Dependency Management in Maven

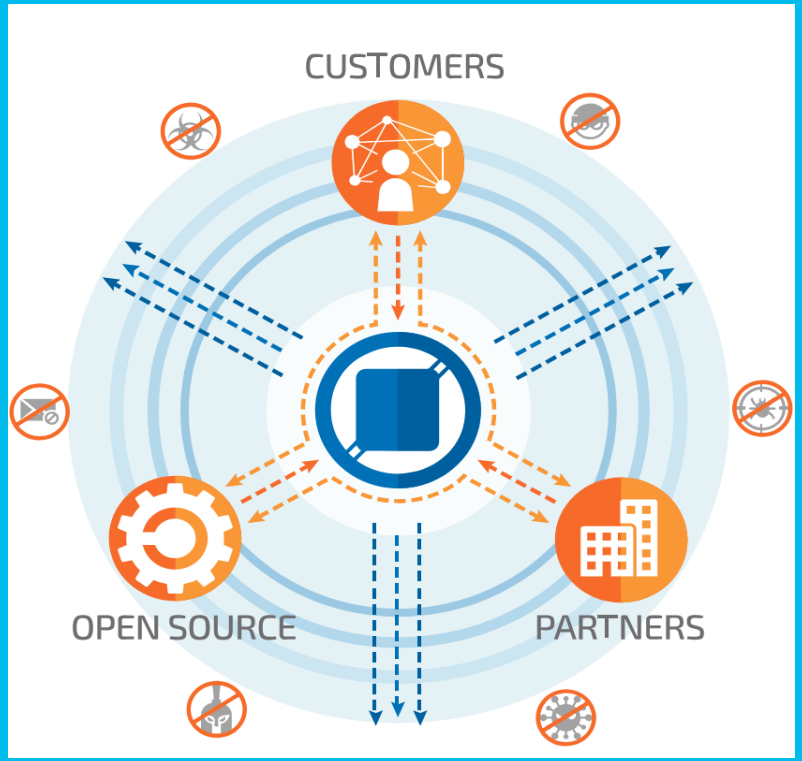
Jay Chow

Software Engineer, Cisco eCommerce, IT

Cisco, San Jose

6th June 2020

Agenda



1. What is Maven?
2. What is a dependency?
3. Docker
4. Kubernetes
5. Microservices
6. Kubernetes with microservices architecture
7. What's next for us?
8. Q & A
9. References

What is Maven?

- A tool that can be used for building and managing any Java-based project
- Something that will make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project
- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices

What is a dependency in Maven?

- In Maven, dependency is another archive—JAR, ZIP, and so on—which your current project needs in order to compile, build, test, and/or to run. The dependencies are gathered in the pom.xml file, inside of a <dependencies> tag.

- When you run a build or execute a maven goal, these dependencies are resolved, and are then loaded from the local repository. If they are not present there, then Maven will download them from a remote repository and store them in the local repository. We are also allowed to manually install the dependencies as well.

What is JAR?

- A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.
- A JAR file is a file that contains compressed version of .class files, audio files, image files or directories. We can imagine a .jar files as a zipped file(.zip) that is created by using WinZip software. Even , WinZip software can be used to used to extract the contents of a .jar . So you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking.
- We can run a jar file using the java command: java –jar sample.jar

Maven Dependency Example

```
<dependencies>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.5.RELEASE</version>
    </dependency>

</dependencies>
```

External Dependency

- Sometimes, you will have to refer jar files which are not in maven repository (neither local, central or remote repository). You can use these jars by placing them in project's lib folder and configure the external dependency like this:

```
<dependency>
  <groupId>extDependency</groupId> // set to name of dependency
  <artifactId>extDependency</artifactId> // set to name of dependency
  <scope>system</scope> // set to system
  <version>1.0</version>
  <systemPath>${basedir}\war\WEB-INF\lib\extDependency.jar</systemPath>
</dependency> // systemPath element refer to location of JAR file
```

Maven Dependency Tree

- Using maven's dependency:tree command, you can view list of all dependencies into your project – transitively. Transitive dependency means that if A depends on B and B depends on C, then A depends on both B and C.

- Transitivity brings a very serious problem when different versions of the same artifacts are included by different dependencies. It may cause **version mismatch** issue in runtime. In this case, dependency:tree command is be very useful in dealing with conflicts of JARs.

- To resolve such version mismatch issues, maven provides <exclusion> tag, in order to break the transitive dependency.

- For example, when you have JUnit 4.12 in classpath and include DBUnit dependency, then you will need to remove JUnit 3.8.2 dependency. It can be done with exclusion tag.

Maven Dependency Exclusion

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.dbunit</groupId>
    <artifactId>dbunit</artifactId>
    <version>${dbunit.version}</version>
    <scope>test</scope>
    <exclusions>
        <!--Exclude transitive dependency to JUnit-3.8.2 -->
        <exclusion>
            <artifactId>junit</artifactId>
            <groupId>junit</groupId>
         </exclusion>
    </exclusions>
</dependency>
```

References

https://howtodoinjava.com/maven/maven-dependency-management/

https://www.geeksforgeeks.org/jar-files-java/