MAVEN

A VCS plays a vital role in any kind of organization, the entire software industry is built around code.

What are we doing with this code ??

- Are we seeing the code when we open the application ?? NO
- Are we seeing the code when we open the app in browser ?? NO

So we are seeing the executable format of the code, that is called build result of the code.

What is build ??
==============
Build is the end result of your source code.

Build tool is nothing but, it takes your source code and converts it into human readable format (executable).

Build
====
The term build may refer to the process by which source code is converted into a stand-alone form that can be run on a computer.

One of the most important steps of a software build is the compilation process, where source code files are converted into executable code.

The process of building software is usually managed by a build tool i.e, maven.

Builds are created when a certain point in development has been reached or the code has been ready for implementation, either for testing or outright release.

Build: *Developers write the code, compile it, compress the code and save it in a compressed folder. This is called Build.*

Release: *As a part of this, starting from System study, developing the software and testing it for multiple cycles and deploy the same in the production server. In short, one release consists of multiple builds.*

Maven Objectives
===============
- A comprehensive model for projects which is reusable, maintainable, and easier to comprehend(understand).
- plugins

Convention over configuration
==========================
Maven uses *Convention* over *Configuration* which means developers are not required to create build process themselves. Developers do not have to mention each and every configuration detail.

Earlier to maven we had ANT, which was pretty famous before maven.

Disadvantages of ANT
==================
ANT - Ant scripts need to be written for building
[build.xml need to tell src & classes]
ANT - There is no dependency management
ANT - No project structure is defined

Advantages of Maven
==================
No script is required for building [automatically generated - pom.xml]
Dependencies are automatically downloaded
Project structure is generated by maven
Documentation for project can be generated

Maven is called as **project management tool** also, the reason is earlier when we used to create projects and we used to create the directory structure and all by yourself, but now maven will take care of that process.

MAVEN has the ability to create project structure.
Maven can generate documentation for the project.

Whenever i generate a project using maven i will get src and test all by default.

MAVEN FEATURES
================

Dependency System
=================
Initially in any kind of a build, whenever a dependency is needed, if i’m using ANT i have to download the dependency then keep it in a place where ANT can understood.

If i’m not giving the dependency manually my build will fail due to dependency issues.

Maven handles dependency in a beautiful manner, there is place called MAVEN CENTRAL. Maven central is a centralized location where all the dependencies are stored over web/internet.

For ex im using a project and i'm having a dependency of junit, whenever my build reaches a phase where it needs junit then it will download the dependencies automatically and it will store those dependencies in your machine. There is a directory called as .m2 created in your machine, where all the dependencies are going to be saved. Next time when it comes across the same dependency, then it doesn't download it coz its already available in .m2 directory.

Plugin Oriented
=============
Maven has so many plugins that i can integrate, i can integrate junit, jmeter, sonarqube, tomcat, cobertura and so many other.

IMPORTANT FILE IN MAVEN
=======================
Projects in maven is defined by **POM** (Project Object Model) pom.xml.

Maven lifecycle phases
===================

What is build life cycle?
The sequence of steps which is defined in order to execute the tasks and goals of any maven project is known as build lifecycle in maven.

The following are most common *default* lifecycle phases executed:

- **validate**: validate the project is correct and all necessary information[dependencies] are available and keep it in local repo
- **compile**: compile the source code of the project
- **test**: Execution of unit tests, 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 to verify the package is valid and meets quality criteria, keeps the HelloWorld.jar in .m2 local repo
- **install**: Deploy to local repo [.m2], install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects. This will push the libraries from .m2 to remote repo.

There are two other Maven lifecycles of note beyond the *default* list above. They are

- clean: cleans up artifacts created by prior builds
- site: generates site documentation for this project

These lifecycle phases are executed sequentially to complete the default life cycle.

POM { will be in XML format }
====

GAV

Maven uniquely identifies a project using:

- **groupID**: Usually it will be the domain name used in reverse format (going to be given by the project manager).
- **artifactID**: This should be the name of the artifact that is going to be generated
- **Version** : Version of the project, Format {Major}.{Minor}.{Maintenance} and add "-SNAPSHOT" to identify in development
  Version can be two things here, a SNAPSHOT and other is RELEASE.

Snapshot - whenever your project is in working condition i mean we are still working on it that would be a snapshot version, you can have multiple snapshots for one single project.
But there would be only one release for it, for example after my 20th snapshot we decided that 8th snapshot should goto release then will remove -SNAPSHOT for 8th.
In real time we will get the basic pom which is already written with groupid, artifactid and version, we can build up required plugins and dependencies.

Packaging
=========
Build type is identified by <packaging> element, this element will tell how to build the project.

Example packaging types: jar, war etc.

Archetype
=========
Maven archetypes are project templates which can be generated for you by Maven.
In other words, when you are starting a new project you can generate a template for that project with Maven.
In Maven a template is called an *archetype*.
Each Maven archetype thus corresponds to a project template that Maven can generate.
Installation
==========

Maven is dependent on java as we are running java applications, so to have maven, we also need to have java in system.

Install java
=========
Java package : java program --- java-1.8.0-openjdk
Java package : java compiler --- java-1.8.0-openjdk-devel

```
    # sudo yum -y install java-1.8.0-openjdk
    # sudo yum -y install java-1.8.0-openjdk-devel
    # java -version        { confirm java version}
```

Install Maven
===========

```
    # yum -y install maven
```

Maven repository are of three types
==============================
For maven to download the required artifacts of the build and dependencies (jar files) and other plugins which are configured as part of any project, there should be a common place. This common shared area is called as Repository in maven.

Local
=====
The repository which resides in our local machine which are cached from the remote/central repository downloads and ready for the usage.

Remote
=======
This repository as the name suggests resides in the remote server. Remote repository will be used for both downloading and uploading the dependencies and artifacts.

Central
======
This is the repository provided by maven community. This repository contains large set of commonly used/required libraries for any java project. Basically, internet connection is required if developers want to make use of this central repository. But, no configuration is required for accessing this central repository.

How does Maven searches for Dependencies?
=======================================
Basically, when maven starts executing the build commands, maven starts for searching the dependencies as explained below :

- It scans through the local repositories for all the configured dependencies. If found, then it continues with the further execution. If the configured dependencies are not found in the local repository, then it scans through the central repository.

- If the specified dependencies are found in the central repository, then those dependencies are downloaded to the local repository for the future reference and usage. If not found, then maven starts scanning into the remote repositories.

- If no remote repository has been configured, then maven will throw an exception saying not able to find the dependencies & stops processing. If found, then those dependencies are downloaded to the local repository for the future reference and usage.

Setting up stand alone project
========================

```
        # cd ~
        # mvn archetype:generate          { generates project structure }
        # we get some number like 1085 beside it we have 2xxx, which means maven
currently supports 2xxx project structures, 1085 is like default project
        # press enter
        # choose a number :: 6 which means latest, so press enter
        # groupId: com.digital.academy    { unique in world, generally domain }
        # artifactId: project1            { project name }
        # version: press enter            { snapshot - intermediate version }
        # package: enter { package name is java package }
        # Press: y
```

Now maven has successfully created a project structure for you:

```
        # tree -a project1
```

We have pom.xml which contains all the definitions for your project generated, this is the main file of the project.

```
# ls -l ~/.m2/repository
```

```
        # mvn validate        { whatever in the pom.xml is correct or not }
```

Let’s make some mistakes and try to fail this phase,

```
        # mv pom.xml pom.xml.bk
        # mvn validate        { build failure }
        # mv pom.xml.bk pom.xml
        # vi App.java         { welcome to Devops }
        # mvn compile         { after changing code we do compilation right }
```

{ this generates a new structure - # tree -a . with class files}

# **mvn test** { test the application }

# **mvn package** { generates the artifact - jar }

# **java -cp target/xxxx.jar groupid(com.digital.proj1).App**

Plugins
=======

We saw maven is only performing phases like validate, compile, test, package, install, deploy but if you remember there is no execution of a jar file,

Can you see any of the phases running jar file, no right ??

Executing jar file is not part not the part of life cycle,
apart from the above phases such as validate, compile, test, package, install, deploy all the other come under <build> under <plugins> </plugins>

These plugins will define, other then regular maven lifecycle phases,

Let's take example, i want to run my jar file,
After </dependencies> in your pom.xml

After </dependencies> in your pom.xml add the following

```
<build>
<plugins>
<plugin>
<groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.2.1</version>
  <configuration>
    <mainClass>com.digi.App</mainClass>
        <arguments>
              <argument>-jar</argument>
              <argument>target/*.jar</argument>
         </arguments>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

In pom.xml after </dependencies> add <build> <plugins> <plugin>

We need to run

# mvn exec:java

WEB APP SETUP

Setting up web project

===================

```
# mvn archetype:generate | grep maven-archetype-webapp
# type the number you get
# tree -a project/
# mvn clean package
# tree -a project
```

You can see the **war** generated under target directory

Tomcat Installation [Binaries]

=========================

Google tomcat 7 download

```
# goto tomcat downloads page and get the binary tar file for the tomcat 7 by wget
# wget <link>
# wget http://www-us.apache.org/dist/tomcat/tomcat-7/v7.0.82/bin/apache-tomcat-7.0.82.tar.gz
# tar xf apache-tomcat-7.0.82.tar.gz
# cd apache-tomcat-7.0.82/bin
# ./startup.sh
# Check for port to be opened in firewall
# netstat -ntpl { # sudo yum -y install net-tools }
# ps -ef | grep tomcat
```

```
# Goto http://ip-address/8080 {click cancel and change tomcat-users.xml file}
        <role rolename="manager-gui"/>
        <user username="tomcat" password="tomcat" roles="manager-gui"/>
        <user username="tomcat1" password="tomcat1" roles="manager-script"/>
# Change the port number in server.xml
# cd apache-tomcat-7.0.81/bin
# ./shutdown.sh
# Copy the generated war file to webapps dir of tomcat
# Refresh the tomcat page
```

Deploy to tomcat maven tomcat plugin

=====================================

Add Manager-Script Role

=====================

Add new role under conf/tomcat-users.xml

# vi **conf/tomcat-users.xml**

```
        <user username="tomcat1" password="tomcat1" roles="manager-script"/>
```

Add Maven-Tomcat Authentication
============================

```
# vi ~/.m2/settings.xml
<settings>
   <servers>
         <server>
               <id>TomcatServer</id>
               <username>tomcat1</username>
               <password>tomcat1</password>
         </server>
   </servers>
</settings>
```

Add Tomcat 7 Maven Plugin
=======================

```
# vi pom.xml
<plugin>
   <groupId>org.apache.tomcat.maven</groupId>
   <artifactId>tomcat7-maven-plugin</artifactId>
   <version>2.2</version>
   <configuration>
         <url>http://localhost:8080/manager/text</url>
         <server>TomcatServer</server>
         <path>/WebApps</path>
   </configuration>
</plugin>
```

```
# mvn tomcat7:deploy
# mvn tomcat7:undeploy
# mvn tomcat7:redeploy
```

Profiles
========

So in general, what is the meaning of profile, let’s take windows as example for each profile there would be some different settings right.

I mean in same machine we can have different different profiles. Similarly in pom.xml, the project is same, but you can create multiple profiles, for multiple purposes.

So for my project i want to have different different profiles like dev, qa and prod env. Here the requirements are different for each and every env, like in dev env we don’t need any of the test to be run.

Let’s say there 4 people who have different different req, now i need to create 4 projects instead of 4 projects, within a single project i can have 4 profiles, that’s the profile concept.

<profiles> will not be there under <build>, they will be below </dependencies>,
So the pom.xml looks like this with <profiles>

```
</dependencies>
<profiles>
        <profile>
                <id>DEV</id>
                <build>
                    <plugins>
                        <plugin>
```

I have keep the configuration here please go through it

https://github.com/ravi2krishna/Maven-Build-Profiles.git