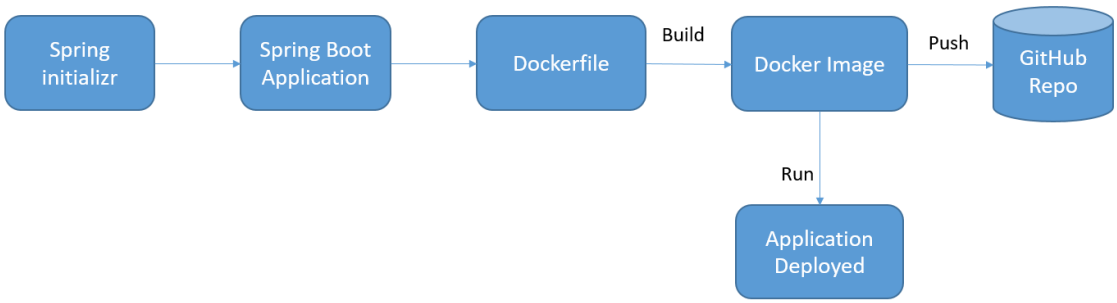
Docker_Project

Requirement：An organization XYZ Private Limited has recently transformed their IT application from Monolithic to Microservice Architecture. Now they have been struggling with deployment in such a complex infrastructure and inconsistency across the system. The organization has hired you to help them with simplifying their deployment process by containerizing their applications. They are using spring boot to develop their microservices. So we need to deploy our application in a container.

Here is the step wise step procedure to implement the solution.



Step 1 : Create and download a Spring Boot application with Spring initializr.

Step 2: Customize the application as per your needs and make a complete application. This application we are going to containerize.

Step 3: Write a Dockerfile for containerizing our application.

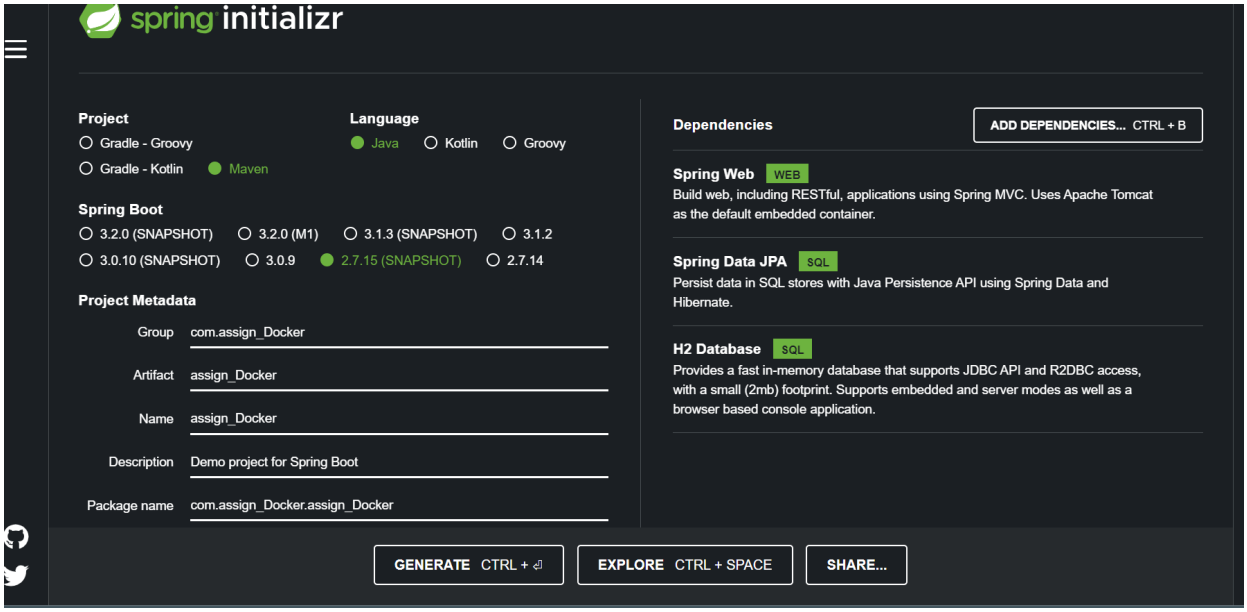
Step 4: Create a Docker image from the Dockerfile.

Step 5 : Push the Docker image to Dockerhub.

Step 6 : make your containerized application in the runnable state.

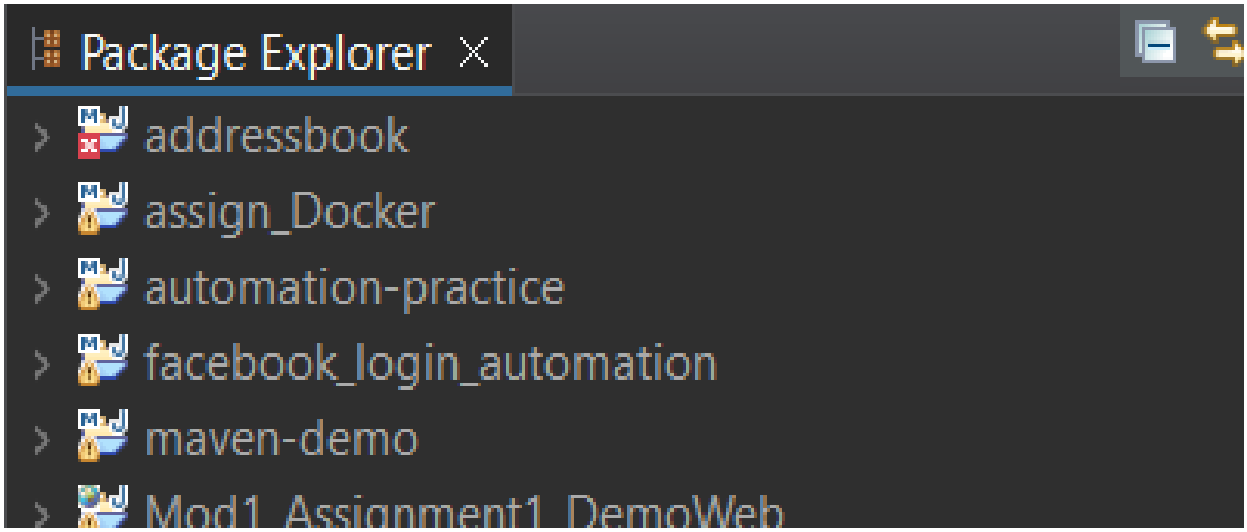
Step 1: Create and download a Spring Boot application with Spring initializr.

[spring initializer]



Click on Generate CTRL. Project zip file will be downloaded. Extract the zip and import to eclipse(file —>import—->maven existing project)

Import the application(assign_Docker) into Eclipse.



Step 2 : Add an index.html to the main/resources/static folder. (Customize the application)

App.java | index.html | App.java | index.html | module-info.java | StudentService.java | index.html

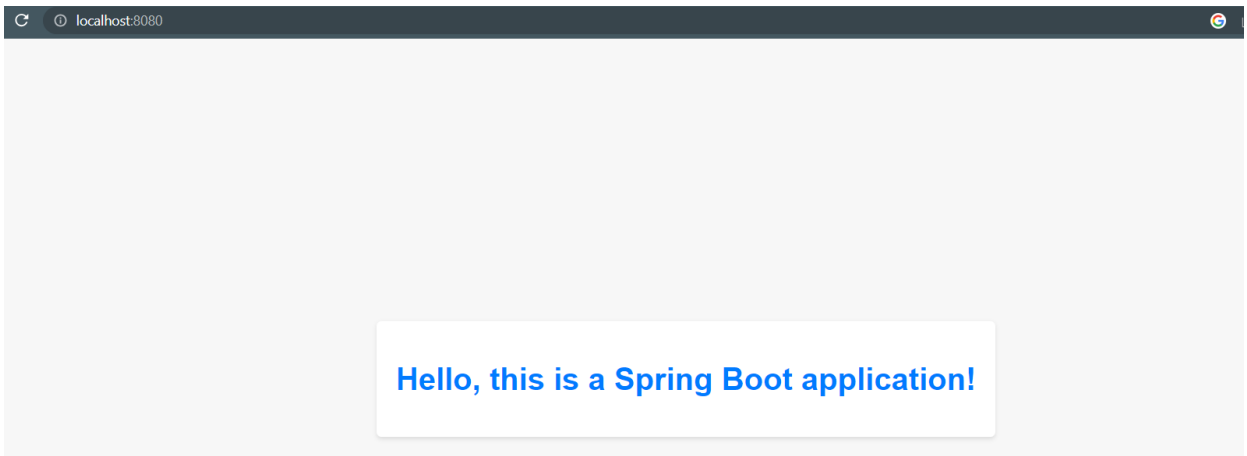
```html
<html>
<head>
    <title>Spring Boot App</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f7f7f7;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }
        .container {
            text-align: center;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
            background-color: #fff;
        }
        h1 {
            color: #007bff;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Hello, this is a Spring Boot application!</h1>
    </div>
</body>
</html>
```

Problems | Javadoc | Declaration | Console

Run the application.

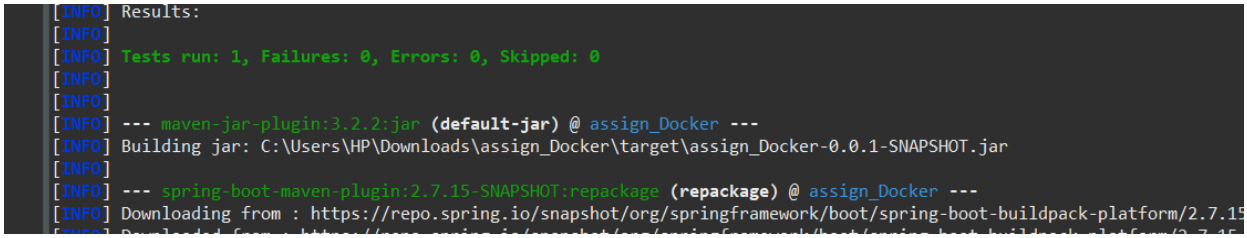
```
main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be perf
main] o.s.b.a.w.s.WelcomePageHandlerMapping    : Adding welcome page: class path resource [static/index.html]
main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
main] c.a.a.AssignDockerApplication            : Started AssignDockerApplication in 6.352 seconds (JVM running for 7.401)
```

Verify the application is running



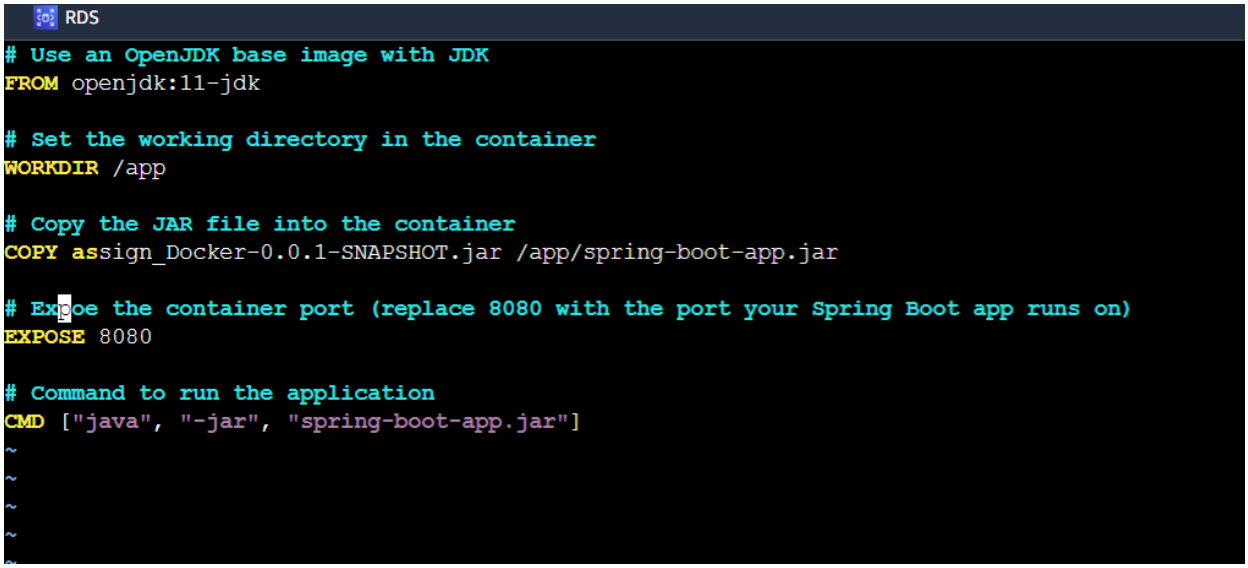
So the application works fine in the localhost next we are going to containerize the application.

In eclipse run as maven build



It will create a jar file, which is used for the deployment of an application.

Step 3: Write a Dockerfile to containerize the application



Step 4 : Create a Docker image from the Dockerfile.

Run the build command to create a docker image from the docker file

sudo docker build -t spring-boot-app-image .

```
ubuntu@ip-172-31-10-201:~$ sudo docker build -t spring-boot-app-image .
Sending build context to Docker daemon  39.17MB
Step 1/5 : FROM openjdk:11-jdk
11-jdk: Pulling from library/openjdk
001c52e26ad5: Pull complete
d9d4b9b6e964: Pull complete
2068746827ec: Pull complete
9daef329d350: Pull complete
d85151f15b66: Pull complete
66223a710990: Pull complete
db38d58ec8ab: Pull complete
Digest: sha256:99bac5bf83633e3c7399aed725c8415e7b569b54e03e4599e580fc9cdb7c21ab
Status: Downloaded newer image for openjdk:11-jdk
```

```
db38d58ec8ab: Pull complete
Digest: sha256:99bac5bf83633e3c7399aed725c8415e7b569b54e03e4599e580fc9cdb7c21ab
Status: Downloaded newer image for openjdk:11-jdk
 ---> 47a932d998b7
Step 2/5 : WORKDIR /app
 ---> Running in fc12e27da902
Removing intermediate container fc12e27da902
 ---> a71dfee38c9b
Step 3/5 : COPY assign_Docker-0.0.1-SNAPSHOT.jar /app/spring-boot-app.jar
 ---> 8dec4f919642
Step 4/5 : EXPOSE 8080
 ---> Running in 1768098d2e22
Removing intermediate container 1768098d2e22
 ---> a6c554d1ea96
Step 5/5 : CMD ["java", "-jar", "spring-boot-app.jar"]
 ---> Running in 0460aa719af3
Removing intermediate container 0460aa719af3
 ---> a2fff3f0a62f
Successfully built a2fff3f0a62f
Successfully tagged spring-boot-app-image:latest
ubuntu@ip-172-31-10-201:~$
```

Check the image is created or not

sudo docker image

```
ubuntu@ip-172-31-10-201:~$ sudo su
root@ip-172-31-10-201:/home/ubuntu# docker images
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
spring-boot-app-image   latest    a2fff3f0a62f   3 minutes ago   693MB
openjdk                 11-jdk    47a932d998b7   11 months ago   654MB
root@ip-172-31-10-201:/home/ubuntu#
```

Next check the running containers using,
docker ps

Presently no containers are running so create a container using run command

Step 7 : Push the docker image to Docker Hub

First tag the image with username of the github account

docker tag spring-boot-app-image sarus23/spring-boot-app-repo:latest

docker login //give the username and password of docker hub

docker push sarus23/spring-boot-app-repo:latest

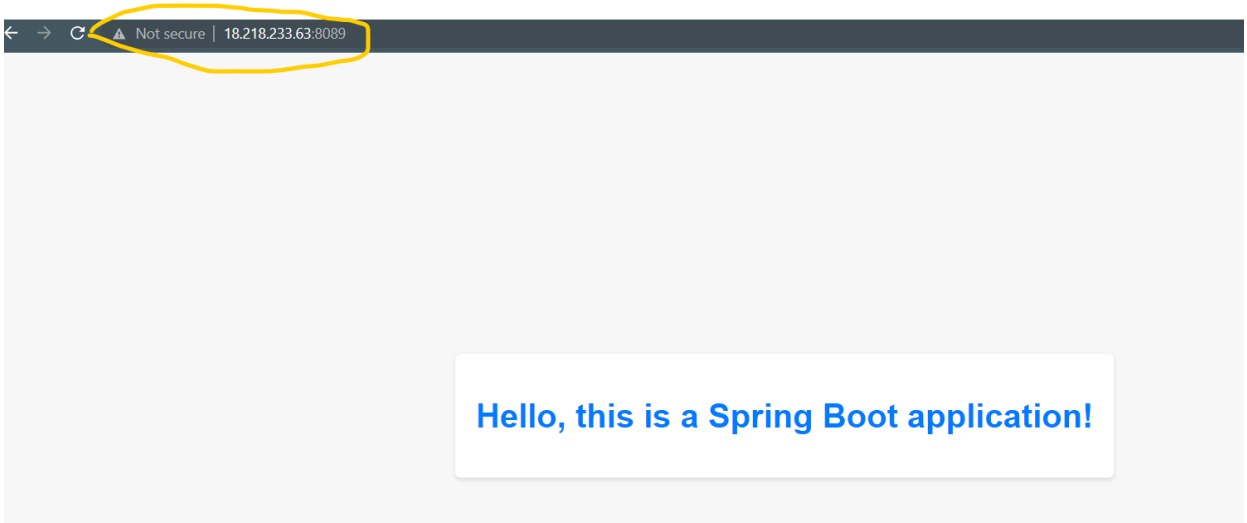
Step 8 : Run on application as container in detached mode and on system port 8089.

docker run -d -p 8089:8080 spring-boot-app-image

Then check if the container is created or not. docker ps will give you the created container id.

```
root@ip-172-31-10-201:/home/ubuntu# docker run -d -p 8089:8080 spring-boot-app-image
6c0d3190b724a89f317d745bc00d000f59fc8469df67167b3aa16b9a249ed91d
root@ip-172-31-10-201:/home/ubuntu#
```

Then go to the browser and check that application is available in the mapped system port(8089)



Application successfully deployed in the container