

Content List

[Content List 1](#_Toc163266109)

[A bit of git/GitHub Recap 2](#_Toc163266110)

[What developers commit to GitHub (or any other SCM): 3](#_Toc163266111)

[Build Tools 4](#_Toc163266112)

[What is Maven? 4](#_Toc163266113)

[Maven 2 5](#_Toc163266114)

[Maven Installation: 5](#_Toc163266115)

[Maven builds and manages applications in our environment 7](#_Toc163266116)

[Differences between standalone, web and enterprise applications 8](#_Toc163266117)

[Task: Build a project in our Maven Server 9](#_Toc163266118)

[Where are these dependencies found/located? 10](#_Toc163266119)

[Interview Question (IQ): Explain the maven lifecycle. 10](#_Toc163266120)

[Securing the Maven local Rep 12](#_Toc163266121)

[Maven 3 12](#_Toc163266122)

A bit of git/GitHub Recap

**Dev = Development:**

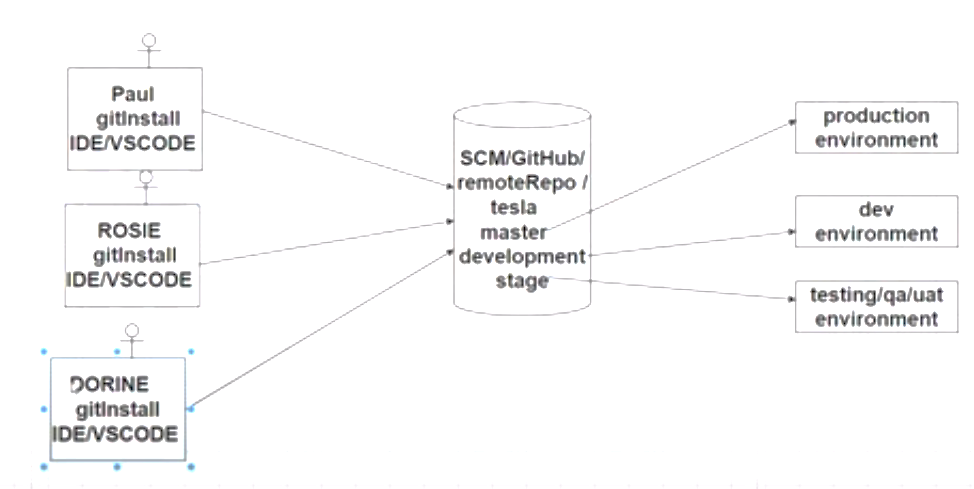
Developers are writing application codes in:

Python, Java, nodeJS, .NET, JAVASCRIPT, C++, Pascal and over a hundred other languages.

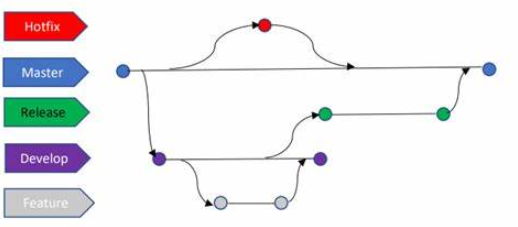
**Developers should implement best practices in coding:**

Examples of such practices are;

1. Use a DVCS – Git for versioning
2. Use CLI over GUI to manage git
3. Use IDE’s
4. Write portable code
5. Avoid duplications by using Functions/packages
6. Use a source code manager (SCM) like GitHub in projects



1. Use git best branching strategy



|  |  |
| --- | --- |
| Branch | Environment where code is deployed |
| Development branch | Dev enviroment |
| Stage branch | UAT/QA/Testing |
| Master/Main branch | Production |
| Bug/hot\_fix\_branch | (can move to testing branch before a merge to the Master branch for production) |
| Feature branch | (can move to Stage or Dev before a merge to the Master branch for production) |

1. Write good READme.md files for better understanding
2. Use .gitignore for files that are not part of the code and shouldn’t be tracked by git
3. Use git diff to review changes before merging

eg; git diff dev

1. Git commands:

git init/status/add/commit/merge/rebase/branch/switch/checkout/stash/log/show/

git pull/push/fetch/tag/cherry-pick/clean/revert/reset

1. Use pull request for review by team members before merging in the master branch
2. Commit code frequently and avoid merge conflicts. Commit before switching branches, do not commit half done work.

**Note**: above so far is mostly what developers do on the daily bases at their own end. Now we will begin looking at where DevOps Engineers fall in.

What developers commit to GitHub (or any other SCM):

1. Source code
2. Unit test cases
3. Build scripts
4. We need the code from GitHub, so we pull or clone, then build in application servers.
5. In the build servers we build and test our application

Build Tools

They are used to create deployable packages eg Tomcat

**What do build tools achieve?**

Raw code + build = packages that deployment servers can interpret

**Interview Question:** What kind of projects are you supporting in your environment?

**Example answer** 🡪 we support java-based projects and a few .NET and NodeJS projects.

**Build tools for java-based projects are:**

MAVEN /Ant /Gradle

**Build tools for .Net projects are:**

NaNt /MSBuild

**Build tools for JAVASCRIPT projects are:**

Gulp /Grunt

**Build tools for Python: projects are:**

PyBuilder

**Build tools for NodeJS projects are:**

npm

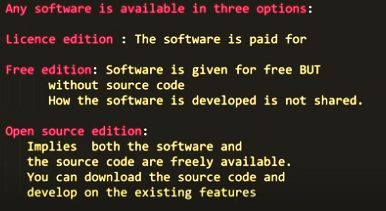
What is Maven?

Maven is an open Java based build tools. Apache Maven is a software project management and comprehension tool. It is based on the concept of a project object model (POM).

Maven can manage a project’s build, reporting and documentation.

**Maven is open source** Java based Build tool. It was originally developed in java for java codes. The vendor of maven is apache.

https: //maven.apache.org/download.cgi



Linux is also Open Source.

Maven 2

Maven Installation:

On a RedHat Linux 8 Server.

Download link: <https://maven.apache.org/download.cgi>.

Maven can be installed on any platform (Multi-platform build tool);

🡪Linux (RedHat, Ubuntu, CentOS, etc),

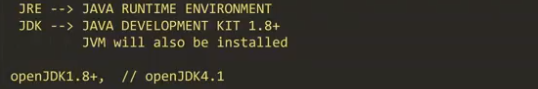
🡪Windows(Server2012/2016/2019),

🡪MacOS,

🡪Solaris

**Note**: We are tasked to install maven on a RedHat Linux 8 Server

✅Java is a pre-requisite for maven to run, so we install the following



You can go to <https://github.com/LandmarkTechnology/package-management>

Look for maven-installation...

Prerequisites;

✅AWS Account

✅Create RedHat EC2 T2 medium instance with 4gb of RAM

✅Create Security Group and open required ports, 22 etc

✅Attach Security Group to EC2 Instance

✅install java OpenJDK 1.8+

################################################################

# Step1. Install Java JDK 11+ and other softares (GIT, wget and tree)

################################################################

# install Java JDK 11+ as a pre-requisit for maven to run.

sudo hostnamectl set-hostname maven

sudo su - ec2-user

cd /opt

sudo yum install wget nano tree unzip git-all -y

sudo yum install java-11-openjdk-devel java-1.8.0-openjdk-devel -y

java -version

git --version

###############################################################

# Step2. Download, extract and Install Maven

###############################################################

sudo wget https://dlcdn.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.zip

sudo unzip apache-maven-3.9.5-bin.zip

sudo rm -rf apache-maven-3.9.5-bin.zip

sudo mv apache-maven-3.9.5/ maven

###############################################################

# Step3. Set Environmental Variable - For Specific User eg ec2-user

###############################################################

vi ~/.bash\_profile # and add the lines below

export M2\_HOME=/opt/maven

export PATH=$PATH:$M2\_HOME/bin

##############################################################

# Step4. Refresh the profile file and Verify if maven is running

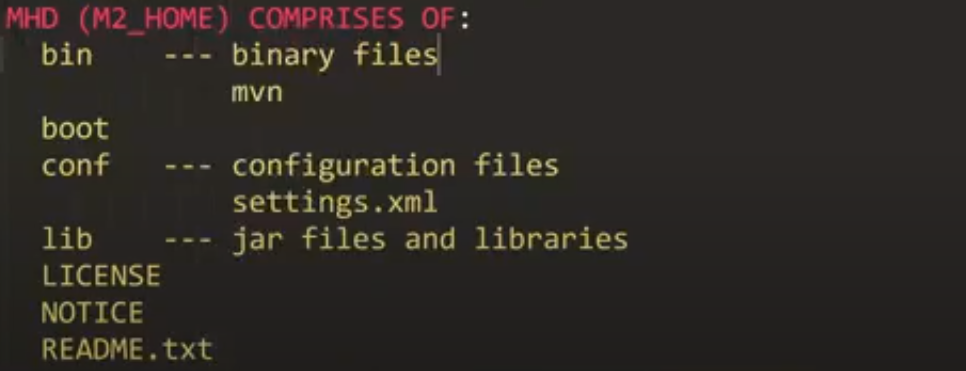
##############################################################

source ~/.bash\_profile

mvn -version

**Note**: Maven home directory here is /opt/maven and can be different for any company as they may choose.

In the maven home directory (MHD), we can find the following files and folders;



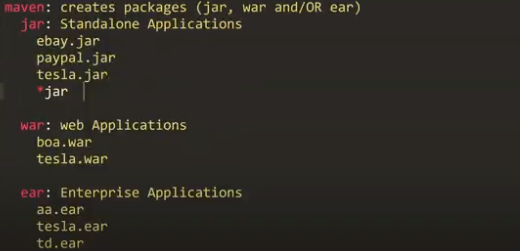
Maven builds and manages applications in our environment

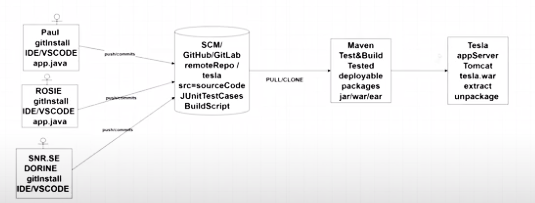
Generally, we have;

✅Maven Standalone applications (.jar)

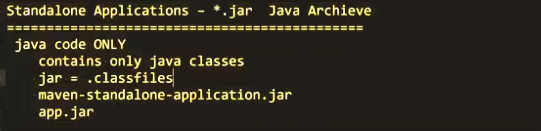
✅Maven Web applications (.war)

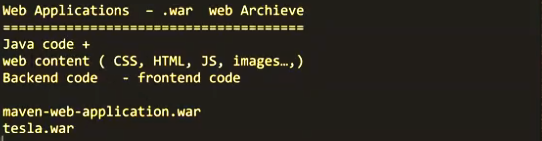
✅Maven Enterprise Applications (.ear)

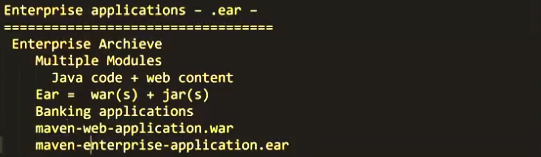




Differences between standalone, web and enterprise applications









**What are we building?**

1. Src 🡪 source code
2. Unit-Test\_Cases:

✅JAVA 🡪Junit test cases

✅.Net 🡪NUnit test cases

✅C++ 🡪CPPUnit test cases

1. BuildScripts 🡪 pom.xml:

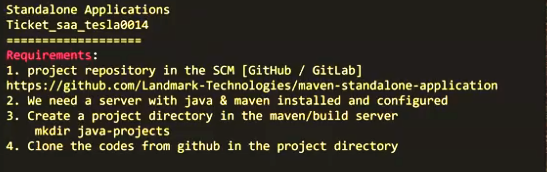
xml = extensible mark-up language – it is dynamic and tags can be created based on the user

html = hyperText mark-up language – comes with pre-defined tags that can’t be changed (eg h1,h2,p etc)

Code examples in bash-shell, html and xml;



Task: Build a project in our Maven Server



Cloning project repo, in your server, run the following commands;

mkdir java-projects

cd java-projects

ll

git clone https://github.com/Landmark-Technologies/maven-standalone-application

ls maven-standalone-application/ # we can see pom.xml and src

ls maven-standalone-application/src/ # to see the content of src

What we have on our server represents the source code written by developers so far

Maven uses dependencies and plugins to test and create packages.

Examples of maven dependencies

✅Selenium

<https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java/4.6.0>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.6.0</version>

</dependency>

✅Junit

<https://mvnrepository.com/artifact/junit/junit>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

✅log4j



Back to the command line;

vi pom.xml # we can observe the various parts of the file and see the dependencies section

Where are these dependencies found/located?

1. Maven local repository

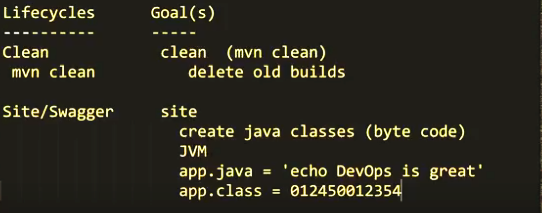
.m2/repository = default

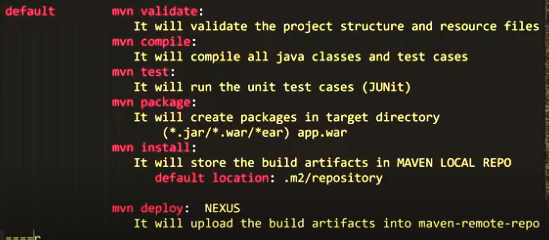
1. Maven remote/proxy repository
2. Maven Central Repository

<https://mvnreopository.com/repos/central>

Interview Question (IQ): Explain the maven lifecycle.

**Answer**; Maven has three lifecycles which are, Clean, site/swagger and default.





The mvn package command will;

1. Validate
2. Compile
3. RunUnitTesting
4. Create the package [jar/war/ear]

Back to the cli;(inside the java-projects folder)

ls

mv maven-standalone-application/ msa

ls

cd msa/

pwd

#run the cmd below to verify if you would find the mavenLocalRepo found in .m2/repository

ls -a /home/ec2-user #or just ls -a ~

mvn package #to create packages

#if build fails, you can vi into pom.xml and verify the dependency versions

#if build succeeds, we can proceed to

ls -a ~ # and we can now see the .m2 file

ls /home/ec2-user/.m2/ #to see our local repo created

ls /home/ec2-user/.m2/repository/ #to see downloaded dependencies and plugins

Securing the Maven local Rep

✅Creating a custom maven local repo

.m2/repository/ # is default

We have to vi into settings.xml and create our own like say /home/ec2-user/mlr

ls /opt/maven/conf/ #we’ll see settings.xml

ll /opt/maven/conf/settings.xml #to check who owns the file

sudo vi /opt/maven/conf/settings.xml #search for the localRepository tag and change

#<localRepository>/path/to/local/repo</localRepository> to

#<localRepository>/home/ec2-user/mlr/</localRepository>

#ensure to remove the above line from comment, save and quit the file

ls /home/ec2-user/ #notice that there is no file like mlr

mvn package #now we build

ls /home/ec2-user/ #notice that there is now a file named mlr

ls /home/ec2-user/mlr #to see the dependencies and plugins downloaded

Maven 3

**mvn clean package vs mvn clean install:**

**mvn clean package:**

* **Clean Phase**: Removes the **target** directory, deleting any previously generated artifacts.
* **Package Phase**: Compiles the source code, runs tests, and packages the compiled code and resources into an executable JAR file (or any other specified format).
* **Behavior**: Running **mvn clean package** on a Maven standalone application project will clean the project, compile the source code, run tests, and package everything into a standalone executable JAR file. The JAR file will be available in the **target** directory.
* **Artifact Generation**: The resulting artifact, usually an executable JAR file, will be located in the **target** directory of your project.
* **Effect of Subsequent Clean**: If you run **mvn clean** after **mvn clean package**, it will delete the **target** directory again, removing the generated artifact.

**mvn clean install:**

* **Install Phase**: In addition to the behaviors of **mvn clean package**, the **install** phase copies the packaged artifact into the local Maven repository (**~/.m2/repository**).
* **Behavior**: Running **mvn clean install** performs the same clean and package operations as **mvn clean package**. Additionally, it copies the resulting artifact (the executable JAR file) into the local Maven repository.
* **Artifact Installation**: After running **mvn clean install**, the packaged artifact is not only available in the **target** directory but also installed in the local Maven repository.
* **Effect of Subsequent Clean**: If you run **mvn clean** after **mvn clean install**, it will delete the **target** directory but won't remove the artifact from the local Maven repository.

**Interview Q1.** Assuming that 699 Testcase passed and 1 fails what can be done for maven to still do a build? OR Some projects are taking too long to build how can we resolve this issue ?

1. **mvn package -DskipTests**: This command skips running the tests during the build process altogether. It compiles the source code and packages it into the desired format (e.g., JAR, WAR) without executing any tests.
2. **mvn package -Dmaven.test.skip=true**: This command also skips running the tests during the build process. However, it differs from the previous option in that it doesn't even compile the tests, whereas **-DskipTests** compiles the tests but skips their execution.

**Understanding Maven Repository Management**

1. **Introduction to Maven and Repositories:**
   * Maven is a build automation tool used primarily for Java projects.
   * It utilizes plugins and dependencies to manage project builds, validations, testing, and artifact creation.
   * These plugins and dependencies are sourced from repositories, including local repositories and remote repositories.
2. **Types of Maven Repositories:**
   * **MavenLocalRepo:** This is the local repository stored on the system where Maven is installed. It typically contains artifacts that are frequently used by the project and have been previously downloaded.
   * **Maven Remote Repositories:** These repositories are hosted online and contain a vast collection of plugins and dependencies. Examples include maven-remote-repo, maven-proxy-repo, and maven-central-repo.
3. **Importance of MavenLocalRepo:**
   * Pulling plugins and dependencies from MavenLocalRepo generally offers the best and fastest results in terms of build time since the artifacts are readily available locally without the need for network requests.
4. **Scenario: Junior Engineer Deletes Maven Local Repository:**
   * If the MavenLocalRepo is accidentally deleted, it necessitates recreating the repository to restore the local artifacts.
   * After recreating the MavenLocalRepo, the initial build using **mvn package** will take longer (around 7 seconds) as it fetches plugins and dependencies from remote repositories.
5. **Optimizing Build Time with MavenLocalRepo:**
   * Subsequent builds after recreating MavenLocalRepo are significantly faster (around 1 second) since Maven can now source the required artifacts locally from MavenLocalRepo.
   * This highlights the importance of maintaining a well-populated and up-to-date MavenLocalRepo to optimize build performance.
6. **Handling Changes in Pom.xml Filename:**
   * Maven expects the project object model (POM) file to be named **pom.xml** by default.
   * If the Pom.xml filename is changed (e.g., to **tesla.xml**), Maven will fail to recognize the file during build commands.
   * To address this issue, the **-f** flag is used to specify the alternative POM file name. For example, **mvn package -f tesla.xml** would correctly identify the renamed POM file and execute the build process.
7. **Conclusion:**
   * Maven repository management, particularly the local repository, plays a crucial role in optimizing build performance.
   * Developers should be cautious about accidental deletions of the local repository and ensure its timely recreation if necessary.
   * Additionally, understanding Maven's conventions, such as default POM file naming, and utilizing appropriate flags can prevent build failures due to filename changes.

**Deploying Java-based applications**

It varies depending on the type of application and the deployment environment. Here's a breakdown:

1. **Java-based Web Applications:**
   * Java-based web applications are typically deployed on application servers like Apache Tomcat, JBoss/Wildfly.
2. **Java-based Enterprise Applications:**
   * Java-based enterprise applications, which typically consist of multiple components and services, are deployed on enterprise application servers like JBoss/Wildfly.
3. **Java-based Standalone Applications:**
   * Java-based standalone applications are typically deployed on servers or systems where the Java Development Kit (JDK) is installed.
   * Deployment of standalone applications often involves copying the JAR files to the target server or system and executing them using the **java** command with appropriate arguments.

**Java** -**jar** artifact-name.**jar**

NB: If you comment out or remove the <finalName> tag from the POM file and then run mvn clean package, Maven will generate the artifact name using the default naming convention.

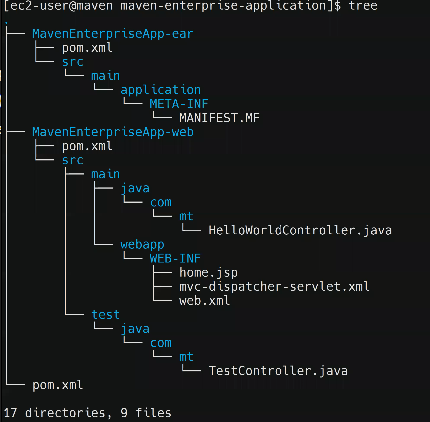
Here's how the default naming convention works:

* The default artifact name consists of the project's **artifactId**, **version**, and **packaging**.
* For example, if your project's **artifactId** is **my-project**, **version** is **1.0**, and **packaging** is **jar**, Maven will generate an artifact named **my-project-1.0.jar**.
* Similarly, for a WAR packaging, the artifact would be named **my-project-1.0.war**, and for EAR packaging, it would be named **my-project-1.0.ear**.

**Maven Enterprise Application:**

**Project Structure and Modules:**

* Maven allows organizing enterprise applications into multiple modules using a parent-child project structure.
* Each module represents a different component of the enterprise application, such as web applications, EJBs, or utility modules.
* The parent project coordinates the build and dependencies among these modules. Running the **tree** command gives us



Running mvn package generate artifacts for MavenEnterpriseApp-ear/ **(****MavenEnterpriseApp-ear.ear)** & MavenEnterpriseApp-web/ **(MavenEnterpriseApp-web.war)**.

How can a specific module be built in maven enterprise application incase we have a change in that module?

mvn **package** **-P profilename**

mvn **package** **-p MavenEnterpr**iseApp-ear

mvn **package** **-p MavenEnterpr**iseApp-web

**Question: How can we troubleshoot a failed build?**

We check our logs to understand the error.