X. Maven

Apache Maven is a java based software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven provides d a complete build lifecycle framework for project management. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

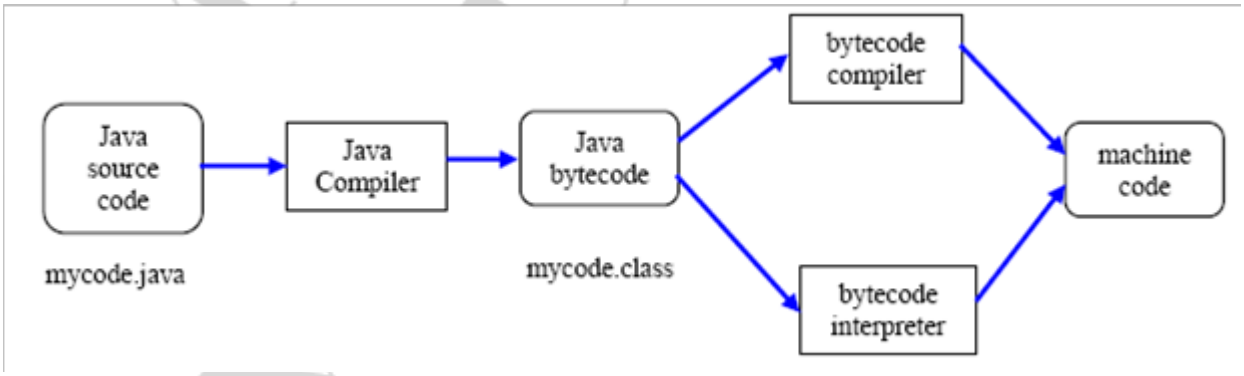
In abstract, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of build related tasks.

1. Build process

The "Build" is a process that covers all the steps required to create a deliverable product of your software into preproduction and production. In the Java world, this typically includes:
Generating source.
Compiling sources.
Executing tests (unit tests, integration tests, etc).
Packaging (into jar, war, ejb-jar, ear).
Running health checks (static analyzers like Checkstyle, Findbugs, PMD, test coverage, etc).
Generating reports.
A defined build process is an essential part of any development cycle because it helps close the gap between the development, integration, test, and production environments. A build process alone will speed the migration of software from one environment to another. It also removes many issues related to compilation, classpath, or properties that cost many projects time and money.

Compilation and execution

Compilation and execution of a Java program is two step processes. During compilation phase Java compiler compiles the source code and generates bytecode. This intermediate bytecode is saved in form of a .class file. In second phase, Java virtual machine (JVM) also called Java interpreter takes the .class as input and generates output by executing the bytecode.



Various build tools available:

12. For java - Ant,Maven,Gradle.
13. For .NET framework - NAnt
14. c# - MsBuild.

2. Ant vs Maven vs Gradle:

Ant

Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. Ant can also be used effectively to build non Java applications, for instance C or C++ applications. More generally, Ant can be used to pilot any type of process which can be described in terms of targets and tasks.

It has very low learning curve thus allowing anyone to start using it without any special preparation. It is based on procedural programming idea. After its initial release, it was improved with the ability to accept plug-ins.

Major drawback was XML as the format to write build scripts. XML, being hierarchical in nature, is not a good fit for procedural programming approach Ant uses. Another problem with Ant is that its XML tends to become unmanageably big when used with all but very small projects.

Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Its goal was to improve upon some of the problems developers were faced while using Ant. Maven continues using XML as the format to write build specification. However, structure is diametrically different.

While Ant requires programmer to write all the commands that lead to the successful execution of some task whereas Maven relies on conventions and provides the available targets (goals) that can be invoked. As the additional, and probably most important addition, Maven introduced the ability to **download dependencies over the network** (later on adopted by Ant through Ivy). Main benefit from Maven is its life-cycle. As long as the project is based on certain standards, with Maven one can pass through the whole life cycle with relative ease. This comes at a cost of flexibility.

Now the interest for DSLs (Domain Specific Languages) continued increasing. The idea is to have languages designed to solve problems belonging to a specific domain. In case of builds, one of the results of applying DSL is Gradle and for e.g. gradle is used in Android for build and packaging.

Gradle

Gradle aims to help organizations ship better software, faster. Faster builds is one of the most direct ways of achieving this;Gradle combines good parts of both tools and builds on top of them with DSL and other improvements. It has Ant's power and flexibility with Maven's life-cycle and ease of use. For example, Google adopted Gradle as the default build tool for the Android OS.

Gradle does not use XML. Instead, it had its own DSL based on Groovy (one of JVM languages). As a result, Gradle build scripts tend to be much shorter and clearer than those written for Ant or Maven. The amount of boilerplate code is much smaller with Gradle since its DSL is designed to solve a specific problem: move software through its life cycle, from compilation through static analysis and testing until packaging and deployment.

3. Understanding the common problem without Maven

There are many problems that we face during the project development. They are discussed below:

- **Adding set of Jars and dependencies in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.
- **Creating and maintaining the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.
- **Building and Deploying the project:** We must have to build and deploy the project so that it may work.

4. What it does?

Maven simplifies and give solution to the above-mentioned problems. It performs mainly following tasks.

- It makes a project easy to build
- It provides uniform build process (maven project can be shared by all the maven projects)
- It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
- It is easy to migrate for new features of Maven

5. What is Build Tool

A build tool takes care of everything for building a process. It does following:

- Generates source code (if auto-generated code is used)
- Generates documentation from source code
- Compiles source code
- Packages compiled code into JAR of ZIP file
- Installs the packaged code in local repository, server repository, or central repository

6. Uses of Apache Maven

- Use as Build Tool
- Use as to manage Project structure
- Building, publishing and deploying
- Documentation
- Reporting
- Releases
- Distribution

7. Setup and Installation for Maven

You can download and install maven on windows, Linux and MAC OS platforms.

Note: Maven is Java based tool, so the very first requirement is to have JDK installed on your machine.

Download Maven archive

https://maven.apache.org/download.cgi

OS	Archive name
Windows	apache-maven-3.5.0-bin.zip
Linux	apache-maven-3.5.0-bin.tar.gz **or sudo apt-get install maven**
Mac	apache-maven-3.5.0-bin.tar.gz

Extract the Maven archive

Extract the archive, to the directory you wish to install Maven 3.5.0. The subdirectory apache-maven-3.5.0 will be created from the archive.

OS	Location (can be different based on your installation)
Windows	C:\Program Files\Apache Software Foundation\apache-maven-3.5.0
Linux	/usr/local/apache-maven
Mac	/usr/local/apache-maven

Set Maven environment variables

Add M2_HOME, M2, MAVEN_OPTS to environment variables.

OS	Output

Windows	Set the environment variables using system properties. M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.5.0 M2=%M2_HOME%\bin MAVEN_OPTS=-Xms256m -Xmx512m
Linux	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.5.0 export M2=$M2_HOME/bin export MAVEN_OPTS='-Xms256m -Xmx512m'
Mac	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.5.0 export M2=$M2_HOME/bin export MAVEN_OPTS=-Xms256m -Xmx512m

Add Maven bin directory location to system path

Now append M2 variable to System Path

OS	Output
Windows	Append the string ;%M2% to the end of the system variable, Path.
Linux	export PATH=$M2:$PATH
Mac	export PATH=$M2:$PATH

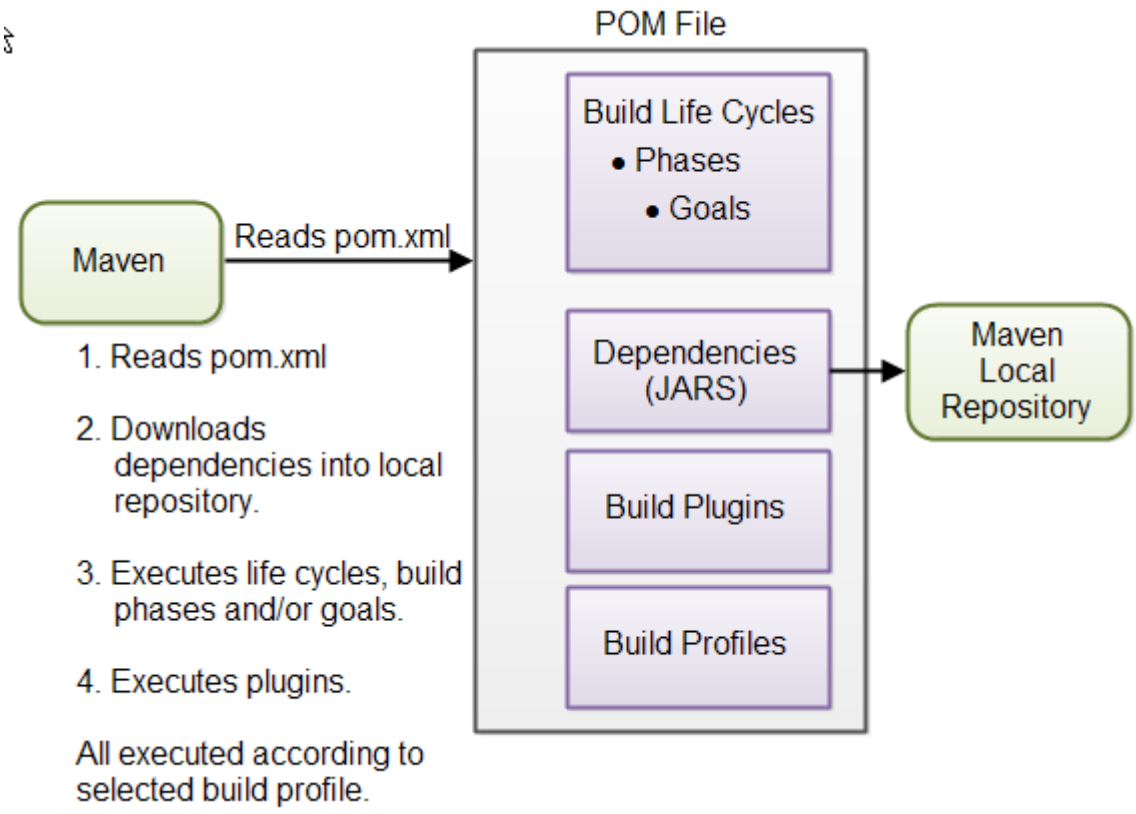
Verify Maven installation

Now open console, execute the following mvn command.

OS	Task	Command
Windows	Open Command Console	c:\> mvn --version
Linux	Open Command Terminal	$ mvn --version
Mac	Open Terminal	machine:~ joseph$ mvn --version

Maven help in project structure ,dependencies management,the reason behind is that maven repositories,maven talk to repositories and form the project structure.

Note: Maven get all its knowledge from maven repositories,how will be the project structure,what is the project type etc.



Maven uses the concept of the repository to hold the jar files. There are two types of repository namely local and remote repository.

Every maven project has a pom.xml file. If u run Your application with maven then, Now it checks your local repository for the jar files. If the jar files are not found it goes to the remote repository and download the jar. After downloading jars it will Execute life cycle, build phases and/or goals.

8. First Sample Application:

(Here ,I use Linux System)

1.Open Command Prompt ctrl+Alt+T

Command : $mvn archetype:generate

(when you run this for 1st time it download lots of maven plugins)

2.choose a number (What selection number is ?)

There are number of archetype, the number is the specific number of a archetype that is available to use.Then the project structure will form accordingly.

For example:

1718: remote -> org.springframework.boot:spring-boot-sample-jetty-archetype (Spring Boot Jetty Sample)
1719: remote -> org.springframework.boot:spring-boot-sample-profile-archetype (Spring Boot Profile Sample)

But here we will go with default which is 981:
(Choose org.apache.maven.archetypes:maven-archetype-quickstart version:)

9. Maven pom.xml file

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

Maven reads the pom.xml file, then executes the goal.

→ In Our case the generated project has this pom.xml file.

```
<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>java.org.wahid</groupId>
  <artifactId>MavenTestApp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>MavenTestApp</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

The simple pom.xml file, contains following elements:

Element	Description
project	It is the root element of pom.xml file.
modelVersion	It is the sub element of project. It specifies the modelVersion. Which is set to 1.0.0.
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 and binary distributions, and WARs.
version	It is the sub element of project. It specifies the version of the artifact under given group.

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.

3. Compile Maven Project:

To compile maven project go to the directory where pom.xml file is there,then execute the command

$ mvn compile

Then it will download all the dependencies and compile the project.after compile project you will a target directory is created which having all the compile classes

4. Package Maven Project:

For Packaging the project just run following command

$ mvn package

So that it will package the project into mentioned format jar, war etc.

Note: When we package the project, it will automatically run test Junit Cases and give the Build result.

After package you see target folder contains: classes, maven-archiver , surefire , surefire-reports test-classes, TestMavenApp-1.0-SNAPSHOT.jar

```
Building jar: /home/vave/workspace/Maven-Project/TestAppMaven/target/TestAppMaven-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

5.To run The Project :

$ java -cp target/TestMavenApp-1.0-SNAPSHOT.jar com.eqv.wahid.App

```
Hello World!
```

6.To run Test Cases :

$ mvn test

After the executing this command it will find the test cases and execute it and give the result

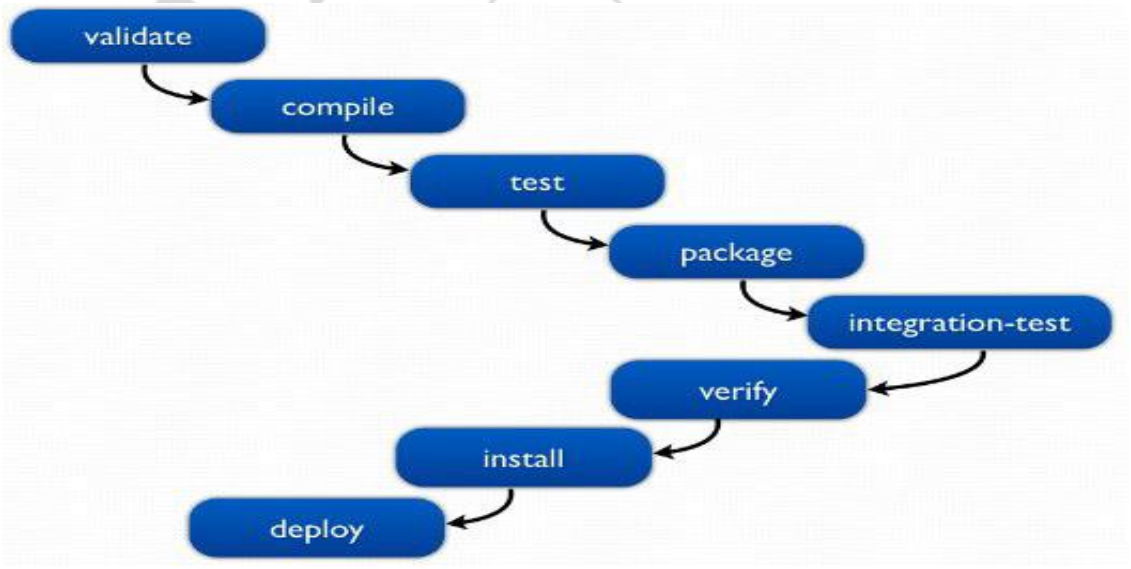
```
--- maven-surefire-plugin:2.10:test (default-test) @ TestMavenApp ---
[INFO] Surefire report directory: /home/vave/workspace/Maven-
Project/TestMavenApp/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.eqv.wahid.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.059 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

10. Maven - Build LifeCycle

A Build Lifecycle is a well-defined sequence of phases which define the order in which the goals are to be executed. Here phase represents a stage in life cycle.
As an example, a typical Maven Build Lifecycle consists of following sequence of phases

A Build Lifecycle is Made Up of Phases:

Each of these build lifecycles is defined by a different list of build phases, where in a build phase represents a stage in the lifecycle.



For example, the default lifecycle comprises of the following phases :

- **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.

Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar,wa), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

Summary:

- ✓ Maven is a software build management and comprehension tool.
- ✓ Maven is based on the concept of a project object model, Maven helps in code build, dependency management, documentation creation, site publication, and distribution publication are all controlled from the declarative file.
- ✓ Maven can be extended by plugins to utilise a number of other development tools for reporting or the build process.
- ✓ Maven Archetype is a set of tools to deal with archetypes, i.e. an abstract representation of a kind of project that can be instantiated into a concrete customized Maven project. An archetype knows which files will be part of the instantiated project and which properties to fill to properly customize the project.