Empaquetado de una aplicación Java en una imagen Docker con Maven

Por Ivan Krizsan | 14 de febrero de 2017 0 Comentario

Contenido [show]

En este artículo voy a mostrar cómo construir una imagen Docker que contiene una aplicación Java utilizando Maven. Como ejemplo, utilizaré una aplicación web de Spring Boot, que se empaquetará en una imagen de Docker. La aplicación empaquetada no necesita ser una aplicación Java, pero puede ser cualquier cosa para la que se pueda crear una imagen Docker. Por ejemplo, he utilizado el proceso descrito para hacer posible la construcción de imágenes regulares de Docker en un servidor de compilación de Jenkins .

El ejemplo de la aplicación Spring Boot

La aplicación web de Spring Boot que usaré como un ejemplo La aplicación Java se crea de la siguiente manera:

- Utilice este enlace Spring Initializr para generar el proyecto de web de Spring Boot desnudo con soporte de plantillas Thymeleaf .
- Abra el proyecto Spring Boot en su IDE favorito.
- En src / main / java en el paquete se.ivankrizsan.spring, cree un paquete denominado "controladores".
- En el nuevo paquete, implemente la clase HelloController como esto:

```
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  }
```

- En src / main / resources / templates, cree un archivo llamado hello.html con los siguientes contenidos:

```
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>
```

La aplicación de ejemplo ahora se puede iniciar ejecutando la clase *SpringbootDockerimageApplication* . Para probar la aplicación, ingrese la URL http: // localhost: 8080 / hello? Name = Ivan en un navegador local. Debe haber un saludo en la ventana del navegador en el siguiente formato:

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

Preparación para la creación de imágenes de Docker

La configuración para crear una imagen Docker para una aplicación Java consta de dos pasos; El primer paso es muy similar a cualquier imagen de Docker en que se crea un archivo Docker y se recopila / crea el contenido estático de la imagen de Docker. La segunda parte consiste en agregar un perfil Maven al archivo pom.xml del proyecto.

Docker Directory y sus contenidos

El archivo Docker y cualquier contenido estático adicional de la imagen Docker se crean en el origen del proyecto. En este ejemplo está en el directorio src / main / resources / docker. La ubicación de este directorio se puede cambiar modificando una propiedad, como veremos más adelante.

- Cree un directorio denominado "docker" en el directorio src / main / resources.
- En el directorio src / main / resources / docker, cree un directorio denominado "application".
- En el directorio src / main / resources / docker / application, cree dos directorios denominados "bin" y "lib".
- En el directorio src / main / resources / docker / application / bin, cree un archivo denominado "start-app.sh" con los siguientes contenidos:

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

- En el directorio src / main / resources / docker, cree un nombre de archivo "Dockerfile" con este contenido:

```
# 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" ]
```

Modifique el Dockerfile en consecuencia si, por ejemplo, desea utilizar otra imagen de base.

Perfil de Maven

Para crear la imagen de Docker que contiene la aplicación Spring Boot, utilizaré los siguientes plug-ins de Maven:

- Maven-resources-plugin
  Copia los archivos en un directorio en el que se construirá la imagen de Docker.
  Los archivos necesarios para construir la imagen de Docker se copian en un directorio del directorio de destino del proyecto.
- Maven-dependency-plugin
  Copia el archivo JAR que contiene la aplicación y cualquier artefacto Maven adicional en el directorio de construcción de imágenes de Docker.
-
  Docker -maven-plugin de Spotify Elimina la imagen de Docker producida por el proyecto. Es necesario desde entonces, en el momento de escribir, el plug-in Maven utilizado para construir la imagen Docker no sobrescribirá una imagen Docker existente.
- Dockerfile-maven-plugin de Spotify
  Genera la imagen de Docker.

En el archivo Maven pom.xml del proyecto, utilizo un perfil de Maven para contener lo que se necesita para construir la imagen de Docker. He intentado confiar en propiedades en el perfil para la configuración de la construcción de imágenes Docker para facilitar la adaptación del perfil a diferentes aplicaciones Java.

El perfil de Maven de imagen de Docker completo se parece a esto:

```
    <profiles>
        <!--
```

```
        This profile builds a Docker image with the Spring Boot application.
        Before being able to build a Docker image using Maven, the environment
        variable DOCKER_HOME need to be set to the endpoint of the local
        Docker API.
        Example *nix: export set DOCKER_HOME=http://localhost:2375
        The Docker image is built using the following command:
        mvn -Pdockerimage package

        If a Docker image with the image name and tag (project version) already
        exists then one of the following may happen depending on the
        environment in which the build is run:
        - The existing image is given the image name and tag <none>.
        - No new Docker image is generated, but the existing image is retained.
        The suggested approach is to first delete any existing Docker image
        using the following Maven command before generating a new image:
        mvn -Pdockerimage clean
    -->
    <profile>
        <id>dockerimage</id>
        <dependencies>
            <!--
                Here you declare dependencies to additional artifacts that
                are to be copied into the Docker image.
                No need to add a dependency to the Spring Boot application JAR
                file here.
            -->
        </dependencies>
        <properties>
            <!-- Name of Docker image that will be built. -->
            <docker.image.name>springboot-webapp</docker.image.name>
            <!--
                Directory that holds Docker file and static content
                necessary to build the Docker image.
            -->
            <docker.image.src.root>src/main/resources/docker</docker.image.src.root>
            <!--
                Directory to which the Docker image artifacts and the Docker
                file will be copied to and which will serve as the root directory
                when building the Docker image.
            -->
            <docker.build.directory>${project.build.directory}/docker</docker.build
        </properties>
        <build>
            <plugins>
                <!--
                    Copy the directory containing static content to build directory
                -->
                <plugin>
                    <artifactId>maven-resources-plugin</artifactId>
                    <executions>
                        <execution>
                            <id>copy-resources</id>
                            <phase>package</phase>
                            <goals>
                                <goal>copy-resources</goal>
                            </goals>
                            <configuration>
                                <outputDirectory>${docker.build.directory}</outputD
                                <resources>
                                    <resource>
                                        <directory>${docker.image.src.root}</direct
                                        <filtering>false</filtering>
                                    </resource>
                                </resources>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
                <!--
```

```
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, version an
90                                              artifact you want to package in the Doc
91                                              In the case of a Spring Boot applicatio
92                                              the same as the project group id, artif
93                                              and version.
94                                          -->
95                                          <groupId>${project.groupId}</groupId>
96                                          <artifactId>${project.artifactId}</artifact
97                                          <version>${project.version}</version>
98                                          <type>jar</type>
99                                          <overWrite>true</overWrite>
100                                         <<outputDirectory>${docker.build.directory}
101                                         <!--
102                                             Specify the destination name as to have
103                                             to refer to in the Docker file.
104                                         -->
105                                         <destFileName>springboot-webapp.jar</destFi
106                                     </artifactItem>
107                                     <!-- Add additional artifacts to be packaged in
108
109                                 </artifactItems>
110                                 <outputDirectory>${docker.build.directory}</outputD
111                                 <overWriteReleases>true</overWriteReleases>
112                                 <overWriteSnapshots>true</overWriteSnapshots>
113                             </configuration>
114                         </execution>
115                     </executions>
116                 </plugin>
117
118                 <!--
119                     Remove any existing Docker image with the image name
120                     and image tag (project version) configured in the properties.
121                 -->
122                 <plugin>
123                     <groupId>com.spotify</groupId>
124                     <artifactId>docker-maven-plugin</artifactId>
125                     <version>0.4.13</version>
126                     <executions>
127                         <execution>
128                             <id>remove-image</id>
129                             <phase>clean</phase>
130                             <goals>
131                                 <goal>removeImage</goal>
132                             </goals>
133                             <configuration>
134                                 <imageName>${docker.image.name}</imageName>
135                                 <imageTags>
136                                     <imageTag>${project.version}</imageTag>
137                                 </imageTags>
138                                 <verbose>true</verbose>
139                             </configuration>
140                         </execution>
```

```
                    </executions>
                </plugin>

                <!--
                    Build the Docker image.
                -->
                <plugin>
                    <groupId>com.spotify</groupId>
                    <artifactId>dockerfile-maven-plugin</artifactId>
                    <version>1.2.2</version>
                    <executions>
                        <execution>
                            <id>default</id>
                            <phase>package</phase>
                            <goals>
                                <goal>build</goal>
                            </goals>
                        </execution>
                    </executions>
                    <configuration>
                        <contextDirectory>${project.build.directory}/docker</contex
                        <writeTestMetadata>false</writeTestMetadata>
                        <dockerInfoDirectory></dockerInfoDirectory>
                        <verbose>true</verbose>
                        <forceCreation>true</forceCreation>
                        <imageName>${docker.image.name}</imageName>
                        <repository>${docker.image.name}</repository>
                        <tag>${project.version}</tag>
                        <forceTags>true</forceTags>
                        <pullNewerImage>false</pullNewerImage>
                        <imageTags>
                            <imageTag>${project.version}</imageTag>
                        </imageTags>
                        <dockerDirectory>${project.build.directory}/docker</dockerD
                    </configuration>
                </plugin>
            </plugins>
        </build>
    </profile>
</profiles>
```

Si observamos las diferentes partes del perfil de Maven, podemos ver:

- Hay un elemento <dependencies>.
  Agregue cualquier dependencia adicional de Maven a, por ejemplo, bibliotecas de terceros que desee incluir en la imagen de Docker en este elemento.
- En el elemento <properties>, hay una propiedad denominada docker.image.name.
  Esta propiedad contiene el nombre de la imagen de Docker que se producirá. Espero que esta propiedad se modifique para cada aplicación. La etiqueta de imagen Docker será la versión del proyecto.
- La propiedad siguiente en el elemento <properties> es docker.image.src.root.
  Esta propiedad contiene la ruta de acceso relativa al directorio que contiene el Dockerfile y cualquier contenido estático adicional que se va a incluir en la imagen Docker.
- La propiedad final en el elemento <properties> es docker.build.directory.
  Esta propiedad contiene la ruta al directorio en el que se construirá la imagen de Docker.
- En la sección <build> del perfil, el primer complemento es maven-resources-plugin.
  Este complemento se utiliza para copiar contenido estático del directorio apuntado por la propiedad docker.image.src.root al directorio cuya ubicación se especifica mediante la propiedad docker.build.directory. Se conserva toda la estructura de directorios.
- El segundo complemento en la sección <build> es el complemento maven-dependency.
  Con este complemento, el archivo JAR que contiene la aplicación de arranque de Spring creada por el

proyecto se copia en el directorio application / lib del directorio en el que se construirá la imagen de Docker. Recuerde que la ubicación del archivo JAR de la aplicación también se almacena en la variable de entorno JAR_PATH en el archivo Docker.
Tenga en cuenta que se especifica un nombre de archivo de destino para la aplicación JAR, con el fin de tener un nombre de archivo fijo para hacer referencia a en el Dockerfile.

- El tercer complemento en la sección <build> es el docker-maven-plugin.
  Dado que la versión actual del dockerfile-maven-plugin que discutiremos más adelante no sobrescribirá las imágenes de Docker existentes, he incluido este plug-in para poder eliminar cualquier imagen Docker existente al limpiar el proyecto Maven.
- El plug-in final es el dockerfile-maven-plugin.
  Este es el plug-in que es responsable de construir la imagen de Docker. La razón principal por la que he elegido este plug-in sobre las alternativas es que me permite crear una imagen de Docker desde un Dockerfile - al igual que funciona cuando estoy creando imágenes de Docker a mano.

Crear la imagen de Docker

Si ahora abrimos una ventana de terminal y vamos al directorio que contiene el archivo pom.xml de la aplicación web Spring Boot de ejemplo, podemos construir la imagen de Docker usando este comando de Maven:

```
1 mvn -Pdockerimage package
```

Después de algún tiempo, el mensaje BUILD SUCCESS debe aparecer en el terminal y si usted lista las imágenes de Docker, debería ver la nueva imagen en la lista:

```
1 $ docker images
2 REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
3 springboot-webapp     0.0.1-SNAPSHOT      16efcb5f8335        3 minutes ago       194.3 M
```

Utilice la imagen Docker

Para iniciar un contenedor Docker usando la imagen de Docker recién producida, utilizo el comando:

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

Abrir la URL http: // localhost: 8080 / hello? Name = Ivan (reemplazar localhost con la IP apropiada si ejecutas Docker en una máquina virtual) en un navegador, veo el mismo tipo de saludo que vi cuando corría La aplicación web Spring Boot fuera del contenedor Docker.

Codificación feliz!

Categoría: Java Etiquetas: docker , java , maven