How can I create an executable JAR with dependencies using Maven?

2134

793

I want to package my project in a single executable JAR for distribution.

How can I make a Maven project package all dependency JARs into my output JAR?

java maven-2 build-process build-automation executable-jar

edited May 31 '18 at 1:59

Rann Lifshitz
2,961 4 14 33

asked Feb 22 '09 at 8:43

soemirno
11.2k 3 15 14

12 Please explain which goal of the dependency plugin you are referring to. I know of no goal which does what the original question requests: to put all the dependencies either A) inside the authors jar via repackaging, or B) make an executable jar that has the others in a classpath of MANIFEST.MF – Matthew McCullough Mar 11 '09 at 15:11

2 You might find this useful rationaljava.com/2015/02/… – Dan Feb 2 '15 at 19:01

1 2 examples: tugay.biz/2015/12/a-standalone-java-web-application-with.html tugay.biz/2016/11/how-did-i-create-executable-jar-with.html – Koray Tugay Mar 30 '17 at 6:54

1 Refer stackoverflow.com/questions/35217128/… – Thanga Sep 11 '17 at 8:18

34 Answers

1 2 next

2074

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artifa
      <configuration>
        <archive>
          <manifest>
            <mainClass>fully.qualified.MainClas
          </manifest>
```

```
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies<,
        </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```

and you run it with

```
mvn clean compile assembly:single
```

Compile goal should be added before assembly:single or otherwise the code on your own project is not included.

See more details in comments.

Commonly this goal is tied to a build phase to execute automatically. This ensures the JAR is built when executing `mvn install` or performing a deployment/release.

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId:
  <configuration>
    <archive>
      <manifest>
        <mainClass>fully.qualified.MainClass</ma
      </manifest>
    </archive>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</des
    </descriptorRefs>
  </configuration>
  <executions>
```

```
    <execution>
      <id>make-assembly</id> <!-- this is used j
      <phase>package</phase> <!-- bind to the pa
      <goals>
        <goal>single</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

edited Feb 21 '13 at 8:37

community wiki
7 revs, 7 users 36%
IAdapter

14 Thanks @IAdapter. Note that you should always do a compile before hand because it will just put whatever is in "target/classes" in the JAR. This will ensure that the JAR includes any changes you recently made to the source code. So, you should do something like: `mvn clean compile assembly:single` . – Michael May 31 '11 at 19:03

8 I've edited the question to include the phase binding. I removed the deprecated assembly goal, because no-one needs to know about that. – Duncan Jones Feb 21 '13 at 8:38

2 I see that this doesn't add the jars to the uber jar, instead this just adds all the class files to the jar. – pitchblack408 Apr 7 '15 at 16:31

116 Tip: you can also add the element `<appendAssemblyId>false</appendAssemblyId>` into the `configuration` to avoid the

annoying "-jar-with-dependencies" suffix in the name – maxivis May 6 '15 at 19:22

2 forget `compile` and you are screwed. – prayagupd Nov 13 '16 at 0:44

316

You can use the dependency-plugin to generate all dependencies in a separate directory before the package phase and then include that in the classpath of the manifest:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifac
    <executions>
        <execution>
            <id>copy-dependencies</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>copy-dependencies</goal>
            </goals>
            <configuration>
                <outputDirectory>${project.build
                <overWriteReleases>false</overWr
                <overWriteSnapshots>false</overW
                <overWriteIfNewer>true</overWrit
            </configuration>
        </execution>
    </executions>
</plugin>
<plugin>
```

```
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath
                <classpathPrefix>lib/</classpath
                <mainClass>theMainClass</mainCla
            </manifest>
        </archive>
    </configuration>
</plugin>
```

Alternatively use `${project.build.directory}/classes/lib` as OutputDirectory to integrate all jar-files into the main jar, but then you will need to add custom classloading code to load the jars.

edited Sep 23 '14 at 12:08

Duncan Jones
45.6k 16 118 176

answered Dec 1 '10 at 10:46

André Aronsen
3,161 1 10 2

3 +1 Excellent. The reason I'm going with maven-dependency-plugin instead of maven-assembly-plugin is that I'm also using buildnumber-maven-plugin, and this way I can store the version number in the manifest of each jar individually. – PapaFreud Sep 15 '11 at 10:16

10 I like Your solution. I use `${project.build.directory}/classes/lib` as `outputDirectory` to have one main .jar with all dependencies inside, but - How to add

custom classloading code to load this jars? I need to make work execution like: `java -jar main-jar-with-deps.jar` . Is this possible ? – marioosh Mar 13 '12 at 19:09

3 @André Aronsen, i used this solution to add the dependencies in a lib folder inside the jar, but i always gets class not found exception, can you please advise how to fix that. – Mahmoud Saleh Aug 1 '12 at 11:36

9 +1 to you!! Looks like maven assembly plugin 'jar-with-dependencies' does not really work well. I was missing some entries from META-INF/spring.schemas in the generated jar. So I scrapped the jar-with-dependencies and used your solution above. Perfect thanks!!! – Derek Aug 6 '12 at 13:35

7 For anyone else encountering this problem, you must include the lib folder in the same directory with your jar where ever you transport the jar to. – Sparticles Oct 25 '14 at 11:13

174

+100

I blogged about some different ways to do this.

See Executable Jar with Apache Maven (WordPress)

or executable-jar-with-maven-example (GitHub)

Notes

Those pros and cons are provided by Stephan.

For Manual Deployment

- Pros
- Cons
  - Dependencies are out of the final jar.

Copy Dependencies to a specific directory

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactI
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>prepare-package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.build.directo
${project.build.finalName}.lib</outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Make the Jar Executable and Classpath Aware

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <configuration>
    <archive>
      <manifest>
```

```
        <addClasspath>true</addClasspath>
        <classpathPrefix>${project.build.finalNa
        <mainClass>${fully.qualified.main.class}
      </manifest>
    </archive>
  </configuration>
</plugin>
```

At this point the `jar` is actually executable with external classpath elements.

```
$ java -jar target/${project.build.finalName}.ja
```

Make Deployable Archives

The `jar` file is only executable with the sibling `...lib/` directory. We need to make archives to deploy with the directory and its content.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-antrun-plugin</artifactId>
  <executions>
    <execution>
      <id>antrun-archive</id>
      <phase>package</phase>
      <goals>
        <goal>run</goal>
      </goals>
      <configuration>
        <target>
          <property name="final.name" value="${p
${project.build.finalName}"/>
          <property name="archive.includes"
value="${project.build.finalName}.${project.pack
${project.build.finalName}.lib/*"/>
          <property name="tar.destfile" value="$
```

```
          <zip basedir="${project.build.director
includes="${archive.includes}" />
          <tar basedir="${project.build.director
includes="${archive.includes}" />
          <gzip src="${tar.destfile}" destfile="
          <bzip2 src="${tar.destfile}" destfile=
        </target>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Now you have `target/${project.build.finalName}.(zip|tar|tar.bz2|tar.gz)` which each contains the `jar` and `lib/*`.

Apache Maven Assembly Plugin

- Pros
- Cons
  - No class relocation support (use maven-shade-plugin if class relocation is needed).

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <archive>
```

```
          <manifest>
            <mainClass>${fully.qualified.main.cl
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</
        </descriptorRefs>
      </configuration>
    </execution>
  </executions>
</plugin>
```

You have `target/${project.bulid.finalName}-jar-with-dependencies.jar` .

Apache Maven Shade Plugin

- Pros
- Cons

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <shadedArtifactAttached>true</shadedArti
        <transformers>
          <transformer
implementation="org.apache.maven.plugins.shade.r
            <mainClass>${fully.qualified.main.cl
          </transformer>
        </transformers>
      </configuration>
    </execution>
```

```
  </executions>
</plugin>
```

You have `target/${project.build.finalName}-shaded.jar`.

onejar-maven-plugin

- Pros
- Cons
  - Not actively supported since 2012.

```
<plugin>
  <!--groupId>org.dstovall</groupId--> <!-- not
  <groupId>com.jolira</groupId>
  <artifactId>onejar-maven-plugin</artifactId>
  <executions>
    <execution>
      <configuration>
        <mainClass>${fully.qualified.main.class}
        <attachToBuild>true</attachToBuild>
        <!-- https://code.google.com/p/onejar-ma
        <!--classifier>onejar</classifier-->
        <filename>${project.build.finalName}-one
      </configuration>
      <goals>
        <goal>one-jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Spring Boot Maven Plugin

- Pros
- Cons
  - Add potential unecessary Spring and Spring Boot related classes.

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifact
  <executions>
    <execution>
      <goals>
        <goal>repackage</goal>
      </goals>
      <configuration>
        <classifier>spring-boot</classifier>
        <mainClass>${fully.qualified.main.class}
      </configuration>
    </execution>
  </executions>
</plugin>
```

You have `target/${project.bulid.finalName}-spring-boot.jar` .

edited May 23 '17 at 12:03

Community ♦

1 1

answered Jun 2 '14 at 3:01

Jin Kwon

10.5k 7 69 107

The only one working for me is the manual deployment. – caiohamamura Sep 16 '16 at 2:48

1 @caiohamamura You can clone the GitHub Repository and see how all profiles work. – Jin Kwon Sep 16 '16 at 11:58

The problem was with the package I was using: stackoverflow.com/a/12622037/2548351 – caiohamamura Sep 17 '16 at 1:48

1 Best answer to the question with multiple ways. – user942640 Nov 25 '18 at 15:32

130

Taking Unanswered's answer and reformatting it, we have:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</g
            <artifactId>maven-jar-plugin</artifa
            <configuration>
                <archive>
                    <manifest>
                        <addClasspath>true</addC
                        <mainClass>fully.qualifi
                    </manifest>
                </archive>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-assembly-plugin</a
            <configuration>
                <descriptorRefs>
                    <descriptorRef>jar-with-depe
                </descriptorRefs>
```

```
            </configuration>
        </plugin>
    </plugins>
</build>
```

Next, I would recommend making this a natural part of your build, rather than something to call explicitly. To make this a integral part of your build, add this plugin to your `pom.xml` and bind it to the `package` lifecycle event. However, a gotcha is that you need to call the `assembly:single` goal if putting this in your pom.xml, while you would call 'assembly:assembly' if executing it manually from the command line.

```
<project>
  [...]
  <build>
      <plugins>
          <plugin>
              <artifactId>maven-assembly-plugin<
              <configuration>
                  <archive>
                      <manifest>
                          <addClasspath>true</ad
                          <mainClass>fully.quali
                      </manifest>
                  </archive>
                  <descriptorRefs>
                      <descriptorRef>jar-with-de
                  </descriptorRefs>
              </configuration>
              <executions>
                  <execution>
                      <id>make-my-jar-with-depen
                      <phase>package</phase>
                      <goals>
                          <goal>single</goal>
```

```
                    </goals>
                </execution>
            </executions>
        </plugin>
      [...]
      </plugins>
    [...]
  </build>
</project>
```

edited Nov 28 '12 at 16:59

Mike Rylander
8,896 16 64 115

answered Feb 26 '09 at 4:31

Matthew McCullough
13.2k 6 33 37

10 Using the approach in this answer results in the following error message: 'Failed to load Main-Class manifest attribute from <jar file>', when trying to run the JAR using 'java -jar <jar file>' – Elmo Sep 10 '10 at 11:37

3 Archive part of the maven-jar-plugin is needed <archive> <manifest> <addClasspath>true</addClasspath> <mainClass>fully.qualified.MainClass</mainClass> </manifest> </archive> – RockyMM Nov 16 '11 at 14:40

4 Sorry, this answer is plain wrong, the mainClass tag has to be on the maven-assembly-plugin entry since you are calling that during the package goal – Alex Lehmann Sep 1 '12 at 23:27

@AlexLehmann true! – RockyMM Nov 22 '12 at 13:24

I'm surprised, why cannot pom.xml already have this included after mvn archetype:generate command? It is kind of annoying to manually copy-paste this every time when I create a new maven project...
– wintermute Jul 23 '16 at 19:32

95

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. After trying to use maven-assembly and maven-jar , I found that this plugin best suited my needs.

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

See example below

```
      <plugins>
    <!-- This plugin provides the capability to
including its dependencies and to shade - i.e. r
dependencies. -->
        <plugin>
            <groupId>org.apache.maven.plugins</g
            <artifactId>maven-shade-plugin</arti
            <version>1.4</version>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
```

```
                        <goal>shade</goal>
                    </goals>
                    <configuration>
                        <artifactSet>
                        <!-- signed jars-->
                            <excludes>
                                <exclude>bouncyc
                            </excludes>
                        </artifactSet>

                         <transformers>
                            <transformer

implementation="org.apache.maven.plugins.shade.r
                                <!-- Main class
                                <mainClass>com.m
                            </transformer>
                            <!-- Use resource tr
-->
                            <transformer

implementation="org.apache.maven.plugins.shade.r
                                <resource>proper
                            </transformer>
                            <transformer

implementation="org.apache.maven.plugins.shade.r
                                <resource>applic
                            </transformer>
                            <transformer

implementation="org.apache.maven.plugins.shade.r
                                <resource>META-I
                            </transformer>
                            <transformer

implementation="org.apache.maven.plugins.shade.r
                                <resource>META-I
                            </transformer>
                     </transformers>
                    </configuration>
                </execution>
            </executions>
```

```
    </plugin>

</plugins>
```

edited Jul 10 '12 at 16:07

Kristian Glass
28.8k 6 33 62

answered Sep 22 '10 at 15:20

Vijay Katam
1,121 8 7

So how does bcprov-jdk15.jar get onto the classpath at runtime, given that it's excluded from the shading process? – Andrew Swan Oct 12 '10 at 6:10

It was getting pulled by cxf-rt-ws-security which is part of my dependencies – Vijay Katam Oct 18 '10 at 22:58

Never heard about this plugin before, but it solved my problem with spring.handlers inside the jars. Thanks! – Alexandre L Telles Mar 26 '11 at 16:02

11 Those who got security exception, exclude DSA's from the Manifest. Check maven.apache.org/plugins/maven-shade-plugin/examples/… – ruhsuzbaykus Jul 19 '11 at 7:41

+1 I have used minijar:ueberjar in the past, but the minijar plugin is now deprecated and replaced by shade – rds Dec 8 '11 at 12:39

18

Long used the **maven assembly plugin**, but I could not find a solution to the problem with `"already added, skipping"` . Now, I'm using another plugin - **onejar-maven-plugin**. Examp below (`mvn package` build jar):

```
<plugin>
    <groupId>org.dstovall</groupId>
    <artifactId>onejar-maven-plugin</artifactI
    <version>1.3.0</version>
    <executions>
        <execution>
            <configuration>
                <mainClass>com.company.MainCla
            </configuration>
            <goals>
                <goal>one-jar</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

You need to add repository for that plugin:

```
<pluginRepositories>
    <pluginRepository>
        <id>onejar-maven-plugin.googlecode.com
        <url>http://onejar-maven-plugin.google
    </pluginRepository>
</pluginRepositories>
```

edited May 23 '17 at 11:33

Community ♦

1 1

answered Mar 13 '12 at 20:55

marioosh

how to get rid of extra messages in the output?
– Alexandr Apr 8 '16 at 12:33

16

You can use maven-dependency-plugin, but the question was how to create an executable JAR. To do that requires the following alteration to Matthew Franglen's response (btw, using the dependency plugin takes longer to build when starting from a clean target):

```
<build>
    <plugins>
        <plugin>
            <artifactId>maven-jar-plugin</artifa
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>fully.qualifi
                    </manifest>
                </archive>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-dependency-plugin<
            <executions>
                <execution>
                    <id>unpack-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>unpack-dependencie
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
    <resources>
        <resource>
```

```
            <directory>${basedir}/target/depende
        </resource>
    </resources>
</build>
```

answered Oct 13 '09 at 12:16

user189057

14

Another option if you really want to repackage the other JARs contents inside your single resultant JAR is the Maven Assembly plugin. It unpacks and then repacks everything into a directory via `<unpack>true</unpack>` . Then you'd have a second pass that built it into one massive JAR.

Another option is the OneJar plugin. This performs the above repackaging actions all in one step.

edited Mar 11 '09 at 15:20

answered Mar 11 '09 at 15:12

Matthew McCullough

13.2k 6 33 37

12

You can add the following to your **pom.xml**:

```
<build>
<defaultGoal>install</defaultGoal>
<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactI
    <version>2.3.2</version>
    <configuration>
      <source>1.6</source>
      <target>1.6</target>
    </configuration>
  </plugin>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>2.3.1</version>
    <configuration>
      <archive>
        <manifest>
          <addClasspath>true</addClasspath>
          <mainClass>com.mycompany.package.MainC
        </manifest>
      </archive>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-assembly-plugin</artifactI
    <configuration>
      <descriptorRefs>
        <descriptorRef>jar-with-dependencies</de
      </descriptorRefs>
      <archive>
        <manifest>
          <mainClass>com.mycompany.package.MainC
        </manifest>
      </archive>
    </configuration>
    <executions>
      <execution>
        <id>make-my-jar-with-dependencies</id>
        <phase>package</phase>
        <goals>
          <goal>single</goal>
```

```
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>
```

Afterwards you have to switch via the console to the directory, where the pom.xml is located. Then you have to execute **mvn assembly:single** and then your executable JAR file with dependencies will be hopefully build. You can check it when switching to the output (target) directory with **cd ./target** and starting your jar with a command similiar to **java -jar mavenproject1-1.0-SNAPSHOT-jar-with-dependencies.jar**.

I tested this with **Apache Maven 3.0.3**.

answered Aug 13 '11 at 18:23

Benny Neugebauer

28.6k 17 150 151

10

You could combine the `maven-shade-plugin` and `maven-jar-plugin` .

- The `maven-shade-plugin` packs your classes and all dependencies in a single jar file.
- Configure the `maven-jar-plugin` to specify the main class of your executable jar (see Set Up The Classpath, chapter "Make The Jar Executable").

Example POM configuration for `maven-jar-plugin`:

```
<plugin>
    <groupId>org.apache.maven.plugins</g
    <artifactId>maven-jar-plugin</artifa
    <version>2.3.2</version>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addC
                <mainClass>com.example.M
            </manifest>
        </archive>
    </configuration>
</plugin>
```

Finally create the executable jar by invoking:

```
mvn clean package shade:shade
```

edited Jan 31 '13 at 0:24

n00begon
3,291 3 21 40

answered Nov 26 '11 at 14:17

Oliver
441 4 10

3 The Shade plugin now has means of specifying the Main-Class entry in the manifest: maven.apache.org/plugins/maven-shade-plugin/examples/… – Chadwick Feb 27 '12 at 21:35

10

I went through every one of these responses looking to make a fat executable jar containing all dependencies and none of them worked right. The answer is the shade plugin, its very easy and straightforward.

```
    <plugin>
      <groupId>org.apache.maven.plugins</group
      <artifactId>maven-shade-plugin</artifact
      <version>2.3</version>
      <executions>
         <!-- Run shade goal on package phase
        <execution>
        <phase>package</phase>
        <goals>
            <goal>shade</goal>
        </goals>
        <configuration>
          <transformers>
             <transformer
implementation="org.apache.maven.plugins.shade
                <mainClass>path.to.MainClass</
             </transformer>
          </transformers>
        </configuration>
          </execution>
      </executions>
    </plugin>
```

Be aware that your dependencies need to have a scope of compile or runtime for this to work properly.

This example came from mkyong.com

edited Nov 23 '15 at 22:04

518 5 13

As I fixed this, would you mind updating your review. I hadn't taken your thoughts into consideration prior to posting and quickly made a fix upon seeing your comment – dsutherland Nov 23 '15 at 22:10

2 The `plugin` element goes in `pom.xml` under `build/plugins` . – isapir Dec 24 '17 at 1:54

9

You can use maven-shade plugin to build a uber jar like below

```
<plugin>
    <groupId>org.apache.maven.plugins</g
    <artifactId>maven-shade-plugin</arti
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>shade</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

answered Oct 26 '17 at 0:57

Minisha

1,123 11 25

But then how is this going to be deployed to repo? – Francesco Gualazzi May 30 '18 at 8:43

8

Ken Liu has it right in my opinion. The maven dependency plugin allows you to expand all the dependencies, which you can then treat as resources. This allows you to include them in the *main* artifact. The use of the assembly plugin creates a secondary artifact which can be difficult to modify - in my case I wanted to add custom manifest entries. My pom ended up as:

```
<project>
 ...
 <build>
  <plugins>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifac
    <executions>
     <execution>
      <id>unpack-dependencies</id>
      <phase>package</phase>
      <goals>
       <goal>unpack-dependencies</goal>
      </goals>
     </execution>
    </executions>
   </plugin>
  </plugins>
  ...
  <resources>
   <resource>
    <directory>${basedir}/target/dependency</dir
    <targetPath>/</targetPath>
   </resource>
  </resources>
```

```
  </build>
  ...
</project>
```

answered Sep 9 '09 at 13:37

Matthew Franglen
3,846 16 27

1 Really nice! Wouldn't it be better to use the generate-resources phase for the unpacking though? – nawroth Aug 8 '13 at 11:58

8

Here's an executable jar plugin for Maven that we use at Credit Karma. It creates a jar of jars with a classloader capable of loading classes from nested jars. This allows you to have the same classpath in dev and prod and still keep all classes in a single signed jar file.

https://github.com/creditkarma/maven-exec-jar-plugin

And here's a blog post with details about the plugin and why we made it: https://engineering.creditkarma.com/general-engineering/new-executable-jar-plugin-available-apache-maven/

answered Oct 14 '16 at 16:47

jchavannes
1,424 14 9

7

It should be like that:

```
<plugin>
        <artifactId>maven-dependency-plu
        <executions>
                <execution>
                        <id>unpack-depen
                        <phase>generate-
                        <goals>
                                <goal>un
                        </goals>
                </execution>
        </executions>
    </plugin>
```

Unpacking have to be in generate-resources phase because, if in package phase, will not be included as resources. Try clean package and you'll see.

edited Feb 4 '18 at 11:45

Masoud Mokhtari
1,238 7 27

answered Mar 5 '10 at 15:39

kac-ani
71 1 1

6

Use onejar plugin to build it as one executable jar file which packages all the dependancy jars in it. That solved my problem which was simila to this. When assembly plugin was used, it

folder and repackage them as a jar, it had over written all the similar implementations I had inside my code which were having the same class names. onejar is an easy solution in here.

edited Jan 8 '13 at 3:05

n00begon
3,291 3 21 40

answered Feb 11 '11 at 3:10

AmilaR
69 1 1

6

Problem with locating shared assembly file with maven-assembly-plugin-2.2.1?

Try using descriptorId configuration parameter instead of descriptors/descriptor or descriptorRefs/descriptorRef parameters.

Neither of them do what you need: look for the file on classpath. Of course you need adding the package where the shared assembly resides on maven-assembly-plugin's classpath (see below). If you're using Maven 2.x (not Maven 3.x), you may need adding this dependency in top-most parent pom.xml in pluginManagement section.

See this for more details.

Class: org.apache.maven.plugin.assembly.io.DefaultAssemblyReader

Example:

```
<!-- Use the assembly plugin to create a
<plugin>
    <artifactId>maven-assembly-plugin</a
    <version>2.2.1</version>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
            <configuration>
                <descriptorId>assembly-z
            </configuration>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>cz.ness.ct.ip.assem
            <artifactId>TEST_SharedAssem
            <version>1.0.0-SNAPSHOT</ver
        </dependency>
    </dependencies>
</plugin>
```

edited Jan 31 '13 at 0:26

n00begon

3,291 3 21 40

answered Mar 13 '11 at 11:40

Rostislav Stříbrný

61 1 1

I won't answer directly the question as other have already done that before, but I really

5

wonder if it's a good idea to embed all the dependencies in the project's jar itself.

I see the point (ease of deployment / usage) bu it depends of the use case of your poject (and there may be alternatives (see below)).

If you use it fully standalone, why not.

But if you use your project in other contexts (lik in a webapp, or dropped in a folder where othe jars are sitting), you may have jar duplicates in your classpath (the ones in the folder, the one the jars). Maybe not a bid deal but i usually avoid this.

A good alternative :

- deploy your application as a .zip / .war : th archive contains your project's jar and all dependent jars ;
- use a dynamic classloader mechanism (se Spring, or you can easily do this yourself) have a single entry point of your project (a single class to start - see the Manifest mechanism on another answer), which wil add (dynamically) to the current classpath all the other needed jars.

Like this, with in the end just a manifest and a "special dynamic classloader main", you can start your project with :

```
java -jar ProjectMainJar.jar com.stackoverflow
```

answered Dec 28 '09 at 16:43

SRG
1,402 15 17

1 How to put the project's jar and all dependent jars into an archive then? – Hai Minh Nguyen Nov 6 '10 at 18:06

4

To resolve this issue we will use Maven Assembly Plugin that will create the JAR together with its dependency JARs into a single executable JAR file. Just add below plugin configuration in your pom.xml file.

```
<build>
   <pluginManagement>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</g
            <artifactId>maven-assembly-plugin</a
            <configuration>
               <archive>
                  <manifest>
                     <addClasspath>true</addClas
                     <mainClass>com.your.package
                  </manifest>
               </archive>
               <descriptorRefs>
                  <descriptorRef>jar-with-depend
               </descriptorRefs>
            </configuration>
            <executions>
               <execution>
                  <id>make-my-jar-with-dependenc
                  <phase>package</phase>
                  <goals>
                     <goal>single</goal>
```

```
                    </goals>
                  </execution>
                </executions>
              </plugin>
            </plugins>
          </pluginManagement>
        </build>
```

After doing this don't forget to run MAVEN tool with this command mvn clean compile assembly:single

http://jkoder.com/maven-creating-a-jar-together-with-its-dependency-jars-into-a-single-executable-jar-file/

answered Sep 1 '16 at 10:21

Anoop Rai
239 3 3

3

If you want if from command Line itself . Just run the below command from the project path

mvn assembly:assembly

answered Sep 14 '10 at 4:37

Mayank
31 1

I think you still need to do some stuff in the `pom.xml` otherwise you get `Error reading assemblies: No assembly descriptors found.` . That's what happens for me anyway.
– Sridhar-Sarnobat Sep 8 '17 at 18:59

2

You can also use this plug-in, it is pretty good and I use it for packaging my jars
http://sonatype.github.io/jarjar-maven-plugin/

answered Jan 29 '14 at 13:54

Adelin
7,455 17 79 122

2

Something that have worked for me was:

```
<plugin>
  <artifactId>maven-dependency-plugin</artifac
  <executions>
    <execution>
      <id>unpack-dependencies</id>
      <phase>prepare-package</phase>
      <goals>
        <goal>unpack-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.build.direc
      </configuration>
    </execution>

  </executions>
</plugin>


<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <executions>
    <execution>
      <id>unpack-dependencies</id>
      <phase>package</phase>
    </execution>
```

```
    </executions>
    <configuration>
      <archive>
        <manifest>
          <addClasspath>true</addClasspath>
          <classpathPrefix>lib/</classpathPrefix
          <mainClass>SimpleKeyLogger</mainClass>
        </manifest>
      </archive>
    </configuration>
  </plugin>
```

I had extraordinary case because my dependency was system one:

```
<dependency>
  ..
  <scope>system</scope>
  <systemPath>${project.basedir}/lib/myjar.jar</
</dependency>
```

I have changed the code provided by @user189057 with changes: 1) maven-dependency-plugin is executed in "prepare-package" phase 2) I am extracting unpacked classess directly to "target/classes"

edited Mar 15 '14 at 15:47

hd1

25.2k 3 57 69

answered Sep 24 '13 at 16:11

fascynacja

537 4 13

2

I tried the most up-voted answer here, and was able to get the jar runnable. But the program didn't run correctly. I do not know what the reason was. When I try to run from `Eclipse` , I get a different result but when I run the jar from command-line I get a different result (it crashes with a program-specific runtime error).

I had a similar requirement as the OP just that I had too many (Maven) dependencies for my project. Fortunately, the only solution that worked for me was that using `Eclipse` . Very simple and very straightforward. This is not a solution to the OP but is a solution for someone who has a similar requirement but with many Maven dependencies,

1) Just right-click on your project folder (in Eclipse) and select `Export`

2) Then select `Java` -> `Runnable Jar`

3) You will be asked to choose the location of the jar file

4) Finally, select the class that has the Main method that you want to run and choose `Package dependencies with the Jar file` and click `Finish`

answered Oct 10 '14 at 22:11

Rocky Inde

1,048 14 20

This is the best way i found:

2

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>2.4</version>
  <configuration>
    <archive>
      <manifest>
      <addClasspath>true</addClasspath>
      <mainClass>com.myDomain.etc.MainClassNam
      <classpathPrefix>dependency-jars/</class
      </manifest>
    </archive>
  </configuration>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifac
  <version>2.5.1</version>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
          <goal>copy-dependencies</goal>
      </goals>
      <configuration>
          <outputDirectory>
             ${project.build.directory}/depend
          </outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

With this configuration, all dependencies will be located in `/dependency-jars`. My application has no `Main` class, just context ones, but one of my dependencies do have a `Main` class (`com.myDomain.etc.MainClassName`) that starts the JMX server, and receives a `start` or a `stop`

parameter. So with this i was able to start my application like this:

```
java -jar ./lib/TestApp-1.0-SNAPSHOT.jar start
```

I wait it be useful for you all.

edited Sep 30 '15 at 17:22

answered Mar 9 '15 at 22:22

EliuX
4,188 23 24

2

I compared the tree plugins mentioned in this post. I generated 2 jars and a directory with all the jars. I compared the results and definitely the maven-shade-plugin is the best. My challenge was that I have multiple spring resources that needed to be merged, as well as jax-rs, and JDBC services. They were all merged properly by the shade plugin in comparison with the maven-assembly-plugin. In which case the spring will fail unless you copy them to your own resources folder and merge them manually one time. Both plugins output the correct dependency tree. I had multiple scopes like test,provide, compile, etc the test and provided were skipped by both plugins. They both produced the same manifest but I was able to consolidate licenses with the shade plugin using

their transformer. With the maven-dependency-plugin of course you don't have those problems because the jars are not extracted. But like some other have pointed you need to carry one extra file(s) to work properly. Here is a snip of the pom.xml

```
            <plugin>
            <groupId>org.apache.maven.plugins</g
            <artifactId>maven-dependency-plugin<
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>prepare-package</phas
                    <goals>
                        <goal>copy-dependencies<
                    </goals>
                    <configuration>

<outputDirectory>${project.build.directory}/lib<
                        <includeScope>compile</i
                        <excludeTransitive>true<
                        <overWriteReleases>false
                        <overWriteSnapshots>fals
                        <overWriteIfNewer>true</
                    </configuration>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</g
            <artifactId>maven-assembly-plugin</a
            <version>2.6</version>
            <configuration>
                <archive>
                    <manifest>
                        <addClasspath>true</addC

<mainClass>com.rbccm.itf.cdd.poller.landingzone.
                    </manifest>
```

```
                </archive>
                <descriptorRefs>
                    <descriptorRef>jar-with-depe
                </descriptorRefs>
            </configuration>
            <executions>
                <execution>
                    <id>make-my-jar-with-depende
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</g
            <artifactId>maven-shade-plugin</arti
            <version>2.4.3</version>
            <configuration>
                <shadedArtifactAttached>false</s

<keepDependenciesWithProvidedScope>false</keepDe
                <transformers>
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                        <resource>META-INF/servi
                    </transformer>
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                        <resource>META-INF/sprin
                    </transformer>
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                        <resource>META-INF/sprin
                    </transformer>
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                        <resource>META-INF/sprin
                    </transformer>
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                        <resource>META-INF/sprin
```

```
                    </transformer>
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                    <transformer
implementation="org.apache.maven.plugins.shade.r
                    </transformer>
                </transformers>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>shade</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
```

answered Feb 24 '16 at 18:12

Fabio

155 2 3 16

2

For anyone looking for options to exclude specific dependencies from the uber-jar, this is a solution that worked for me:

```
<project...>
<dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.11</artifac
            <version>1.6.1</version>
            <scope>provided</scope> <===========
        </dependency>
</dependencies>
<build>
```

```
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugi
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-
                    </descriptorRefs>
                    <archive>
                        <manifest>
                            <mainClass>...</main
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

So it's not a configuration of the mvn-assembly-plugin but a property of the dependency.

edited Jun 18 '16 at 14:50

user3083022

149 1 2 13

answered May 3 '16 at 8:09

Paul Bormans

854 10 16

2

There are millions of answers already, I wanted to add you don't need `<mainClass>` if you don't need to add entryPoint to your application. **For example APIs may not have necessarily have `main` method.**

maven plugin config

```
<build>
  <finalName>log-enrichment</finalName>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artif
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies
        </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```

build

```
mvn clean compile assembly:single
```

verify

```
ll target/
total 35100
drwxrwx--- 1 root vboxsf     4096 Sep 29 16:25 .
drwxrwx--- 1 root vboxsf     4096 Sep 29 16:25 .
drwxrwx--- 1 root vboxsf        0 Sep 29 16:08 a
drwxrwx--- 1 root vboxsf        0 Sep 29 16:25 c
drwxrwx--- 1 root vboxsf        0 Sep 29 16:25 g
drwxrwx--- 1 root vboxsf        0 Sep 29 16:25 g
```

```
-rwxrwx--- 1 root vboxsf 35929841 Sep 29 16:10 l
drwxrwx--- 1 root vboxsf        0 Sep 29 16:08 m
```

answered Sep 29 '16 at 23:27

prayagupd

20.4k 8 93 143

2

Add to pom.xml:

```
<dependency>
        <groupId>com.jolira</groupId>
        <artifactId>onejar-maven-plugin</a
        <version>1.4.4</version>
</dependency>
```

and

```
<plugin>
    <groupId>com.jolira</groupId>
    <artifactId>onejar-maven-plugin</artifa
    <version>1.4.4</version>
    <executions>
        <execution>
            <goals>
                <goal>one-jar</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Thats it. Next mvn package will also create one fat jar additionally, including all dependency jars.

edited Nov 7 '18 at 7:17

Biswajit Roy
311 1 4 14

Aydin K.
1,647 20 34

1

The maven-assembly-plugin worked great for me. I spent hours with the maven-dependency-plugin and couldn't make it work. The main reason was that I had to define in the configuration section explicitly the artifact items which should be included as it is described in the documentation. There is an example there for the cases when you want to use it like: `mvn dependency:copy` , where there are not included any artifactItems but it doesn't work.

answered May 24 '13 at 19:42
Chris
184 2 13

1

This could also be an option,You will be able to build your jar file

```
<build>
    <plugins>
        <plugin>
            <!-- Build an executable JAR -->
            <groupId>org.apache.maven.plugins</g
            <artifactId>maven-jar-plugin</artifa
            <version>2.4</version>
            <configuration>
```

```
                <archive>
                    <manifest>
                        <addClasspath>true</addC
                        <classpathPrefix>lib/</c
                        <mainClass>WordListDrive
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
```

answered Nov 25 '15 at 10:40

salmanbw

920 1 12 17

1 2 next

protected by Will Apr 14 '11 at 18:34

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?