Dockerize Spring Boot Application

Deploy Spring Boot Application to Docker/ Containerize Spring Boot App to Docker

In this example I will be creating simple hello world Spring Boot Application and containerizing using Docker.First, let us see what are all the technologies and tool used for this example.

Technologies and Tools Used:

1. Java 1.8, JRE 1.8 (To download goto http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)

2. Maven 4.0.0

3. Spring Boot Version 1.5.8

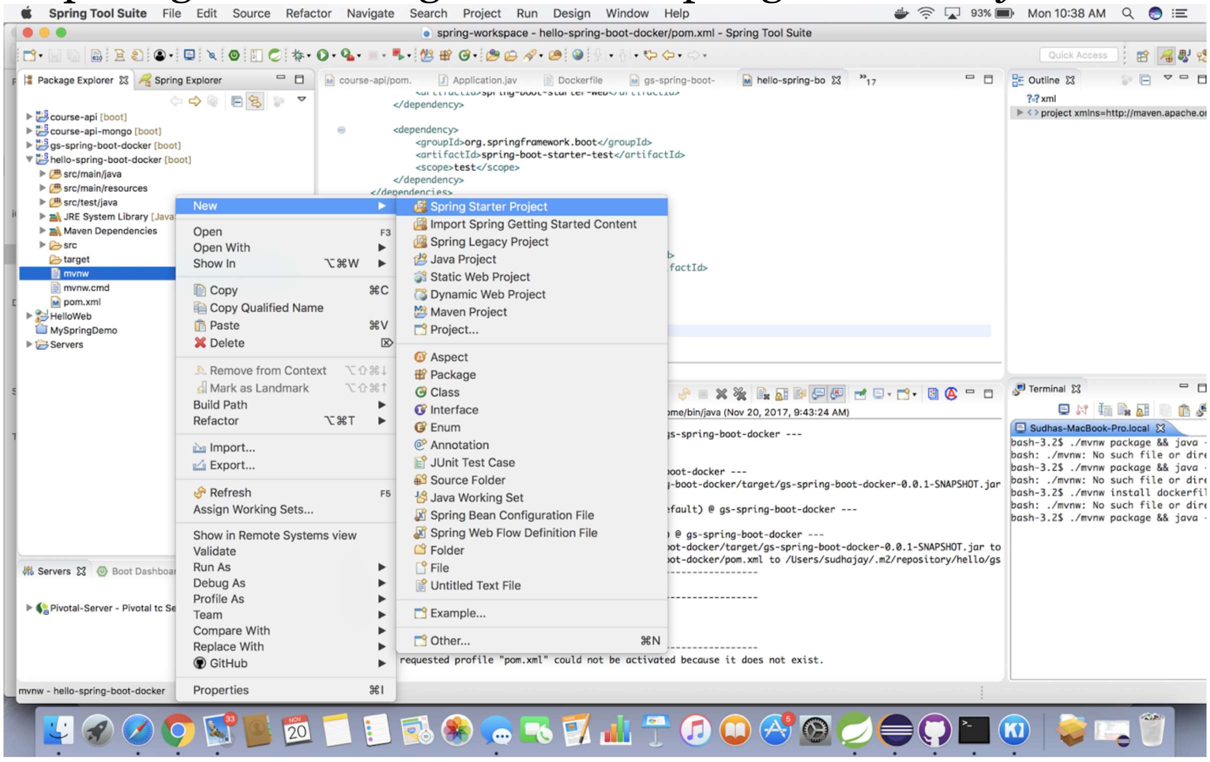
4. Spring Tool Suite(Version: 3.9.1.RELEASE(to download: https://spring.io/tools/sts/all)

5. Docker Community Edition Version: Version 17.09.0-ce-mac35 (19611) (To Install Docker: https://docs.docker.com/docker-for-mac/install/)

The tool or IDE used to create a project is STS. Follow the steps below to create a simple hello-spring-boot-dockers project.

Note: Assumption here is that one has basic knowledge on how to use Eclipse/ STS IDE and know how to create a simple Project.

Step1: Right click and go to New-> Spring Starter Project

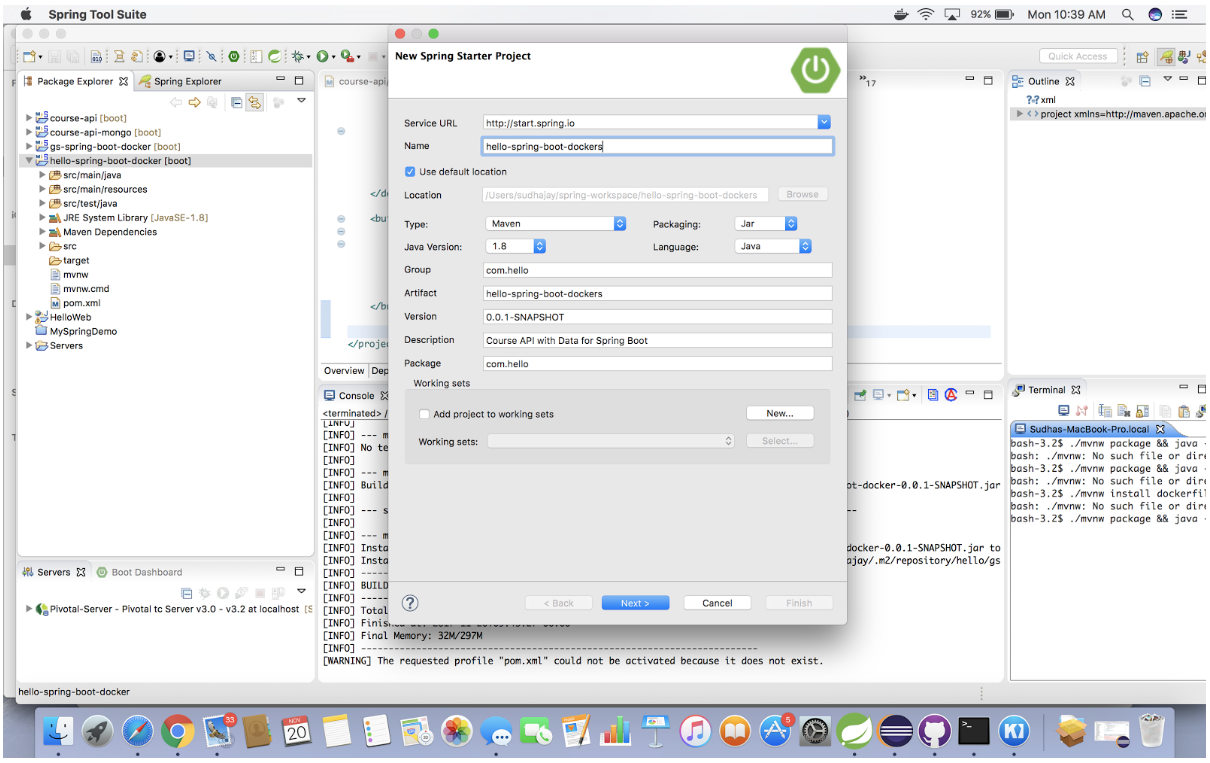


```
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

```
ome/bin/java (Nov 20, 2017, 9:43:24 AM)
js-spring-boot-docker ---

oot-docker ---
j-boot-docker/target/gs-spring-boot-docker-0.0.1-SNAPSHOT.jar

efault) @ gs-spring-boot-docker ---

) @ gs-spring-boot-docker ---
ot-docker/target/gs-spring-boot-docker-0.0.1-SNAPSHOT.jar to
ot-docker/pom.xml to /Users/sudhajay/.m2/repository/hello/gs
-----------------

-----------------

-----------------
requested profile "pom.xml" could not be activated because it does not exist.
```

```
bash-3.2$ ./mvnw package && java
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw package && java
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw install dockerfil
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw package && java
```

Step 2: Next enter the values in the following fields: Name, Type, Java Version, Group id, Artifact Id, Version, Package.



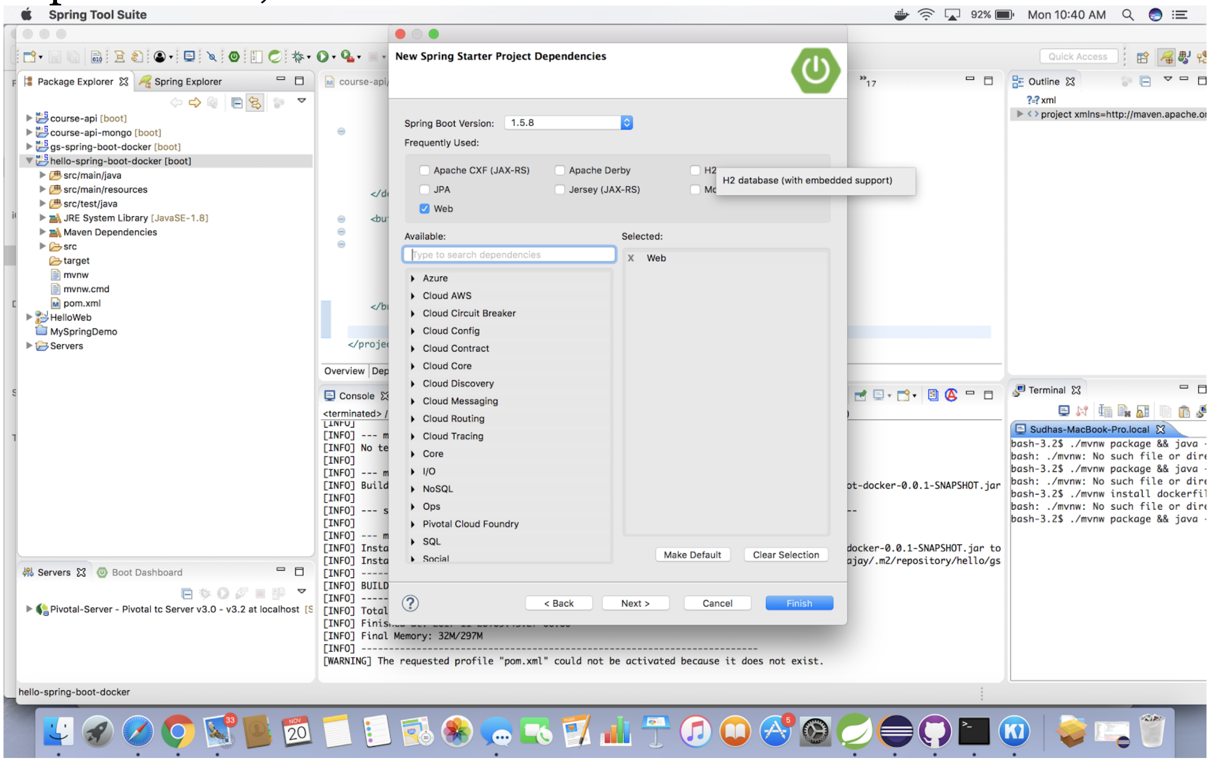
New Spring Starter Project

Service URL: http://start.spring.io
Name: hello-spring-boot-dockers
Use default location [✓]
Location: /Users/sudhajay/spring-workspace/hello-spring-boot-dockers
Type: Maven
Packaging: Jar
Java Version: 1.8
Language: Java
Group: com.hello
Artifact: hello-spring-boot-dockers
Version: 0.0.1-SNAPSHOT
Description: Course API with Data for Spring Boot
Package: com.hello
Working sets
Add project to working sets
Working sets:

```
[INFO] Final Memory: 32M/297M
[INFO] ------------------------------------------------------------------------
[WARNING] The requested profile "pom.xml" could not be activated because it does not exist.
```

```
bash-3.2$ ./mvnw package && java -
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw package && java -
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw install dockerfil
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw package && java -
```

Step 3: Choose Spring Boot Version. Here I am using 1.5.8. Since we are using rest we will choose Web for project dependencies, so I selected web from the list.

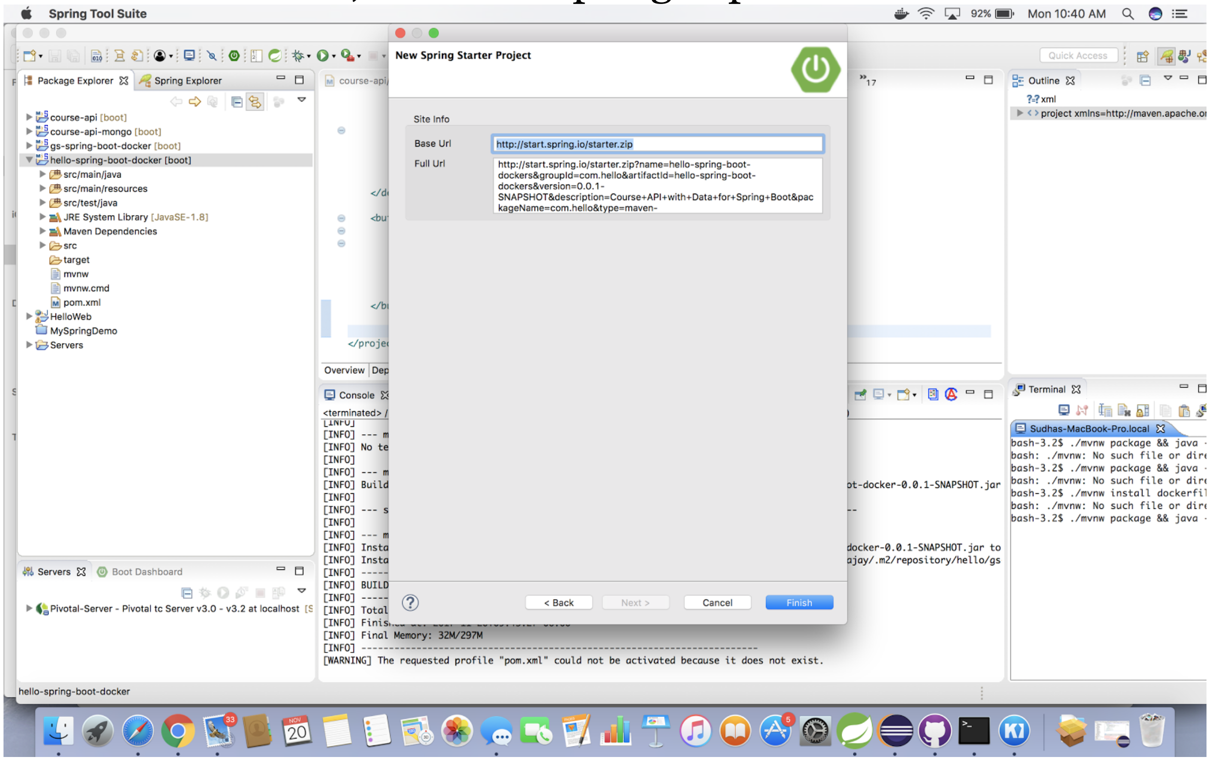


New Spring Starter Project Dependencies

Spring Boot Version: 1.5.8
Frequently Used:
Apache CXF (JAX-RS)
Apache Derby
H2
JPA
Jersey (JAX-RS)
[✓] Web
H2 database (with embedded support)
Available:
Type to search dependencies
Azure
Cloud AWS
Cloud Circuit Breaker
Cloud Config
Cloud Contract
Cloud Core
Cloud Discovery
Cloud Messaging
Cloud Routing
Cloud Tracing
Core
I/O
NoSQL
Ops
Pivotal Cloud Foundry
SQL
Social
Selected:
X Web

```
[INFO] Final Memory: 32M/297M
[INFO] ------------------------------------------------------------------------
[WARNING] The requested profile "pom.xml" could not be activated because it does not exist.
```

Step 4: Click Finish. This will create Spring Boot Maven project with all the Maven, Web and Spring dependencies.



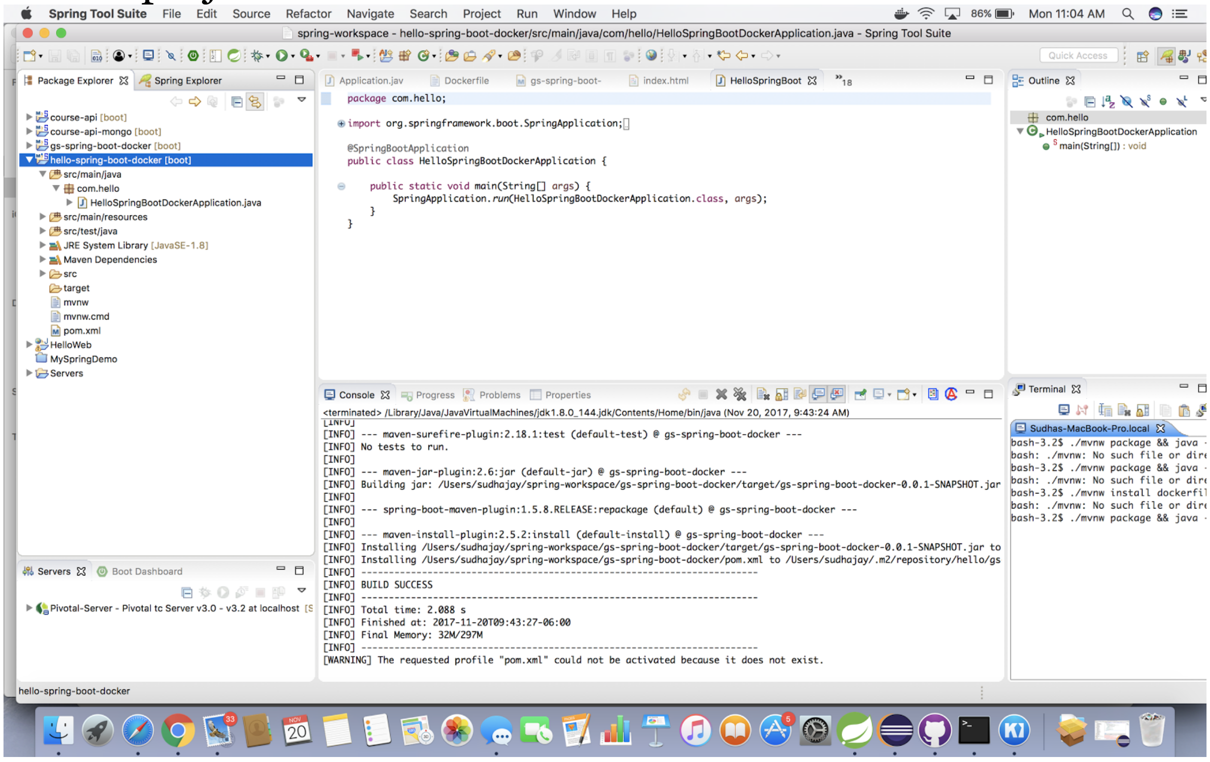
New Spring Starter Project

Site Info

Base Url: http://start.spring.io/starter.zip

Full Url: http://start.spring.io/starter.zip?name=hello-spring-boot-dockers&groupId=com.hello&artifactId=hello-spring-boot-dockers&version=0.0.1-SNAPSHOT&description=Course+API+with+Data+for+Spring+Boot&packageName=com.hello&type=maven-

Once complete, you workspace will have the hello-spring-boot-docker project.



spring-workspace - hello-spring-boot-docker/src/main/java/com/hello/HelloSpringBootDockerApplication.java - Spring Tool Suite

```
hello-spring-boot-docker [boot]
  src/main/java
    com.hello
      HelloSpringBootDockerApplication.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  mvnw
  mvnw.cmd
  pom.xml
```

```java
package com.hello;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class HelloSpringBootDockerApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloSpringBootDockerApplication.class, args);
    }
}
```

```
<terminated> /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (Nov 20, 2017, 9:43:24 AM)
[INFO] --- maven-surefire-plugin:2.18.1:test (default-test) @ gs-spring-boot-docker ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.6:jar (default-jar) @ gs-spring-boot-docker ---
[INFO] Building jar: /Users/sudhajay/spring-workspace/gs-spring-boot-docker/target/gs-spring-boot-docker-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:1.5.8.RELEASE:repackage (default) @ gs-spring-boot-docker ---
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ gs-spring-boot-docker ---
[INFO] Installing /Users/sudhajay/spring-workspace/gs-spring-boot-docker/target/gs-spring-boot-docker-0.0.1-SNAPSHOT.jar to
[INFO] Installing /Users/sudhajay/spring-workspace/gs-spring-boot-docker/pom.xml to /Users/sudhajay/.m2/repository/hello/gs
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.088 s
[INFO] Finished at: 2017-11-20T09:43:27-06:00
[INFO] Final Memory: 32M/297M
[INFO] ------------------------------------------------------------------------
[WARNING] The requested profile "pom.xml" could not be activated because it does not exist.
```

```
bash-3.2$ ./mvnw package && java -
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw package && java -
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw install dockerfil
bash: ./mvnw: No such file or dire
bash-3.2$ ./mvnw package && java -
```

The Maven pom.xml that is generated looks like this:

```
<?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/4.0.0htt
p://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.hello</groupId>
 <artifactId>hello-spring-boot-docker</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>
 <name>hello-spring-boot-docker</name>
 <description>Hello Spring Boot Docker
Sample</description>
<parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>1.5.8.RELEASE</version>
 <relativePath/> <!—lookup parent from repository →
 </parent>
 <properties>
 <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
 <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
 <java.version>1.8</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-starter-test</artifactId>
 <scope>test</scope>
 </dependency>
 </dependencies>
```

```
<build>
 <plugins>
 <plugin>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-maven-plugin</artifactId>
 </plugin>
 </plugins>
 </build>
</project>
```

The *HelloSpringBootDockerApplication.java* is created under the package com.hello;

```
package com.hello;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicati
on;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;
@SpringBootApplication
public class HelloSpringBootDockerApplication {
public static void main(String[] args) {
SpringApplication.run(HelloSpringBootDockerApplication.cla
ss, args);
}
}
```

To make it rest add the *@RestController* to the *HelloSpringBootDockerApplication.java* class. Now, the class looks like this:

```
package com.hello;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicati
on;
import
```

```
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;
@SpringBootApplication
@RestController
public class HelloSpringBootDockerApplication {

 @RequestMapping("/hello")
 public String sayHello() {
 return "Hello Docker World";
 }
 public static void main(String[] args) {
 SpringApplication.run(HelloSpringBootDockerApplication.cla
ss, args);
 }
}
```

If you are using Maven, execute:

```
$ ./mvnw package && java -jar target/hello-spring-boot-
docker-0.1.0.jar
```

This will create jar file. Else using IDE(STS) Run as Maven Clean and Maven Install to create the jar file.

Now, your spring boot app is ready to be containerized.

How to Containerize It

Docker has a simple Dockerfile file format that it uses to specify the "layers" of

an image. So let’s go ahead and create a Dockerfile in our Spring Boot project:

Dockerfile

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG JAR_FILE
ADD ${JAR_FILE} hello-spring-boot-docker-0.0.1-SNAPSHOT.jar
ENTRYPOINT [“java”,”-Djava.security.egd=file:/dev/./urandom”,”-jar”,”/hello-spring-boot-docker-0.0.1-SNAPSHOT.jar”]
```

This Dockerfile is very simple, but that’s all you need to run a Spring Boot app:

just Java and a JAR file. The project JAR file is ADDed to the container as “hello-spring-boot-docker-0.0.1-SNAPSHOT.jar” and then executed in the ENTRYPOINT.

We added a VOLUME pointing to “/tmp” because that is where a Spring Boot application creates working directories for Tomcat by default. The effect is to create a temporary file on your host under “/var/lib/docker” and link it to the container under “/tmp”. This step is optional for the simple app that we wrote here, but can be necessary for other Spring Boot applications if they need to actually write in the filesystem.

To reduce Tomcat startup time we added a system property pointing to “/dev/urandom” as a source of entropy.

To build the image you can use some tooling for Maven or Gradle from the community

Build a Docker Image with Maven

In the Maven pom.xml you should add a new plugin like this

pom.xml

```
<properties>
 <docker.image.prefix>springio</docker.image.prefix>
</properties>
<build>
 <plugins>
 <plugin>
 <groupId>com.spotify</groupId>
 <artifactId>dockerfile-maven-plugin</artifactId>
 <version>1.3.6</version>
 <configuration>
 <repository>${docker.image.prefix}/${project.artifactId}</r
epository>
 <buildArgs>
 <JAR_FILE>target/${project.build.finalName}.jar</JAR_FI
LE>
 </buildArgs>
 </configuration>
 </plugin>
 </plugins>
</build>
```

The configuration specifies 3 things:

- The repository with the image name, which will end up here as springio/hello-spring-boot-docker
- The name of the jar file, exposing the Maven configuration as a build argument for docker.
- Optionally, the image tag, which ends up as latest if not specified. It can be set to the artifact id if desired.

You can build a tagged docker image using the command line like this:

```
$ ./mvnw install dockerfile:build
```

You do NOT have to register with docker or publish anything to run a docker image. You still have a locally tagged image, and you can run it like this:

```
$ docker run -p 8080:8080 -t springio/hello-spring-boot-
docker
```

The application is then available on http://localhost:8080 and it says “Hello Docker World”.