**Apache Maven** is a very popular choice for most of the Java based application development. Before Maven existed, there was a necessity for:

* A well-defined project structure to place contents of the project
* A uniform way to build projects
* A simple method to publish the project information and a convenient way to make JARs shared across different projects

**The primary aim of Apache Maven is to:**

* Simplify the process of build activity of projects
* Manage the dependencies efficiently and use the resources optimally
* Providing guidelines for best practices in project development

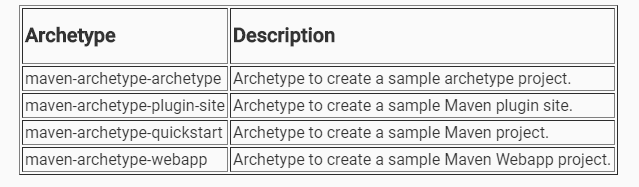
**Apache Maven** is an open source software project management tool that helps the developer in creation, building and management of Java based projects.

Maven is often considered to be an **Intelligent,** **Comprehensive**, **Project management**and also as a **Build tool**.

# Maven Archetype: -

# Archetype in Maven is a project templating toolkit. Archetype decides the folder structure of any Maven project.

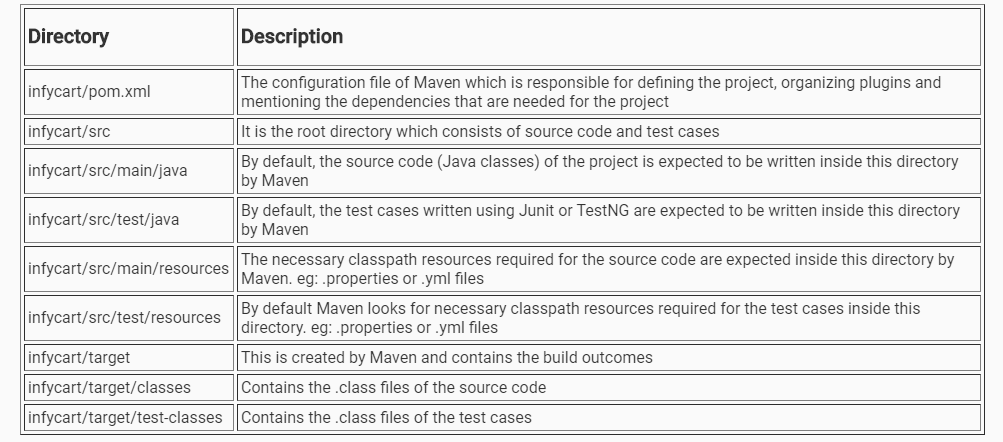
* Archetype helps maintaining the consistency among Maven projects
* Archetype enables developers to stick onto the best practices employed by the project/organization
* Archetype helps developers arrive at Maven projects in a fraction of seconds



**Note:**If we are in a network where firewall restrictions exist, a settings.xml file having appropriate configuration to maven repository needs to be placed in the user's .m2 folder.

If connected to open network, this step can be skipped completely.

**Standard Directory Structure of a Maven-based Java Project**



# ****What is POM?****

* **Project Object Model** is the most important, logical unit of work in any Maven project.
* It is basically an XML configuration file named pom.xml with the information about project and the necessary configurations.
* pom.xml is often described as the declarative descriptor of Maven projects. It is mandatory for this file to be placed in the root folder of the project.
* It not only describes what to build, but, how to build as well.

A Maven project's pom.xml has the following sections:

1. **Project Information:** Mandatory section which contains the project co-ordinates to uniquely identify the project.
2. **Dependencies:** Optional section which refers to the libraries/artifacts/jars needed for the application.
3. **Build Plugin:** Optional section which refers to the plugin configurations needed for the application.
4. **Build Profile:**Itis anoptional section. This section usually has the various configuration settings using which the build can be customized for different environments - production, test, development, etc.

Every Maven project is uniquely identified using the following set of co-ordinates:

**groupId :**

* This element normally has group name or organization name which owns the project
* Typically, the naming convention followed is the reverse of the organization's domain name. For example,
  + com.infosys.demos - contains the list of all related projects owned by infosys
  + org.apache - contains the list of all related projects owned by apache

**artifactId:** This element gives the information about the name of the application. eg: infycart

**version:**

* This element reveals the version of the distribution unit (jar/war/ear).
  + “**SNAPSHOT**” is the general convention used to mention that the project is in active development phase. eg: infycart-**0.0.1 - SNAPSHOT**.jar
  + **RELEASE** is the convention used to mention that the build of the project is the base-lined, stable version. eg: Spring Context - **4.3.14.RELEASE**.jar

**packaging:** This element decides the type of the distribution unit. eg: jar/war/ear etc.

# ****What is a repository?****

Repository in Maven is referred to as Artifactory for the following reasons:

* It is the source where artifacts required for building the projects are located
* It is the target where artifacts generated out of build process are deployed

**Local repository** refers to the location/folder in the developer’s machine where the copy of dependencies gets downloaded and stored.

* When Maven starts downloading the project artifacts for the very first time, local repository automatically gets created. Hence, it need not be created manually.
* If the dependencies needed for the project are already present in the local repository, Maven allows the developers to work offline.  Hence local repository makes usage of network optimally.
* We are free to clean the contents of local repository as it will get filled again, when the next Maven download happens.
* When the project build occurs, local repository is contacted first. Only when some dependencies are found missing, Maven will take a chance to contact remote (if configured) or central repository to get the contents downloaded. This reduces the time being consumed during the build process to a great extent. Also, disk space is utilized effectively because multiple projects in the machine refer to this centralized folder for dependencies.
* However, we can have the local repository anywhere in the local file system based on our preference. Just add the below entry in settings.xml to customize the local repository location.

**Remote repository** as the name infers, this repository is usually a location which is remote to the developer's machine.

* Maven requires Intranet/Internet connection to access Remote repository
* To resolve the dependencies for project builds, Maven tries to find the dependency in remote repository only in the absence of dependencies in the local repository. To connect to remote repository, Maven requires extra configuration either in pom.xml or settings.xml
* Remote repository gets content from the central repository and this is highly abstracted from the developers whose projects contact the remote repository

**Central Repository** is the public website/repository given by Maven. It has all the most-commonly used libraries for project builds.

* Central repository exists outside the developer's machine
* Internet connection is mandatory to access Central repository
* Maven contacts the Central repository when build process for the project occurs for the very first time. But, this is true only when the developer is in an open network. If the developer is behind any firewall, then the following configuration should be enabled in the settings.xml to access Central repository

**<proxy>**

**<active>true</active>**

**<protocol>http</protocol>**

**<username>proxyuser</username>**

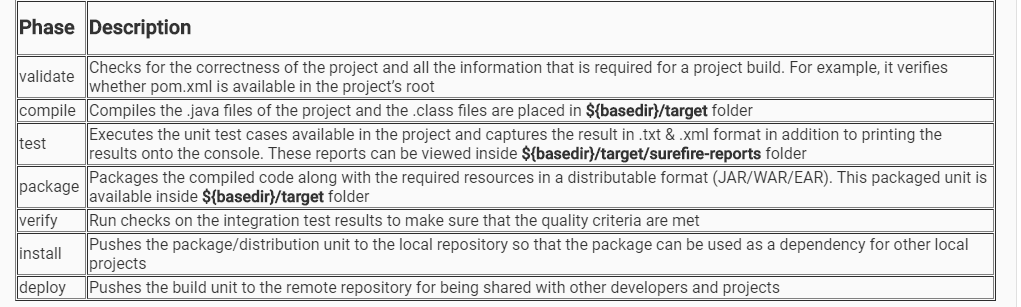
**<password>proxypass</password>**

**<host>proxy.host.net</host>**

**<port>80</port>**

**</proxy>**

**Maven Life Cycle: -**



**Maven Plugins and Goals: -**

Some of the available Maven plugins are as follows:

* **maven-compiler-plugin:** has goals for compiling source code and unit tests
* **maven-surefire-plugin:** has goals for execution of unit tests and publishing its reports
* **maven-jar-plugin:** has goals for creating JAR files
* **maven-clean-plugin:** has goal for the deletion of target folder
* One of the notable observations is, none of the dependencies mentioned in the pom.xml is bundled with this JAR by default. If these dependencies are also to be included, **maven-assembly-plugin** needs to be used.

**exec-maven-plugin:**  helps in executing Java programs and has the following important goals:

* exec - executes programs and Java programs in a separate process.
* java -  executes Java programs in the same VM.

**maven-install-plugin:**is used during the install phase to add artifact(s) to the local repository. The install Plugin uses the information in the POM (groupId, artifactId, version) to determine the proper location for the artifact within the local repository. It has the following important goals:

* install - is used to install the project's main artifact (the JAR, WAR or EAR), its POM and attached artifacts if any (sources, javadoc, etc) in the local repository.
* install-file - is used to install an externally created artifact into the local repository, along with its POM.
* help - displays help information on maven-install-plugin.