Title: spring-boot : Exclude dependencies on packaging

Post Body:

I am working on a spring boot project (Project A) that would be included in other projects (Project B, Project C ...) . I have several dependencies in Project A, but in the project importing Project A, some or only one may be required. I am trying to find a way to exclude the jar dependencies while packaging Project A so that the required ones will be provided by Project B during run time. I would like to have the dependencies available when the Project A is run independently for testing purposes.

Already tried the following

I have tried using:

```
<scope>provided</scope> <optional>true</optional>
```

Still the jars end up in the final artifact.

Also tried adding the following to the spring-boot-maven-plugin

```
        <plugins>               <plugin>                        <groupId>org.springframework.boot</groupId>               <artifac
```

This would just remove the spring-boot dependency , but the jars for the children of this dependency would still end up in the final artifact.

Accepted Answer:

In our current project we have the requirement to create a war file for the application, which has to be deployed in a JEE server. The war file must include only the needed jar files, not including any API or implementation already provided by the JEE server.

But, we want to retain the possibility to generate an executable war or jar file as provided by default by Boot, for testing purposes.

To achieve it, we've set all optional dependencies as *provided*. For example, we have some direct dependencies used in development, like the JDBC driver, we don't want to include in the deployed war file. Also there are some boot main starters which provide dependencies with other starters and libraries we don't need in a JEE server. This is the case of the *spring-boot-starter-tomcat* and *spring-boot-starter-jdbc* starters. In our project, we have the followind dependencies in our *pom.xml* file:

```
<dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-tomcat</artifactId>      <scop
```

This way those dependencies won't be included in the original jar/war file, but the spring boot maven plugin will include them in the *lib-provided* folder of the repackaged jar/war.

Those dependencies won't be seen by the JEE server, but make the packaged application bigger than needed. The solution is to tell the spring boot maven plugin to create the repackaged file with another name, as well as excluding the development tools:

```
<plugin>    <groupId>org.springframework.boot</groupId>    <artifactId>spring-boot-maven-plugin</artifactId>    <configuration>
```

This way maven will generate two packages for your application:

- The default jar/war package, without all the provided dependencies.
- A repackaged file whose name ends with _exec.jar/.war, with all provided dependencies in the *lib-provided* folder and the support to run the application with *java -jar file*

In your case you could use the same technique to be able to generate the package for the Project A to be included in Project B, and the package for Project A to be run as standalone.

If you don't need to create the package for Project A to be run by itself, and only test it in your IDE, you might even remove the spring boot maven plugin from your *pom.xml*.

Highest Rated Answer:

You can **add `exclude` block into spring-boot-maven-plugin**

```
 <plugin>       <groupId>org.springframework.boot</groupId>       <artifactId>spring-boot-maven-plugin</artifactId>       <configur
```

Official documentation : https://docs.spring.io/spring-boot/docs/2.1.13.RELEASE/maven-plugin/examples/exclude-dependency.html