Running Java Applications using Docker

Steps
Open Eclipse and install the plugin Eclipse Docker Tooling, Restart the Eclipse

Spring Boot App (Maven Build tool), Generate Jar & Running a Class File (Using OpenJDK Image)

Step 1: Create a Spring Boot Maven project in Eclipse (Name as auto-start-app)
Step 2: Create a Class and replace a simple Java Code

```
package com.example.demo.example;
public class Test {
	public static void main(String[] args) {
		for (int i = 0; i < 100; i++) {
			System.out.println("Hello World Ping " + i);
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

Step 3: Generate a jar from Java App (Generated jar found in target folder and named as auto-start-app-0.0.1-SNAPSHOT.jar.original) (Can be renamed in POM.XML or Rename manually)
The jar name will be used in Dockerfile
Step 4: Create a Dockerfile inside project directory

```
# Updated as of Nov 21, 2020
# Install FROM JDK IMAGE
FROM openjdk:15

#Author of the Docker File
# MAINTAINER Nagaraj Note: MAINTAINER has been deprecated for LABEL,
# LABEL is a key value pair
LABEL "Maintainer"="Nagaraj"

# ADD a directory called auto start app inside the JDK IMAGE where you will be moving all of these files under this
# DIRECTORY to
ADD . /usr/local/auto-start-app

# AFTER YOU HAVE MOVED ALL THE FILES GO AHEAD CD into the directory
RUN cd /usr/local/auto-start-app

#THE CMD COMMAND tells docker the command to run when the container is started up from the image. In this case we are
# executing the java program as is to print Hello World.
```

```
CMD ["java", "-cp", "/usr/local/auto-start-app/target/auto-start-app-0.0.1-SNAPSHOT.jar.original", "com.example.demo.example.Test"]
```

Step 5:

Running as Container. Goto project path in terminal

```
docker build -t auto-start-app .
docker run -itd --name auto-start-app -p 8080:8080 auto-start-app:latest
```

Where first auto-start-app is container name and second name is image name

```
docker logs <container-id>
```

Spring Boot App (Maven Build tool) & RESTFul Service (Using OpenJDK Image)

Step 1: Create a Spring Boot Maven project in Eclipse (Name as rest-call-app)
Step 2: Create a Class and replace a simple Java Code

```
package com.example.demo.example;

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

@RestController
@RequestMapping("/test")
public class TestRESTService {

	@GetMapping("")
	public String test() {
		return "REST Call Ping";
	}
}
```

Step 3: Generate a jar from Java App (Generated jar found in target folder and named as rest-call-app-0.0.1-SNAPSHOT.jar.original) (Can be renamed in POM.XML or Rename manually)
The jar name will be used in Dockerfile
Step 4: Create a Dockerfile inside project directory

```
# Updated as of Nov 21, 2020
# Install FROM JDK IMAGE
FROM openjdk:15

#Author of the Docker File
# MAINTAINER Nagaraj Note: MAINTAINER has been deprecated for LABEL,
# LABEL is a key value pair
LABEL "Maintainer"="Nagaraj"

# ADD a directory called auto start app inside the JDK IMAGE where you will be moving all of these files under this
# DIRECTORY to
ADD . /usr/local/rest-call-app
```

```
# AFTER YOU HAVE MOVED ALL THE FILES GO AHEAD CD into the directory
RUN cd /usr/local/rest-call-app

#THE CMD COMMAND tells docker the command to run when the container
is started up from the image. In this case we are
# executing the java program as is to print Hello World.
CMD ["java", "-jar", "/usr/local/rest-call-app/target/rest-call-
app-0.0.1-SNAPSHOT.jar"]
```

Running as Container

```
docker build -t rest-call-app .
docker run -itd --name rest-call-app -p 8081:8080 rest-call-
app:latest
```

#Note: 8081/8080 Left port number is Exposing port number(Can be used to send the request), and the right port number is Container port number (Which is actually running in docker)

Rest call on browser

http://localhost:8081/test

Maven App (Maven Build tool) & Running a Class File (Maven assembly)

Step 1: Create a Maven project in Eclipse (Select maven-archetype-quickstart archetype) called docker-git-hello-world (Package name: org.pictolearn.docker and same Group ID and Artifact ID)
Step 2: Delete contents inside test package
Step 3: Select the jdk version from build path
Step 4: Delete contents inside test package
Step 5: We are going to use maven assembly plugin which allows you to run a jar file as an executable. We are going to run our project in a Container.
Replace the contents of pom.xml file with the below code

```
<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 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>docker-git-hello-world</groupId>
     <artifactId>docker-git-hello-world</artifactId>
     <version>0.0.1-SNAPSHOT</version>
     <packaging>jar</packaging>

     <name>docker-git-hello-world</name>
     <url>http://maven.apache.org</url>

     <properties>
          <project.build.sourceEncoding>UTF-8</
project.build.sourceEncoding>
     </properties>

     <build>
```

```
          <plugins>
               <plugin>
                    <artifactId>maven-assembly-plugin</artifactId>
                    <version>2.6</version>
                    <configuration>
                         <descriptorRefs>
                              <descriptorRef>jar-with-
dependencies</descriptorRef>
                         </descriptorRefs>
                    </configuration>
               </plugin>
          </plugins>
     </build>
</project>

Step 6: Create a Dockerfile inside project directory
# Updated as of Aug 16, 2017
# Install FROM UBUNTU IMAGE
FROM ubuntu:16.04

#Author of the Docker File
# MAINTAINER Pictolearn Note: MAINTAINER has been deprecated for
LABEL,
# LABEL is a key value pair
LABEL "Maintainer"="Pictolearn"

# RUN COMMAND BASICALLY runs the command in the terminal and creates
an image.
# Install all the updates for UBUNTU
RUN apt-get update && apt-get install -y python-software-properties
software-properties-common

# Install all the updates for UBUNTU
RUN apt-get install -y iputils-ping

# Adds the repository where JDK 8 can be obtained for UBUNTU
RUN add-apt-repository ppa:webupd8team/java

# INSTALL THE VI EDITOR AND MYSQL-CLIENT
RUN apt-get install -y vim
RUN apt-get install -y mysql-client

# NOTE and WARNING: ORACLE JDK is no longer licensed. Please install
default jdk or OPEN JDK.
# The initial set up was to get this working with JDK 7 but when the
licensing terms for oracle changing we will install the default JDK
# INSTALL ORACLE JDK 8 BY ACCEPTING YES
# RUN echo "oracle-java8-installer shared/accepted-oracle-license-
v1-1 boolean true" | debconf-set-selections
#INSTALL ALL the updates again and install MAVEN and JDK 8
# RUN apt-get update && apt-get install -y oracle-java8-installer
maven
RUN apt-get update && apt-get install -y default-jdk maven
```

```
# ADD a directory called docker-git-hello-world inside the UBUNTU
IMAGE where you will be moving all of these files under this
# DIRECTORY to
ADD . /usr/local/docker-git-hello-world

# AFTER YOU HAVE MOVED ALL THE FILES GO AHEAD CD into the directory
and run mvn assembly.
# Maven assembly will package the project into a JAR FILE which can
be executed
RUN cd /usr/local/docker-git-hello-world && mvn assembly:assembly

#THE CMD COMMAND tells docker the command to run when the container
is started up from the image. In this case we are
# executing the java program as is to print Hello World.
CMD ["java", "-cp", "/usr/local/docker-git-hello-world/target/
docker-git-hello-world-0.0.1-SNAPSHOT-jar-with-dependencies.jar",
"org.pictolearn.docker.HelloWorldPing"]
```

Step 7:

Running as Container. Goto project path in terminal

```
docker build -t custom-app .
docker run -itd --name custom-app -p 8080:8080 custom-app:latest
docker logs <container-id>
```

Spring Boot App (Maven Build tool), Generate Jar & RESTFul Service (Using Ubuntu Image)

Step 1: Create a Spring Boot Maven project in Eclipse (Name as spring-boot-app)
Step 2: Create a Class and replace a simple Java Code

```
package com.example.demo.example;

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

@RestController
@RequestMapping("/test")
public class TestRESTService {

	@GetMapping("")
	public String test() {
		return "REST Call Ping";
	}
}
```

Step 3: Now we are going to Create a Dockerfile inside project directory.
Docker file -> 1) Pull Ubuntu image first from Docker Hub then installing updates for ubuntu
2) Install OpenJDK 11 (Not able to install OpenJDK 15)
3) Install Maven build tool in Ubuntu OS
4) Copying the project to Internal Docker path

5) Run Cd to project and Generate Jar for the micro service
6) Running the generated jar using java command

```
# Updated as of Nov 21, 2020
FROM ubuntu:20.04

#Author of the Docker File
# MAINTAINER Nagaraj Note: MAINTAINER has been deprecated for LABEL,
# LABEL is a key value pair
LABEL "Maintainer"="Nagaraj"

# install packages
RUN apt-get update && \
    apt-get install -y curl \
    wget

RUN apt-get install -y openjdk-11-jdk
RUN apt install -y maven

# ADD a directory called auto start app inside the JDK IMAGE where
you will be moving all of these files under this
# DIRECTORY to
ADD . /usr/local/spring-boot-app

# AFTER YOU HAVE MOVED ALL THE FILES GO AHEAD CD into the directory
RUN cd /usr/local/spring-boot-app && mvn install

#THE CMD COMMAND tells docker the command to run when the container
is started up from the image. In this case we are
# executing the java program as is to print Hello World.
CMD ["java", "-jar", "/usr/local/spring-boot-app/target/spring-boot-
app-0.0.1-SNAPSHOT.jar”]
```

Step 4: Now we are going to Create a image and running it as Container. Goto project path in terminal

```
docker build -t spring-boot-app .
docker run -itd --name custom-app -p 8081:8080 spring-boot-
app:latest
docker logs <container-id>
```

Rest call on browser

http://localhost:8082/test

View containers

```
docker ps
CONTAINER ID        IMAGE                       COMMAND
CREATED             STATUS              PORTS
NAMES
0fe7b23527b9        spring-boot-app1:latest   "java -jar /usr/loca…"
4 hours ago         Up 4 hours          0.0.0.0:5555->8080/tcp
boot-cont-1

docker exec -it 0f /bin/bash
root@0fe7b23527b9:/# ls
```

```
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt
opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@0fe7b23527b9:/# cd /usr/local/spring-boot-app
root@0fe7b23527b9:/usr/local/spring-boot-app# ls
Dockerfile  HELP.md  mvnw  mvnw.cmd  pom.xml  src  target
root@0fe7b23527b9:/usr/local/spring-boot-app# ls target/
classes  generated-sources  generated-test-sources  maven-archiver
maven-status  spring-boot-app-0.0.1-SNAPSHOT.jar  spring-boot-
app-0.0.1-SNAPSHOT.jar.original  surefire-reports  test-classes
```