
Apache
maven

❖ APACHE-MAVEN:

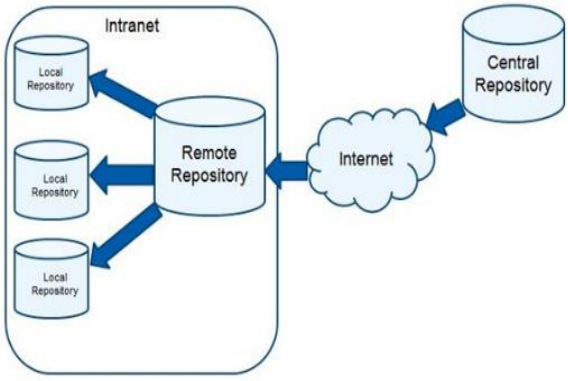
- Apache Maven is an open-source **software project management** and **comprehension tool.**
- It was originally started as an attempt to simplify the build processes in the Jakarta Turbine project.
- Maven can manage a project's build, reporting and documentation from a central piece of information.

MAVEN OBJECTIVIES:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

➢ MAVEN REPOSITORIES:

- A repository in Maven holds build artifacts and dependencies of varying types.
- It is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.
- **Maven repositories are:** Local Repository, Central Repository & Remote Repository



LOCAL REOSITORY:

- Maven local repository keeps your project's all dependencies (library jars, plugin jars etc.).
- Maven automatically downloads all the dependency jars into the local repository when you run it.
- It helps to avoid references to dependencies stored on remote machine every time a project is build.

CENTRAL REPOSITORY:

- It is located on the web. It has been created by the Apache maven community itself.
- The path of central repository is: **https://repo.maven.apache.org/maven2/**
- It contains a lot of common libraries that can be viewed by this url**: http://search.maven.org/**

REMOTE REPOSITORY:

- Maven remote repository is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in **pom.xml** file.

➢ BUILD TOOLS:

- These are helps to create an **executable application** from the **source code.**
- The build tool is needed for the following processes:
  - Generating source code
  - Generating documentation from the source code
  - Compiling source code
  - Packaging the compiled codes into JAR files
  - Installing the packaged code in the local repo, server, or central repo.

STAGES OF BUILD:

src -->

Compile-->

Package it -->

Archive --> Deploy to server/JVM

➢ MAVEN INSTALLATION ON CENTOS / RHEL 9:

STEP1: To update and change the hostname:

#yum update -y

#hostname Maven

#vim /etc/hostname

Maven

STEP 2: Security-Enhanced Linux is being disabled or in permissive mode:

#sed -i 's/SELINUX=.*/SELINUX=disabled/g' /etc/selinux/config

#setenforce 0

STEP 3: Installing Java-21 Package

#yum install java-21 -y

#java --version

STEP 5: Download Binary file

#cd /opt

#wget https://dlcdn.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz

#tar -xzvf apache-maven-3.9.2-bin.tar.gz

#ls

Rename extracted directory:

#mv apace-maven-3.9.2 maven3

#ls

STEP6: Setting maven Paths:

#echo $PATH

#vim ~/.bashrc

M2_HOME=/opt/maven3

M2=/opt/maven3/bin

export PATH=$PATH:$M2_HOME:$M2

#source ~/.bashrc

#mvn --version

➢ BUILDING A PROJECT:

Maven Archetype:

- Archetype is a Maven project templating toolkit. An archetype is defined as an original pattern or model from which all other things of the same kind are made.

NOTE: Maven provides several archetype artifacts

https://maven.apache.org/archetypes/

MAVEN CORE CONCEPTS (POM FILE):

- Maven is centered around the concept of POM file (Project Object Model).
- A POM file is an XML representation of project resources like source code, test code, dependencies (External JARs used) etc.
- It contains a detailed description of your project, including information about versioning and configuration management, dependencies, application and testing resources, team members and structure, and much more.
- POM file should be located in the root directory of the project it belongs to.
- Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on.

POM.XML FILE ATTRIBUTES:

- Before maven 2, it was named as **project.xml** file. But, since maven 2 (also in maven 3), it is renamed as **pom.xml**.

  PROJECT: It is the root element of pom.xml file.

  MODEL VERSION: It is a model version of the project.

  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.

➢ MAVEN BUILD LIFECYCLE PHASES:

- Maven is based around the central concept of a build lifecycle. What this means: building and distributing a particular artifact is clearly defined.

  VALIDATE: validate the project is correct & all necessary info 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.

➢ GENERATE A NEW PROJECT:

STEP 1: Setting Up the Directory Structure:

#mvn archetype:generate -DgroupId=sysgeeks.app -DartifactId=/opt/maven-app -DarchetypeArtifactId=maven-archetype-quickstart -Dinteractivemode=false

#cd /opt/maven-app

#ls -l

#tree

- The following is an explanation of what each directory's purpose is:
  Building a Jar File:

`src`	This will contain all types of source files under the following subdirectories.
`src/main`	This is intended for the main part of the application - the code that would be part of the final distributable.
`src/main/java`	This is the java code that makes up the application. Below this should be a package structure.
`src/main/resources`	These are additional resources for copying into the final distributable. This may have a subdirectory structure that is maintained, including using a package structure. For example, you might have a `META-INF/MANIFEST.MF` file in here (although Maven does create a default one for you so this isn't usually necessary).
`src/test`	This contains everything needed to unit test your application. You may have additional similar directories later if you add other types of tests such as integration tests using Cactus.
`src/test/java`	This is the java code that makes up the unit tests for the application. Below this should be a package structure.
`src/test/resources`	As for `src/main/resources`, but only made available to the unit tests. Not used in this example.
`xdocs`	This contains the documentation that will be transformed into HTML and published as a project site. **Note**: by defaulting the xdocs location to the top level directory, Maven 1.x violates the directory structure conventions adopted in Maven 2, where it defaults to `src/site/xdoc`. You can make your project layout compatible with Maven 2 by overriding the `maven.docs.src` property.

STEP 2: Create a Project description:

- The most important file to Maven is project.xml. While you can run Maven without it, it will not know anything about your project - so is only useful for project-independant goals.

#vim pom.xml

```
<project xmlns="http://maven.apache.org/POM/3.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/3.0.0
http://maven.apache.org/maven-v3_0_0.xsd">
  <pomVersion>3</pomVersion>
  <groupId>sample</groupId>
  <artifactId>sample-echo</artifactId>
  <name>Sample</name>
  <currentVersion>1.0-SNAPSHOT</currentVersion>
  <inceptionYear>2005</inceptionYear>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.8</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
    <unitTestSourceDirectory>src/test/java</unitTestSourceDirectory>
    <resources>
      <resource>

        <directory>src/main/resources</directory>
      </resource>
    </resources>
    <unitTest>
      <includes>
        <include>**/*Test.java</include>
      </includes>
    </unitTest>
  </build>
</project>
```

- **The following goals will perform some standard behaviors:**

  maven java:compile: This will compile the code and check for errors - nothing more.

  maven test: This will compile the code and tests, then run all of the unit tests

  maven jar: This will build a JAR from your code, after running the tests as above

  maven site: Even now, you can generate a site in target/docs and see what it will look like

- **COMMON TASKS:**

Cleans the Maven project by deleting the target directory:

#mvn clean

Compiles the Java source classes of the Maven project:

mvn compiler:compile

To compiles the test classes of the Maven project:

#mvn compiler:testcompile

To builds the Maven project and packages it into a JAR, WAR, etc.:

#mvn package

#ls taget

To builds the Maven project and installs the project files (JAR, WAR, pom.xml, etc.) to the local repository:

#mvn install

To deploys the artifact to the remote repository:

#mvn deploy

validates the Maven project to ensure that everything is correct and all the necessary information is available:

#mvn validate

To analyzes the maven project to identify the unused declared and used undeclared dependencies:

#mvn dependency:analyze

runs the test cases of the project:

#mvn test

To clean up after a build and start fresh, simply run:

#mvn clean

#ls target