How to automate Docker container deployment via Maven

Ravindu Nirmal Fernando

.

Follow

Published in

We've moved to freeCodeCamp.org/news

.

4 min read

.

Apr 13, 2019

--

2

Listen

Share

Background Image Courtesy — https://cdn.pixabay.com/ | Created Via https://www.canva.com/

This article is intended for people who are using Maven as a build and dependency management tool for JAVA applications. It will show you how to integrate docker container build, tag and push workflows into their existing Maven build management ecosystem.

Having the ability to build, tag and push your application as a container right off from the Maven lifecycle commands itself is a pretty cool thing to have. It just makes things easy and quick if you are trying to bring in the power of containers to deploy your applications and all-ready using Maven for dependency management.

If we take a look at existing solutions for integrating docker container deployment into Maven, there are several ones out there, like spotify maven docker plugin, fabric8io docker maven plugin etc. But all these solutions bring in unwanted complexity, additional learning curve and too much change into your existing application code. Yet there is a simpler and easy way to achieve this without the use of any third-party plugin.

If you note Maven's Ant plugin, it allows us to run external commands. So by using the Ant plugin, we have the capability to run docker build, tag, push or just any command as you wish. The only thing that we have to do is to provide a proper Dockerfile for building the Docker image for your application and necessary set of commands and Maven configurations into the pom.xml file.

> For explaining the steps involved in this process, I will use a sample JAVA application. It contains all the code samples used in the following steps. You can clone it from here.

Step 1 | Create the Dockerfile

> Dockerfile should be stored within the path ***src/main/docker/Dockerfile*** of your JAVA application.

```
# Pull base image
FROM tomcat:8.0.30-jre7

# Maintainer
MAINTAINER "ravindu@emojot.com"# Set Environment properties
ENV JAVA_OPTS=-Denvironment=production

# Copy war file to tomcat webapps folder
COPY /dockermavensample.war /usr/local/tomcat/webapps/
```

Step 2 | Update the pom.xml to copy all Docker-related resources into the target directory

> We can use maven-resource-plugin to copy resources.

```
<plugin>
   <artifactId>maven-resources-plugin</artifactId>
   <executions>
       <execution>
           <id>copy-resources</id>
           <phase>validate</phase>
           <goals>
               <goal>copy-resources</goal>
           </goals>
           <configuration>
               <outputDirectory>${basedir}/target</outputDirectory>
               <resources>
                   <resource>
                       <directory>src/main/docker</directory>
                       <filtering>true</filtering>
                   </resource>
               </resources>
           </configuration>
       </execution>
   </executions>
</plugin>
```

Step 3 | Update the pom.xml to allow build and tag the Docker image via Maven's Ant plugin

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-antrun-plugin</artifactId>
   <version>1.6</version>
   <executions>
       <execution>
           <id>prepare-package</id>
           <phase>package</phase>
           <inherited>false</inherited>
           <configuration>
               <target>
                   <exec executable="docker">
                       <arg value="build"/>
                       <arg value="-t"/>
                       <arg value="dockermavensample:${project.version}"/>
                       <arg value="target"/>
                   </exec>
               </target>
           </configuration>
           <goals>
               <goal>run</goal>
           </goals>
       </execution>
   </executions>
</plugin>
```

Maven's Ant plugin will execute the docker command in the package phase of Maven lifecycle in the following order, which will build the docker image from the Dockerfile which was copied into the target folder in step 2.

```
docker build -t dockermavensample:1.0.0 target
```

Step 4 | Update the pom.xml file to allow pushing Docker Image to remote Docker repository

> Ideally for production, you would have to push your Docker images into your own private Docker registry or use a third party Docker image repository which allows storing private Docker images so that others cannot pull your Docker images directly.

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-antrun-plugin</artifactId>
   <version>1.6</version>
   <executions>
       <execution>
           <phase>install</phase>
           <inherited>false</inherited>
```

```
            <configuration>
                <target>
                    <exec executable="docker">
                        <arg value="tag"/>
                        <arg value="dockermavensample:${project.version}"/>
                        <arg value="dockermavensample:latest"/>
                    </exec>
                    <exec executable="docker">
                        <arg value="push"/>
                        <arg value="dockermavensample:latest"/>
                    </exec>
                </target>
            </configuration>
            <goals>
                <goal>run</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

In addition to the above steps, you may want to have control over how you are running these docker related commands in your Maven lifecycle. For that, you can use Maven profiles to logically divide above plugin definitions. Then execute those only when the profile related to that action is invoked.

Take a look at following sample profiles:

```
<profile>
   <id>dockerBuild</id>

   <build>
       <plugins>
           <plugin>
               <artifactId>maven-resources-plugin</artifactId>
               <executions>
                   <execution>
                       <id>copy-resources</id>
                       <phase>validate</phase>
                       <goals>
                           <goal>copy-resources</goal>
                       </goals>
                       <configuration>
                           <outputDirectory>${basedir}/target</outputDirectory>
                           <resources>
                               <resource>
                                   <directory>src/main/docker</directory>
                                   <filtering>true</filtering>
                               </resource>
                           </resources>
                       </configuration>
                   </execution>
               </executions>
           </plugin>

           <plugin>
               <groupId>org.apache.maven.plugins</groupId>
               <artifactId>maven-antrun-plugin</artifactId>
               <version>1.6</version>
               <executions>
                   <execution>
                       <id>prepare-package</id>
                       <phase>package</phase>
                       <inherited>false</inherited>
                       <configuration>
                           <target>
                               <exec executable="docker">
                                   <arg value="build"/>
                                   <arg value="-t"/>
                                   <arg value="dockermavensample:${project.version}"/>
                                   <arg value="target"/>
                               </exec>
                           </target>
```

```
                    </configuration>
                    <goals>
                        <goal>run</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>    <activation>
    <activeByDefault>true</activeByDefault>
</activation>
</profile><!-- docker Image push and release profile -->
<profile>
    <id>dockerRelease</id>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-antrun-plugin</artifactId>
                <version>1.6</version>
                <executions>
                    <execution>
                        <phase>install</phase>
                        <inherited>false</inherited>
                        <configuration>
                            <target>
                                <exec executable="docker">
                                    <arg value="tag"/>
                                    <arg value="dockermavensample:${project.version}"/>
                                    <arg value="dockermavensample:latest"/>
                                </exec>
                                <exec executable="docker">
                                    <arg value="push"/>
                                    <arg value="dockermavensample:latest"/>
                                </exec>
                            </target>
                        </configuration>
                        <goals>
                            <goal>run</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</profile>
```

After completing the above steps, just run

```
mvn clean install -P dockerBuild,dockerRelease
```

Now your JAVA application is packaged as a container and pushed into a remote docker repository as well. You can test whether the image you created is working by running following commands,

After running dockerBuild profile, the Docker image should be available locally

Run dockermavensample:1.0.0 Docker Image

Apache Tomcat Home Page

Kaboom! :)

As you can see we can use already available Maven features and plugins to create a well-structured build pipeline for deploying our applications as containers.

Sample Project:

rav94/dockermavensample

Demo Project for showcasing Automating Container Deployment via Maven - rav94/dockermavensample

github.com

Thanks for reading!