How to create Java+Spring+Cucumber-jvm based "executable.jar" with all required Dependencies, Properties and Resources using Maven-shade-plugin?

http://www.automationace.com/

07 April, 2016

Recently I had to create a jar file of the cucumber-jvm project with all the dependent libraries bundled into it. The biggest advantage of creating jar file like this is it can be distributed to anyone in the team (like BAs) who can run the tests from their machines without having the project to be installed.

In other words:
1. How to create spring/cucumber -based executable jar with maven?
2. Creating a Runnable Binary Distribution With Maven for Cucumber-jvm + Java + Spring + Other dependencies
3. How to Build Cucumber-jvm + Java + Spring Projects with Maven?

Normally in Maven, we rely on dependency management. An artifact just contains the classes/resources of itself. Maven will be responsible to find out all artifacts (JARs etc) that the project is dependent, when compiling and for running etc.

With the help of **maven-shade-plugin** I was able to achieve this goal. It provides the capability to package the artifact in an uber-jar, including its dependencies and to shade – i.e. rename – the packages of some of the dependencies.

An **uber-jar** is something that take all dependencies, and extract the content of the dependencies and put it in the one JAR, with the classes/resources of the project itself. By having such uber-jar, it is easy for execution, because you will need only one big JAR instead of tons of small JARs to run your app. It also ease the distribution in some case.

Let's take a look at complete **pom.xml** file. For the ease of reading I have extracted only the required dependencies and the plugins from the pom.xml.

```
<dependency>
 <groupId>info.cukes</groupId>
 <artifactId>cucumber-junit</artifactId>
 <version>1.2.4</version>
</dependency>
<dependency>
 <groupId>info.cukes</groupId>
 <artifactId>cucumber-spring</artifactId>
 <version>1.2.4</version>
</dependency>
<dependency>
 <groupId>info.cukes</groupId>
 <artifactId>cucumber-java</artifactId>
 <version>1.2.4</version>
</dependency>
<dependency>
 <groupId>info.cukes</groupId>
 <artifactId>cucumber-core</artifactId>
 <version>1.2.4</version>
</dependency>
<dependency>
 <groupId>org.seleniumhq.selenium</groupId>
 <artifactId>selenium-java/artifactId>
```

```
 <version>2.45.0</version>
</dependency>
<dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>4.11</version>
</dependency>
<dependency>
 <groupId>com.google.guava</groupId>
 <artifactId>guava</artifactId>
 <version>18.0</version>
</dependency>

 <plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <version>1.9</version>
  <executions>
   <execution>
    <id>add-resource</id>
    <phase>generate-resources</phase>
    <goals>
     <goal>add-resource</goal>
    </goals>
    <configuration>
     <resources>
      <resource>
      <directory>${basedir}/target/test-classes</directory>
     </resource>
    </resources>
   </configuration>
  </execution>
 </executions>
</plugin>


<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-shade-plugin</artifactId>
 <version>2.4</version>
 <executions>
   <execution>
   <id>package-jar-with-dependencies</id>
   <phase>package</phase>
   <goals>
    <goal>shade</goal>
   </goals>
   <configuration>
    <appendAssemblyId>false</appendAssemblyId>
    <descriptorRefs>
     <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
   <transformers>
     <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
        <mainClass>cucumber.api.cli.Main</mainClass>
     </transformer>
```

```
        <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
        <resource>META-INF/spring.handlers</resource>
      </transformer>
        <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
        <resource>META-INF/spring.schemas</resource>
      </transformer>
    </transformers>
    <filters>
      <filter>
        <artifact>*:*</artifact>
        <excludes>
          <exclude>META-INF/*.SF</exclude>
          <exclude>META-INF/*.DSA</exclude>
          <exclude>META-INF/*.RSA</exclude>
        </excludes>
      </filter>
    </filters>
  </configuration>
 </execution>
</executions>
</plugin>
```

As you can see here I'm using Resource Transformers in pom.xml file. Aggregating classes/resources from several artifacts into one uber JAR is straight forward as long as there is no overlap. Otherwise, some kind of logic to merge resources from several JARs is required. This is where resource transformers kick in.

Once you add maven-shade-plugin to your pom.xml file then just use below command to build your project.

mvn clean package
mvn package (yes run only package this time. Need to run it twice else the source classes didn't copy)

And you should see one *.jar file created with all dependencies mentioned in pom.xml file inside.

Review It
List out the content of MyApp-1.0.0.jar
$ jar tf target/ MyApp-1.0.0.jar

Run the app with the cucumber-jvm options

1. Copy the jar file to any location

2. If you are running your tests on IE browser then store your IEDriverServer.exe at some location and point your code to that location as the *.exe cannot be added it to the zip.

3. Execute the below command

java -jar target/MyApp-1.0.0-.jar src/test/resources/features/ --glue com.xxx.stepdefs --monochrome --plugin json:target/test-report.json --plugin pretty --plugin html:target/test-report --plugin junit:target/test-report.xml --plugin com.xxx.resources.CustomFormatter --tags @Smoke

ANDING tags

java -jar target/MyApp-1.0.0-.jar src/test/resources/features/ --glue com.xxx.stepdefs --monochrome --plugin

json:target/test-report.json --plugin pretty --plugin html:target/test-report --plugin junit:target/test-report.xml --plugin com.xxx.resources.CustomFormatter --tags @Smoke --tags @Web

ORING tags

java -jar target/MyApp-1.0.0-.jar src/test/resources/features/ --glue com.xxx.stepdefs --monochrome --plugin json:target/test-report.json --plugin pretty --plugin html:target/test-report --plugin junit:target/test-report.xml --plugin com.xxx.resources.CustomFormatter --tags @Smoke,@Critical

Hurray! One monolithic artifact to deliver, with everything included!!!

Additional notes if you need to understand the above setup:
What Eclipse calls a “Runnable JAR file” is known in the Maven world as a “shaded” JAR. Assuming that your project is structured as a Maven project, you would add a snippet like this underneath <project><build><plugins>:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>2.4</version>
  <executions>
    <execution>
    <id>package-jar-with-dependencies</id>
    <phase>package</phase>
    <goals>
      <goal>shade</goal>
    </goals>
    <configuration>
      <appendAssemblyId>false</appendAssemblyId>
      <descriptorRefs>
        <descriptorRef>jar-with-dependencies</descriptorRef>
      </descriptorRefs>
     <transformers>
       <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
          <mainClass>cucumber.api.cli.Main</mainClass>
       </transformer>
          <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
          <resource>META-INF/spring.handlers</resource>
       </transformer>
          <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
          <resource>META-INF/spring.schemas</resource>
       </transformer>
     </transformers>
     <filters>
       <filter>
        <artifact>*:*</artifact>
        <excludes>
          <exclude>META-INF/*.SF</exclude>
          <exclude>META-INF/*.DSA</exclude>
```

```
          <exclude>META-INF/*.RSA</exclude>
        </excludes>
      </filter>
    </filters>
  </configuration>
 </execution>
</executions>
</plugin>
```

Under the section, note the element with a “ManifestResourceTransformer” attribute. This element expects a <mainClass> child element, which contains the name of the class which starts your application (i.e. the one with the “static void main” method).

The shade plugin is bound to the “package” phase of the Maven lifecycle. So when you execute:

mvn package

… Maven will generated monolithic “shaded” JAR under your project’s “/target” directory. You will actually find two JAR files there:

original-MyApp-1.0.0.jar
MyApp-1.0.0.jar

As should be obvious by the file sizes, the one with “original” in the filename is the ordinary (non-“shaded”) JAR file that would be built in any normal Maven build. The file without the “original” prefix is the “shaded” JAR, suitable for distribution as-is with all dependencies bundled. You can run your application with a command like this:

java -jar MyApp-1.0.0.jar

Advanced Cases
When you build a “shaded” JAR file… you are basically unzipping all your dependency JAR’s, copying their contents to the same directory root, and zipping that back up. What about cases where more than one dependency contains a file with the same name?

Well, one of those will overwrite the other(s), which may not be what you want to happen. The Maven “shade” plugin comes with the concept of “transformers”, which enable you to merge conflicting files together in the monolithic JAR rather than having one overwrite the other. There are different types of transformers… with the most common being:

♣ org.apache.maven.plugins.shade.resource.AppendingTransformer — simply appends one text file onto the end of another
♣ org.apache.maven.plugins.shade.resource.XmlAppendingTransformer — appends XML together while keeping the format sane

Now when you change your pom.xml snippet to look like this:

```
      <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
      <resource>META-INF/spring.handlers</resource>
    </transformer>
      <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
      <resource>META-INF/spring.schemas</resource>
    </transformer>
```

… your “shaded” JAR file will contain a “META-INF/spring.handlers” file and “META-INF/spring.schemas” file… made up of all the “META-INF/spring.handlers” files and “META-INF/spring.schemas” files found in all your dependencies, appended together as one.

Problems faced:
1. When I started working on it, I used maven assembly plugin to create jar with its dependencies but it was overwriting files with same name. For example it was overwriting spring.handlers and spring.schemas files in META-INF directory and application was unable to find some namespaces. Meta info like META-INF/spring.handlers and META-INF/spring.schemas have same path in all spring jars.

2. Then I came across maven shade plugin. You can use the maven-shade-plugin to package all dependencies into one uber-jar. It can also be used to build an executable jar by specifying the main class.

3. I found this plugin particularly useful as it merges content of specific files instead of overwriting them. This is needed when there are resource files that are have the same name across the jars and the plugin tries to package all the resource files.

4. I was getting exception early on as my source class were not included in Jar file when I extracted files from the jar. This particular exception was resolved by adding additional resources to the jar.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <version>1.9</version>
  <executions>
    <execution>
      <id>add-resource</id>
      <phase>generate-resources</phase>
      <goals>
        <goal>add-resource</goal>
      </goals>
     <configuration>
       <resources>
         <resource>
         <directory>${basedir}/target/test-classes</directory>
       </resource>
      </resources>
     </configuration>
    </execution>
  </executions>
</plugin>
```