



Apache Maven is a powerful build automation tool primarily used for Java projects. It simplifies the process of building and managing Java-based projects by providing a standard way to manage project dependencies, build processes, and project documentation.

Maven is a powerful project management tool that is based on POM (project object model). It simplifies the build process like ANT. But it is too much advanced than ANT. Maven, a Yiddish word meaning accumulator of knowledge.

PYTHONLIFE



## Maven Introduction:

Apache Maven is a cornerstone of Java development, and the *most used build management tool for Java*. Maven's streamlined, XML-based configuration model enables developers to rapidly describe or grasp the outlines of any Java-based project, which makes starting and sharing new projects a snap. Maven also supports test-driven development, long-term project maintenance, and its declarative configuration and wide range of plugins make it a popular option for CI/CD. This article is a quick introduction to Maven, including the Maven POM and directory structure, and commands for building your first Maven project.

**Meant for:** Individuals and small DevOps teams.

### **Key features:**

Developer-centricity: Maven streamlines new developer ramp-up. It also supports multiple coding languages, with a large developer community available for support.

Cloud readiness: It is compatible with software development projects in all environments, including major public clouds, thanks to its open-source architecture.

CI/CD compatibility: It includes a mechanism for technology reusability between projects to address backward compatibility issues for CI/CD.

Ease of integration: It can be integrated with virtually any platform, computing environment, or DevOps tool through open-source code.

Collaboration support: Maven supports multiple concurrent projects and helps create centralized repositories and code across distributed teams.

The biggest unique selling point (USP) of Apache Maven is its open-source nature and its ease of use. With Apache infrastructure looking after the build, you can get started with Maven in as little as ten minutes.

**Pricing:** Maven is free for use.

## Java Project Structure:

### Steps/Process Involved in Building a Project

Here are the steps to follow when building a Maven project:

- Add or write the code to create the application creation, and process it into the source code repository.
- Edit any necessary configuration / pom.XML / plugin details.

- Build the actual application.
- Save your build process output as either a WAR or EAR file to a local server or other location.
- Access the file from the local location or server and deploy it to the production or client site.
- Update the application document by changing the date and updated application version number, if necessary.
- Create and generate a report as requested for the application or the requirement.



## Problems without Maven:

Following are the problems discussed below which are faced without Maven:

**1) Adding a set of Jars in each project:** Set of jar files are to add in each project whether the case of spring, struts or hibernate frameworks. Also, all the dependencies of jars are to be included.

**2) Creating the right project structure:** Right project should be created in struts or servlet so as to execute it properly.

**3) Building and Deploying the project:** To make the project work properly, it should be built and deployed properly.

## Maven's Objectives

The primary goal of Maven is to allow a user to comprehend the complete state of a development effort in the shortest period of time.

- Makes the build process easier
- Provides a uniform build system
- Provides quality project information
- Provides guidelines for best development practices
- Allows transparent migration to new features



## What Maven Does:

Maven does a lot of helpful task like

- We can easily build a project using maven.
- We can add jars and other dependencies of the project easily using the help of maven.
- Maven provides project information (log document, dependency list, unit test reports etc.)
- Maven is very helpful for a project while updating central repository of JARs and other dependencies.
- With the help of Maven, we can build any number of projects into output types like the JAR, WAR etc. without doing any scripting.
- Using maven, we can easily integrate our project with source control system (such as Subversion or Git)

## What is Build tool:

A build tool is essential for the process of building, as these are the tools that automate the process of creating applications from the source code. The build tool compiles and packages the code into an executable form.

A build tool does the following tasks:

- Generates source code.
- Generates documentation from the source code.
- Compiles source code.
- Packages the compiled codes into JAR files.
- Installs the packaged code in the local repository, server, or central repository.

## Xml File:

POM stands for **Project Object Model**. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml. And has all the configuration settings for the project build.

Maven reads the pom.xml file, then executes the goal. Before maven 2, it was named as project.xml file. But, since maven 2 (also in maven 3), it is renamed as pom.xml.

Elements of maven pom.xml file

For creating the simple pom.xml file, you need to have following elements:

Element	Description
<b>project</b>	It is the root element of pom.xml file.
<b>Model Version</b>	It is the sub element of project. It specifies the model Version. It should be set to 4.0.0.
<b>groupId</b>	It is the sub element of project. It specifies the id for the project group.
<b>artifactId</b>	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.
<b>version</b>	It is the sub element of project. It specifies the version of the artifact under given group.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<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.javatpoint.application1</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>

  </project>

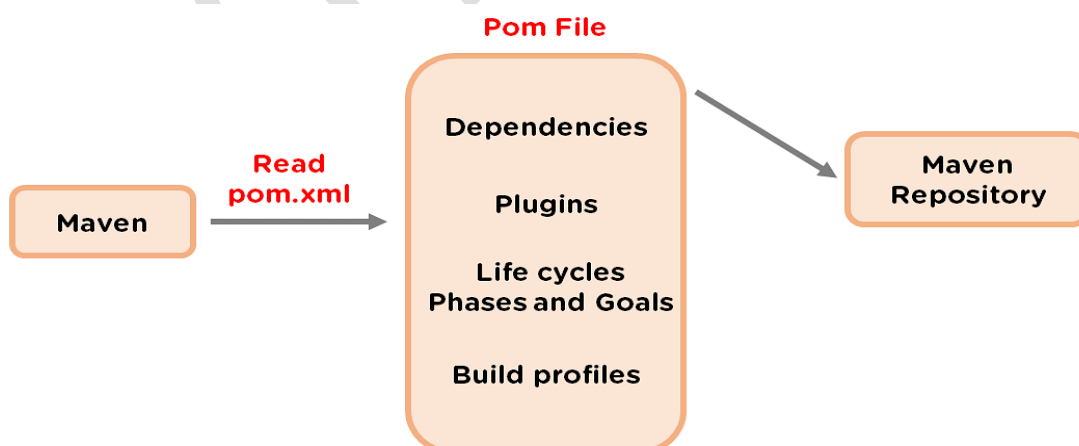
```

#### Requirements for Build:

1. Source Code: Present in work space
2. Compiler :Remote repo □Local repo □Work space
3. Dependencies: Remote repo □Local repo □Work space

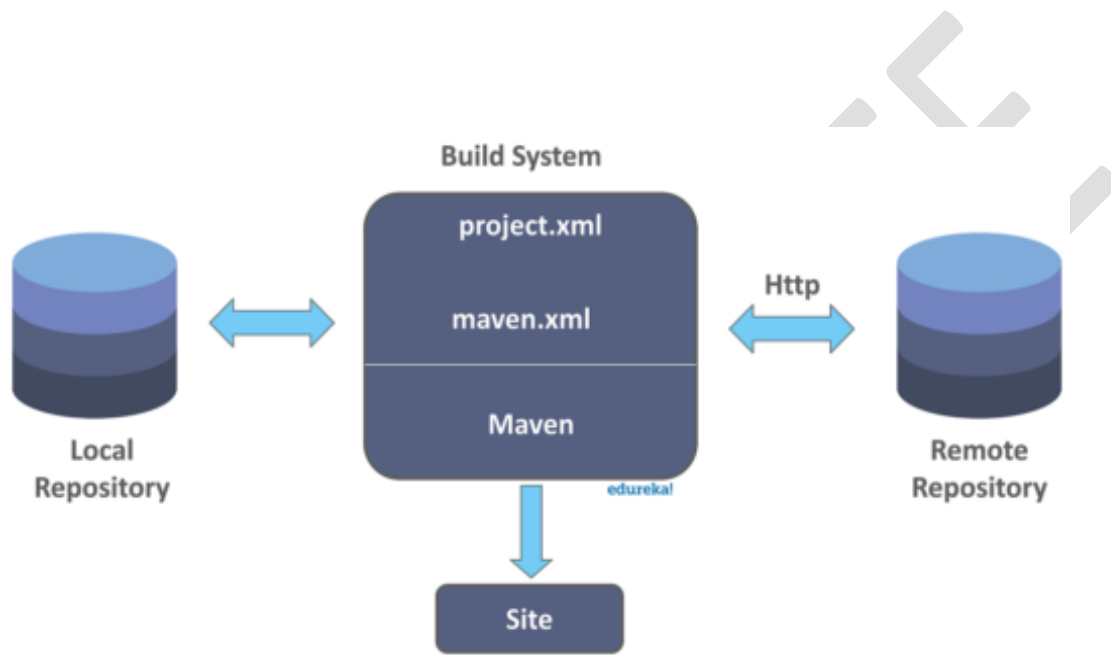
#### Maven Architecture:

The projects created in Maven contain POM files that describe the aspect of the project essentials. Maven architecture shows the process of creating and generating a report according to the requirements and executing lifecycles, phases, goals, plugins, and so on—from the first step.



#### How Does the Maven Architecture Work?

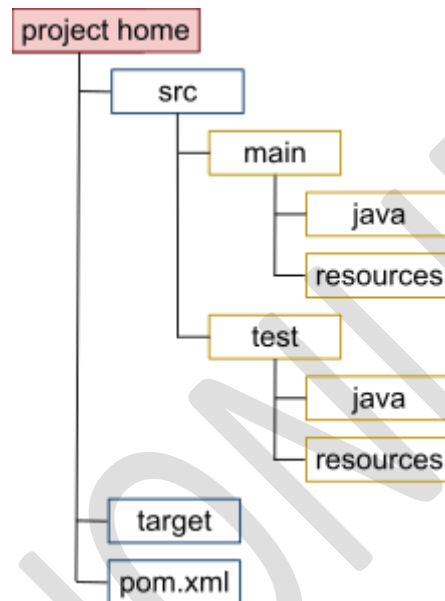
1. The first step refers to configuring Maven, which is stored in a pom.xml file.
2. The POM file includes all of the configurations that Maven needs. The second step is to download the dependencies defined in pom.xml into the local repository from the central repository.
3. After the user starts working in Maven, the tool provides various default settings, so there is no need to add every configuration in the pom.xml.





## Maven Directory Structure:

Maven has own standard for directory structure to create in a project. So we no need specify any directory on the project level for sources, resources, tests, plugins etc. And also we don't need to configure for creating directory structure on the pom.xml file it is internal functionality of maven.



## Generating War File:

The Maven WAR plugin is responsible for collecting and compiling all the dependencies, classes, and resources of the web application into a web application archive.

There are some defined goals in the Maven WAR plugin:

war: This is the default goal that is invoked during the packaging phase of the project. It builds a WAR file if the *packaging* type is *war*.

exploded: This goal is normally used in the project development phase to speed up the testing. It generates an exploded web app in a specified directory.

inplace: This is a variant of the *exploded* goal. It generates an exploded web app inside the web application folder.

Let's add the Maven WAR plugin to our *pom.xml* file:

```
<plugin>
  <artifactId>maven-war-
plugin</artifactId>
  <version>3.3.1</version>
</plugin>
```




Now, once we execute the *mvn install* command, the WAR file will be generated inside the target folder.

Using the *mvn:war:exploded* command, we can generate the exploded WAR as a directory inside the target directory. This is a normal directory, and all the files inside the WAR file are contained inside the exploded WAR directory.

### Important Concepts:

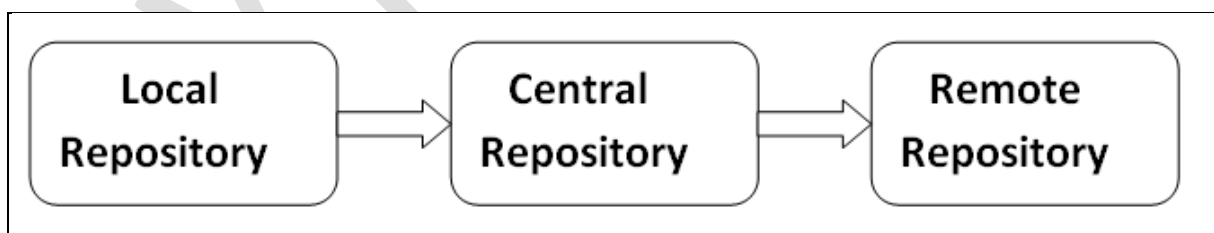
#### 1. Maven Repository

A maven repository is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

-  Local Repository
-  Central Repository
-  Remote Repository

Maven searches for the dependencies in the following order:

Local repository then Central repository then Remote repository.



The local repository is a directory on the developer's machine where Maven stores downloaded dependencies and plug-in. By default, it is located in the *.m2* directory within the user's home directory.

When Maven downloads artifacts from remote repositories for the first time, it caches them in the local repository. Subsequent builds can then access these artifacts locally without needing to re-download them, improving build performance and allowing offline development.



## Remote Repository

Remote repositories are server-based repositories that host project dependencies, plugins, and other artifacts. These repositories are accessible over the internet and provide a centralized location for developers to access commonly used libraries and tools.

The Central Repository (Maven Central) is the default remote repository used by Maven. It contains a vast collection of open-source Java libraries and plug-in.

Additionally, organizations and projects may host their own remote repositories to distribute proprietary libraries or custom artifacts internally.

Repository is where the build artifacts are stored. Build artifacts means, the dependent files (Ex: dependent jar files) and the build outcome (the package we build out of a project).

There are two types of repositories, local and remote. Local maven repository(.m2) is in the user's system. It stores the copy of the dependent files that we use in our project as dependencies. Remote maven repository is setup by a third party(nexus) to provide access and distribute dependent files. Ex: repo.maven.apache.org from internet.

### 2. Maven Dependencies:

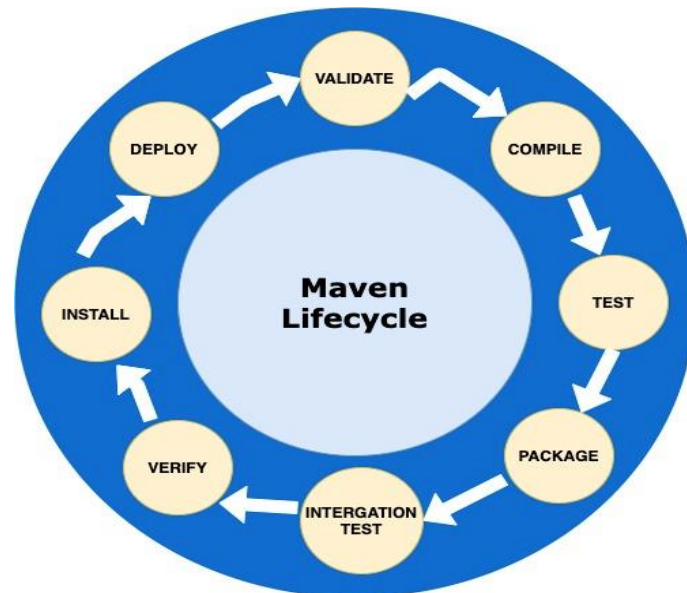
There is an element available for declaring dependencies in project pom.xml. This is used to define the dependencies that will be used by the project. Maven will look for these dependencies when executing in the local maven repository. If not found, then Maven will download those dependencies from the remote repository and store it in the local maven repository.

### 3. Maven Plugins

All the execution in Maven is done by plugins. A plugin is mapped to a phase and executed as part of it. A phase is mapped to multiple goals. Those goals are executed by a plugin. We can directly invoke a specific goal while Maven execution. A plugin configuration can be modified using the plugin declaration.

## MAVEN BUILD LIFE CYCLE:

In Maven, a build process is organized into predefined phases called the build lifecycle. Each lifecycle consists of a sequence of phases, and each phase represents a specific stage in the build process. When a Maven command is executed, it traverses through these phases in a predetermined order, executing associated goals and tasks



**Clean:** Deletes the target directory, removing all compiled classes, generated files, and other artifacts created during the build process.

**Validate:** Validates the project's structure and configuration, ensuring it meets Maven's standards.

**Compile:** Compiles the project's source code into binary class files.

**Test:** Runs unit tests using a testing framework such as JUnit.

**Package:** Packages compiled code and resources into distributable format (e.g., JAR, WAR).

**Install:** Installs the generated artifact into the local repository for use in other projects.

**Deploy:** Copies the generated artifact to a remote repository for sharing with other developers or deployment to production servers.

## Difference between Ant and Maven



Ant and Maven both are build tools provided by Apache. The main purpose of these technologies is to ease the build process of a project.

Ant	Maven
Ant <b>doesn't</b> has <b>formal conventions</b> , so we need to provide information of the project structure in build.xml file.	Maven <b>has a convention</b> to place source code, compiled code etc. So we don't need to provide information about the project structure in pom.xml file.
Ant is <b>procedural</b> , you need to provide information about what to do and when to do through code. You need to provide order.	Maven is <b>declarative</b> , everything you define in the pom.xml file.
There is <b>no life cycle</b> in Ant.	There is <b>life cycle</b> in Maven.
It is <b>a tool</b> box.	It is <b>a framework</b> .
It is <b>mainly a build tool</b> .	It is <b>mainly a project management tool</b> .
The ant scripts are <b>not reusable</b> .	The maven plugins are <b>reusable</b> .
It is <b>less preferred</b> than Maven.	It is <b>more preferred</b> than Ant.

## MAVEN INSTALLATION:

### Maven Installation in Windows:

```
-----
--> install java above 7.1 version
--> Download java JDK & JRE (or)
http://www.oracle.com/technetwork/java/javase/downloads/index.html
--> Go to-->mycomputer-->properties-->Advanced system settings--
>environment variables-->system variables
--> path ;C:\Program Files\Java\jdk1.8.0_131\bin;C:\Program
Files\Java\jre1.8.0_131\bin;D:\Apache_Maven\bin ---> to system
variables PATH by seperater ;
        JAVA_HOME should point to JDK(without bin)

--> install Maven
Go to this website to downloab(Zip)--> maven.apache.org/download.cgi
        D:\Apache_Maven ---> MAVEN_HOME in system variables
        path--> ;D:\Apache_Maven\bin
```

### maven installation in ec2

=====

1. yum update -y  
sudo -i
2. Login to instance as a root user and install Java(openJdk)



```
- /usr/sbin/alternatives --config java
Note:- it'll show you how many java versions are installed in your
machine and you can select which one want to use.
- sudo su - root (or) sudo su - (or) sudo -i
- yum install java-17-openjdk-devel -y
- java and javac (Java Compiler)
- java -version
- which java
- whereis java
- ls -l /usr/bin/java
- ls -l /etc/alternatives/java
- java path --->> /usr/lib/jvm/jre-17-openjdk-17.0.5.0.8-
2.el9_0.x86_64/bin/java
- yum list installed | grep java
3. Install Maven in /opt directory
   cd /opt
4. Download apache maven
   wget https://dlcdn.apache.org/maven/maven-3/3.8.7/binaries/apache-
maven-3.8.7-bin.tar.gz
   note:- yum install wget -y (by default wget package will not
install in redhat Linux)

5. tar -xvzf apache-maven-3.9.0-bin.tar.gz
6. for permanent configuration
   vi /etc/profile.d/maven.sh
   export MAVEN_HOME=/opt/apache-maven-3.8.7
   export PATH=$PATH:$MAVEN_HOME/bin
6. source /etc/profile.d/maven.sh
mvn --version

o/p:- Apache Maven 3.8.7
(1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-17T19:33:14+01:00)
Maven home: /usr/local/src/apache-maven
Java version: 9.0.4, vendor: Oracle Corporation, runtime:
/opt/java/jdk-9.0.4
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.17.6-1.el7.elrepo.x86_64",
arch: "amd64", family: "unix"

verify whether java/maven is installed or not in CMD prompt by typing
below commands
Javac --> compiler
java --> keyword
java -version --> runtime environment
mvn --version
```