Maven

By

Srikanth Pragada

What is Maven?

- Maven is a software project management and comprehension tool.
- It automates tasks as downloading dependencies, putting additional jars on a classpath, compiling source code into byte code, running tests, packaging compiled code into deployable artifacts such as JAR, WAR, and ZIP files, and deploying these artifacts to an application server or repository minimizing the risk of humans making errors while building the software manually and separating the work of compiling and packaging our code from that of code construction.

Why Maven?

- **Simple project setup that follows best practices**: Maven tries to avoid as much configuration as possible, by supplying project templates (named archetypes)
- **Dependency management**: It includes automatic updating, downloading and validating the compatibility, as well as reporting the dependency closures (known also as transitive dependencies)
- **Isolation between project dependencies and plugins**: Project dependencies are retrieved from the dependency repositories while any plugin's dependencies are retrieved from the plugin repositories, resulting in fewer conflicts when plugins start to download additional dependencies
- **Central repository system**: Project dependencies can be loaded from the local file system or public repositories, such as Maven Central

Project Attributes

groupId
A unique base name of the company or group that created the project

artifactId
A unique name of the project

version
A version of the project

packaging
A packaging method (e.g. *WAR*/*JAR*/*ZIP*)

Repository

- A repository in Maven is used to hold build artifacts and dependencies of varying types.
- The default local repository is located in the ***.m2/repository*** folder under the home directory of the user.
- If an artifact or a plugin is available in the local repository, Maven uses it.
- Otherwise, it is downloaded from a central repository and stored in the local repository.
- The default central repository is Maven Central.
- If a library is not available in Central then you need to specify the repository as follows:

```
<repositories>
   <repository>
    <id>JBoss repository</id>
    <url>http://repository.jboss.org/nexus/content/groups/public/</url>
   </repository>
</repositories>
```

Properties

- Custom properties can help to make your pom.xml file easier to read and maintain.
- They are declared using <properties> sections and used with ${name}.

```
<properties>
    <spring.boot>3.0</spring.boot>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <version>${spring.boot}</version>
    </dependency>

</dependencies>
```

Maven Plugins

- Maven is - at its heart - a plugin execution framework; all work is done by plugins.
- Build plugins will be executed during the build and they should be configured in the <build/> element from the POM.

Maven Plugins

- ❑ clean — Clean up after the build.
- ❑ compiler — Compiles Java sources.
- ❑ deploy — Deploy the built artifact to the remote repository.
- ❑ install — Install the built artifact into the local repository.
- ❑ surefire — Run the JUnit unit tests in an isolated classloader.
- ❑ verifier — Useful for integration tests - verifies the existence of certain conditions.

Maven Lifecycle Phases

- Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.
- For example, if we run the deploy phase, which is the last phase in the default build lifecycle, it'll execute all the phases before the deploy phase as well, which is the entire default lifecycle.
- A Maven phase represents a stage in the Maven build lifecycle.
- Each phase is responsible for a specific task.

Maven Lifecycle Phases

- The default lifecycle comprises of the following :
  - ✓ **validate** - validate the project is correct and all necessary information is available
  - ✓ **compile** - compile the source code of the project
  - ✓ **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
  - ✓ **package** - take the compiled code and package it in its distributable format, such as a JAR.
  - ✓ **verify** - run any checks on results of integration tests to ensure quality criteria are met
  - ✓ **install** - install the package into the local repository, for use as a dependency in other projects locally
  - ✓ **deploy** - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

Maven Goals

- Each phase is a sequence of goals, and each goal is responsible for a specific task.
- When we run a phase, all goals bound to this phase are executed in order.

```
>mvn <PLUGIN>:help   -- lists all goals associated with plugin

Example:

>mvn spring-boot:help
```

Maven Commands

mvn clean
Cleans the project and removes all files generated by the previous build.
mvn compile
Compiles source code of the project.
mvn test
Runs tests for the project.
mvn package
Creates JAR or WAR file for the project to convert it into a distributable format.
mvn install
Deploys the packaged JAR/ WAR file to the local repository.
mvn validate
Validate the project's POM and configuration.
mvn deploy
Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.

Spring Boot Maven

- The Spring Boot Maven Plugin provides Spring Boot support in Apache Maven.
- It allows you to package executable jar or war archives, run Spring Boot applications, generate build information and start your Spring Boot application prior to running integration tests.

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

Spring Boot Maven Plugin Goals

spring-boot:repackage

- ✓ Repackage existing JAR and WAR archives so that they can be executed from the command line using java -jar.
- ✓ With layout=NONE can also be used simply to package a JAR with nested dependencies (and no main class, so not executable).

spring-boot:run

- ✓ Run an application in place.

spring-boot:test-run

- ✓ Run an application in place using the test runtime classpath.
- ✓ The main class that will be used to launch the application is determined as follows: The configured main class, if any.
- ✓ Then the main class found in the test classes directory, if any.
- ✓ Then the main class found in the classes directory, if any.