**MAVEN**

* Maven is a project management and comprehension tool, which is based on project object model(**POM**). It can manage project build, reporting and documentation from a central piece of information. We can build and manage java projects using maven.
* The advantage of maven is:
* Unlike other build tools, It gives you a default directory structure to place your source code instead of creating a directory structure manually.
* **Pom.xml** = It is an xml file which contains information about the project and configurations used by maven to build the project. It also contains goals and plugins. While executing, maven looks for pom file in current directory and reads the file and gets needed configuration information and executes the goals as defined in the pom file.
* In pom.xml, you will see all your configurations like **packaging** (jar,war, ear, pom), **group** **ID**, **artifact** **ID**, **name**, **version**, **plugins**, **dependencies** etc**.**
* You can add any new **dependencies**, **plugins** in pom file if you want for the project.
* Maven actually runs on plugin extension framework. Where every task in maven is done by plugins. A plugin provides a set of goals, which can be executed.
* There are 2 types of plugins:
* **Build** **plugins** = these plugins execute during the build stage and should be configured in **<build/>** elements of pom.xml.
* **Reporting** **plugins** = these plugins execute during the site generation and should be configured in **<reporting/>** elements of pom.xml.
* After configuring, maven will download the plugins from the central repository.
* You need to pass a goal to maven and if there is any plugin corresponding to the goal, maven will execute it.
* You have to define a plugin like this

**<groupId>name</groupId>**

**<artifactId>artifact</artifactId>**

**<version>1.0.1</version>**

* You can also define phase below this plugin. So maven will execute this plugin in that phase.

**<phase>compile</phase>**

**MAVEN REPOSITORIES**

* Maven have 3 types of repositories.

**LOCAL :**

* By default, maven stores all your **dependencies**(**plugins**, **jars** and other downloaded by maven) in a local folder. In simple words, when you build a maven project, all the dependency files will be stored in **maven** **local** **repository**. It will store as versions based on your snapshot versions.
* The default local folder in linux is **~/.m2**. In this directory, you can see all your dependencies installed by maven.
* This local repository will be created after you build your first project.
* You can change this default repo dir by adding a line in setting.xml file.
* Go to settings.xml file, add a line below local repository arrow line.

**<localRepository>/path/to/repo</localRepository>**

**CENTRAL :**

* It is a repository which is provided by **maven** **community**. It contains large number of commonly used libraries.
* When maven doesn’t find any dependency in local repository , it searches in central repo using this url **-** [**https://repo1.maven.org/maven2/**](https://repo1.maven.org/maven2/)**.**

**REMOTE:**

* If maven doesn’t find dependency in both local and central repo, it gives you an error by stoping the build. To prevent this, maven provides concept of remote repository, which is our own custom repository containing dependencies and other jars.
* We have to mention remote repo url in pom file and it will search the remote repo for dependencies while building the project.
* Maven will search for dependencies from local repo to central, if its there in central, it will download the dependency. If its not there, it will search in remote only if we configured remote repo in pom.xml file.
* Once we configured the remote repo, it will search and download the dependencies to local.

**BUILD LIFE CYCLE**:

* Maven is based around the central concept of build lifecycle. It is the sequence of the phases(stages), which define the order to execute the goals.
* The phases are:
* **Validate** = validating the information.
* **Compile** = Compiling the source code. (compile means converting your source code into machine–code (or) low-level code in which the program can be executed).
* **Test** = Testing the compiled source code.
* **Packaging** = Create jar/war files as mentioned in pom.xml.
* **Integrated test =** Process and deploy the package into an environment if necessary, where integrate test can run. Test results are stored in **/target/surefire-reports** dir.
* **Verify =** Run any check-ups to verify the package is valid and meets quality criteria.
* **Install** = Install the package in local/maven repo**(.m2).**
* **Deploy** = Copy the final package to maven repo.
* Maven follow this life cycle to build a project. If you do maven install directly, it won’t skip the above stages, it will perform all the stages above install stage and finally installs it.
* To install maven, we need java. So, Install java first, set home path for java and download maven from official website, extract and set home path.
* Test wether maven is installed (or) Not = **mvn --version.**
* You can use maven commands along with the stage names…**mvn** **install**, **mvn** **package** etc.
* There is another goal in maven called “**clean**”. It is used to clean(remove) the working directory. You can run clean with maven commands…….**mvn clean.**
* **mvn clean install =** cleans the target dir and install once again newly and saves in local repo.
* When you installed maven for the first time, you don’t get any directory structures and pom.xml. you have to generate a sample(**free-sytle**) project.
* This free-style project is going to create a dummy project for you which will have all directory structure and pom.xml, which we can use as base for all the projects.
* To create a dummy project = **mvn archetype:generate**
* It’s just creating a template. You can use this template for your projects.
* When you type the generate command, you will be seen asking some values
* **Choose a number** = you have to specify a number based on the type of your project. The number list will be shown above. After you mentioned a number maven will create a dummy project based on the your specifications. If you don’t know what type of project you want leave it default.
* **Version (or) apply filter** = leave it default for maven.
* **groupID** = it’s just your company name (or) business name (your wish).
* **ArtifactID** = name of the deliverable artifact.
* **Version** = artifact version.
* **Package** = type of package (jar, war, ear). By default, it will take jar.
* It will show you all the configuration specified and click **y** if you want to proceed with those configuration.
* You can also generate architecture type with a command as an options.

**mvn archetype:generate –Dgroupid = groupname –Dartifactid = artifcatname –Darchetypartifactid = type of maven project –Dversion = version –Dinteractivemode = false**

* Once you click YES, maven will create a directory structure based on the name you specified for artifactID.
* Go to that directory, you can see pom.xml and src directory where you have to copy all your source code.
* Your **source** code should be in **src/main/java**.
* Your **unit** **test** code should be in **src/test/java**.
* **Pom.xml** should be in the root folder.
* You have to run maven commands from where the pom.xml file is located . When you run maven commands, it first looks for pom.xml file. Based on the configuration inside the pom.xml maven will build the project. If there is no pom.xml in that dir, build will be failed.
* Maven will follow the above directory structure for every project. You should follow and place your code according to that structure.
* Once the dummy project is created, you can use maven lifecycle commands to build the project. Maven will pick the code according to its directory structure.
* When you build a maven project, it will create a **target** directory under your project directory. This target directory contains all your **builds, classes** and **test** **reports**.

**MULTI MODULE PROJECTS:**

* You can create and build multiple modules in a project with the help of maven.
* To create modules, create a dir inside your project dir and copy (or) move pom and src dir to that module dir. Now you have a module in your project.
* You will have separate pom.xml for each modules and you can go to each module dir and build the modules manually.
* Instead of doing this manually, you can automate building all modules by writing a pom.xml for parent project.
* In the parent pom.xml, you have to specify the packaging as **pom** and specify **modules** to build sequentially.

**<modules>**

**<module>module1</module>**

**<module>module2</module>**

**</modules>**

* In the above example, we have mentioned all our modules in parent pom.xml and maven will build these modules in order.

**DEPENDENCY MANAGEMENT:**

* If you have dependencies for each project in maven. the dependencies will be build first and the project will build later.
* For ex, you have 2 modules 1 and 2. Module2 is dependent on module1.
* You have to add module1 **GAV** in module2 pom.xml under dependencies section.

**<dependency>**

**<groupId>reddy</groupId>**

**<artifactId>mod1</artifactId>**

**<version>1.0.2-SNAPSHOT</version>**

**</dependency>**

* So, when you tries to build module2 , it will check for module1 **jar** in **local** and **remote** repo. If module1 jar is present in local/remote repo, it will take the jar and build module2. If it’s not present in local/remote repo module2 build will fail.
* If you want to build these modules using parent pom along with the dependencies, specify parent’s **GAV** in all modues pom by creating parent section.

**<parent>**

**<groupId>reddy</groupId>**

**<artifactId>eswar</artifactId>**

**<version>1.0.3-SNAPSHOT</version>**

**</parent>**

* Now parent will have full control over modules and it will check for dependencies in pom.xml for all modules before building them. If there are any dependencies for a module, it will build the dependency first and build the module after this. We called it as “**maven reactor**”.