Maven

* Maven is a powerful *project management tool* based on POM (project object model).
* It is used for project build, dependency, and documentation.

# Why Maven?

* **Adding a set of Jars in each project is easier.**
* **Creating the right project structure**
* **Integration with other tools for build and deployment**
* **Easy updates to new versions of jar files**

# Maven POM.xml file

**POM** is an acronym for **Project Object Model**. The pom.xml file contains information on the project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals, etc.

|  |  |
| --- | --- |
| Element | Description |
| Project | It is the root element of pom.xml file. |
| modelVersion | It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0. |
| groupId | It is the sub element of project. It specifies the id for the project group. |
| artifactId | It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source |
| Version | It is the sub element of project. It specifies the version of the artifact under given group. |

**<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.google.application1**</groupId>**

**<artifactId>**my-app**</artifactId>**

**<version>**1**</version>**

**</project>**

## Additional elements

|  |  |
| --- | --- |
| Element | Description |
| packaging | defines packaging type such as jar, war etc. |
| name | defines name of the maven project. |
| url | defines url of the project. |
| dependencies | defines dependencies for this project. |
| dependency | defines a dependency. It is used inside dependencies. |
| Scope | defines scope for this maven project. It can be compile, provided, runtime, test and system. |

**<packaging>**jar**</packaging>**

**<name>**Maven Quick Start Archetype**</name>**

**<url>**http://maven.apache.org**</url>**

**<dependencies>**

**<dependency>**

**<groupId>**junit**</groupId>**

**<artifactId>**junit**</artifactId>**

**<version>**4.8.2**</version>**

**<scope>**test**</scope>**

**</dependency>**

**</dependencies>**

# Maven Build life cycle

Maven build cycle comprises of:

|  |  |  |
| --- | --- | --- |
| Phase | Handles | Description |
| prepare-resources | resource copying | Resource copying can be customized in this phase. |
| validate | Validating the information | Validates if the project is correct and if all necessary information is available. |
| compile | Compilation | Source code compilation is done in this phase. |
| Test | Testing | Tests the compiled source code that is suitable for the testing framework. |
| package | Packaging | This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml. |
| install | Installation | This phase installs the package in local/remote maven repository. |
| Deploy | Deploying | Copies the final package to the remote repository. |

# Maven Phase plugin and goals

|  |  |  |
| --- | --- | --- |
| Phase | Plugin | Goals |
| Clean | Maven Clean Plugin | clean |
| Compile | Maven Compiler Plugin | compile |
| Test | Maven Surefire Plugin | test |
| Package | Varies based on the packaging. Ex Maven JAR plugin | jar |
| Install | Maven Install plugin | install |

Here, the plugin is a jar file containing class files with methods that are goals and these goals call methods that will perform the required task such as clean or compile operation.

Each phase is invoking one or more goals which is associated with a plugin.

# Maven Archetypes

Maven provides users, a very large list of different types of project templates (614 in numbers) using the concept of **Archetype**. Maven helps users to quickly start a new java project using the following command.

mvn archetype:generate

What is Archetype?

Archetype is a Maven plugin whose task is to create a project structure as per its template. We are going to use quickstart archetype plugin to create a simple java application

mvn archetype:generate

* Maven will start processing and will ask to choose the required archetype.
* Press Enter to choose the default option (1843: maven-archetype-quickstart)
* Maven will ask for a particular version of archetype.
* Press Enter to choose the default option (8: maven-archetype-quickstart:1.4)
* Now Maven will start creating the project structure

# Creating a project in Maven

* Maven uses **archetype** plugins to create projects.
* To create a simple java application, we'll use maven-archetype-quickstart plugin.

mvn archetype:generate "-DgroupId=com.google" "-DartifactId=CubeGenerator" "-DArtifactId=simple"

This will create a default file with default java code files.

To compile the code go the location with the project (Ex: **C:\Users\SSS IT\CubeGenerator)**

mvn clean compile

To run go to the target folder in (Ex: **C:\Users\SSS IT\CubeGenerator\target\classes)**

java com.google.App

Jar file is created in the target folder(From location : **C:\Users\SSS IT\CubeGenerator)**:

java -classpath target\CubeGenerator-1.0-SNAPSHOT.jar;.; com.javatpoint.App

## Building and packaging a maven project

The **mvn package** command completes the build life cycle of the maven project such as:

1. validate
2. compile
3. test
4. package
5. integration-test
6. verify
7. install
8. deploy

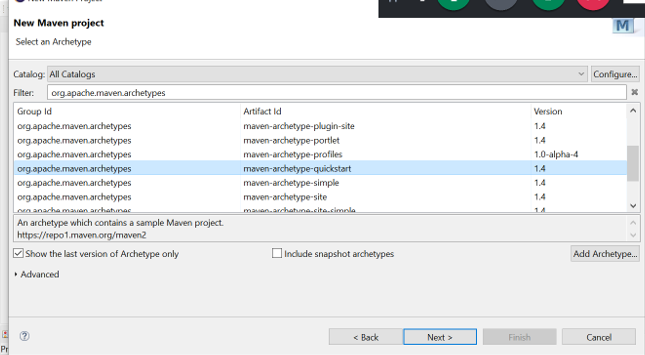
Execute the command mvn package from project location (Ex: C:\Users\MEGHANA\Downloads\apache-maven-3.8.3-bin\maven\CubeGenerator)

A jar file is created in the target folder.

# Running and configuring Maven Test

To create a Maven Project,

* Go to New 🡪 Project 🡪 Maven 🡪 Maven Project
* Select the below archtype



* Click Next
* Enter groupID, artifactID and package details
* Group ID 🡪 ID of the organization (domain name in reverse order)
* Artifact ID 🡪 Project Name
* Package 🡪 Group ID + Artifact ID
* Add JUnit dependency in pom.xml
* Update the Maven project by doing Right click 🡪 Maven 🡪 Update Project
* To run, right click 🡪 Run as 🡪 Maven Test
* Check the reports in target 🡪 surefire-reports

Maven Tests can be executed using the command prompt as well

* Go to the location where maven project is saved.
* Run the command to execute the test

mvn test

# Dependency Scope

There are two types of dependencies in Maven: direct and transitive. Direct dependencies are the ones that we explicitly include in the project.

These can be included using *<dependency>* tags

On the other hand, transitive dependencies are required by direct dependencies. Maven automatically includes required transitive dependencies in our project.

We can list all dependencies including transitive dependencies in the project using *mvn dependency:tree* command.

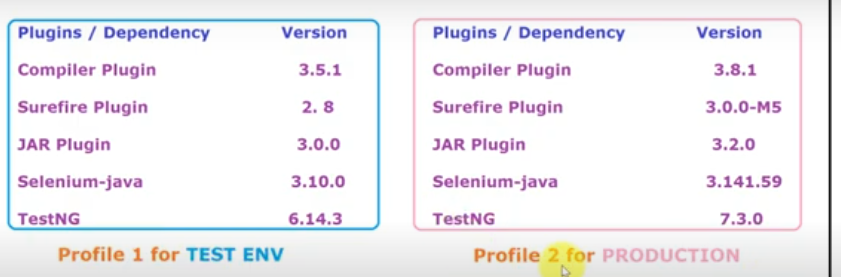
## Dependency scope

1. **Compile** - Dependencies with this scope are available on the classpath of the project in all build tasks
2. **Provided** - We use this scope to mark dependencies that should be provided at runtime by JDK
3. **Runtime** - The dependencies with this scope are required at runtime**.** But we don’t need them for the compilation of the project code.

A JDBC driver is a good example of dependencies that should use the runtime scope

1. **System** - System scope is very similar to the provided scope. The main difference is that system requires us to directly point to a specific jar on the system.
2. **Test** - The standard use case for this scope is adding a test library such as JUnit to our application.

# Maven Profiles



[Maven](https://www.baeldung.com/maven) profiles can be used to create customized build configurations.

Creating profiles:

1. Adding profiles in pom.xml
2. Add Profiles and profile tags with plugins and dependencies.

Ex:

<profiles>

<profile>

<id>TestEnv</id>

<build>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>3.2.0</version>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.10</version>

<scope>test</scope>

</dependency>

</dependencies>

</profile>

</profiles>

1. If we run the Maven project as a Maven test without mentioning the profile, it will run with the default profile.
2. In order to run a maven test with a particular profile, right-click on maven project 🡪 maven build and provide the below command

install -P TestEnv

where **TestEnv** is the name of the profile.

If running from the command prompt, provide the following command

mvn install -P TestEnv

# Site life cycle

Maven site lifecycle handles everything related to generating documentation for a project.

|  |  |
| --- | --- |
| pre-site | execute processes needed prior to the actual project site generation |
| Site | generate the project’s site documentation |
| post-site | execute processes needed to finalize the site generation, and to prepare for site deployment |
| site-deploy | deploy the generated site documentation to the specified web server |

