Title: javax.servlet in Springboot Application

Post Body:

I have a SpringBoot application. One class needs to import javax.servlet.filter and implements the filter for customization. It builds successfully locally, but failed to start up on the cloud server and throws llegalStateExceptions. It's the filter class caused the issue. Once I remove the filter, the app runs successfully locally and on the cloud.

I think it might be possible that app could not find javax.servlet api in maven, or it's a servlet/embedded tomcat incompatibility issue. We have most of the dependencies from org.springframework.boot. The only tomcat related part we have is:

```
<dependency>     <groupId>org.apache.tomcat</groupId>     <artifactId>tomcat-jdbc</artifactId>     <version>LATEST</version> <
```

We don't have the following:

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

I tried to add them (individually and both) and changed scope to compile, or remove scope. All sorts of dependencies minor changes, but nothing works so far. Please let me know if you have any good suggestions.

Accepted Answer: None

Highest Rated Answer:

The `provided` scope means that you intend that library to be provided by the environment in which you deploy your code. Why would you do such a thing? Well, by default tomcat puts libs containing javax.servlet classes on the classpath for you, and many other servers do the same. It's a way of allowing container providers (tomcat, jboss, websphere, etc.) to provide custom implementations of a library that specific to their container.

Maven interprets the `provided` scope as meaning that you do not want to include the library in any bundles or deployments, uberjars, zips, or whatever. Spring-boot produces an uber-jar -- that is, it packages all your application dependencies in your application's single artefact (as opposed to keeping them in separate files which is how we used to do java in the ancient past before we knew better).

So, if you find that you're missing that package at run time it may be because it isn't actually `provided` by your container and you have to supply it yourself by removing the scope tag completely. This will tell maven to bundle it up in your uber-jar for use at runtime.

Looking at your specific problem, neither of those libraries should be `provided` so you should remove the scope tags completely from all your dependencies. If that doesn't work, you have other problems.

BTW: As an aside, it's a really bad idea to use `LATEST` as the version number for any dependency. It violates dozens of best-practices and against certain binary stores (nexus for example) it barely works and isn't guaranteed. You should instead find out which version of tomcat-jdbc you're meant to be using and use the absolute version instead. In this case the `LATEST` version is almost certainly the wrong one to be using.