Packaging a Java Application in a Docker Image with Maven

By Ivan Krizsan | February 14, 2017 0 Comment

Contents [show]

In this article I am going to show how to build a Docker image containing a Java application using Maven. As an example, I will use a Spring Boot web application, which will be packaged into a Docker image. The application packaged does not need to be a Java application, but can be anything for which a Docker image can be created. I have, for instance, used the process described to make it possible to build regular Docker images on a Jenkins build server.

The Example Spring Boot Application

The Spring Boot web application I will use as an example Java application is created as follows:

- Use this Spring Initializr link to generate the bare-bones Spring Boot web project with Thymeleaf templating support.
- Open the Spring Boot project in your favourite IDE.
- In src/main/java in the package se.ivankrizsan.spring, create a package named “controllers”.
- In the new package, implement the class HelloController as this:

```
1  package se.ivankrizsan.spring.controllers;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.ui.Model;
5  import org.springframework.web.bind.annotation.RequestMapping;
6  import org.springframework.web.bind.annotation.RequestParam;
7
8  import java.util.Date;
9
10 /**
11  * Simple Spring MVC controller that says hello.
```

```
12  */
13 @Controller
14 public class HelloController {
15     /* Constant(s): */
16     protected static final String HELLO_VIEW_NAME = "hello";
17     protected static final String GREETING_PLACEHOLDER = "greeting";
18
19     @RequestMapping("/hello")
20     public String hello(
21         @RequestParam(value="name", required=false, defaultValue="Anonymous")
22         final String inName,
23         final Model inModel) {
24         final StringBuilder theMessageBuilder = new StringBuilder();
25         theMessageBuilder
26             .append("Hello ")
27             .append(inName)
28             .append(", the time is now ")
29             .append(new Date().toString())
30             .append(".");
31         inModel.addAttribute(GREETING_PLACEHOLDER, theMessageBuilder.toString());
32         return HELLO_VIEW_NAME;
33     }
34
35 }
```

- In src/main/resources/templates, create a file named hello.html with the following contents:

```
1  <!DOCTYPE HTML>
2  <html xmlns:th="http://www.thymeleaf.org">
3      <head>
4          <title>Greetings!</title>
5          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
6      </head>
7      <body>
8          <p th:text="${greeting}"/>
9      </body>
10 </html>
```

The example application can now be started by running the *SpringbootDockerimageApplication* class. To test the application, enter the http://localhost:8080/hello?name=Ivan URL in a local browser. There should be a greeting in the browser window on the following format:

```
1 Hello Ivan, the time is now Sat Feb 11 22:57:33 CET 2017.
```

Prepare for Docker Image Creation

Setting up to create a Docker image for a Java application consists of two steps; the first step is much like for any Docker image in that a Docker file is created and the static contents of the Docker image is gathered/created. The second part consists of adding a Maven profile to the project's pom.xml file.

The Docker file and any additional static contents of the Docker image is created in the source of the project. In this example it is in the src/main/resources/docker directory. The location of this directory can be changed by modifying a property, as we will see later.

- Create a directory named “docker” in the src/main/resources directory.
- In the src/main/resources/docker directory, create a directory named “application”.
- In the src/main/resources/docker/application directory, create two directories named “bin” and “lib”.
- In the src/main/resources/docker/application/bin directory, create a file named “start-app.sh” with the following contents:

```
#! /bin/sh
java -jar ${JAR_PATH}
```

- In the src/main/resources/docker directory, create a file name “Dockerfile” with this contents:

```
# Docker image that contains a Spring Boot application.
#
FROM anapsix/alpine-java:8_jdk_unlimited

# Absolute path to the JAR file to be launched when a Docker container is started.
ENV JAR_PATH=/application/lib/springboot-webapp.jar

# Create directory to hold the application and all its contents in the Docker image.
RUN mkdir /application
# Copy all the static contents to be included in the Docker image.
COPY ./application/ /application/
# Make all scripts in the bin directory executable. Includes start-script.
RUN chmod +x /application/bin/*.sh

# Web port.
EXPOSE 8080

CMD [ "/application/bin/start-app.sh" ]
```

Modify the Dockerfile accordingly if, for instance, you want to use another base image.

Maven Profile

To create the Docker image containing the Spring Boot application, I’ll use the following Maven plug-ins:

- maven-resources-plugin

  Copies files to a directory in which the Docker image will be built.

  The files needed to build the Docker image are copied to a directory in the target directory of the project.

age build directory.

- Spotify's docker-maven-plugin

  Removes the Docker image produced by the project. Needed since, at the time of writing, the Maven plug-in used to build the Docker image will not overwrite an existing Docker image.
- Spotify's dockerfile-maven-plugin

  Builds the Docker image.

In the project's Maven pom.xml file, I use a Maven profile to contain what is needed to build the Docker image. I have tried to rely on properties in the profile for configuration of the Docker image build in order to make it easier to adapt the profile to different Java applications.

The complete Docker image building Maven profile looks like this:

```
1       <profiles>
2           <!--
3               This profile builds a Docker image with the Spring Boot application.
4               Before being able to build a Docker image using Maven, the environment
5               variable DOCKER_HOME need to be set to the endpoint of the local
6               Docker API.
7               Example *nix: export set DOCKER_HOME=http://localhost:2375
8               The Docker image is built using the following command:
9               mvn -Pdockerimage package
10
11              If a Docker image with the image name and tag (project version) already
12              exists then one of the following may happen depending on the
13              environment in which the build is run:
14              - The existing image is given the image name and tag <none>.
15              - No new Docker image is generated, but the existing image is retained.
16              The suggested approach is to first delete any existing Docker image
17              using the following Maven command before generating a new image:
18              mvn -Pdockerimage clean
19          -->
20          <profile>
21              <id>dockerimage</id>
22              <dependencies>
23                  <!--
24                      Here you declare dependencies to additional artifacts that
25                      are to be copied into the Docker image.
26                      No need to add a dependency to the Spring Boot application JAR
27                      file here.
28                  -->
29              </dependencies>
30              <properties>
31                  <!-- Name of Docker image that will be built. -->
32                  <docker.image.name>springboot-webapp</docker.image.name>
33                  <!--
34                      Directory that holds Docker file and static content
35                      necessary to build the Docker image.
36                  -->
37                  <docker.image.src.root>src/main/resources/docker</docker.image.src.
38                  <!--
39                      Directory to which the Docker image artifacts and the Docker
```

```
43                  <docker.build.directory>${project.build.directory}/docker</docker.b
44              </properties>
45              <build>
46                  <plugins>
47                      <!--
48                          Copy the directory containing static content to build direc
49                      -->
50                      <plugin>
51                          <artifactId>maven-resources-plugin</artifactId>
52                          <executions>
53                              <execution>
54                                  <id>copy-resources</id>
55                                  <phase>package</phase>
56                                  <goals>
57                                      <goal>copy-resources</goal>
58                                  </goals>
59                                  <configuration>
60                                      <outputDirectory>${docker.build.directory}</out
61                                      <resources>
62                                          <resource>
63                                              <directory>${docker.image.src.root}</di
64                                              <filtering>false</filtering>
65                                          </resource>
66                                      </resources>
67                                  </configuration>
68                              </execution>
69                          </executions>
70                      </plugin>
71                      <!--
72                          Copy the JAR file containing the Spring Boot application
73                          to the application/lib directory.
74                      -->
75                      <plugin>
76                          <groupId>org.apache.maven.plugins</groupId>
77                          <artifactId>maven-dependency-plugin</artifactId>
78                          <executions>
79                              <execution>
80                                  <id>copy</id>
81                                  <phase>package</phase>
82                                  <goals>
83                                      <goal>copy</goal>
84                                  </goals>
85                                  <configuration>
86                                      <artifactItems>
87                                          <artifactItem>
88                                              <!--
89                                                  Specify groupId, artifactId, versio
90                                                  artifact you want to package in the
91                                                  In the case of a Spring Boot applic
92                                                  the same as the project group id, a
93                                                  and version.
94                                              -->
95                                              <groupId>${project.groupId}</groupId>
96                                              <artifactId>${project.artifactId}</arti
97                                              <version>${project.version}</version>
98                                              <type>jar</type>
99                                              <overWrite>true</overWrite>
100                                             <<outputDirectory>${docker.build.direct
```

```
104                                         -->
105                                         <destFileName>springboot-webapp.jar</de
106                                     </artifactItem>
107                                     <!-- Add additional artifacts to be package
108
109                                 </artifactItems>
110                                 <outputDirectory>${docker.build.directory}</out
111                                 <overWriteReleases>true</overWriteReleases>
112                                 <overWriteSnapshots>true</overWriteSnapshots>
113                             </configuration>
114                         </execution>
115                     </executions>
116                 </plugin>
117
118                 <!--
119                     Remove any existing Docker image with the image name
120                     and image tag (project version) configured in the propertie
121                 -->
122                 <plugin>
123                     <groupId>com.spotify</groupId>
124                     <artifactId>docker-maven-plugin</artifactId>
125                     <version>0.4.13</version>
126                     <executions>
127                         <execution>
128                             <id>remove-image</id>
129                             <phase>clean</phase>
130                             <goals>
131                                 <goal>removeImage</goal>
132                             </goals>
133                             <configuration>
134                                 <imageName>${docker.image.name}</imageName>
135                                 <imageTags>
136                                     <imageTag>${project.version}</imageTag>
137                                 </imageTags>
138                                 <verbose>true</verbose>
139                             </configuration>
140                         </execution>
141                     </executions>
142                 </plugin>
143
144                 <!--
145                     Build the Docker image.
146                 -->
147                 <plugin>
148                     <groupId>com.spotify</groupId>
149                     <artifactId>dockerfile-maven-plugin</artifactId>
150                     <version>1.2.2</version>
151                     <executions>
152                         <execution>
153                             <id>default</id>
154                             <phase>package</phase>
155                             <goals>
156                                 <goal>build</goal>
157                             </goals>
158                         </execution>
159                     </executions>
160                     <configuration>
161                         <contextDirectory>${project.build.directory}/docker</co
```

```
165                             <forceCreation>true</forceCreation>
166                             <imageName>${docker.image.name}</imageName>
167                             <repository>${docker.image.name}</repository>
168                             <tag>${project.version}</tag>
169                             <forceTags>true</forceTags>
170                             <pullNewerImage>false</pullNewerImage>
171                             <imageTags>
172                                 <imageTag>${project.version}</imageTag>
173                             </imageTags>
174                             <dockerDirectory>${project.build.directory}/docker</doc
175                         </configuration>
176                     </plugin>
177                 </plugins>
178             </build>
179         </profile>
180     </profiles>
```

If we look at the different parts of the Maven profile, we can see:

- There is a <dependencies> element.
  Add any additional Maven dependencies to, for instance, third party libraries that you want to include in the Docker image in this element.
- In the <properties> element, there is a property named docker.image.name.
  This property contains the name of the Docker image that will be produced. I expect this property to be modified for each application. The Docker image tag will be the project's version.
- The next property in the <properties> element is docker.image.src.root.
  This property holds the relative path to the directory that contains the Dockerfile and any additional static contents that is to be included in the Docker image.
- The final property in the <properties> element is docker.build.directory.
  This property holds the path to the directory in which the Docker image will be built.
- In the <build> section of the profile, the first plug-in is the maven-resources-plugin.
  This plug-in is used to copy static contents from the directory pointed at by the docker.image.src.root property to the directory which location is specified by the docker.build.directory property. The entire directory structure is preserved.
- The second plug-in in the <build> section is the maven-dependency-plugin.
  Using this plug-in, the JAR file that contains the Spring Boot application built by the project is copied to the application/lib directory in the directory in which the Docker image will be built. Recall that the location of the application JAR file is also stored into the environment variable JAR_PATH in the Dockerfile.
  Note that a destination filename is specified for the application JAR, in order to have a fixed filename to refer to in the Dockerfile.
- The third plug-in in the <build> section is the docker-maven-plugin.
  Since the current version of the dockerfile-maven-plugin that we will discuss later does not overwrite existing Docker images, I have included this plug-in to be able to remove any existing Docker

This is the plug-in that is responsible for building the Docker image. The main reason that I chose this plug-in over the alternatives is that it allows me to create a Docker image from a Dockerfile – just like it works when I am creating Docker images by hand.

Create the Docker Image

If we now open a terminal window and go to the directory that contains the pom.xml file of the example Spring Boot web application, we can build the Docker image using this Maven command:

```
1 mvn -Pdockerimage package
```

After some time, the message BUILD SUCCESS should appear in the terminal and if you list the Docker images, you should see the new image listed:

```
1 $ docker images
2 REPOSITORY            TAG                 IMAGE ID            CREATED             SIZ
3 springboot-webapp     0.0.1-SNAPSHOT      16efcb5f8335        3 minutes ago       194
```

Use the Docker Image

To start a Docker container using the newly produced Docker image, I use the command:

```
1 docker run -p 8080:8080 springboot-webapp:0.0.1-SNAPSHOT
```

Opening the URL http://localhost:8080/hello?name=Ivan (replace localhost with the appropriate IP if you run Docker in a virtual machine) in a browser, I see the same type of greeting that I saw when I was running the Spring Boot web application outside of the Docker container.

Happy coding!

Category: Java Tags: docker, java, maven