In this blog on Maven tutorial, we will cover the following topics:

1. Why do we need Maven?
2. What is Maven?
3. Maven Architecture
4. Maven life cycle, phases and goals
5. Demo project.

Why do we need Maven?

If you are working on Java projects then most of the time you need dependencies. Dependencies are nothing but libraries or JAR files. You need to download and add them manually. Also, the task of upgrading the software stack for your project was done manually before Maven. So there was a need for a better build tool that would handle such issues.

This where Maven comes into the picture. Maven can solve all your problems related to dependencies. You just need to specify the dependencies and the software version that you want in pom.xml file in Maven and Maven will take care of the rest. So now let us try to understand what exactly Maven is.

What is Maven?

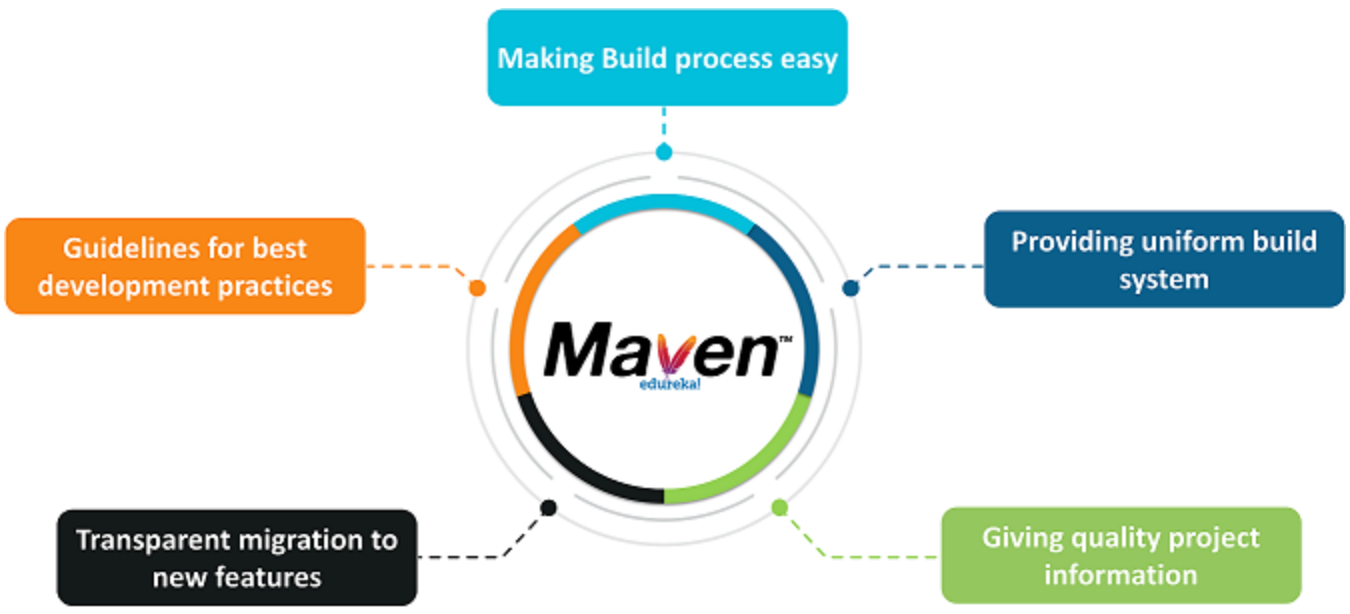
The Maven project is developed by Apache Software Foundation where it was formerly a part of the Jakarta project. Maven is a powerful build automation tool that is primarily used for Java-based projects. Maven helps you tackle two critical aspects of building software –

- It describes how software is built
- It describes the dependencies.

Maven prefers convention over configuration. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven Central Repository and stores them in a local cache. The artifacts of the local projects can also be updated with this local cache. Maven can also help you build and manage projects written in C#, Ruby, Scala, and other languages.

Project Object Model(POM) file is an XML file that contains information related to the project and configuration information such as dependencies, source directory, plugin, goals, etc. used by Maven to build the project. When you execute a maven command you give maven a POM file to execute the commands. Maven reads the pom.xml file to accomplish its configuration and operations.

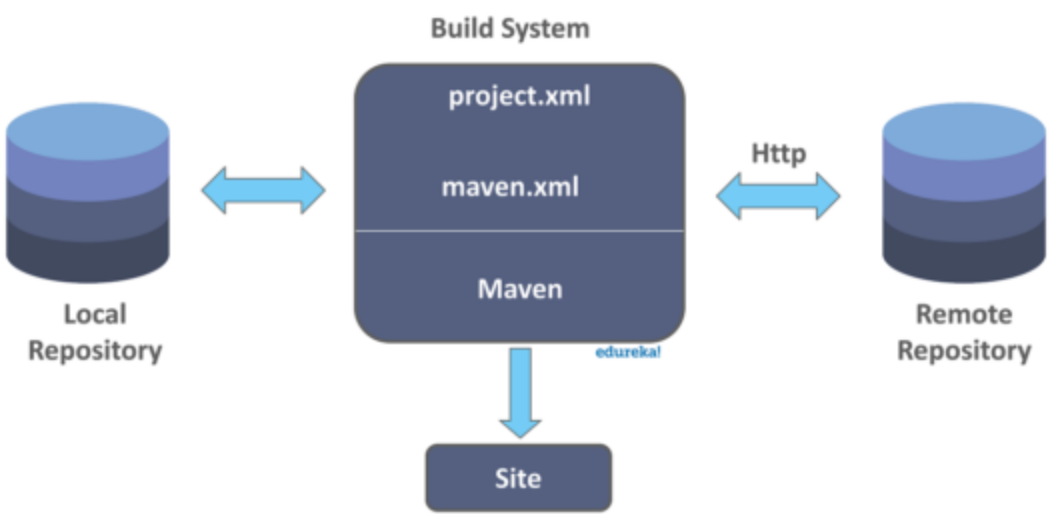
Maven Objectives



When should someone use Maven?

1. If there are too many dependencies for the project.
2. When the dependency version update frequently.
3. Continuous builds, integration, and testing can be easily handled by using maven.
4. When one needs an easy way to generate documentation from the source code, compiling the source code, packaging compiled code into JAR files or ZIP files.

Maven Architecture



Maven life cycle, phases and goals

1. Maven life cycle

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

edureka!

There is a specific life cycle that Maven follows to deploy and distribute the target project.

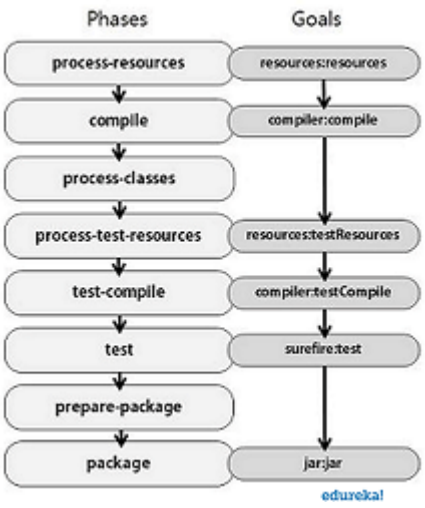
There are three built-in life cycles:

- **default** – This is the main life cycle of Maven as it is responsible for project deployment.
- **clean** – This life cycle is used to clean the project and remove all files generated by the previous build.
- **site** – The aim of this life cycle is to create the project's site documentation.

Each life cycle is made up of a sequence of phases. The default build life cycle consists of 23 phases as it is the main build life cycle of Maven

On the other hand, clean life cycle consists of 3 phases, while the site life cycle is made up of 4 phases.

2. Maven Phases



A Maven phase is nothing but a stage in the Maven build life cycle. Each phase executes a specific task.

Here are a few important phases in the default build life cycle –

- **validate** – This phase checks if all information necessary for the build is available
- **compile** – This phase compiles the source code
- **test-compile** – This phase compiles the test source code
- **test** – This phase runs unit tests
- **package** – This phase packages compiled source code into the distributable format (jar, war)
- **integration-test** – This phase processes and deploys the package if needed to run integration tests
- **install** – This phase installs the package to a local repository
- **deploy** – This phase copies the package to the remote repository

Maven executes phases in a specific order. This means that if we run a specific phase using the command such as mvn <phase>, this won't only execute the specified phase but all the preceding phases as well.

For example, if you run the command mvn deploy, that is, the deploy phase which is the last phase in the default build life cycle, then this will execute all phases prior to the deploy phase as well.

3. Maven Goals

A sequence of goals constitutes a phase and each goal executes a specific task. When you run a phase, then Maven executes all the goals in an order that are associated with that phase. The syntax used is plugin:goal. Some of the phases and the default goals bound to them are as follows :

- compiler:compile – compile phase
- compiler:test – test-compile phase
- surefire:test – test phase
- install:install – install phase
- jar and war:war – package phase

A Maven plugin is a group of goals. However, these goals aren’t necessarily all bound to the same phase. For example, the Maven Failsafe plugin which is responsible for running integration tests. For unit testing, you need Maven surefire plugin.