In a traditional day to day usage of Docker we build the images from our existing maven application, push them to a docker image and run them. What if we have the facility to create a docker image with out code from maven build and we simply need to run the image to test our application. Spotify maven plug-in provides us the way to do this. In this article we will develop a sample application using maven and Spotify plug-in and finally start our docker image with the application that we built.

1. Generate a Sample web application using the maven archetype as,

mvn archetype:generate -DgroupId=com.jags.dockerapp -DartifactId=dockerapp -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

2. Once the src code is generated along with the pom.xml file use the below pom configuration (Copy the below code to the pom.xml file)

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.jags.dockerapp</groupId>
  <artifactId>dockerapp</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>dockerapp</name>
  <url>http://maven.apache.org</url>

  <dependencies>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
  </dependencies>

<build>
  <plugins>
   <plugin>

  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
<version>2.3.2</version>
  <configuration>
     <archive>
        <manifest>
           <mainClass>com.jags.dockerapp.App</mainClass>
        </manifest>
     </archive>
   </configuration>
  </plugin>

   <plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.4.10</version>
  <executions>
```

```
      <execution>
        <phase>package</phase>
        <goals>
          <goal>build</goal>
        </goals>
      </execution>
    </executions>

    <configuration>
      <imageName>jags/${project.artifactId}</imageName>
      <baseImage>docker.io/java</baseImage>
      <maintainer>jagadesh manchala(jagadesh.manchala@gmail.com)</maintainer>
      <cmd>java -jar /opt/${project.build.finalName}.jar</cmd>

      <!--
      <entrypoint>
        ["java", "-jar", "/opt/${project.build.finalName}.jar"]
      </entrypoint>
      -->

      <serverid>docker-hub</serverid>
      <registryurl>https://index.docker.io/v1/</registryurl>

      <resources>
        <resource>
          <targetPath>/opt/</targetPath>
          <directory>${project.build.directory}</directory>
          <excludes>
            <exclude>target/**/*</exclude>
            <exclude>pom.xml</exclude>
            <exclude>*.iml</exclude>
          </excludes>
          <include>${project.build.finalName}.jar</include>
        </resource>
      </resources>
    </configuration>

    </plugin>
  </plugins>
</build>

</project>
```

The above snippet is simple to understand. The most important thing is the configuration element. The elements defined inside the configuration element goes to the Dockerfile that we use to create the Docker image.

```
    <configuration>
      <imageName>jags/${project.artifactId}</imageName>
      <baseImage>docker.io/java</baseImage>
      <maintainer>jagadesh manchala(jagadesh.manchala@gmail.com)</maintainer>
      <cmd>java -jar /opt/${project.build.finalName}.jar</cmd>

      <resources>
        <resource>
          <targetPath>/opt/</targetPath>
          <directory>${project.build.directory}</directory>
```

```
            <excludes>
               <exclude>target/**/*</exclude>
               <exclude>pom.xml</exclude>
               <exclude>*.iml</exclude>
            </excludes>
            <include>${project.build.finalName}.jar</include>
         </resource>
      </resources>
   </configuration>
```

The ImageName sets the name of the image
baseImage is the image that we use to create our docker image
maintainer is the author of the image
cmd tells the command that needs to execute once the docker image is ran

The resources element tells the resources that we need to copy to the docker image. We are copying the generated jar file to the /opt location in the docker image.

3) Once the configuration is done, run the maven clean install

```
[puppet@root$:/work/dockerapp/dockerapp]$  mvn clean install
[INFO] Scanning for projects...
[WARNING]
******************

******************
 [INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ dockerapp ---
[INFO] Building jar: /work/dockerapp/dockerapp/target/dockerapp-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- docker-maven-plugin:0.4.10:build (default) @ dockerapp ---
[INFO]      Copying      /work/dockerapp/dockerapp/target/dockerapp-1.0-SNAPSHOT.jar      ->
/work/dockerapp/dockerapp/target/docker/opt/dockerapp-1.0-SNAPSHOT.jar
[INFO] Building image jags/dockerapp
Step 1 : FROM docker.io/java
 ---> 861e95c114d6
Step 2 : MAINTAINER jagadesh manchala(jagadesh.manchala@gmail.com)
 ---> Running in 8dc4b0a195ec
 ---> 2a35d5cc5191
Removing intermediate container 8dc4b0a195ec
Step 3 : ADD /opt/dockerapp-1.0-SNAPSHOT.jar /opt/
 ---> 9f6a63e0ce04
Removing intermediate container 8feb0bc43678
Step 4 : CMD java -jar /opt/dockerapp-1.0-SNAPSHOT.jar
 ---> Running in c90a9c77b2ff
 ---> 2823967dc477
Removing intermediate container c90a9c77b2ff
Successfully built 2823967dc477
[INFO] Built jags/dockerapp
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ dockerapp ---
[INFO]      Installing      /work/dockerapp/dockerapp/target/dockerapp-1.0-SNAPSHOT.jar      to
/root/.m2/repository/com/jags/dockerapp/dockerapp/1.0-SNAPSHOT/dockerapp-1.0-
SNAPSHOT.jar
[INFO]                Installing                /work/dockerapp/dockerapp/pom.xml                to
/root/.m2/repository/com/jags/dockerapp/dockerapp/1.0-SNAPSHOT/dockerapp-1.0-
SNAPSHOT.pom
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 27.980 s
[INFO] Finished at: 2017-02-06T08:32:58-05:00
[INFO] Final Memory: 28M/365M
[INFO] ------------------------------------------------------------------------
```

The above maven command not just build the code , makes a jar out of it it also builds our image from the configuration that we defined. If we check the output of the build we can see building of the docker image as

```
[INFO] Building image jags/dockerapp
Step 1 : FROM docker.io/java
 ---> 861e95c114d6
Step 2 : MAINTAINER jagadesh manchala(jagadesh.manchala@gmail.com)
 ---> Running in 8dc4b0a195ec
 ---> 2a35d5cc5191
Removing intermediate container 8dc4b0a195ec
Step 3 : ADD /opt/dockerapp-1.0-SNAPSHOT.jar /opt/
 ---> 9f6a63e0ce04
Removing intermediate container 8feb0bc43678
Step 4 : CMD java -jar /opt/dockerapp-1.0-SNAPSHOT.jar
 ---> Running in c90a9c77b2ff
 ---> 2823967dc477
Removing intermediate container c90a9c77b2ff
Successfully built 2823967dc477
```

NOTE – Make sure the docker daemon on the local host is running. The Spotify plug-in will try to connect to docker hub using the docker daemon to download our base image. If the docker daemon is not running the build fails.

4. Now in the target directory created after the build we can see a docker directory which contains the Dockerfile with contents as,

```
[puppet@root$:/work/dockerapp/dockerapp/target/docker]$  cat $PWD/Dockerfile
FROM docker.io/java
MAINTAINER jagadesh manchala(jagadesh.manchala@gmail.com)
ADD /opt/dockerapp-1.0-SNAPSHOT.jar /opt/
CMD java -jar /opt/dockerapp-1.0-SNAPSHOT.jar
```

5. Run the docker image as ,

```
docker run --rm jags/dockerapp
Hello World
```

This is how we can use maven along with docker to deploy our code to the docker images and run them. More to come. Happy learning