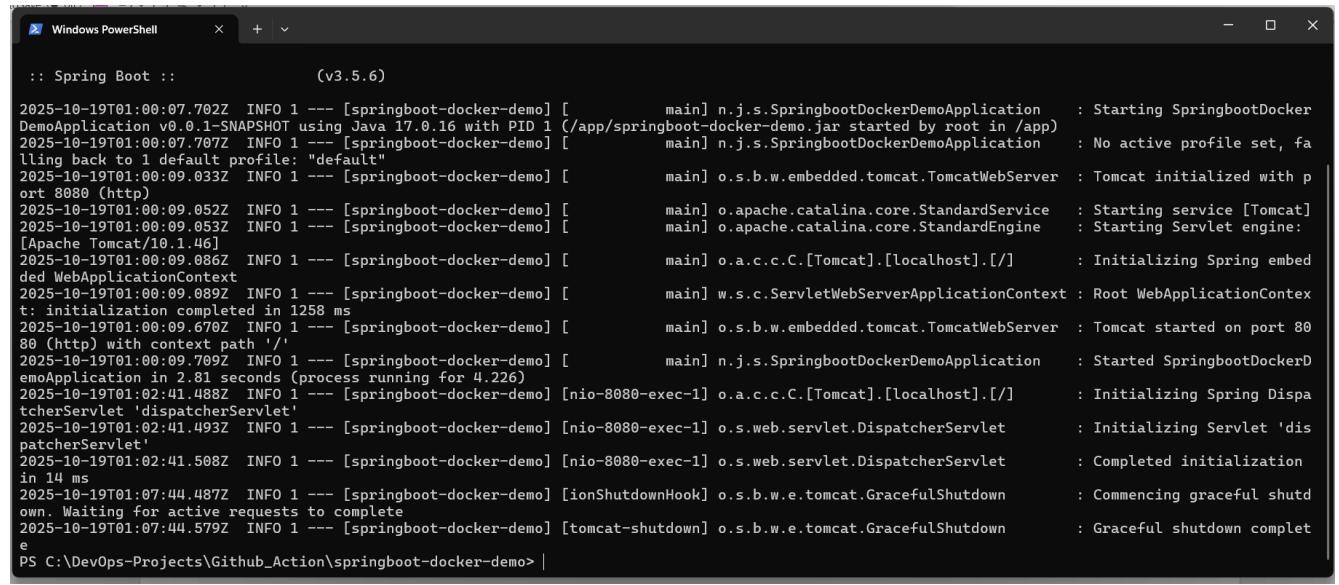
```
docker run -p 8080:8080 springboot-docker-demo
```

The `-p` is used to map the operating system port 8080 to the docker container port 8080. The image name is “**springboot-docker-demo**”

Now, let us go to our browser and test using **localhost:8080/docker**



The application is running on our container. You can go back to the command mode in the terminal using “**ctrl + c**”. This will also stop the docker container



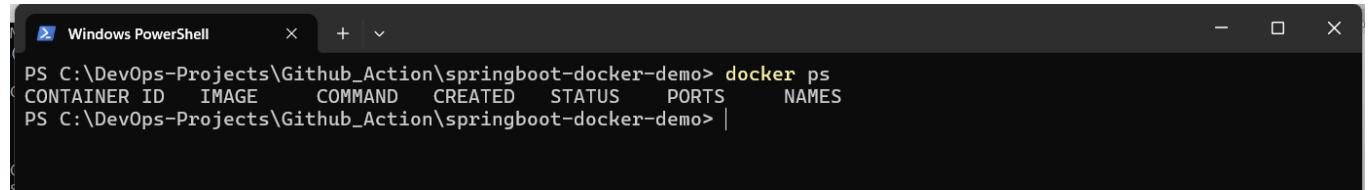
```
Windows PowerShell

:: Spring Boot ::          (v3.5.6)

2025-10-19T01:00:07.702Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Starting SpringbootDocker
DemoApplication v0.0.1-SNAPSHOT using Java 17.0.16 with PID 1 (/app/springboot-docker-demo.jar started by root in /app)
2025-10-19T01:00:07.707Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : No active profile set, falling back to 1 default profile: "default"
2025-10-19T01:00:09.033Z INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-10-19T01:00:09.052Z INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-10-19T01:00:09.053Z INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]
2025-10-19T01:00:09.086Z INFO 1 --- [springboot-docker-demo] [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2025-10-19T01:00:09.089Z INFO 1 --- [springboot-docker-demo] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext initialization completed in 1258 ms
2025-10-19T01:00:09.670Z INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 80 (http) with context path '/'
2025-10-19T01:00:09.709Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Started SpringbootDockerDemoApplication in 2.81 seconds (process running for 4.226)
2025-10-19T01:02:41.488Z INFO 1 --- [springboot-docker-demo] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-10-19T01:02:41.493Z INFO 1 --- [springboot-docker-demo] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-10-19T01:02:41.508Z INFO 1 --- [springboot-docker-demo] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 14 ms
2025-10-19T01:07:44.487Z INFO 1 --- [springboot-docker-demo] [ionShutdownHook] o.s.b.w.e.tomcat.GracefulShutdown : Commencing graceful shutdown
2025-10-19T01:07:44.579Z INFO 1 --- [springboot-docker-demo] [tomcat-shutdown] o.s.b.w.e.tomcat.GracefulShutdown : Graceful shutdown completed
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo>
```

Run the command to check if any container is running:

`docker ps`

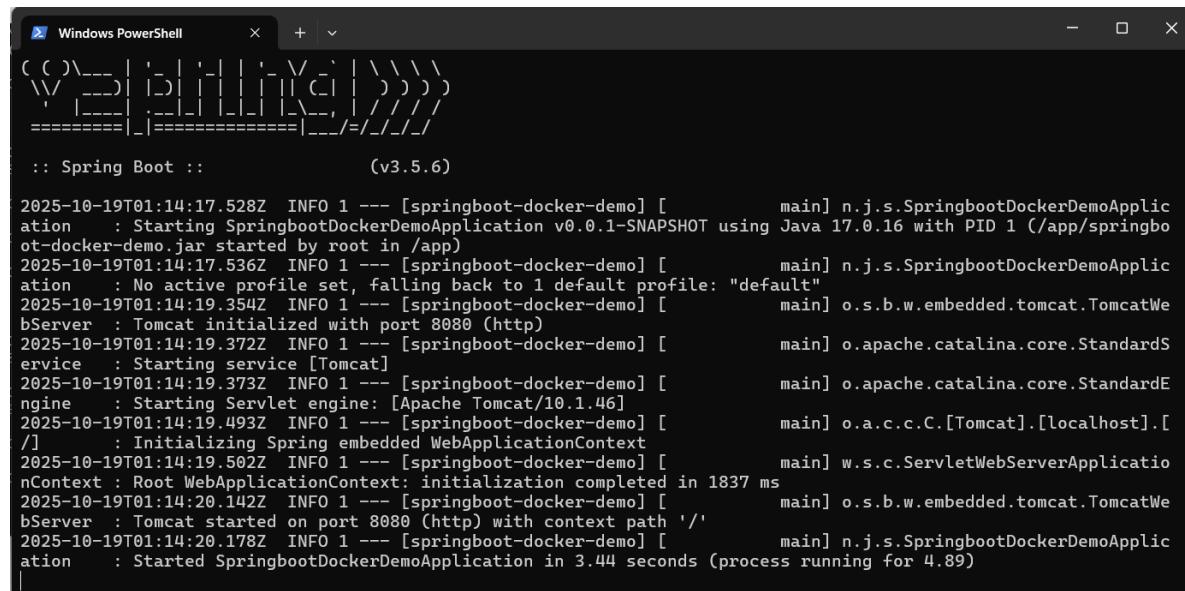


```
Windows PowerShell

PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo>
```

You can see that no container is running. Let us try to run the container on another port now. We will use port 8081. So, our command will be

`docker run -p 8081:8080 springboot-docker-demo`

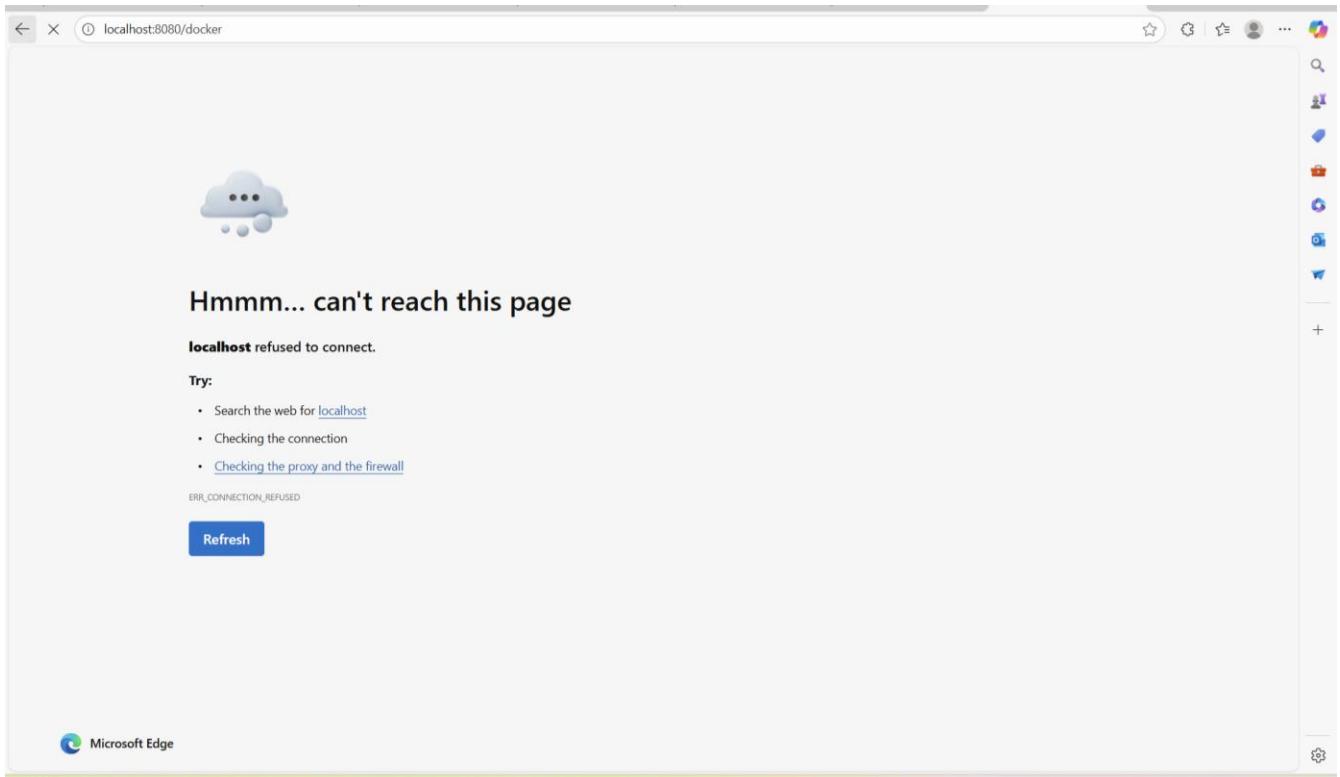


```
Windows PowerShell

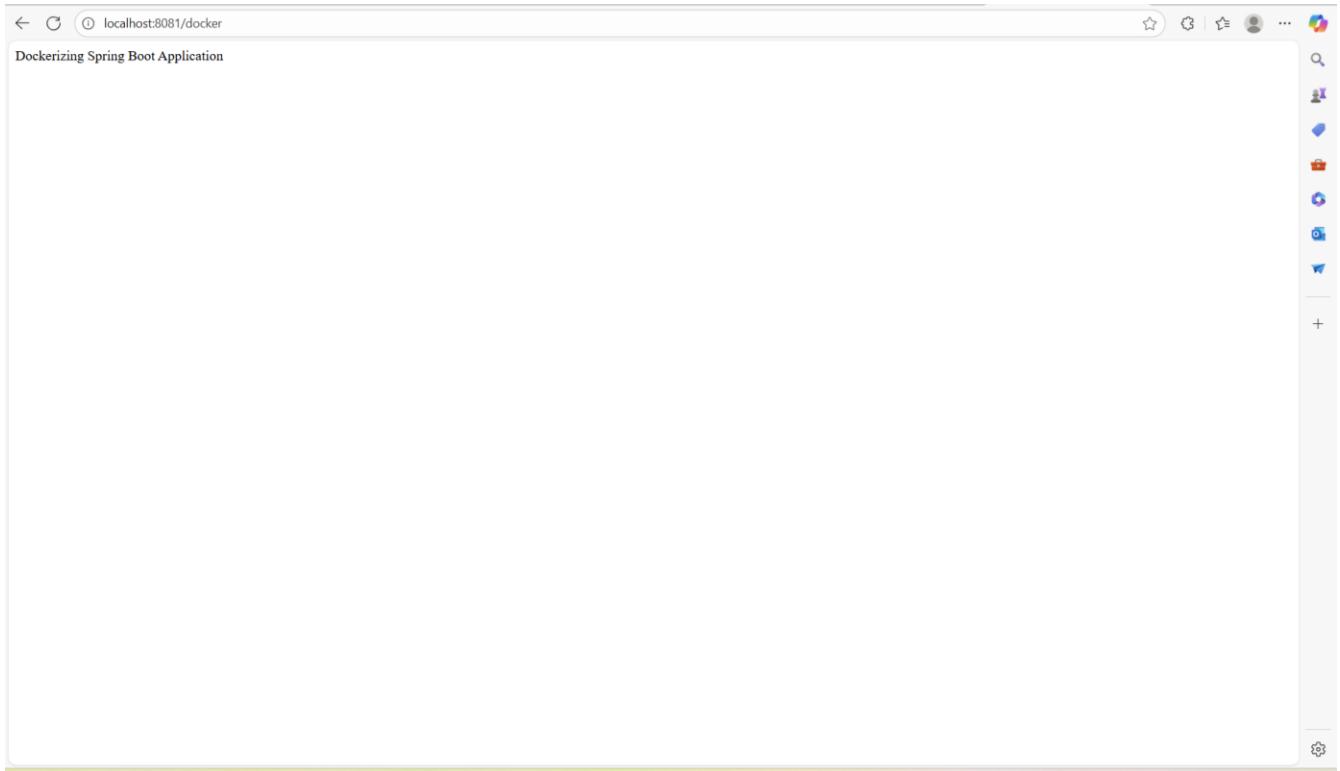
:: Spring Boot ::          (v3.5.6)

2025-10-19T01:14:17.528Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Starting SpringbootDockerDemoApplication v0.0.1-SNAPSHOT using Java 17.0.16 with PID 1 (/app/springboot-docker-demo.jar started by root in /app)
2025-10-19T01:14:17.536Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : No active profile set, falling back to 1 default profile: "default"
2025-10-19T01:14:19.354Z INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-10-19T01:14:19.372Z INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-10-19T01:14:19.373Z INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]
2025-10-19T01:14:19.493Z INFO 1 --- [springboot-docker-demo] [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2025-10-19T01:14:19.502Z INFO 1 --- [springboot-docker-demo] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1837 ms
2025-10-19T01:14:20.142Z INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-10-19T01:14:20.178Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Started SpringbootDockerDemoApplication in 3.44 seconds (process running for 4.89)
```

Let us try to test our application on port 8080 again using localhost:8080/docker



You can see that the application is not running on port 8080 anymore. Let us try to test it on port 8081 now using localhost:8081/docker

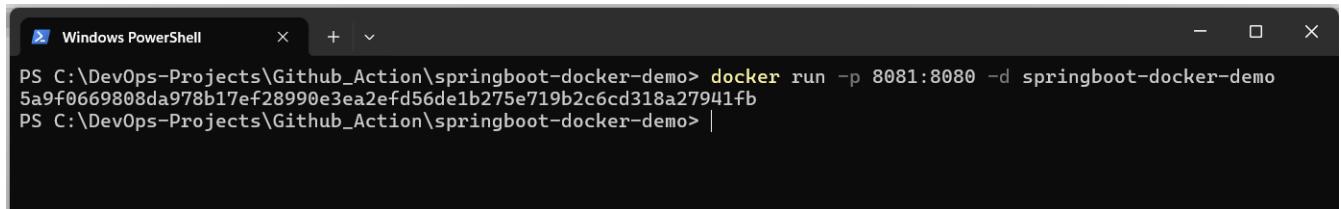


You can see that the application is now running on Port 8081.

STEP 7: Run the Docker container in Detached mode

Running the docker container in detached mode means running it in the background. To do this we run the command:

```
docker run -p 8081:8080 -d springboot-docker-demo
```



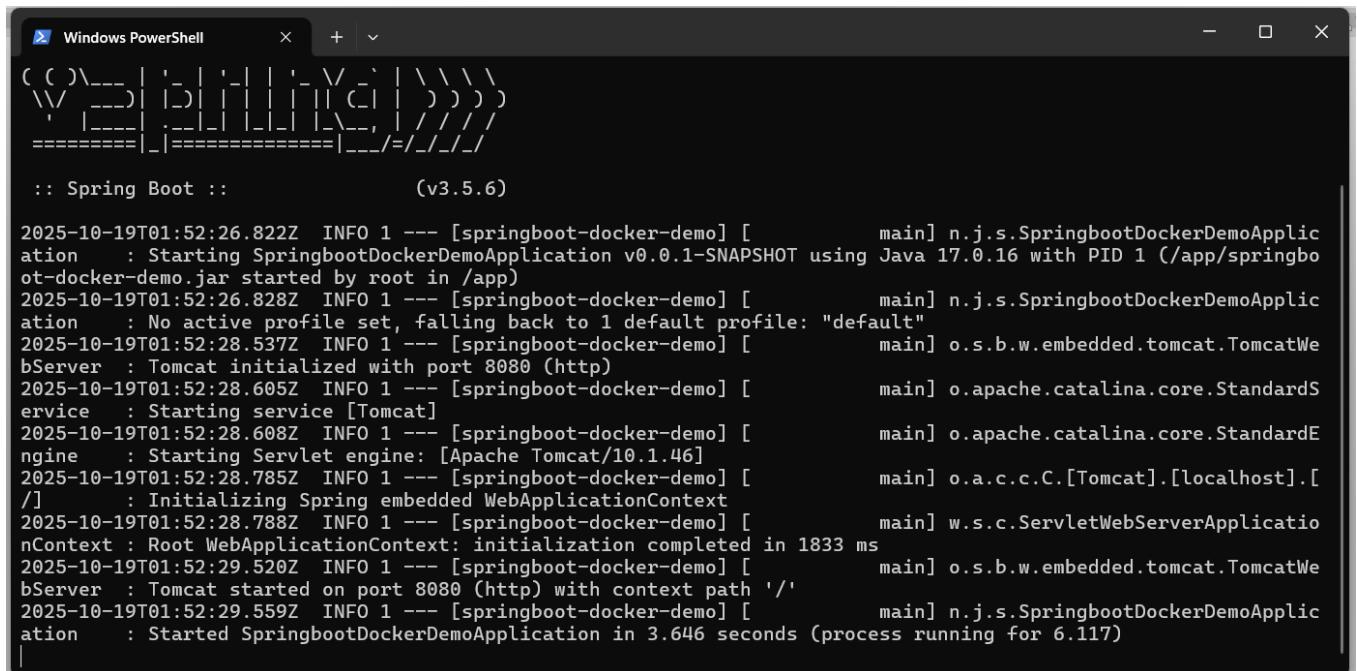
A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "docker run -p 8081:8080 -d springboot-docker-demo" is entered, followed by the output "5a9f0669808da978b17ef28990e3ea2efd56de1b275e719b2c6cd318a27941fb". The prompt "PS C:\DevOps-Projects\Github_Action\springboot-docker-demo>" is visible at the bottom.

You can see the container's ID

We can now check the logs of the docker container using the command:

```
docker logs -f 5a9f
```

5a9f is the first four digits of the docker container's ID



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "docker logs -f 5a9f" is entered, followed by the log output. The log shows the application starting up, including the Spring Boot logo and various INFO messages about the application context and Tomcat server initialization. The log ends with the message "Started SpringbootDockerDemoApplication in 3.646 seconds (process running for 6.117)".

```
(v3.5.6)

2025-10-19T01:52:26.822Z  INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Starting SpringbootDockerDemoApplication v0.0.1-SNAPSHOT using Java 17.0.16 with PID 1 (/app/springboot-docker-demo.jar started by root in /app)
2025-10-19T01:52:26.828Z  INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : No active profile set, falling back to 1 default profile: "default"
2025-10-19T01:52:28.537Z  INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-10-19T01:52:28.605Z  INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-10-19T01:52:28.608Z  INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]
2025-10-19T01:52:28.785Z  INFO 1 --- [springboot-docker-demo] [/] : Initializing Spring embedded WebApplicationContext
2025-10-19T01:52:28.788Z  INFO 1 --- [springboot-docker-demo] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1833 ms
2025-10-19T01:52:29.520Z  INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-10-19T01:52:29.559Z  INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Started SpringbootDockerDemoApplication in 3.646 seconds (process running for 6.117)
```

This is the log of the container. If you use "ctrl + c" now, you will exit to command mode but the container will still be running in the background

```
Windows PowerShell + - X
:: Spring Boot ::          (v3.5.6)

2025-10-19T01:52:26.822Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Starting SpringbootDockerDemoApplication v0.0.1-SNAPSHOT using Java 17.0.16 with PID 1 (/app/springboot-docker-demo.jar started by root in /app)
2025-10-19T01:52:26.828Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : No active profile set, falling back to 1 default profile: "default"
2025-10-19T01:52:28.537Z INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-10-19T01:52:28.605Z INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-10-19T01:52:28.608Z INFO 1 --- [springboot-docker-demo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]
2025-10-19T01:52:28.785Z INFO 1 --- [springboot-docker-demo] [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2025-10-19T01:52:28.788Z INFO 1 --- [springboot-docker-demo] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1833 ms
2025-10-19T01:52:29.520Z INFO 1 --- [springboot-docker-demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-10-19T01:52:29.559Z INFO 1 --- [springboot-docker-demo] [main] n.j.s.SpringbootDockerDemoApplication : Started SpringbootDockerDemoApplication in 3.646 seconds (process running for 6.117)
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo>
```

Verify using the command:

```
docker ps
```

```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
5a9f0669808d      springboot-docker-demo   "java -jar /app/spri..."   9 minutes ago     Up 9 minutes      0.0.0.0:8081->80
80/tcp, [::]:8081->8080/tcp
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

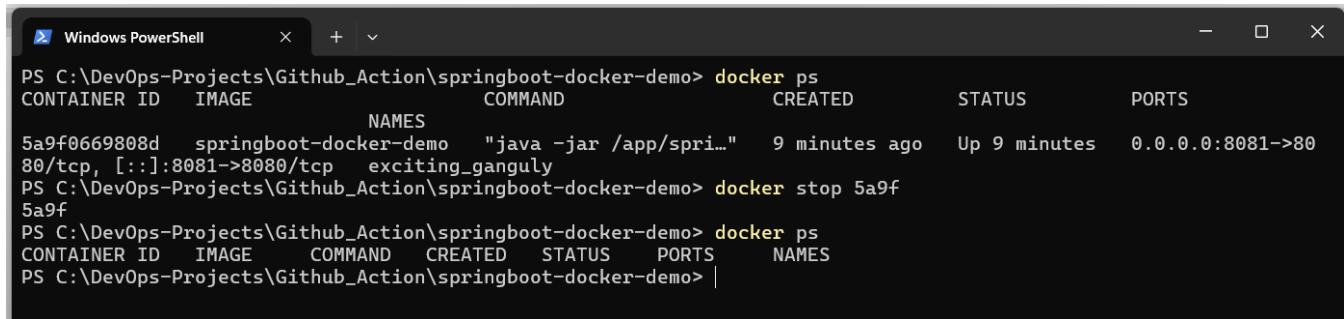
You can see that the container is still running in the background. To stop the container from running completely, you use the command:

```
docker stop 5a9f
```

```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
5a9f0669808d        springboot-docker-demo   "java -jar /app/spring-boot-docker-demo.jar"   9 minutes ago    Up 9 minutes     0.0.0.0:8081->80
80/tcp, [::]:8081->8080/tcp
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker stop 5a9f
5a9f
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

If you run the command to check the containers now, that is

```
docker ps
```



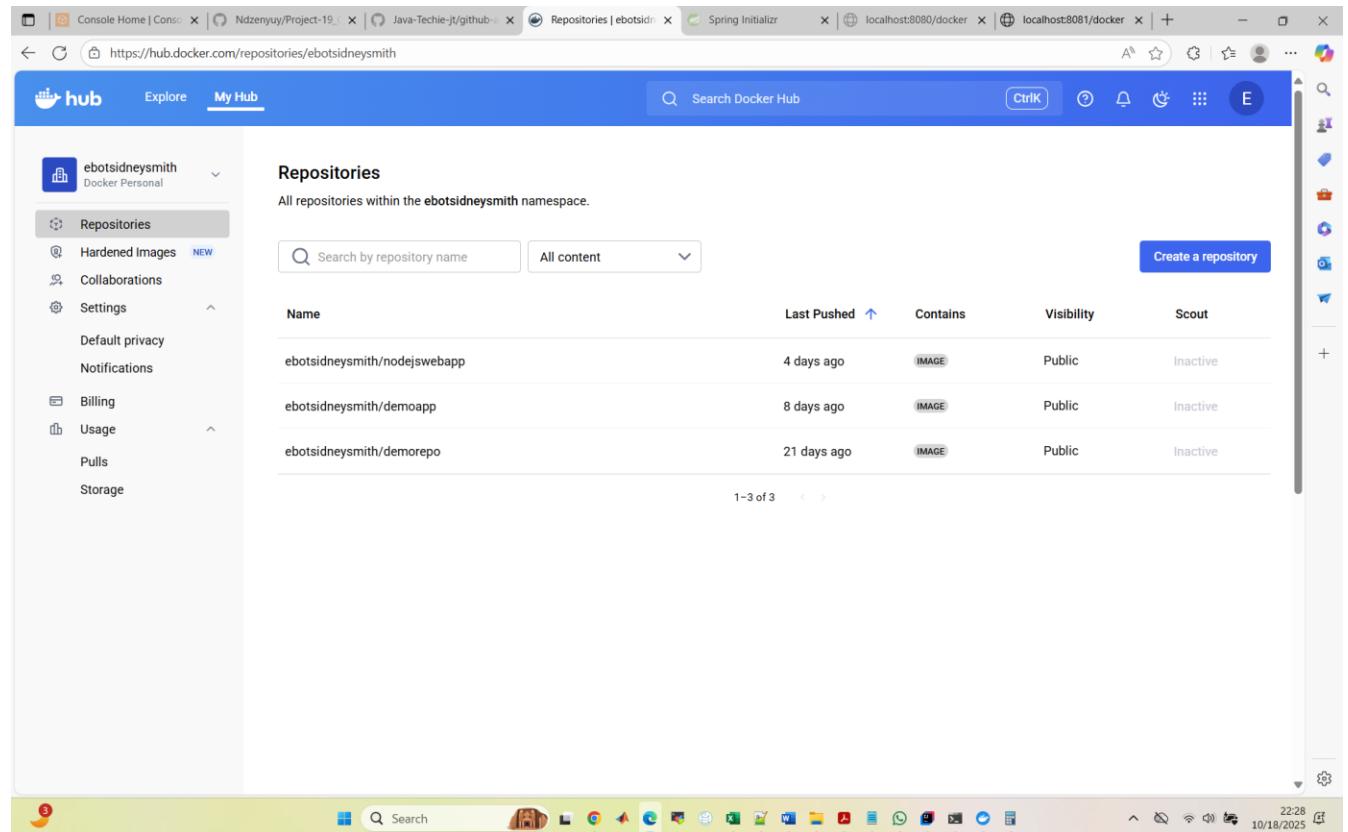
A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "docker ps" is run, showing one container named "exciting_ganguly" with the ID "5a9f0669808d". It's running a Java application from the "springboot-docker-demo" image, mapping port 8081 to 8080. The container was created 9 minutes ago and is currently up. The command "docker stop 5a9f" is then run to stop the container. Finally, "docker ps" is run again, showing no containers are running.

```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED        STATUS        PORTS
 NAMES
5a9f0669808d  springboot-docker-demo "java -jar /app/spri..."  9 minutes ago   Up 9 minutes   0.0.0.0:8081->80
80/tcp, [::]:8081->8080/tcp   exciting_ganguly
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker stop 5a9f
5a9f
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker ps
CONTAINER ID   IMAGE       CREATED      STATUS    PORTS     NAMES
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

You can see no container is running.

STEP 8: Push Docker Image from local Machine to Docker Hub

Finally, we will push the Docker image to the Docker hub. To do this you have to first create a account in Docker hub. I already have an account in Docker hub. So, I will skip that step.



The screenshot shows the Docker Hub interface. On the left, there's a sidebar with options like 'ebotsidneysmith Docker Personal', 'Repositories', 'Hardened Images', 'Collaborations', 'Settings', 'Default privacy', 'Notifications', 'Billing', 'Usage', 'Pulls', and 'Storage'. The main area is titled 'Repositories' and shows three entries:

Name	Last Pushed	Contains	Visibility	Scout
ebotsidneysmith/nodejswebapp	4 days ago	IMAGE	Public	Inactive
ebotsidneysmith/demoapp	8 days ago	IMAGE	Public	Inactive
ebotsidneysmith/demorepo	21 days ago	IMAGE	Public	Inactive

At the bottom of the page, it says '1-3 of 3'.

Whenever you push a docker image to a docker hub, a new repository is created in your Docker hub to hold the image.

Let us check the docker images using the command:

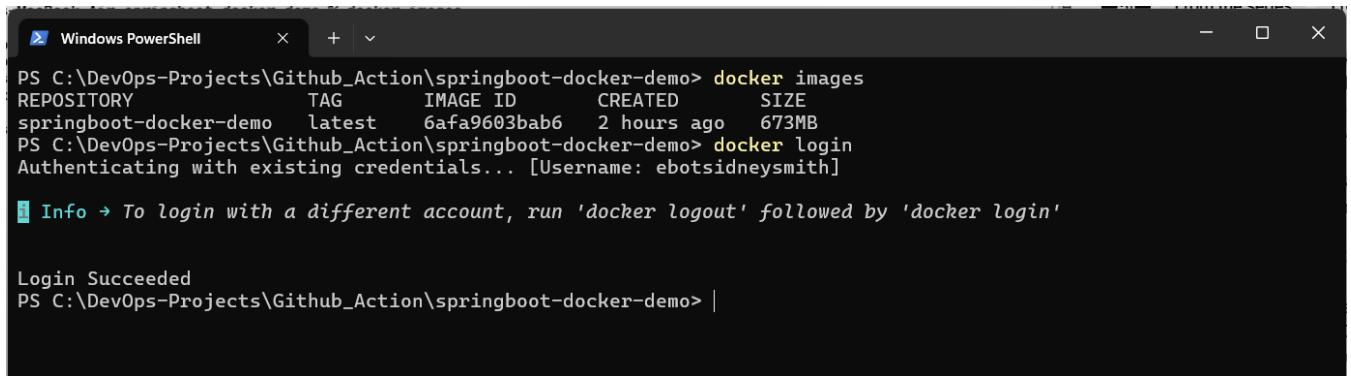
```
docker images
```



```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
springboot-docker-demo  latest   6afa9603bab6  2 hours ago  673MB
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

We have one docker image. Now, let us push this image to the docker hub. To do this, we have to login to docker in our terminal. We can do this using the command:

```
docker login
```



```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker images
REPOSITORY          TAG      IMAGE ID   CREATED    SIZE
springboot-docker-demo  latest   6afa9603bab6  2 hours ago  673MB
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker login
Authenticating with existing credentials... [Username: ebotsidneysmith]

Info → To login with a different account, run 'docker logout' followed by 'docker login'

Login Succeeded
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

I have logged into docker. Now, let us first associate our docker image with the Docker hub using the command:

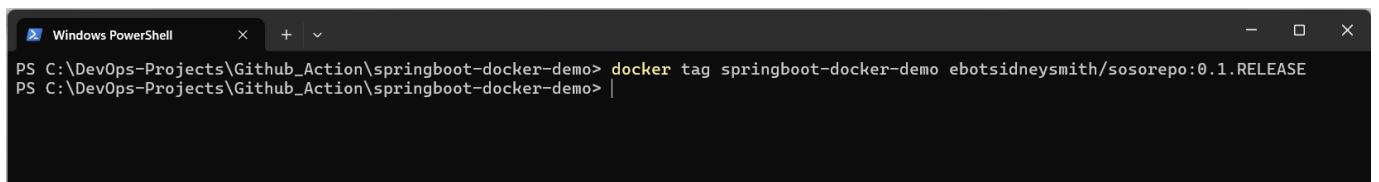
```
docker tag springboot-docker-demo ebotsidneysmith/sosorepo:0.1.RELEASE
```

springboot-docker-demo is the image name

ebotsidneysmith is my docker account ID

sosorepo is the repository to be created in Docker hub to hold the image

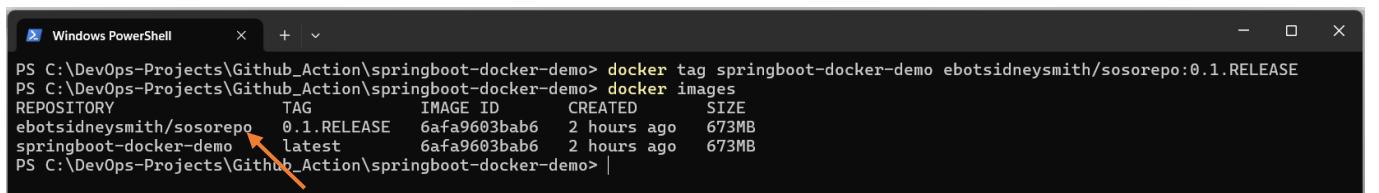
0.1.RELEASE is our image tag



```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker tag springboot-docker-demo ebotsidneysmith/sosorepo:0.1.RELEASE
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

Run the command:

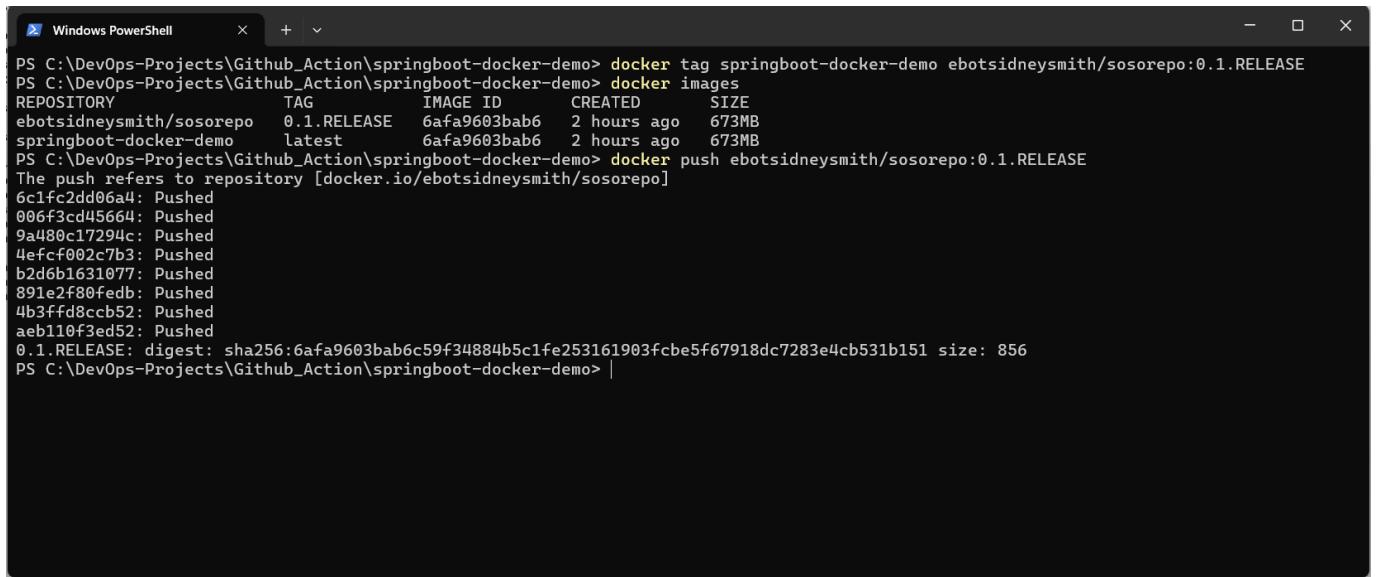
```
docker images
```



```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker tag springboot-docker-demo ebotsidneysmith/sosorepo:0.1.RELEASE
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker images
REPOSITORY          TAG      IMAGE ID   CREATED    SIZE
ebotsidneysmith/sosorepo  0.1.RELEASE  6afa9603bab6  2 hours ago  673MB
springboot-docker-demo  latest   6afa9603bab6  2 hours ago  673MB
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

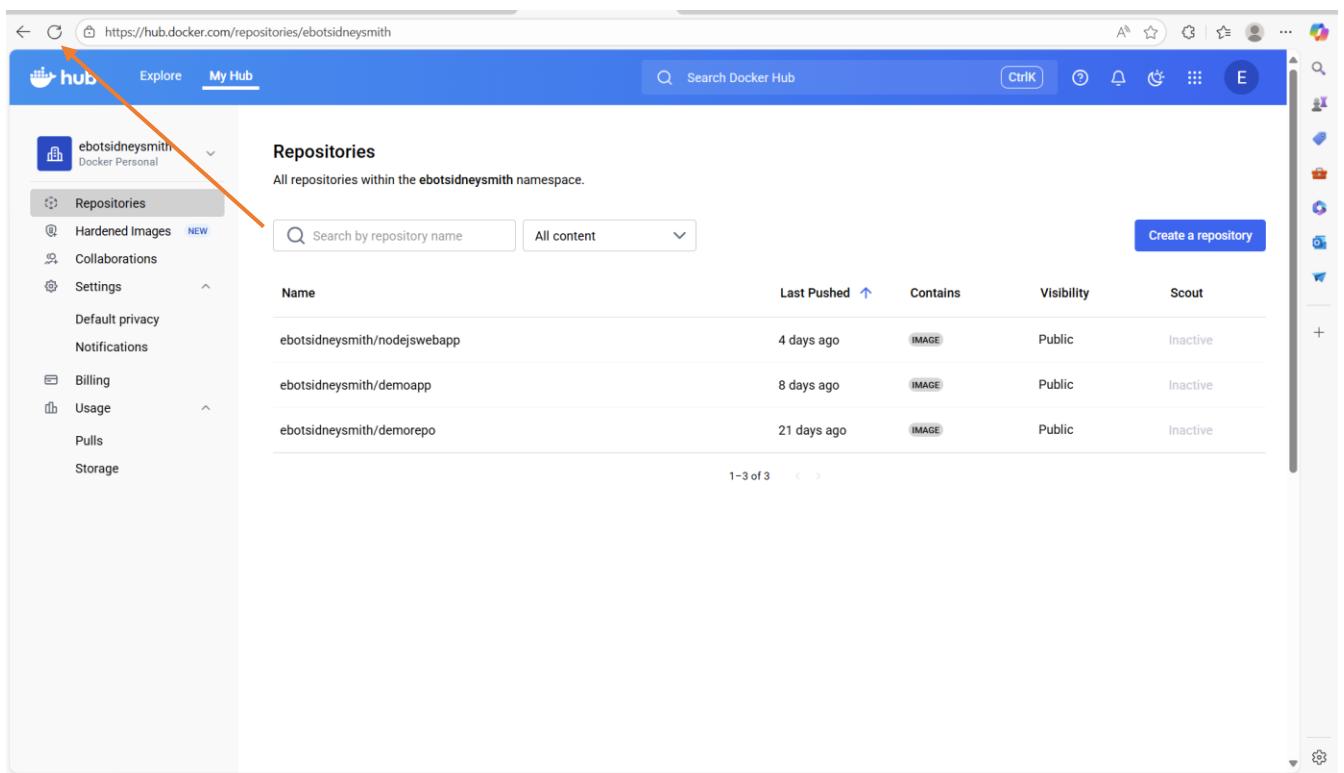
You can see that the repository ebotsidneysmith/sosorepo. Then push the image to the docker hub using the command:

```
docker push ebotsidneysmith/sosorepo:0.1.RELEASE
```



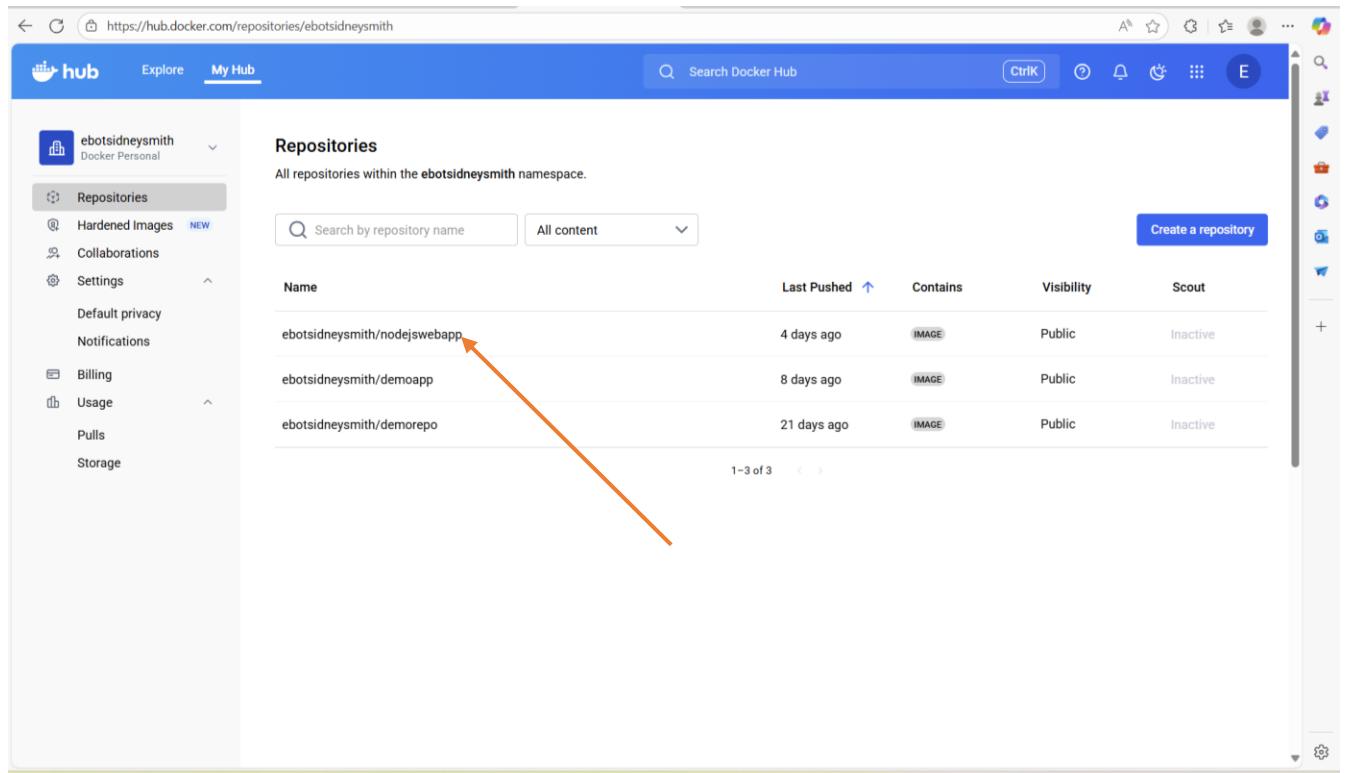
```
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker tag springboot-docker-demo ebotsidneysmith/sosorepo:0.1.RELEASE
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
ebotsidneysmith/sosorepo  0.1.RELEASE  6afa9603bab6  2 hours ago  673MB
springboot-docker-demo  latest    6afa9603bab6  2 hours ago  673MB
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> docker push ebotsidneysmith/sosorepo:0.1.RELEASE
The push refers to repository [docker.io/ebotsidneysmith/sosorepo]
6c1fc2dd06a4: Pushed
006f3cd45664: Pushed
9a480c17294c: Pushed
4e-fc-f002c7b3: Pushed
b2d6b1631077: Pushed
891e2f80fedb: Pushed
4b3ffd8ccb52: Pushed
aeb110f3ed52: Pushed
0.1.RELEASE: digest: sha256:6afa9603bab6c59f34884b5c1fe253161903fcbe5f67918dc7283e4cb531b151 size: 856
PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |
```

Let us head to our Docker hub now



The screenshot shows the Docker Hub interface. On the left, there is a sidebar with navigation links: 'Explore', 'My Hub', 'ebotsidneysmith' (selected), 'Repositories', 'Hardened Images', 'Collaborations', 'Settings', 'Default privacy', 'Notifications', 'Billing', 'Usage', 'Pulls', and 'Storage'. The main content area is titled 'Repositories' and displays three repositories under the heading 'All repositories within the ebotsidneysmith namespace.' The repositories listed are 'ebotsidneysmith/nodejswebapp', 'ebotsidneysmith/demoapp', and 'ebotsidneysmith/demorepo'. Each repository entry includes a preview image, the last pushed date, visibility status (Public), and a 'Scout' link.

Refresh the page



The screenshot shows the Docker Hub interface for the user ebotsidneysmith. The left sidebar contains navigation links for Repositories, Hardened Images, Collaborations, Settings, Default privacy, Notifications, Billing, Usage, Pulls, and Storage. The main area is titled 'Repositories' and displays three repositories under the namespace 'ebotsidneysmith'. The repositories listed are 'nodejswebapp', 'demoapp', and 'demorepo'. An orange arrow points to the first repository, 'nodejswebapp'. The table headers are Name, Last Pushed, Contains, Visibility, and Scout. The data for each repository is as follows:

Name	Last Pushed	Contains	Visibility	Scout
ebotsidneysmith/nodejswebapp	4 days ago	IMAGE	Public	Inactive
ebotsidneysmith/demoapp	8 days ago	IMAGE	Public	Inactive
ebotsidneysmith/demorepo	21 days ago	IMAGE	Public	Inactive

You can see our repository now.

STEP 9: Pull an Image from a Docker Hub

We can also pull an image from a docker hub and deploy it on different environments. This will be demonstrated here.

The screenshot shows the Docker Hub interface for the user ebotsidneysmith. On the left, there's a sidebar with options like Repositories, Hardened Images, Collaborations, Settings, Billing, Usage, Pulls, and Storage. The main area is titled 'Repositories' and shows three entries:

Name	Last Pushed	Contains	Visibility	Scout
ebotsidneysmith/nodejswebapp	4 days ago	IMAGE	Public	Inactive
ebotsidneysmith/demoapp	8 days ago	IMAGE	Public	Inactive
ebotsidneysmith/demorepo	21 days ago	IMAGE	Public	Inactive

An orange arrow points to the first repository, 'nodejswebapp'.

Click on the docker repository

The screenshot shows the Docker Hub repository page for 'ebotsidneysmith/sosorepo'. The left sidebar is identical to the previous screen. The main area shows the repository details:

ebotsidneysmith/sosorepo ← (highlighted by an orange arrow)

Last pushed 9 minutes ago · Repository size: 206.3 MB

Add a description ⓘ ⓘ
Add a category ⓘ ⓘ

General Tags Image Management BETA Collaborators Webhooks Settings

Tags
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
0.1.RELEASE	Image	Image	less than 1 day	9 minutes

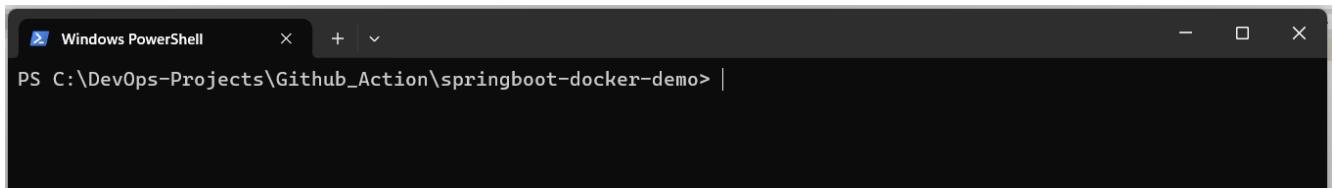
[See all](#)

Using 0 of 1 private repositories.
Docker commands
To push a new tag to this repository:
`docker push ebotsidneysmith/sosorepo:tagname`

buildcloud
Build with Docker Build Cloud
Accelerate image build times with access to cloud-based builders and shared cache.
Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.
Get faster builds through shared caching across your team, native multi-platform support, and encrypted data transfer - all without managing infrastructure.
[Go to Docker Build Cloud →](#)

Copy the docker ID and the repository: **ebotsidneysmith/sosorepo**

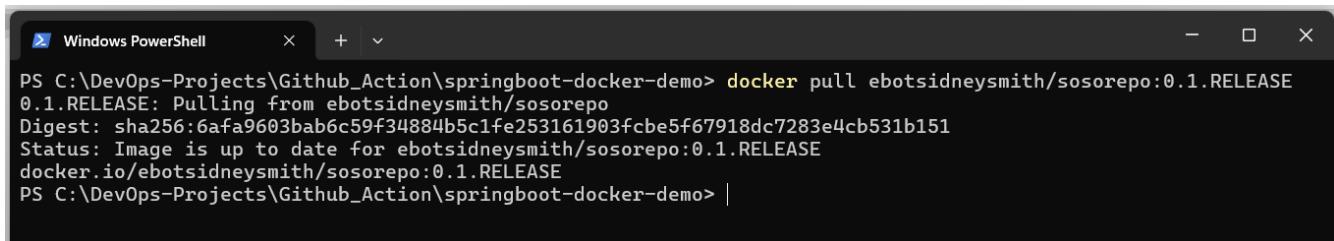
Head back to our terminal



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command prompt shows "PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |". The window has standard minimize, maximize, and close buttons at the top right.

Then run the command:

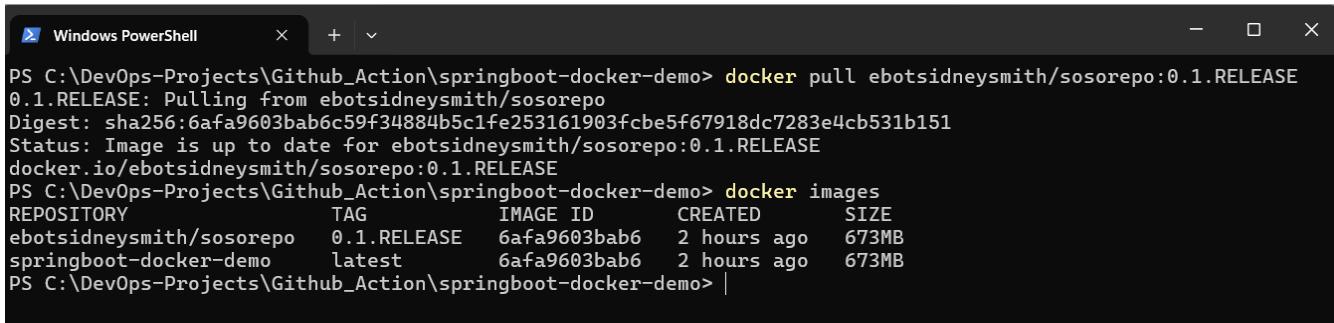
```
docker pull ebotsidneysmith/sosorepo:0.1.RELEASE
```



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "docker pull ebotsidneysmith/sosorepo:0.1.RELEASE" is run, followed by its output: "0.1.RELEASE: Pulling from ebotsidneysmith/sosorepo", "Digest: sha256:6afa9603bab6c59f34884b5c1fe253161903fcbe5f67918dc7283e4cb531b151", "Status: Image is up to date for ebotsidneysmith/sosorepo:0.1.RELEASE", and "docker.io/ebotsidneysmith/sosorepo:0.1.RELEASE". The command prompt then shows "PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |".

We have successfully pulled the docker image from the docker hub. Let us check using the command:

```
docker images
```



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "docker pull ebotsidneysmith/sosorepo:0.1.RELEASE" is run again, followed by "0.1.RELEASE: Pulling from ebotsidneysmith/sosorepo", "Digest: sha256:6afa9603bab6c59f34884b5c1fe253161903fcbe5f67918dc7283e4cb531b151", "Status: Image is up to date for ebotsidneysmith/sosorepo:0.1.RELEASE", and "docker.io/ebotsidneysmith/sosorepo:0.1.RELEASE". Then, the command "docker images" is run, showing a table of images:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ebotsidneysmith/sosorepo	0.1.RELEASE	6afa9603bab6	2 hours ago	673MB
springboot-docker-demo	latest	6afa9603bab6	2 hours ago	673MB

The command prompt then shows "PS C:\DevOps-Projects\Github_Action\springboot-docker-demo> |".