Maven

Quickstart
2022

What is Maven?

- English meaning: Accumulator of Knowledge
- Provides
  - A standard way to build projects,
  - A clear definition of what the project consists of
  - An easy way to publish project information and a way to share JARs across several projects
- In summary, it is a tool that can now be used for building and managing any Java-based project.

What is Maven?

- Now what exactly does a build tool do? Maven does three things rather well:
  - **Dependency Management:** Maven lets you easily include 3rd party dependencies (think libraries/frameworks such as Spring) in your project. An equivalent in other languages would be Javascript's npm, Ruby's gems or PHP's composer.
  - **Compilation through convention:** In theory you could compile big Java projects with a ton of classes, by hand with the javac command line compiler (or automate that with a bash script). This does however only work for toy projects. Maven expects a certain directory structure for your Java source code to live in and when you later do a mvn clean install , the whole compilation and packaging work will be done for you.
  - **Everything Java:** Maven can also run code quality checks, execute test cases and even deploy applications to remote servers, through plugins. Pretty much every possible task you can think of.

Directory Layout

- pom.xml
  - Technically, any directory that contains a pom.xml file is also a valid Maven project. A pom.xml file contains everything needed to describe your Java project. Let's have a look at a minimal version:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.marcobehler</groupId>
    <artifactId>my-project</artifactId> (1)
    <version>1.0-SNAPSHOT</version> (2)

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source> (3)
        <maven.compiler.target>1.8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency> (4)
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

1. We are defining a project called 'my-project'
2. With a version number of 1.0-SNAPSHOT, i.e. work-in-progress
3. Using Java 1.8 for compilation
4. With one dependency needed for unit testing: junit in version 4.12

Directory Layout

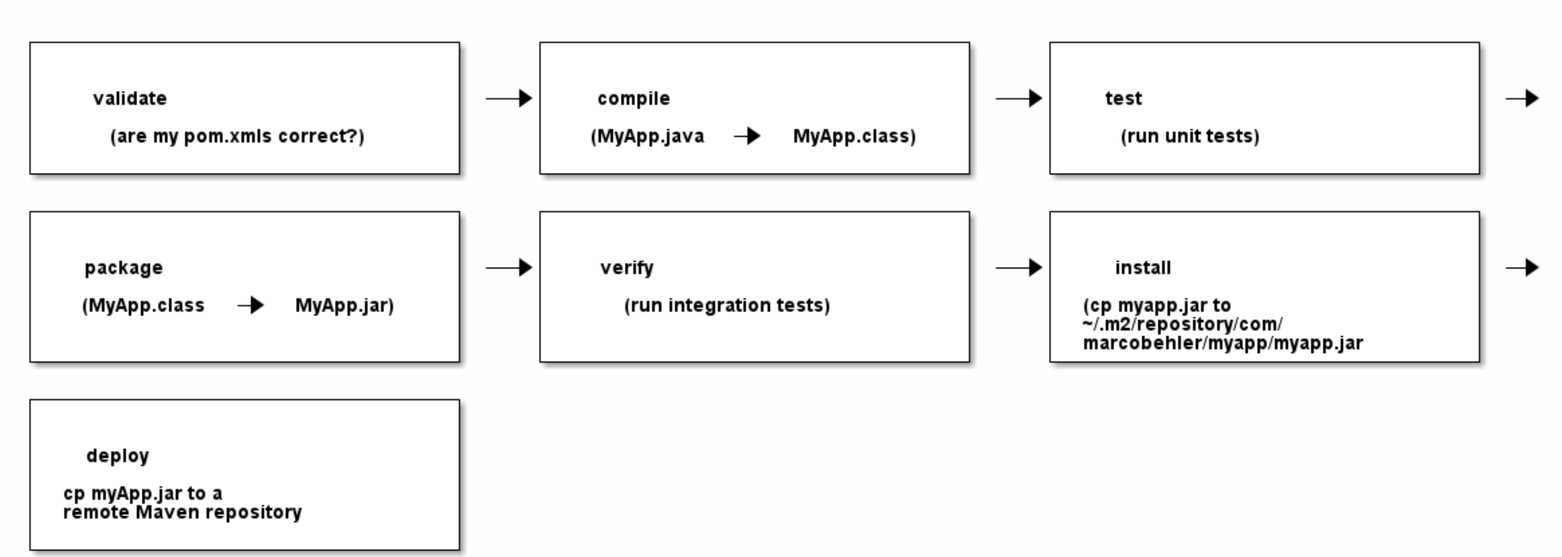
- Maven’s src & target folders
  - Apart from a pom.xml file, you also need Java source code for Maven to do its magic, whenever you are calling mvn clean install. By convention:
    - Java source code is to be meant to live in the "/src/main/java" folder
    - Maven will put compiled Java classes into the "target/classes" folder
    - Maven will also build a .jar or .war file, depending on your project, that lives in the "target" folder.
- In the end, your project will look like this:

```
+ myproject
    + -- src
        + -- main
            + -- java
                MyApp.java
    + -- target
        + -- classes (after 'mvn compile')
             MyApp.class

        myproject.jar (upon mvn package or mvn install)

    pom.xml
```

Maven Build Lifecycle: Phases

- Now what really happens when you execute a mvn clean install in your project? Maven has the concept of a build lifecycle, which is made up of different phases.

- Here's what Maven's default lifecycle looks like (note: it is missing 'clean').



- The phases are sequential and dependent on each other.

Maven Build Lifecycle: Example

- When you call mvn deploy, mvn will also execute every lifecycle phase before deploy, in order: validate, compile, test, package, verify, install.
- Same for verify: validate, compile, test, package. Same for all other phases.
- And as clean is not part of Maven's default lifecycle, you end up with commands like mvn clean install or mvn clean package. Install or package will trigger all preceding phases, but you need to specify clean in addition.

Where does Maven store 3rd party libraries?

- Contrary to other languages, where project dependencies are stored in your project's directory, Maven has the concept of repositories.
- There are local repositories (in your user's home directory: ~/.m2/) and remote repositories. Remote repositories could be internal, company-wide repositories like Artifactory or Nexus or the (reference) global repo at https://repo.maven.apache.org/maven2/.
- Maven will always download your project dependencies into your local maven repository first and then reference them for your build. When you think back at your pom.xml file from before:

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

Then mvn will, once you try and "mvn test" your project, download the junit dependency into ~/.m2/repository/junit/junit/4.12/junit-4.12.jar and reference it via Java's classpath mechanism for your build.