1

Apache Maven

Objectives

After completing this lesson, you should be able to do the following:

- An Introduction to Maven
- Installing Maven
- Core Concepts
- The Project Object Model (POM)
- Custom Builds

Introduction to Maven

Introduction

- Building a software project typically consists of such tasks as downloading dependencies, putting additional jars on a classpath, compiling source code into binary code, running tests, packaging compiled code into deployable artifacts such as JAR, WAR, and ZIP files, and deploying these artifacts to an application server or repository.
- Apache Maven automates these tasks, minimizing the risk of humans making errors while building the software manually and separating the work of compiling and packaging our code from that of code construction.

- Maven is a popular open-source build tool that the Apache Group developed for building, publishing, and deploying several projects.
- Maven is written in Java and is used to create projects written in C#, Scala, Ruby, and so on.
- The tool is used to build and manage any Java-based project. It simplifies the day-to-day work of Java developers and helps them with various tasks.
- Maven is a powerful *project management tool* that is based on POM (project object model). It is used for projects build, dependency and documentation.
- It simplifies the build process like ANT. But it is too much advanced than ANT.

History

- Maven was originally designed to simplify building processes in Jakarta Turbine project. There were several projects and each project contained slightly different ANT build files. JARs were checked into CVS.
- Apache group then developed **Maven** which can build multiple projects together, publish projects information, deploy projects, share JARs across several projects and help in collaboration of teams.

Understanding the problem without Maven

There are many problems that we face during the project development. They are discussed below:

1) Adding set of Jars in each project: In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

2) Creating the right project structure: We must create the right project structure in servlet, struts etc, otherwise it will not be executed.

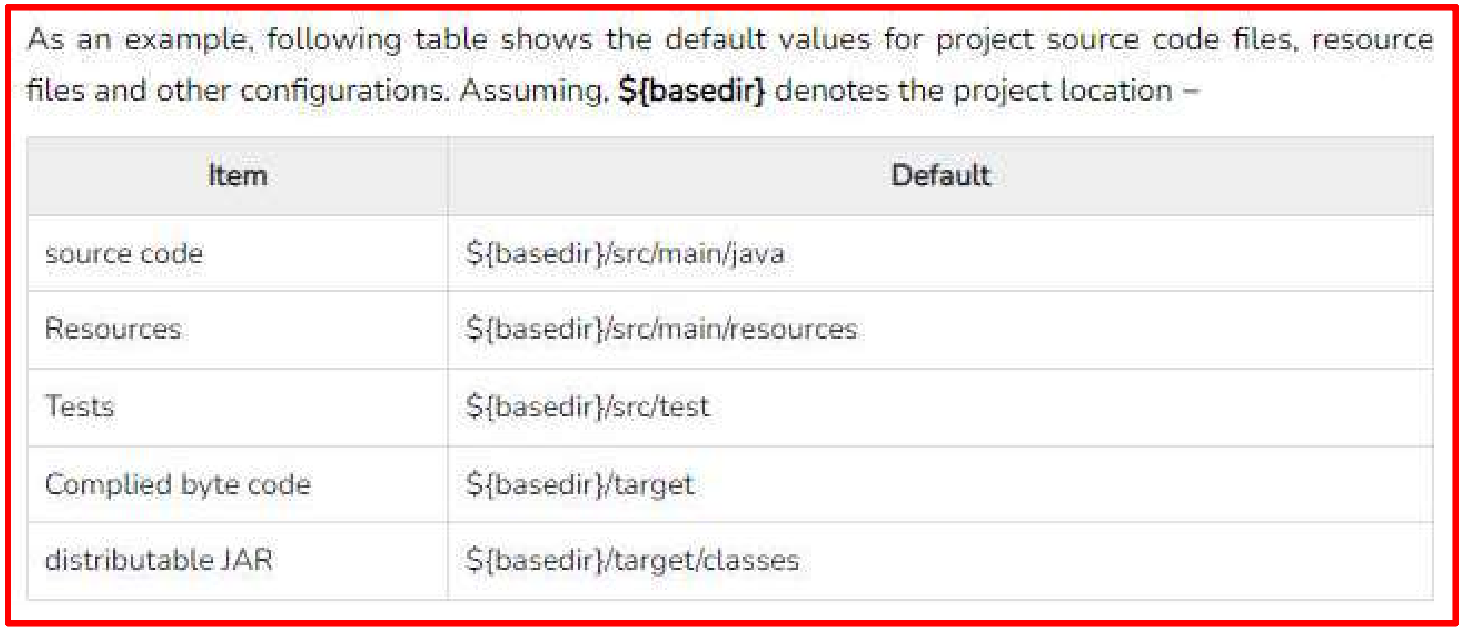
3) Building and Deploying the project: We must have to build and deploy the project so that it may work.

- Maven simplifies the previously mentioned problems. It does mainly following tasks.
  1. It makes a project easy to build
  2. It provides uniform build process (maven project can be shared by all the maven projects)
  3. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
  4. It is easy to migrate for new features of Maven

- ➢ Maven is based on Project Object Model (POM) and focuses on simplification and standardization of the building process.
- ➢ During the process, Maven takes care of the following elements:
  1. Builds
  2. Dependencies
  3. Reports
  4. Distribution
  5. Releases
  6. Mailing list

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. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.


As an example, following table shows the default values for project source code files, resource files and other configurations. Assuming, **${basedir}** denotes the project location –

Item	Default
source code	${basedir}/src/main/java
Resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Complied byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

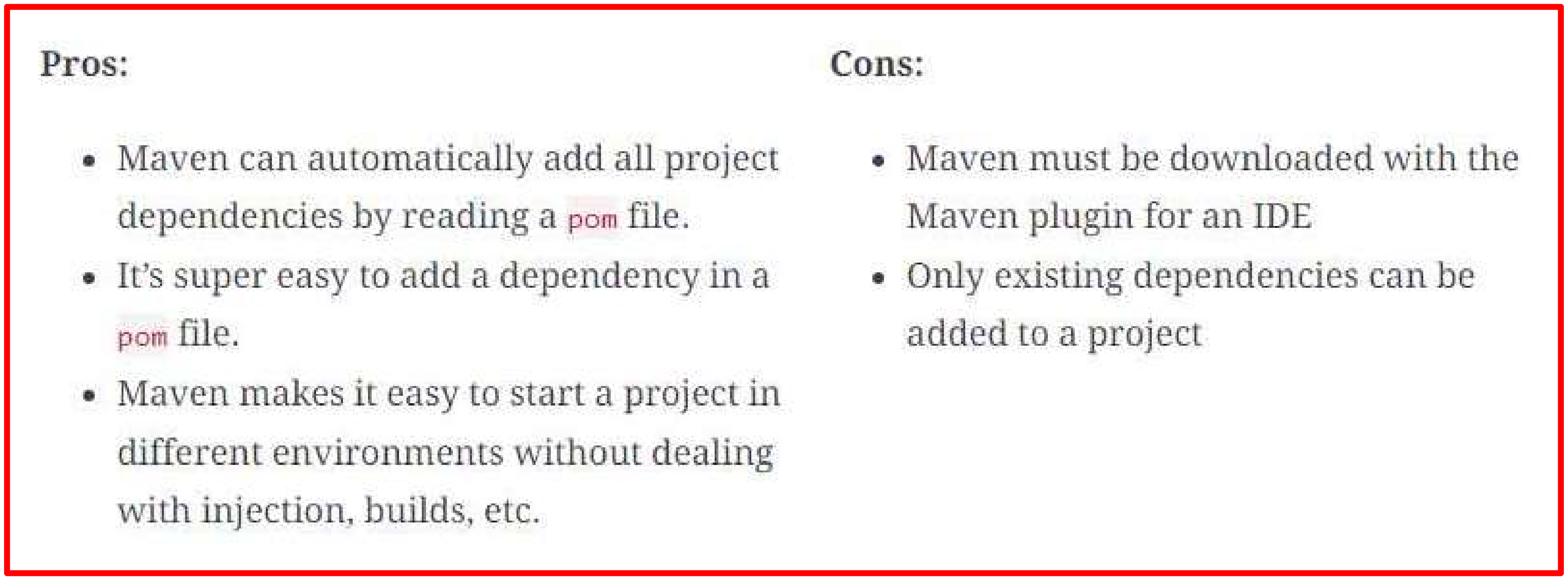
Why Use Maven?

1. **simple project setup that follows best practices:** Maven tries to avoid as much configuration as possible, by supplying project templates (named *archetypes*)
2. **dependency management:** it includes automatic updating, downloading and validating the compatibility, as well as reporting the dependency closures (known also as transitive dependencies)
3. **isolation between project dependencies and plugins:** with Maven, project dependencies are retrieved from the *dependency repositories* while any plugin's dependencies are retrieved from the *plugin repositories,* resulting in fewer conflicts when plugins start to download additional dependencies
4. **central repository system:** project dependencies can be loaded from the local file system or public repositories, such as **Maven Central**

What is Build Tool ?

- A build tool takes care of everything for building a process. It does following:
  1. Generates source code (if auto-generated code is used)
  2. Generates documentation from source code
  3. Compiles source code
  4. Packages compiled code into JAR of ZIP file
  5. Installs the packaged code in local repository, server repository, or central repository

Maven Pros and Cons



Pros:

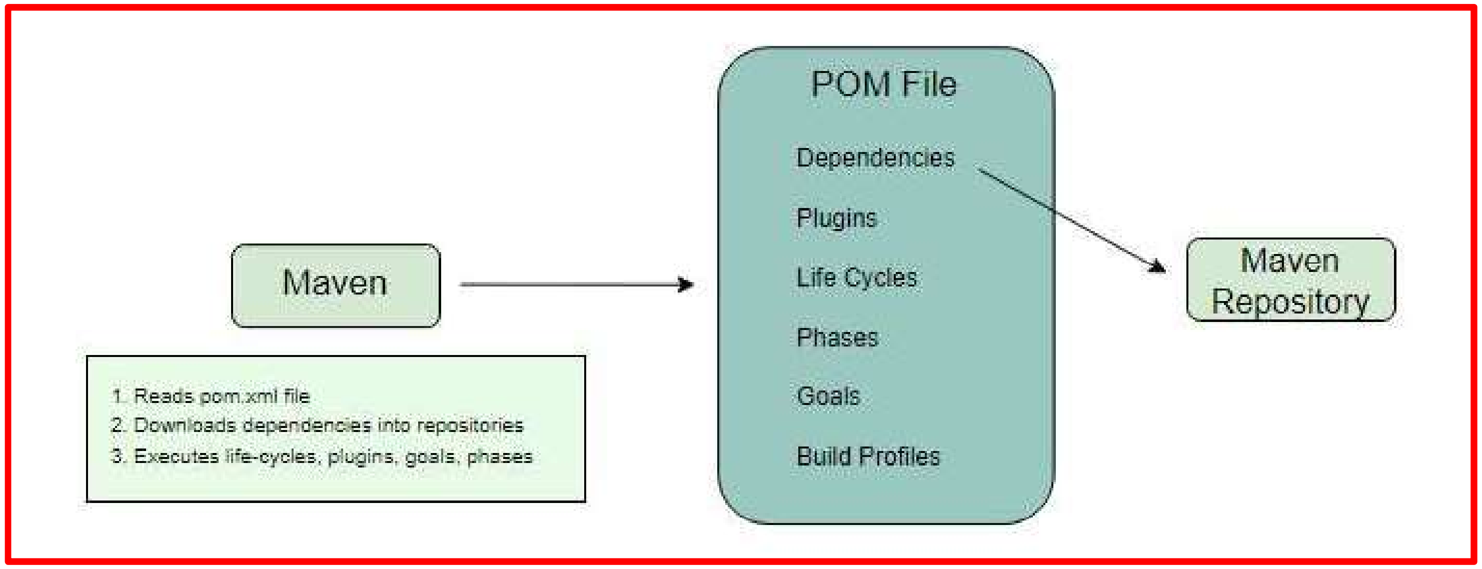
- Maven can automatically add all project dependencies by reading a `pom` file.
- It's super easy to add a dependency in a `pom` file.
- Maven makes it easy to start a project in different environments without dealing with injection, builds, etc.

Cons:

- Maven must be downloaded with the Maven plugin for an IDE
- Only existing dependencies can be added to a project

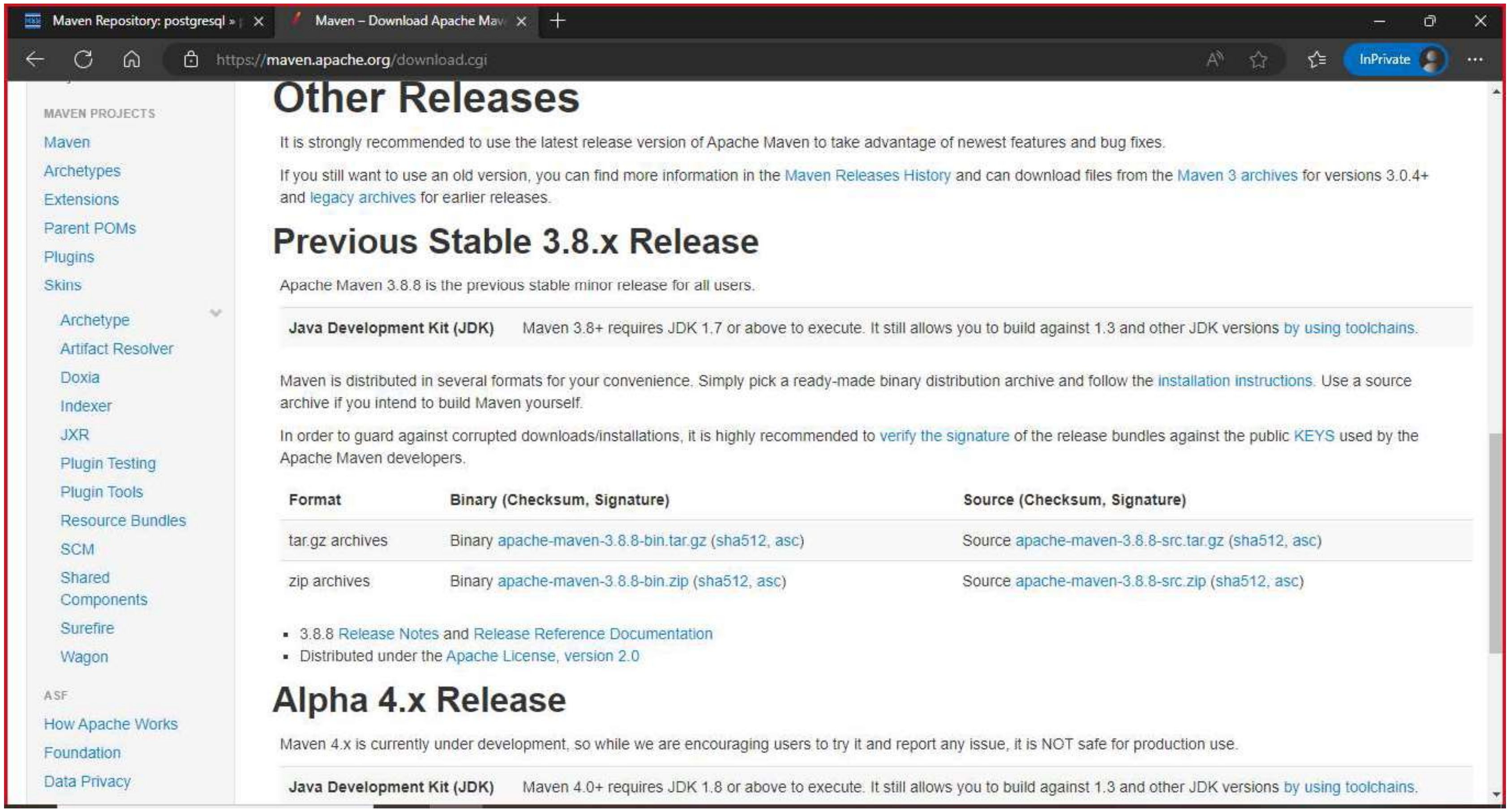
Salient features of Maven

- **POM Files:** Project Object Model (POM) Files are XML files containing project and configuration information. Maven POM files are used to execute the commands.
- **Dependencies and Repositories:** Dependencies are external Java libraries, while repositories are the directories for packaged JAR files. Maven repository.
- **Build Plugins:** Build plugins perform specific goals for your project. These are added to the POM file. Maven provides standard plugins, or you can implement your own.
- **Life Cycles, Phases, and Goals:** A build life cycle is made up of multiple build phases, which are simply a sequence of project goals. The build lifecycle is named a Maven command.
- **Build Profiles:** Build profiles a set of configuration values that allow you to build with different configurations. You add build profiles to your POM files using the profiles elements.


Maven
1. Reads pom.xml file
2. Downloads dependencies into repositories
3. Executes life-cycles, plugins, goals, phases
POM File
Dependencies
Plugins
Life Cycles
Phases
Goals
Build Profiles
Maven Repository

Installation of Maven

Download from [https://maven.apache.org/download.cgi]



Other Releases

It is strongly recommended to use the latest release version of Apache Maven to take advantage of newest features and bug fixes.

If you still want to use an old version, you can find more information in the Maven Releases History and can download files from the Maven 3 archives for versions 3.0.4+ and legacy archives for earlier releases.

Previous Stable 3.8.x Release

Apache Maven 3.8.8 is the previous stable minor release for all users.

Java Development Kit (JDK)	Maven 3.8+ requires JDK 1.7 or above to execute. It still allows you to build against 1.3 and other JDK versions by using toolchains.

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

Format	Binary (Checksum, Signature)	Source (Checksum, Signature)
tar.gz archives	Binary apache-maven-3.8.8-bin.tar.gz (sha512, asc)	Source apache-maven-3.8.8-src.tar.gz (sha512, asc)
zip archives	Binary apache-maven-3.8.8-bin.zip (sha512, asc)	Source apache-maven-3.8.8-src.zip (sha512, asc)

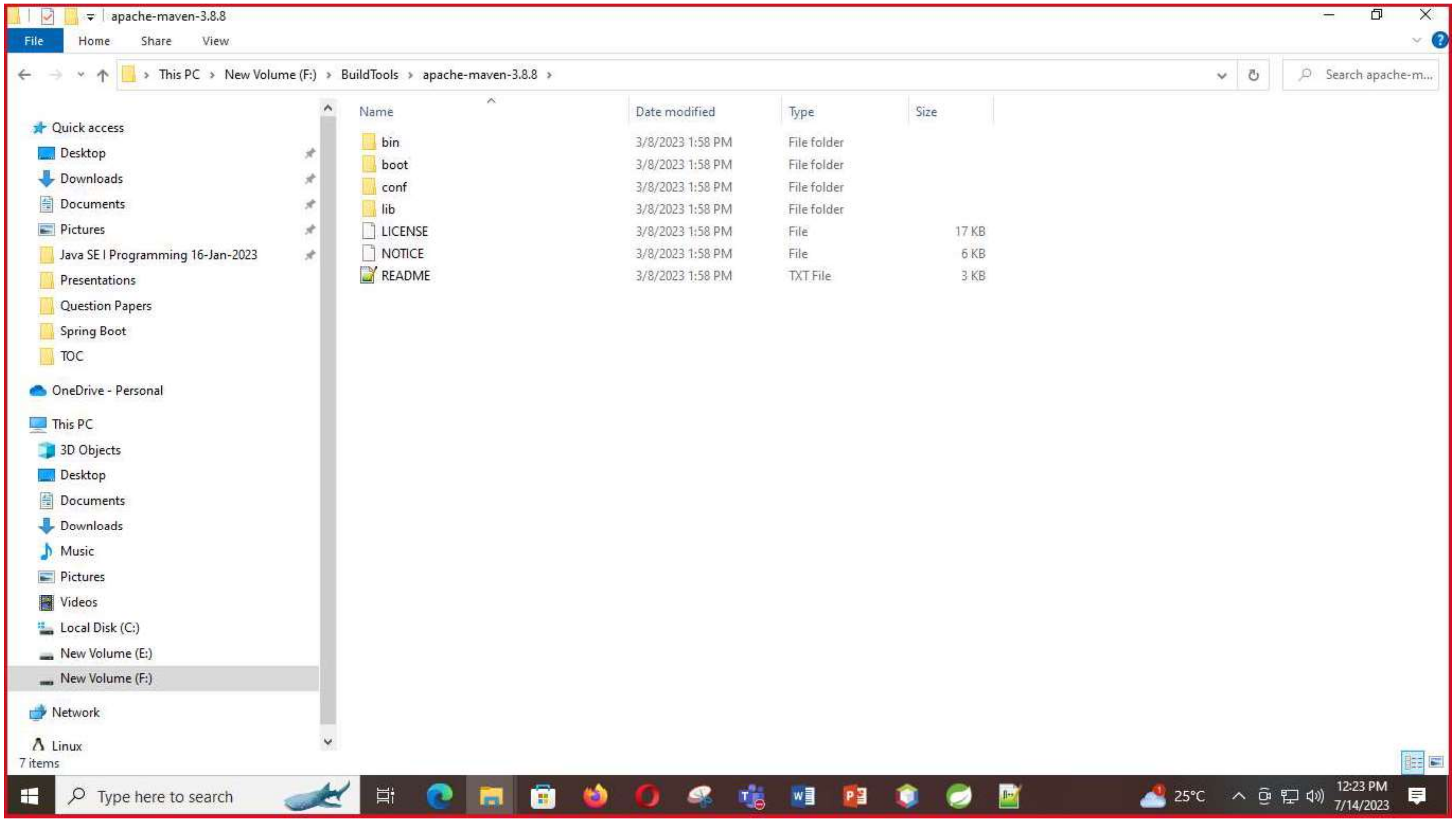
- 3.8.8 Release Notes and Release Reference Documentation
- Distributed under the Apache License, version 2.0

Alpha 4.x Release

Maven 4.x is currently under development, so while we are encouraging users to try it and report any issue, it is NOT safe for production use.

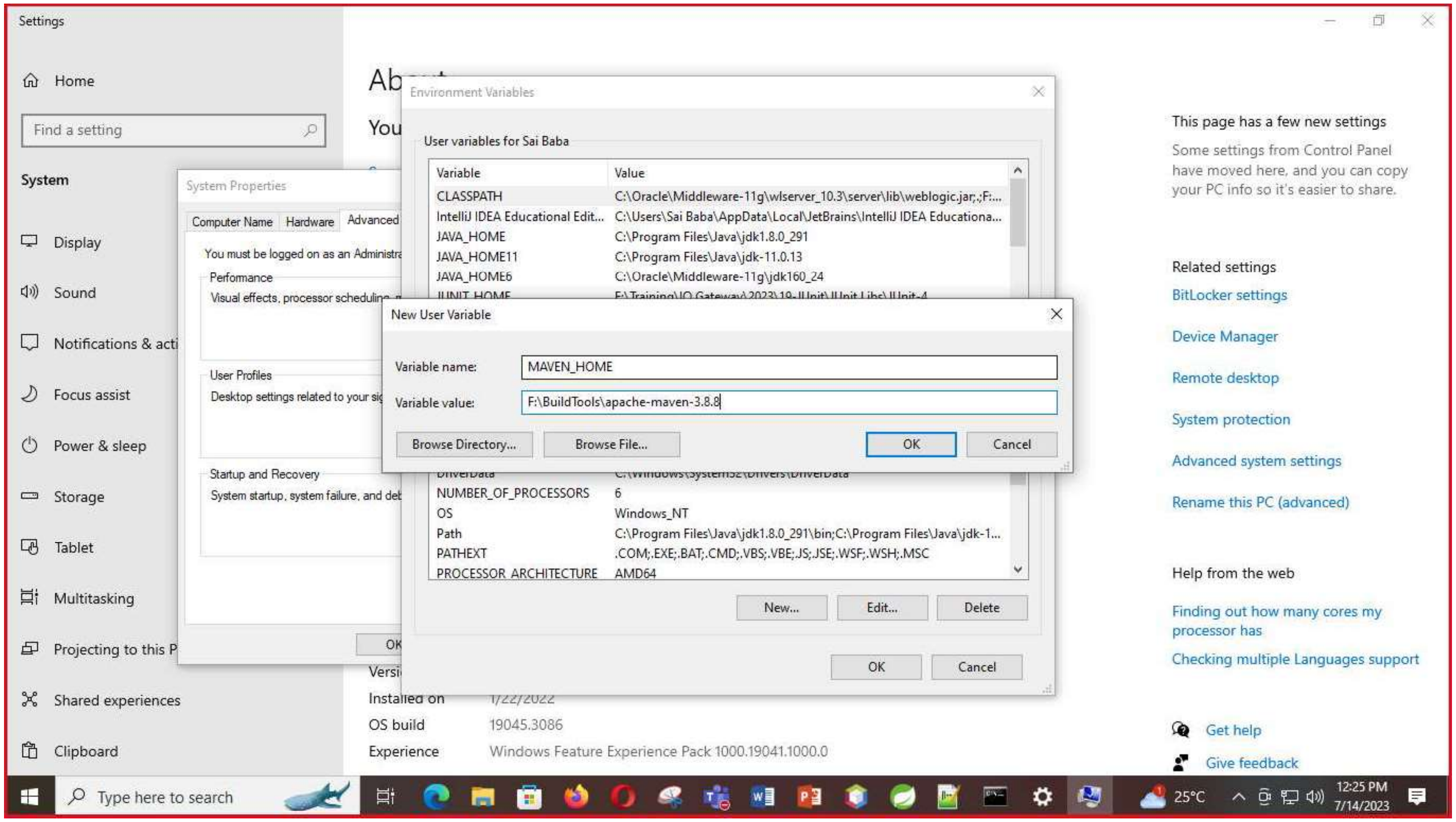
Java Development Kit (JDK)	Maven 4.0+ requires JDK 1.8 or above to execute. It still allows you to build against 1.3 and other JDK versions by using toolchains.

Upack the Zip/ Tar File



Name	Date modified	Type	Size
bin	3/8/2023 1:58 PM	File folder	
boot	3/8/2023 1:58 PM	File folder	
conf	3/8/2023 1:58 PM	File folder	
lib	3/8/2023 1:58 PM	File folder	
LICENSE	3/8/2023 1:58 PM	File	17 KB
NOTICE	3/8/2023 1:58 PM	File	6 KB
README	3/8/2023 1:58 PM	TXT File	3 KB

Configure Maven Installation [MAVEN_HOME] : Env Variables



New User Variable

Variable name: MAVEN_HOME

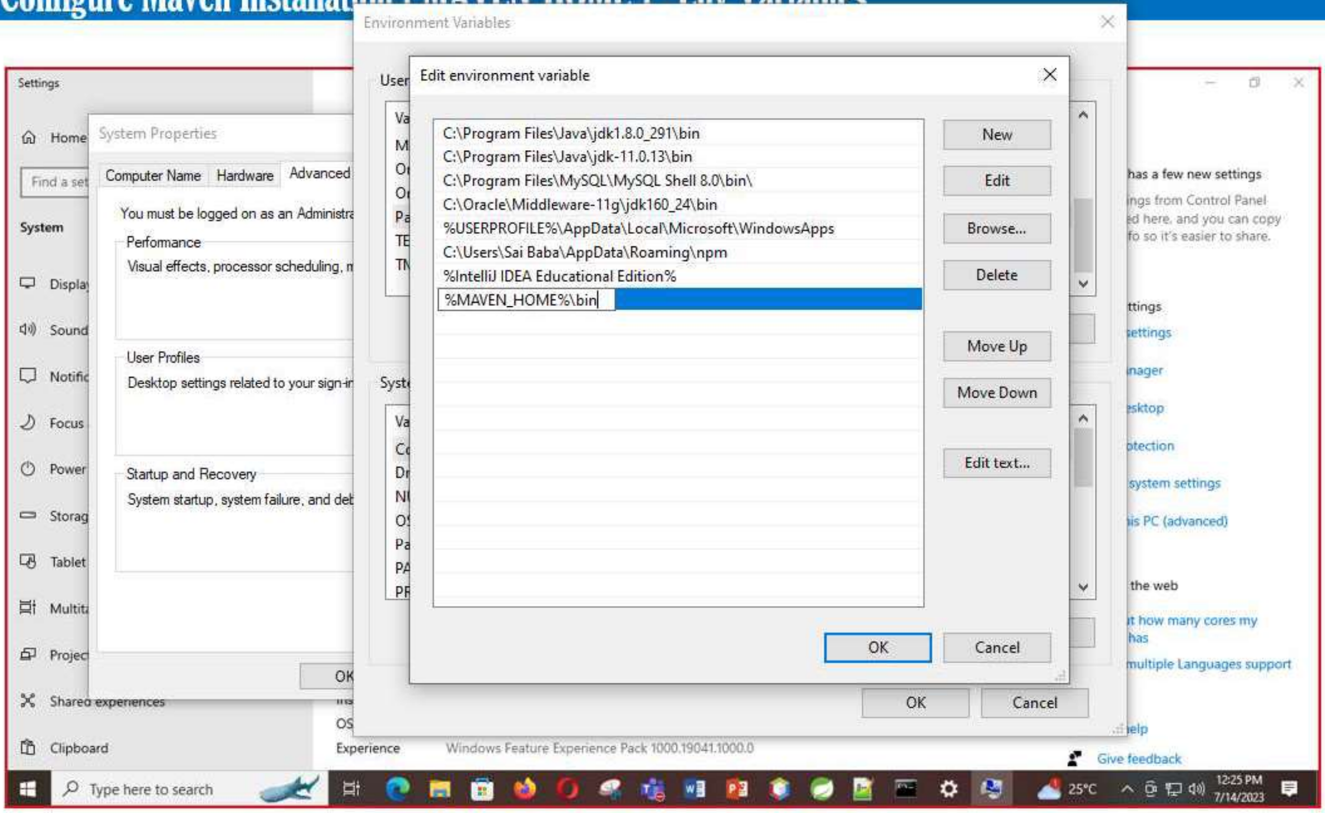
Variable value: F:\BuildTools\apache-maven-3.8.8

User variables for Sai Baba

Variable	Value
CLASSPATH	C:\Oracle\Middleware-11g\wlserver_10.3\server\lib\weblogic.jar;.;F:...
IntelliJ IDEA Educational Edit...	C:\Users\Sai Baba\AppData\Local\JetBrains\IntelliJ IDEA Educationa...
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_291
JAVA_HOME11	C:\Program Files\Java\jdk-11.0.13
JAVA_HOME6	C:\Oracle\Middleware-11g\jdk160_24
NUMBER_OF_PROCESSORS	6
OS	Windows_NT
Path	C:\Program Files\Java\jdk1.8.0_291\bin;C:\Program Files\Java\jdk-1...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

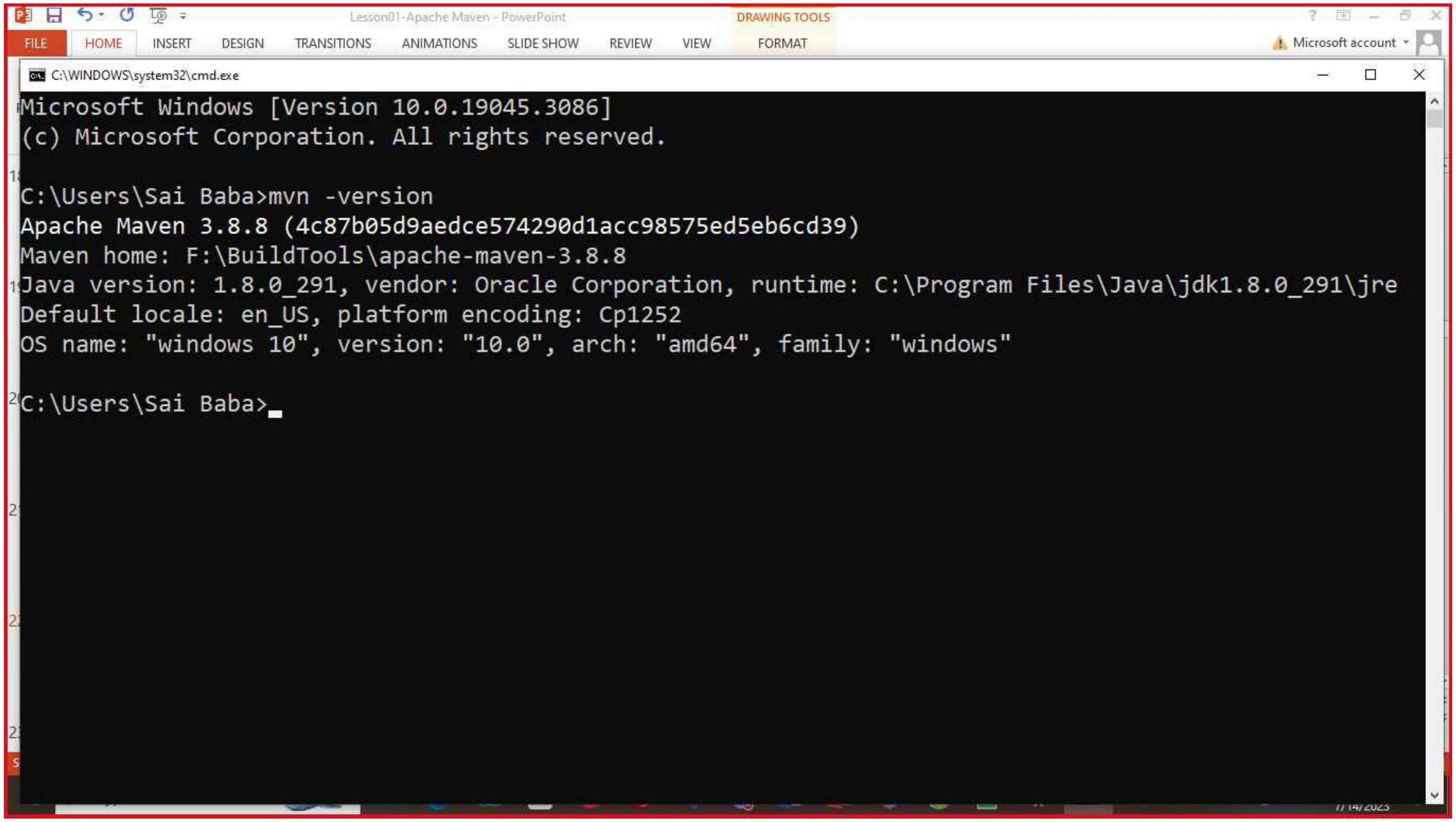
MentorLabs

Configure Maven Installation [MAVEN_HOME] : Env Variables


Edit environment variable

```
C:\Program Files\Java\jdk1.8.0_291\bin
C:\Program Files\Java\jdk-11.0.13\bin
C:\Program Files\MySQL\MySQL Shell 8.0\bin\
C:\Oracle\Middleware-11g\jdk160_24\bin
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Sai Baba\AppData\Roaming\npm
%IntelliJ IDEA Educational Edition%
%MAVEN_HOME%\bin
```

Verify Maven



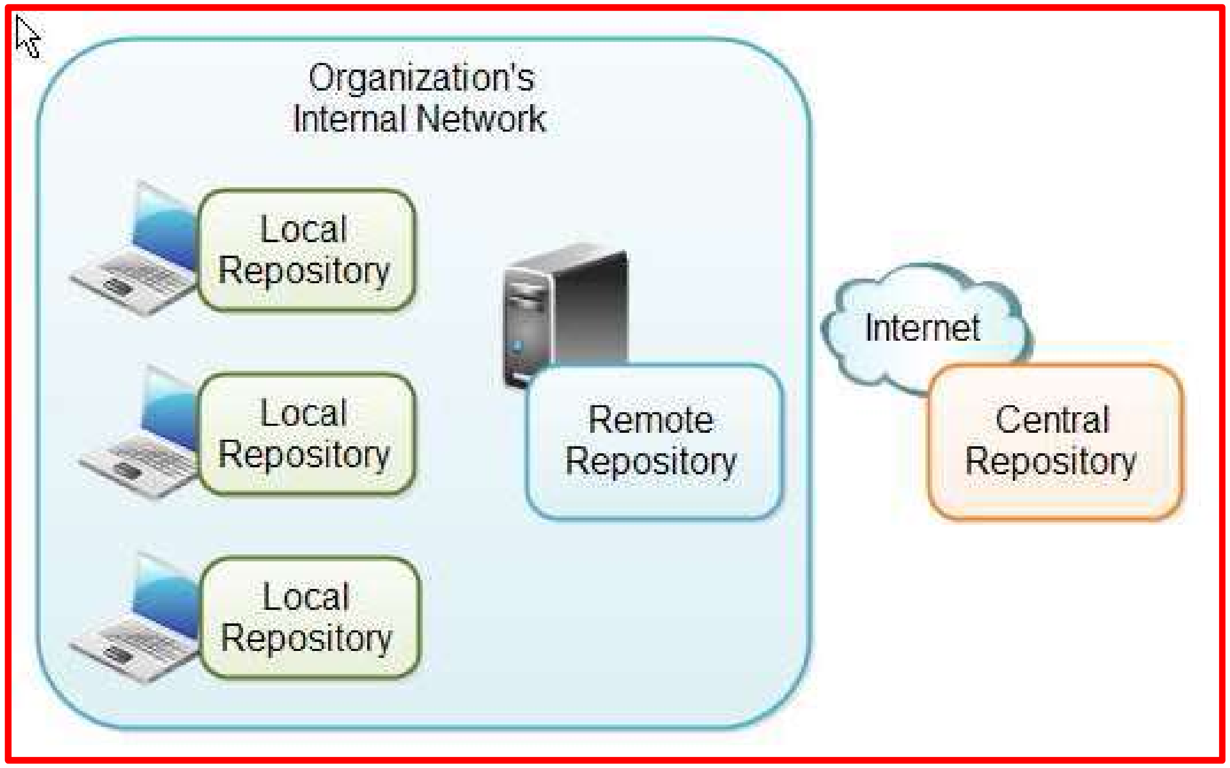
```
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Sai Baba>mvn -version
Apache Maven 3.8.8 (4c87b05d9aedce574290d1acc98575ed5eb6cd39)
Maven home: F:\BuildTools\apache-maven-3.8.8
Java version: 1.8.0_291, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_291\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\Sai Baba>_
```

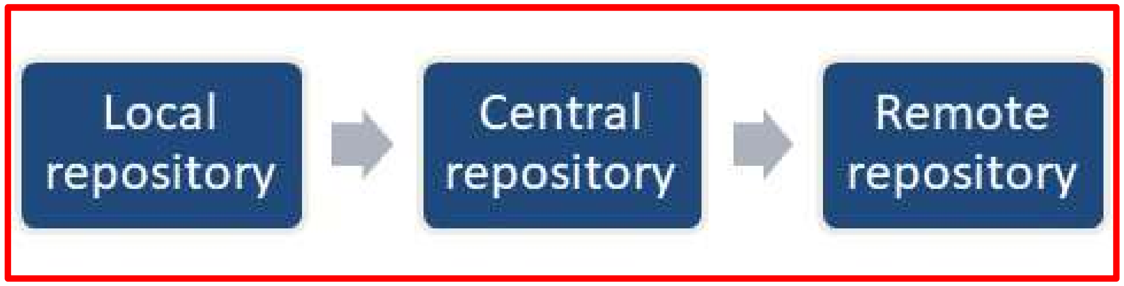
Maven Repositories

- Maven repositories refer to the directories of packaged JAR files that contain metadata. The metadata refers to the POM files relevant to each project. This metadata is what enables Maven to download dependencies.
- There are 3 types of maven repository:
  1. Local Repository
  2. Central Repository
  3. Remote Repository


Organization's Internal Network
Local Repository
Local Repository
Local Repository
Remote Repository
Internet
Central Repository

- Maven searches for the dependencies in the following order:

- **Local repository** then **Central repository** then **Remote repository**.

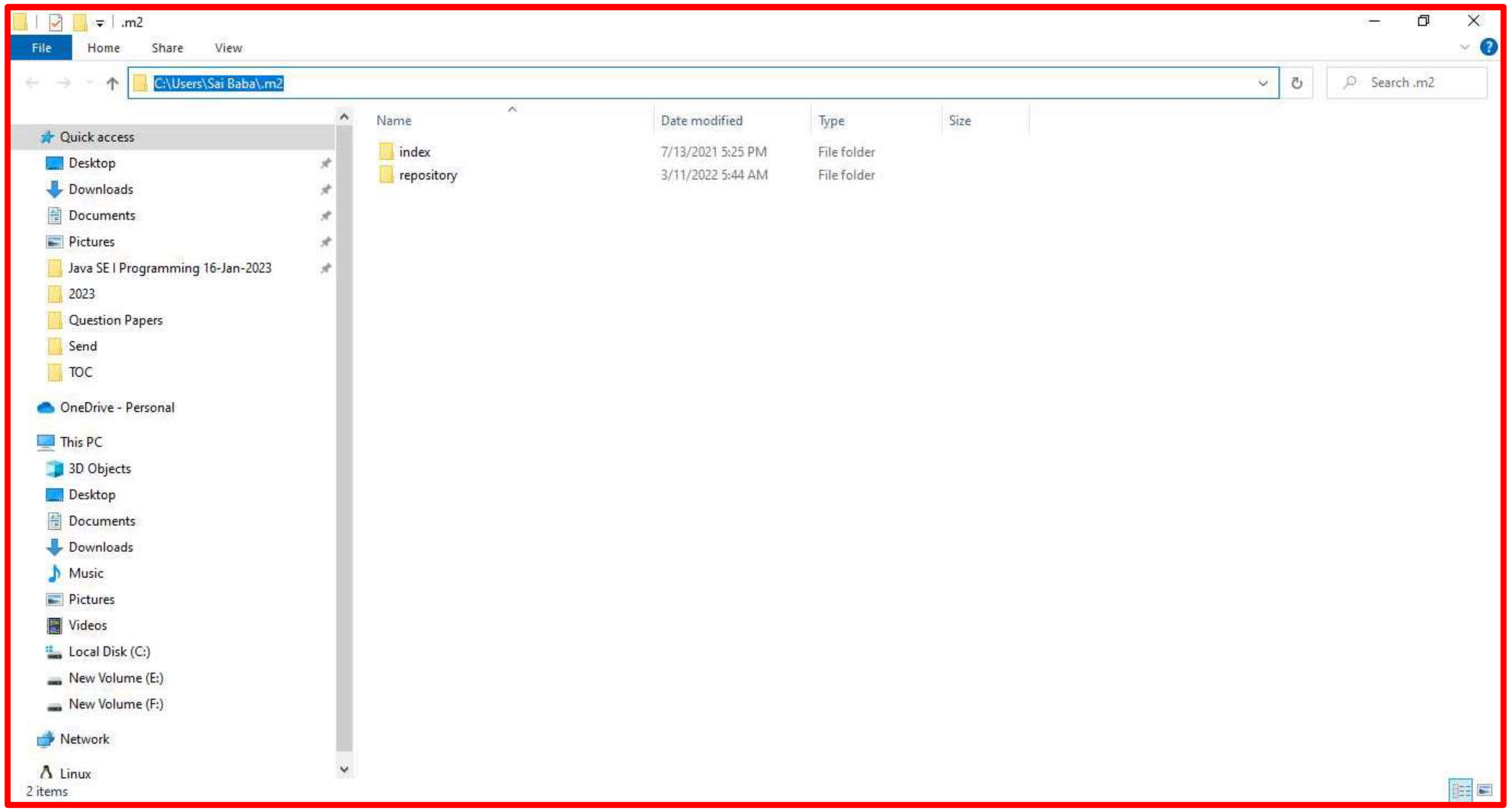


- ***If dependency is not found in these repositories, maven stops processing and throws an error.***

1. Local Repository:

- Local repository refers to the developer's machine, which is where all the project material is saved. This repository contains all the dependency jars.

- Maven **local repository** is located in your local system. It is created by the maven when you run any maven command.
- By default, maven local repository is %USER_HOME%/.m2 directory. For example: **C:\Users\SaiBaba\.m2**.

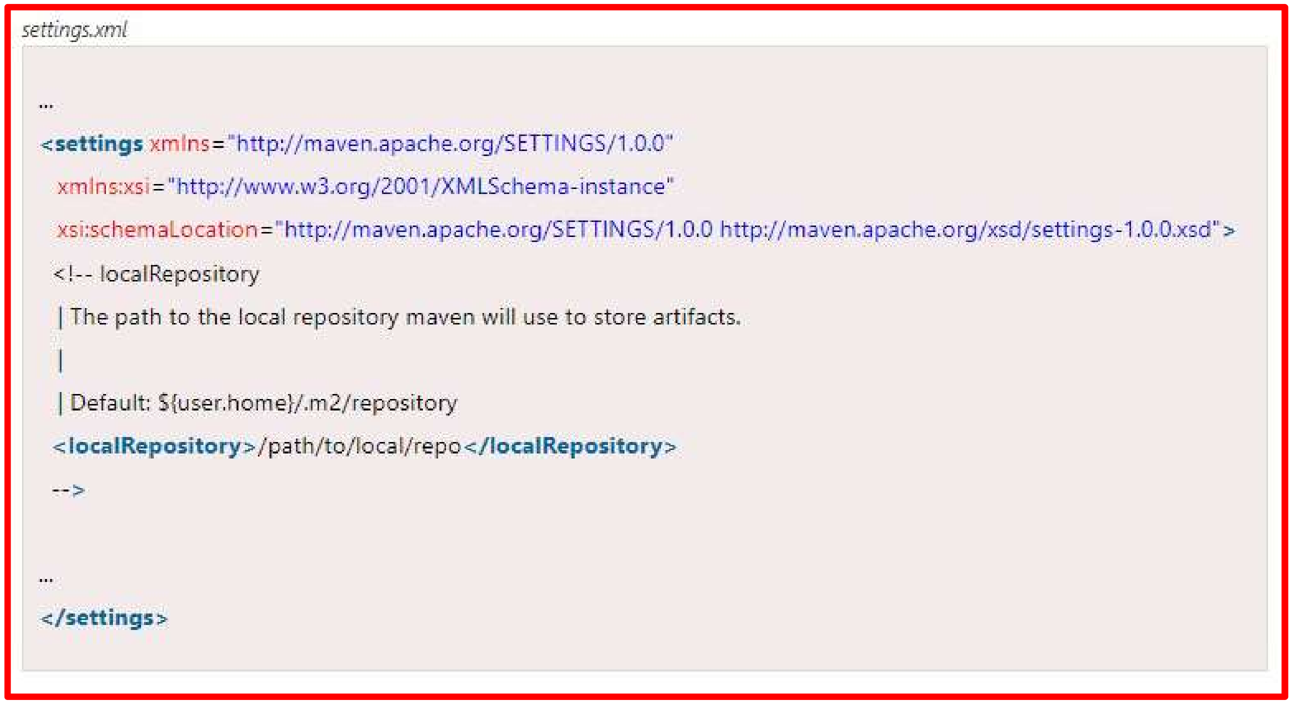


C:\Users\Sai Baba\.m2

Name	Date modified	Type	Size
index	7/13/2021 5:25 PM	File folder	
repository	3/11/2022 5:44 AM	File folder	

Update location of Local Repository

- We can change the location of maven local repository by changing the **settings.xml** file. It is located in **MAVEN_HOME/conf/settings.xml**, for example: **E:\apache-maven-3.1.1\conf\settings.xml**.

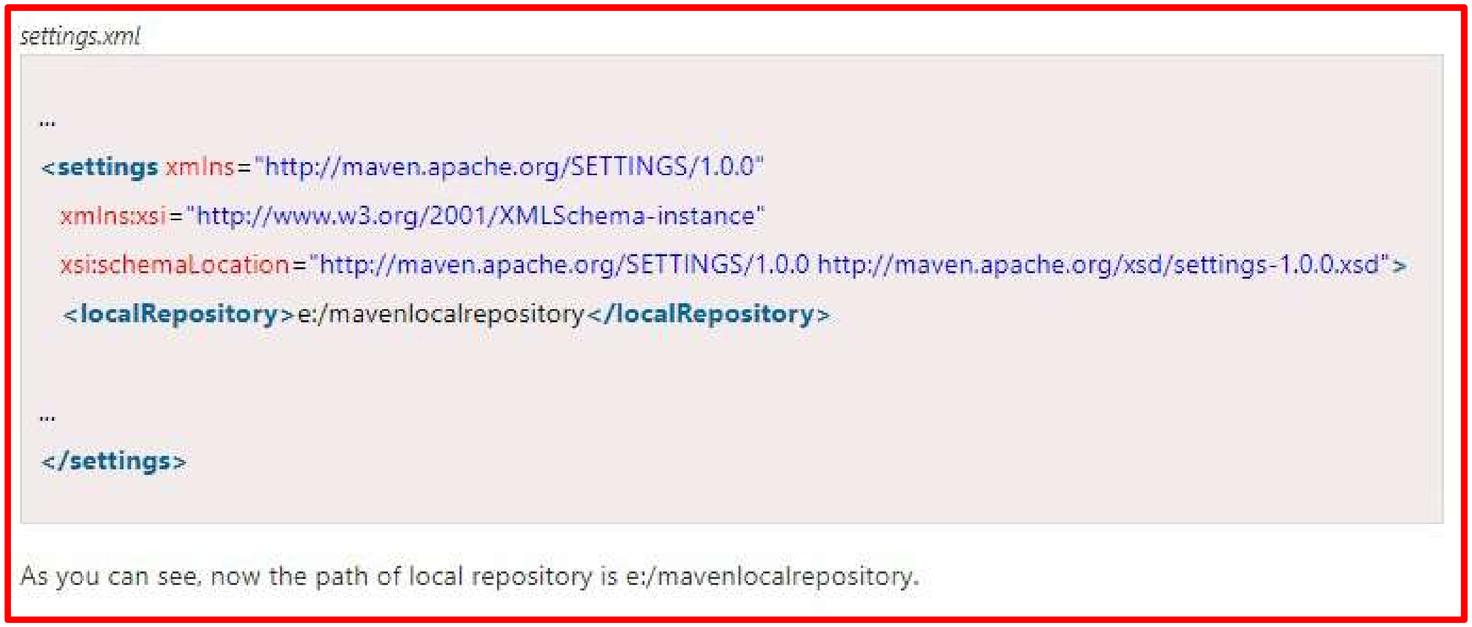


settings.xml

```
...
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
 <!-- localRepository
  | The path to the local repository maven will use to store artifacts.
  |
  | Default: ${user.home}/.m2/repository
 <localRepository>/path/to/local/repo</localRepository>
 -->

...
</settings>
```

- Now change the path to local repository. After changing the path of local repository, it will look like this:



settings.xml

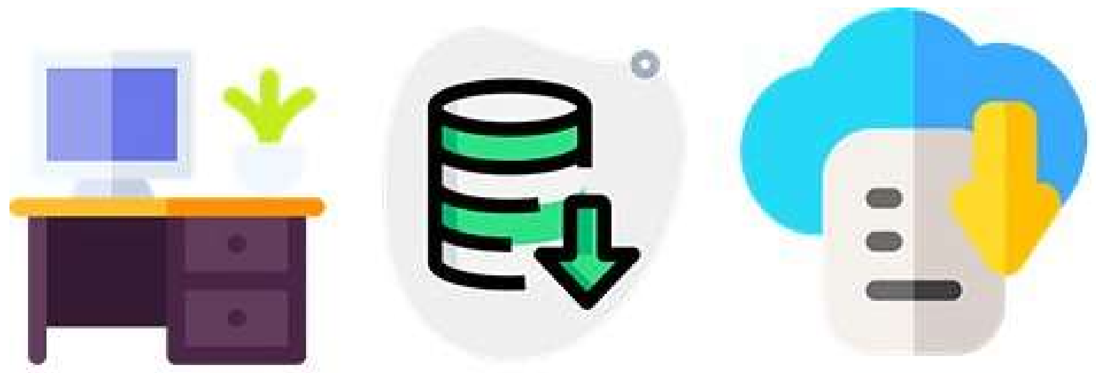
```
...
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository>e:/mavenlocalrepository</localRepository>


...
</settings>
```

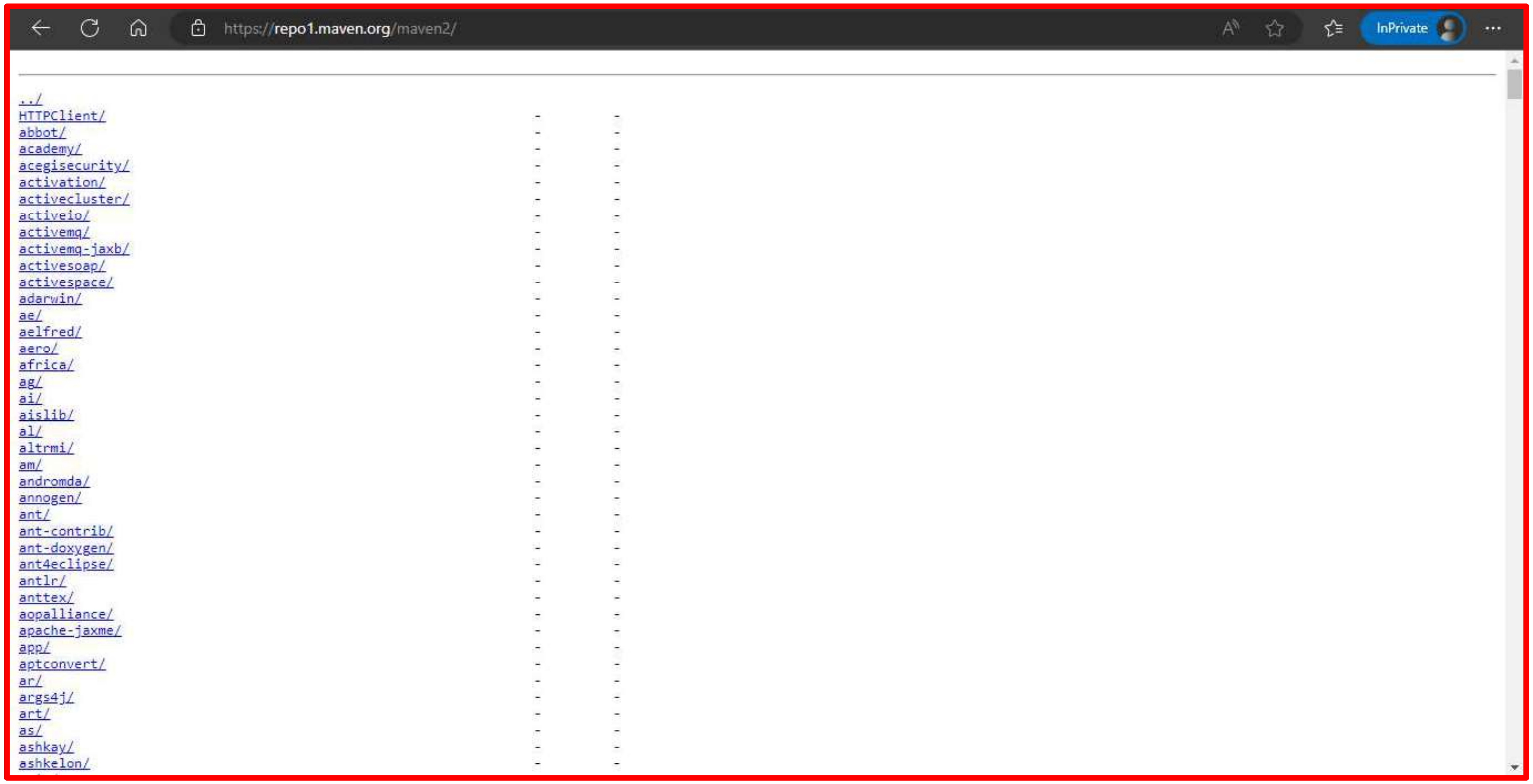
As you can see, now the path of local repository is e:/mavenlocalrepository.

2. Central Repository:

- Central repository refers to the Maven community that comes into action when there is a need for dependencies, and those dependencies cannot be found in the local repository. Maven downloads the dependencies from here whenever needed.



- Maven **central repository** is located on the web. It has been created by the apache maven community itself.
- The path of central repository is: https://repo1.maven.org/maven2/.
- The central repository contains a lot of common libraries



https://repo1.maven.org/maven2/

InPrivate

```
../
HTTPClient/                                                       -          -
abbot/                                                            -          -
academy/                                                          -          -
acegisecurity/                                                    -          -
activation/                                                       -          -
activecluster/                                                    -          -
activeio/                                                         -          -
activemq/                                                         -          -
activemq-jaxb/                                                    -          -
activesoap/                                                       -          -
activespace/                                                      -          -
adarwin/                                                          -          -
ae/                                                               -          -
aelfred/                                                          -          -
aero/                                                             -          -
africa/                                                           -          -
ag/                                                               -          -
ai/                                                               -          -
aislib/                                                           -          -
al/                                                               -          -
altrmi/                                                           -          -
am/                                                               -          -
andromda/                                                         -          -
annogen/                                                          -          -
ant/                                                              -          -
ant-contrib/                                                      -          -
ant-doxygen/                                                      -          -
ant4eclipse/                                                      -          -
antlr/                                                            -          -
anttex/                                                           -          -
aopalliance/                                                      -          -
apache-jaxme/                                                     -          -
app/                                                              -          -
aptconvert/                                                       -          -
ar/                                                               -          -
args4j/                                                           -          -
art/                                                              -          -
as/                                                               -          -
ashkay/                                                           -          -
ashkelon/                                                         -          -
```

3. Remote Repository:

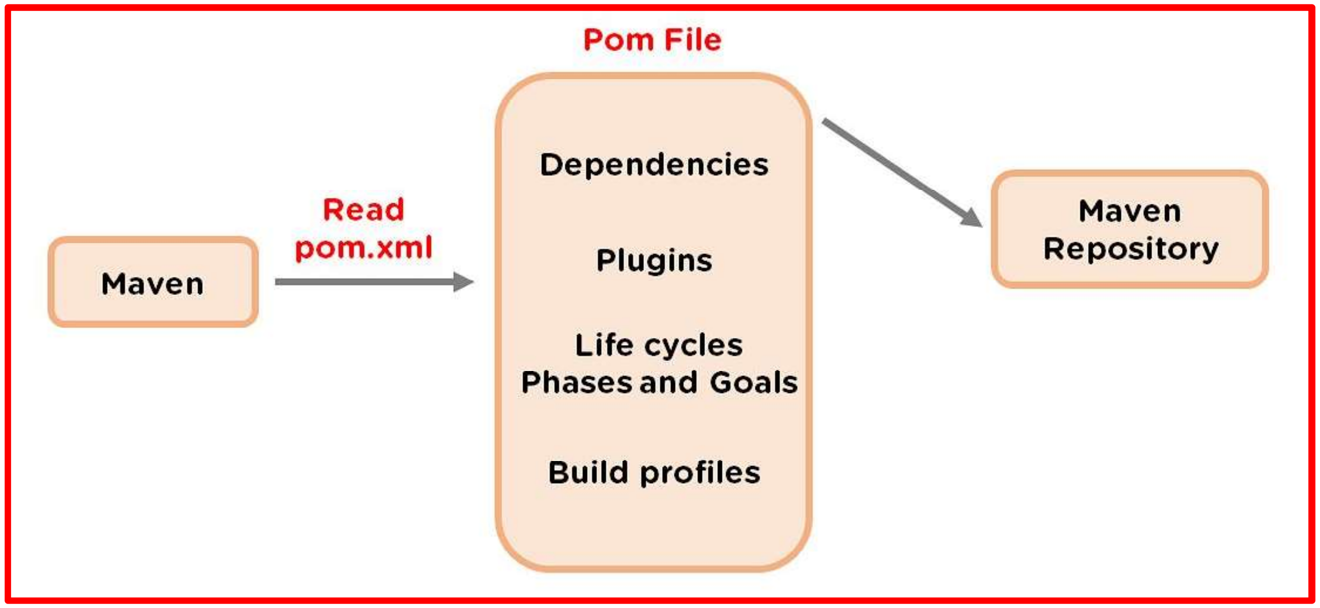
- The remote repository refers to the repository present on a web server, which is used when Maven needs to download dependencies.

- This repository works the same as the central repository. Whenever anything is required from the remote repository, it is first downloaded to the local repository, and then used.

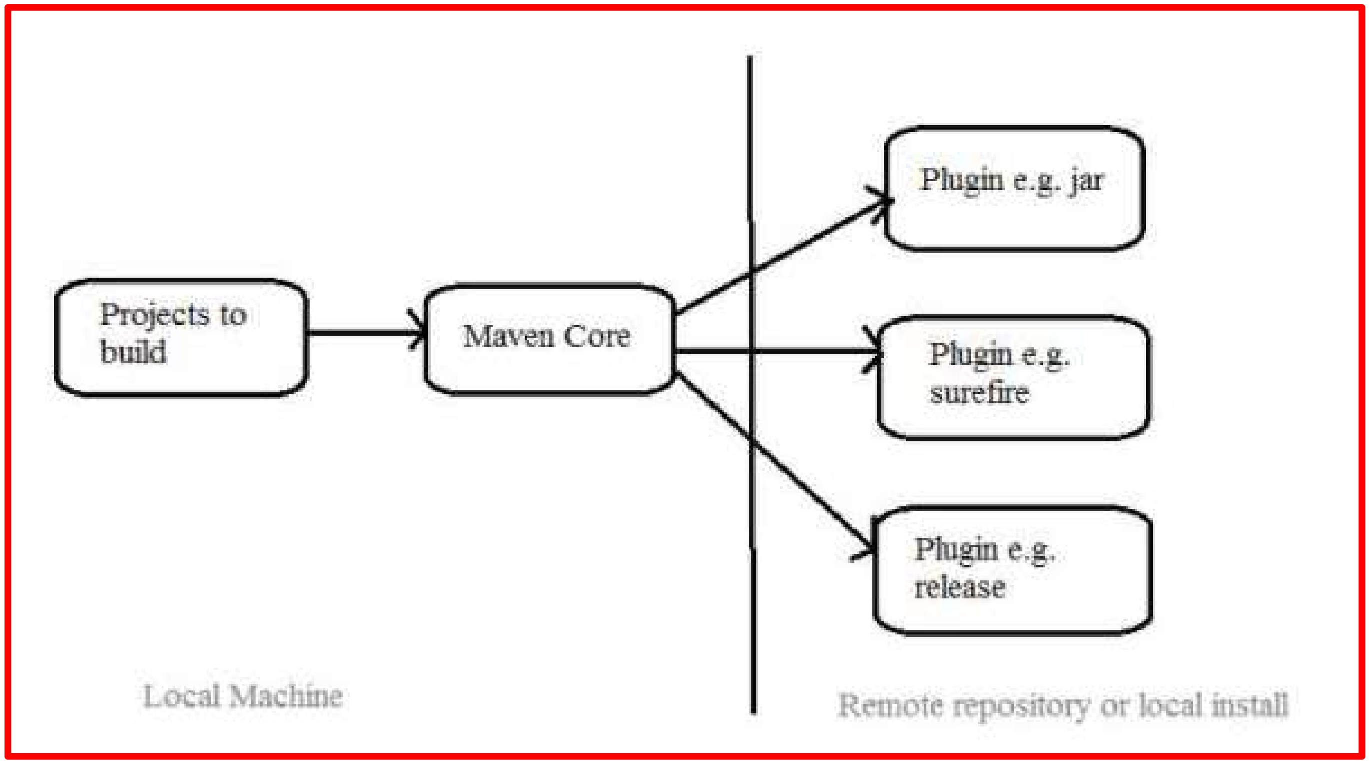
- Maven **remote repository** is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in pom.xml file.

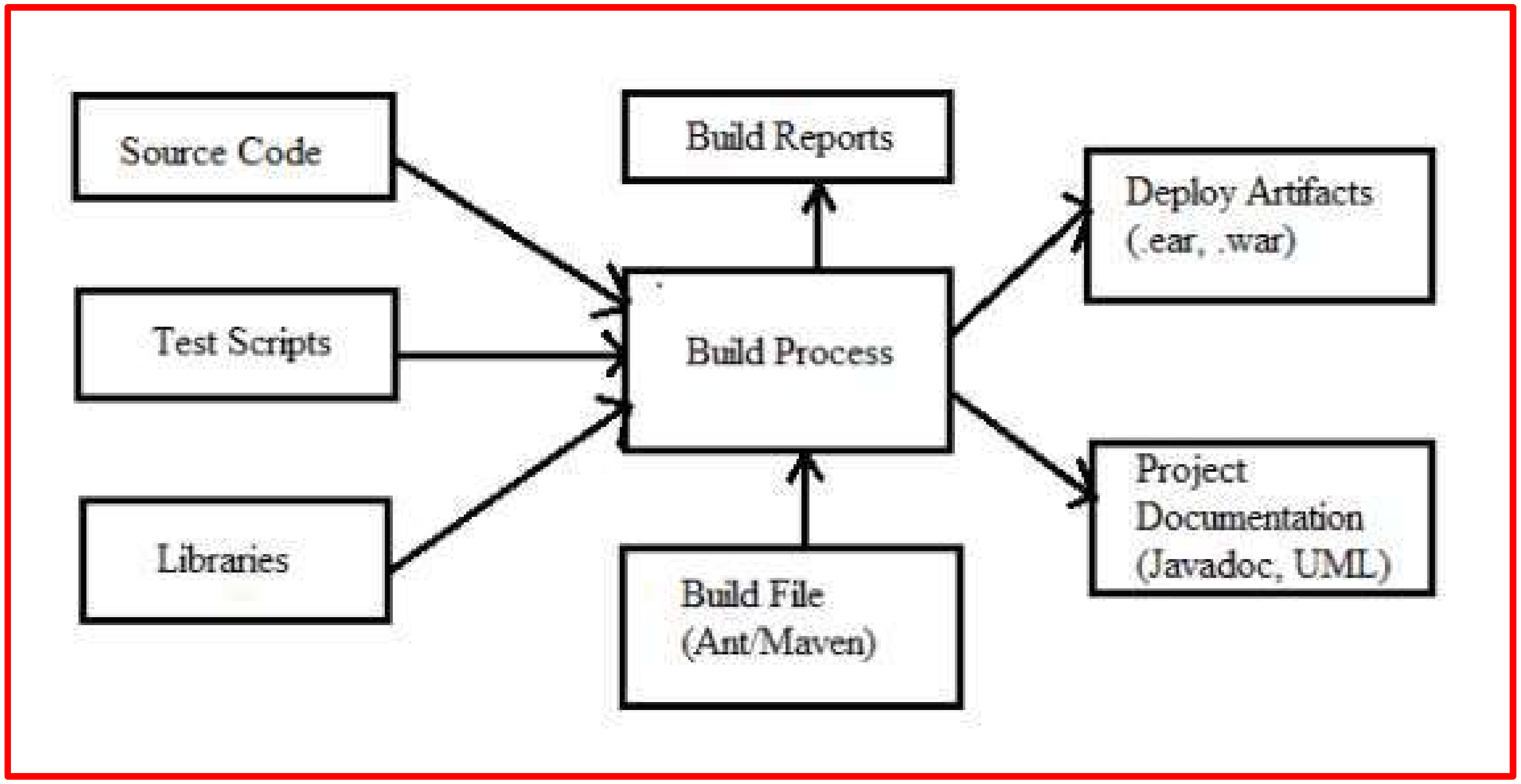
Maven Architecture

- The projects created in Maven contain POM files that describe the aspect of the project essentials. Maven architecture shows the process of creating and generating a report according to the requirements and executing lifecycles, phases, goals, plugins, and so on—from the first step.



Maven Architecture




Source Code
Test Scripts
Libraries
Build Reports
Build Process
Build File (Ant/Maven)
Deploy Artifacts (.ear, .war)
Project Documentation (Javadoc, UML)

How to use Maven

1. To configure the Maven in Java, you need to use Project Object Model, which is stored in a pom.xml-file.
2. POM includes all the configuration setting related to Maven. Plugins can be configured and edit in the <plugins> tag of a pom.xml file. And developer can use any plugin without much detail of each plugin.
3. When user start working on Maven Project, it provides default setting of configuration, so the user does not need to add every configuration in pom.xml

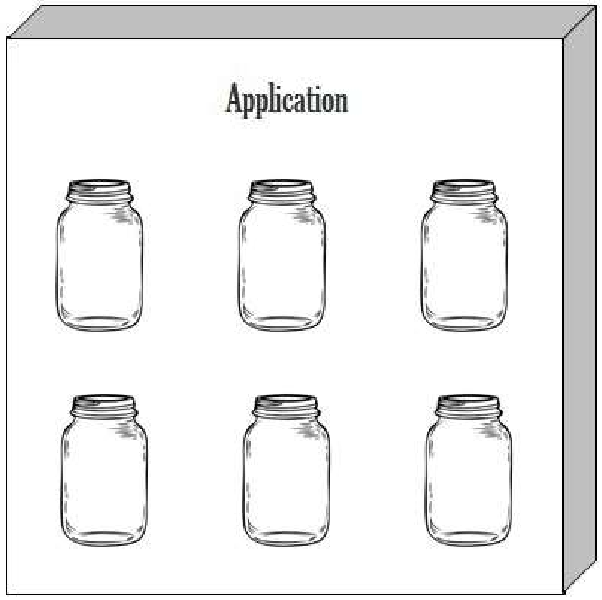
Steps/process involved in building the project:

- Add / Write the code for application creation and process that into source code repository
- Edit configuration / pom.XML / plugin details
- Build the application
- Save the build process output as WAR or EAR file to a local location or server
- Get the file from local location or server and deploy the file to the production site or
- client site Updated the application document with date and updated version number of the application
- create and generate a report as per the application or requirement.

POM.xml

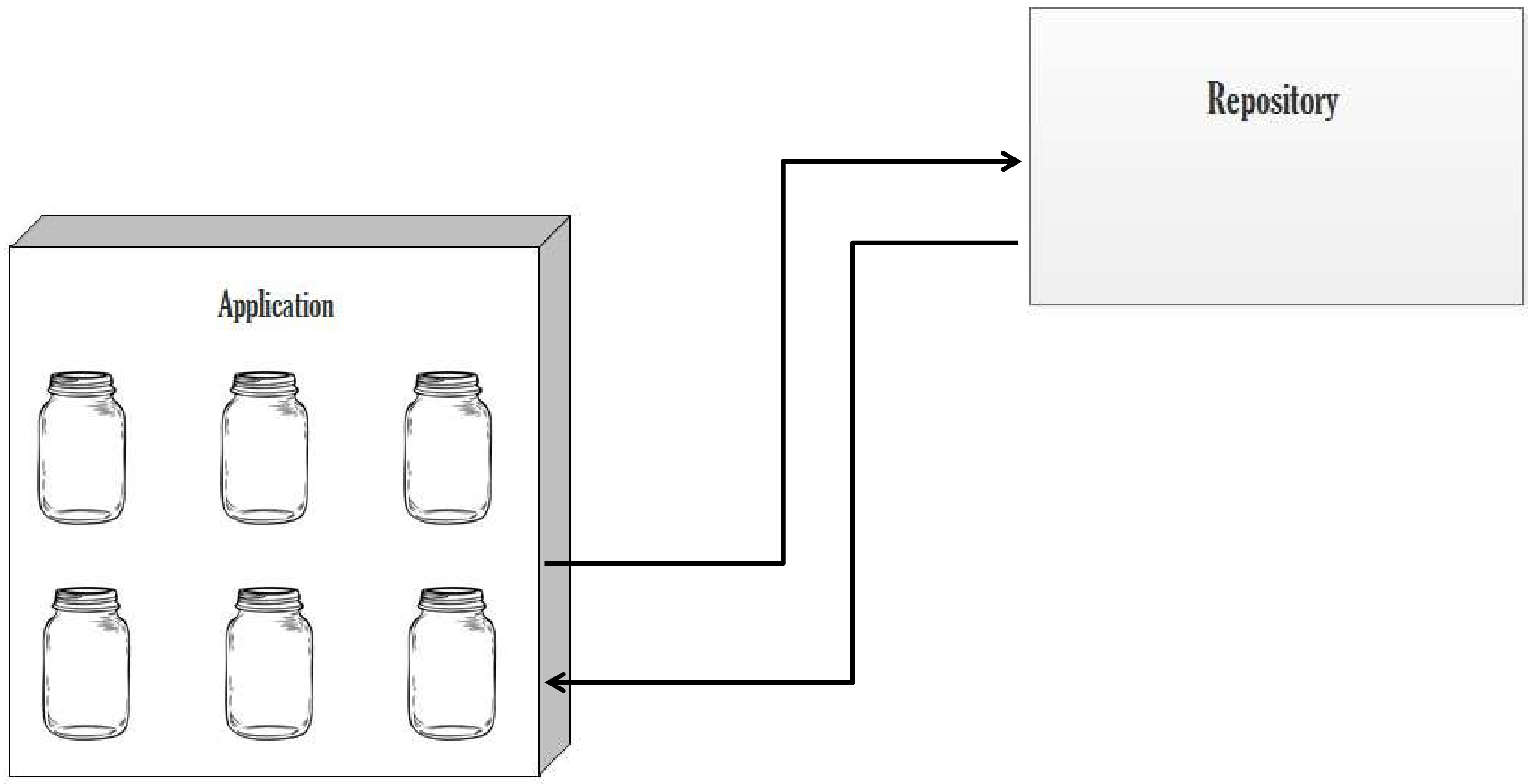
Java Dependencies

- When we build our Applications we need to manage dependencies
- We need to import the Jarz that are required for the Spring Applications to work
- Typically what we do is, we get the Jarz and Add it to the Classpath



Achieving it with Maven

- Maven lets u declare all the dependencies that you want in a Single file.
- We Don’t need to download Jarz and add it to our Classpath
- We Just Mention Which all Jarz We Need in one file [POM.xml]
- This file contains list of all the dependencies, Maven needs to know [Dependency Management Tool]
- So When we run the Maven Command, Maven is going to look at your dependencies, [These are all the things that this App Needs]
- It goes to the repository, which contains all the dependencies that we ever need, and Repository is going to provide all of them.

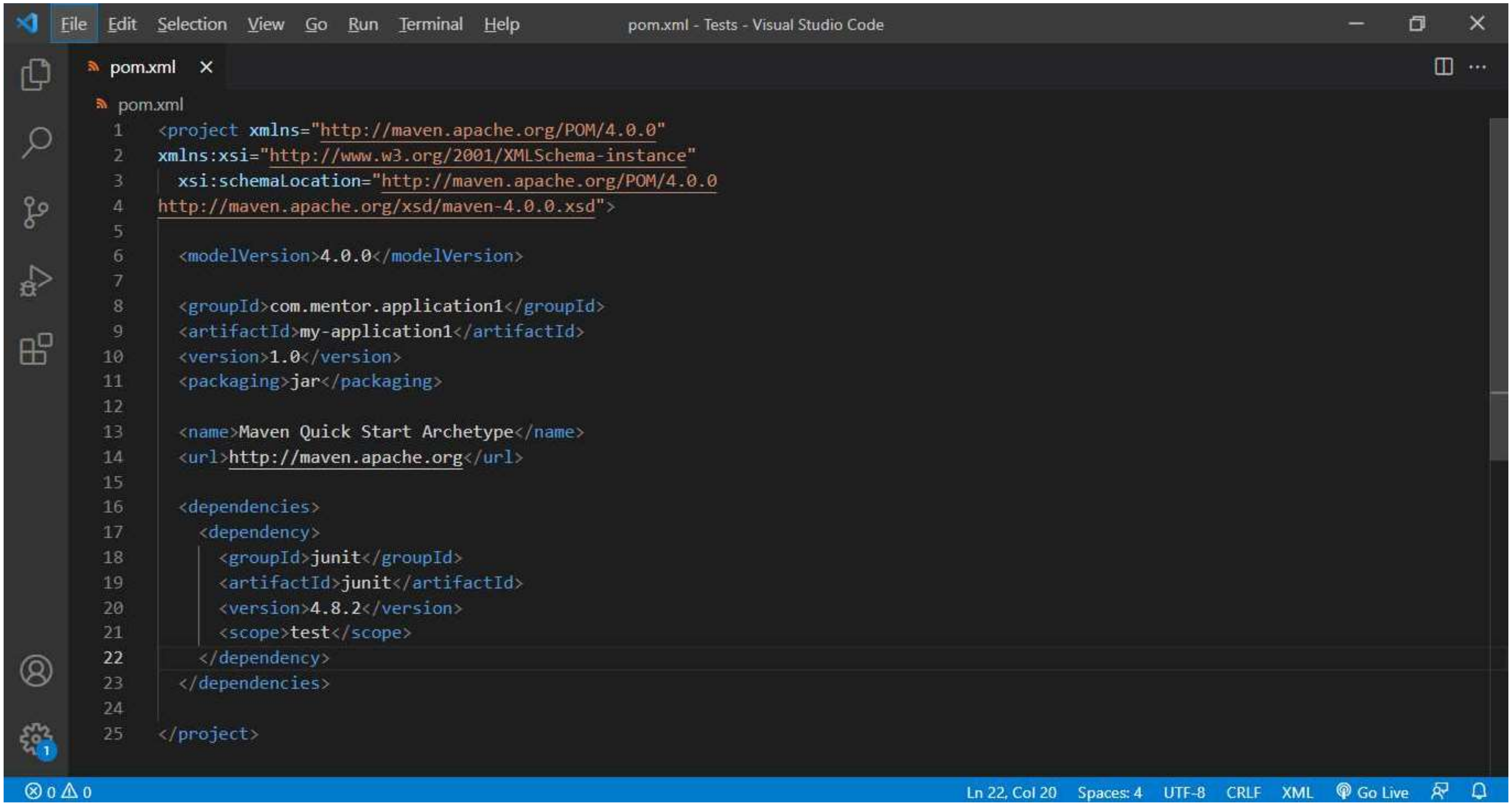

Repository
Application

Project Object Model (POM)

1. Project Object Model (POM) refers to the XML files with all the information regarding project and configuration details
2. It contains the project description, as well as details regarding the versioning and configuration management of the project
3. The XML file is in the project home directory. Maven searches for the POM in the current directory when any given task needs to be executed

- The configuration of a Maven project is done via a *Project Object Model (POM)*, represented by a *pom.xml* file. The *POM* describes the project, manages dependencies, and configures plugins for building the software.

- The *POM* also defines the relationships among modules of multi-module projects.


File Edit Selection View Go Run Terminal Help
pom.xml - Tests - Visual Studio Code
pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mentor.application1</groupId>
  <artifactId>my-application1</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

Ln 22, Col 20 Spaces: 4 UTF-8 CRLF XML Go Live

Project Identifiers

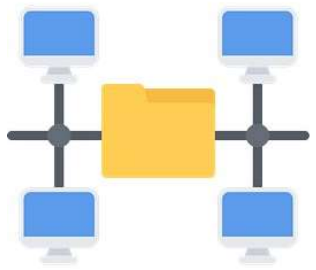
- Maven uses a set of identifiers, also called coordinates, to uniquely identify a project and specify how the project artifact should be packaged:
  1. *groupId* – a unique base name of the company or group that created the project
  2. *artifactId* – a unique name of the project
  3. *version* – a version of the project
  4. *packaging* – a packaging method (e.g. *WAR*/*JAR*/*ZIP*)
- The first three of these (*groupId:artifactId:version*) combine to form the unique identifier and are the mechanism by which you specify which versions of external libraries (e.g. JARs) your project will use.

Dependencies & Repositories

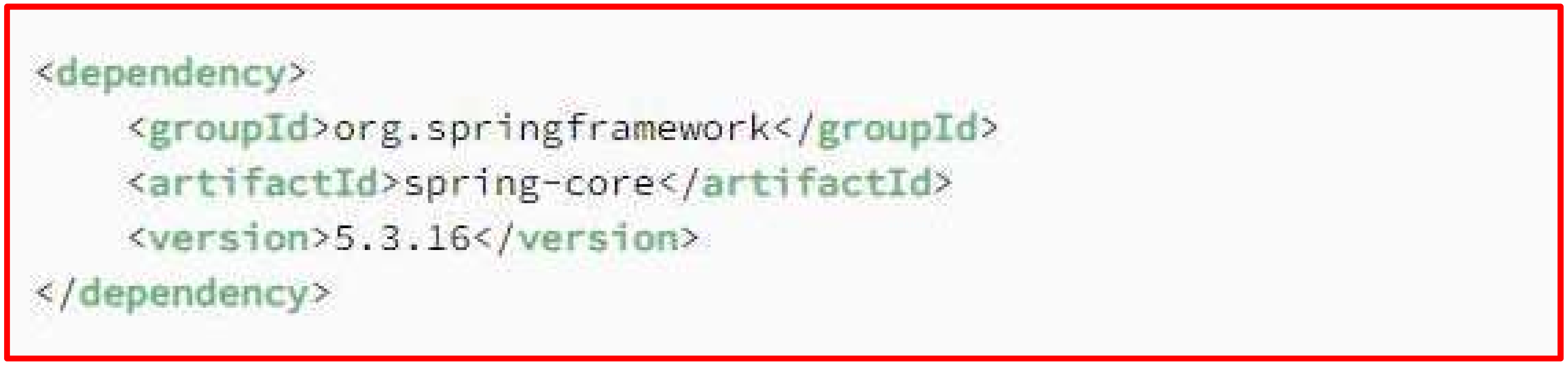
- These external libraries that a project uses are called dependencies. The dependency management feature in Maven ensures the automatic download of those libraries from a central repository, so you don't have to store them locally.

- Dependencies refer to the Java libraries required for the project. Repositories refer to the directories of packaged JAR files.

- If the dependencies are not present in your local repository, then Maven downloads them from a central repository and stores them in the local repository.



- This is a key feature of Maven and provides the following benefits:
  1. **uses less storage by significantly reducing the number of downloads off remote repositories**
  2. **makes checking out a project quicker**
  3. **provides an effective platform for exchanging binary artifacts within your organization and beyond without the need for building artifacts from source every time**
- In order to declare a dependency on an external library, you need to provide the *groupId, artifactId*, and *version* of the library

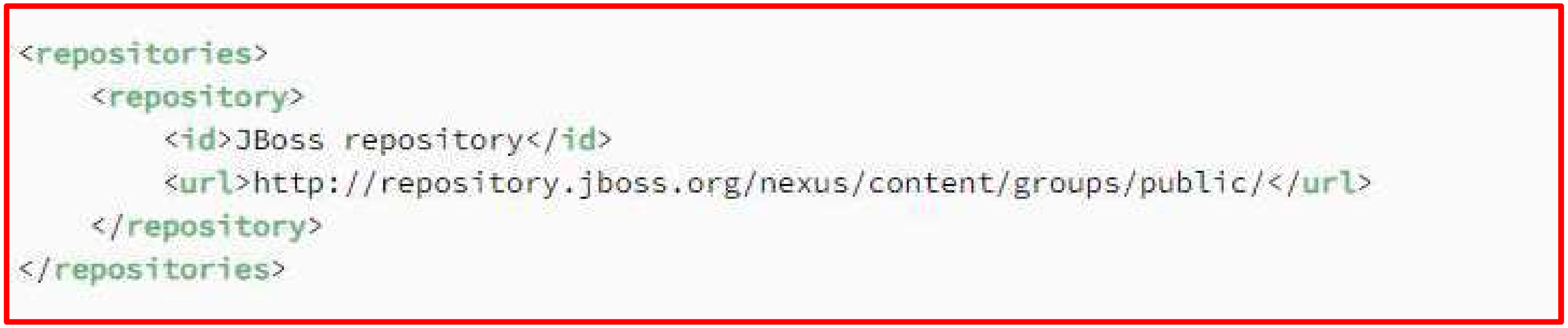


```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.16</version>
</dependency>
```

- As Maven processes the dependencies, it will download the Spring Core library into your local Maven repository.

Repositories

- A repository in Maven is used to hold build artifacts and dependencies of varying types. The default local repository is located in the *.m2/repository* folder under the home directory of the user.
- If an artifact or a plugin is available in the local repository, Maven uses it. Otherwise, it is downloaded from a central repository and stored in the local repository. The default central repository is Maven Central.
- Some libraries, such as the JBoss server, are not available at the central repository but are available at an alternate repository. For those libraries, you need to provide the URL to the alternate repository inside the *pom.xml* file:

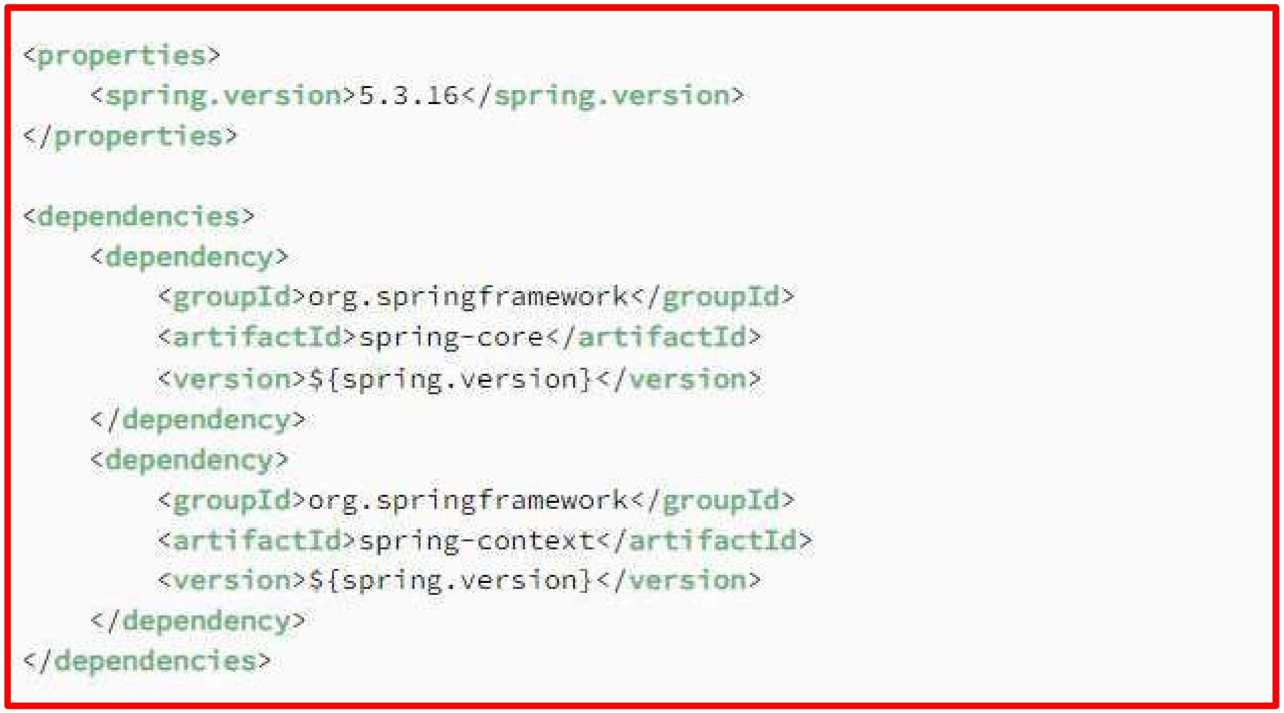


```
<repositories>
    <repository>
        <id>JBoss repository</id>
        <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    </repository>
</repositories>
```

- **Note that you can use multiple repositories in your projects.**

Properties

- Custom properties can help to make your *pom.xml* file easier to read and maintain. In the classic use case, you would use custom properties to define versions for your project's dependencies.
- **Maven properties are value-placeholders and are accessible anywhere within a *pom.xml* by using the notation *${name}***, where *name* is the property.

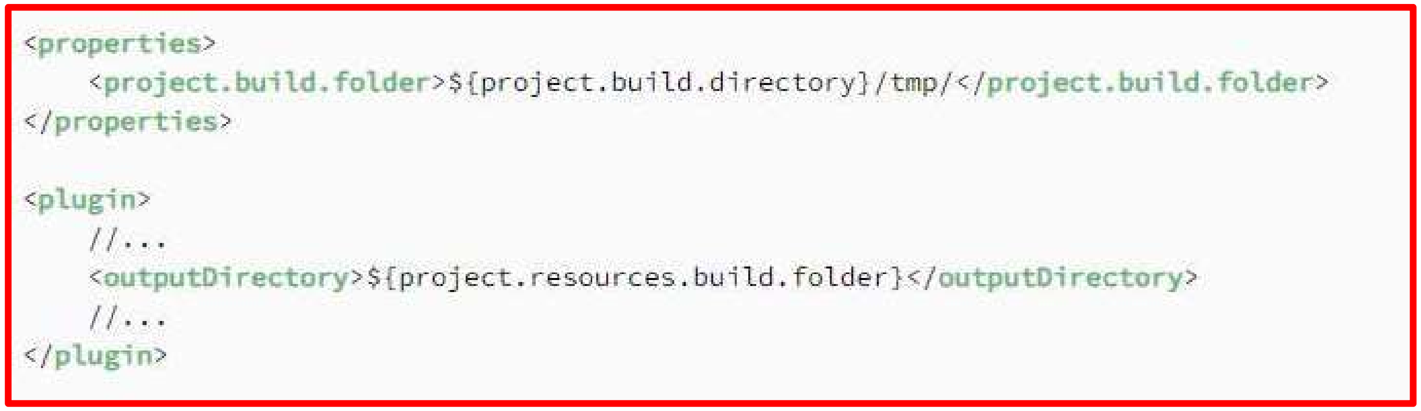


```xml
<properties>
    <spring.version>5.3.16</spring.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
</dependencies>
```

- **Now if you want to upgrade Spring to a newer version, you only have to change the value inside the<*spring.version*> property tag and all the dependencies using that property in their <*version*> tags will be updated.**

- Properties are also often used to define build path variables

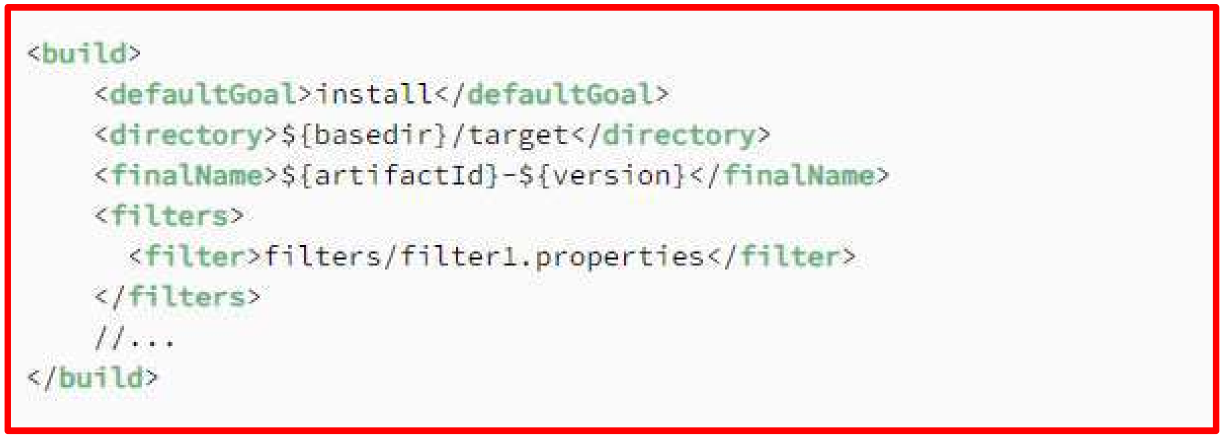


```
<properties>
    <project.build.folder>${project.build.directory}/tmp/</project.build.folder>
</properties>

<plugin>
    //...
    <outputDirectory>${project.resources.build.folder}</outputDirectory>
    //...
</plugin>
```

Build

- The *build* section is also a very important section of the Maven *POM.* It provides information about the default Maven *goal*, the directory for the compiled project, and the final name of the application. The default *build* section looks like this:



```
<build>
    <defaultGoal>install</defaultGoal>
    <directory>${basedir}/target</directory>
    <finalName>${artifactId}-${version}</finalName>
    <filters>
      <filter>filters/filter1.properties</filter>
    </filters>
    //...
</build>
```

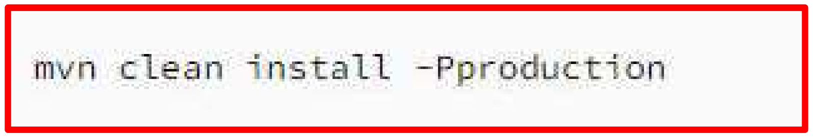
- **The default output folder for compiled artifacts is named *target*, and the final name of the packaged artifact consists of the *artifactId* and *version*, but you can change it at any time.**

Build Profiles

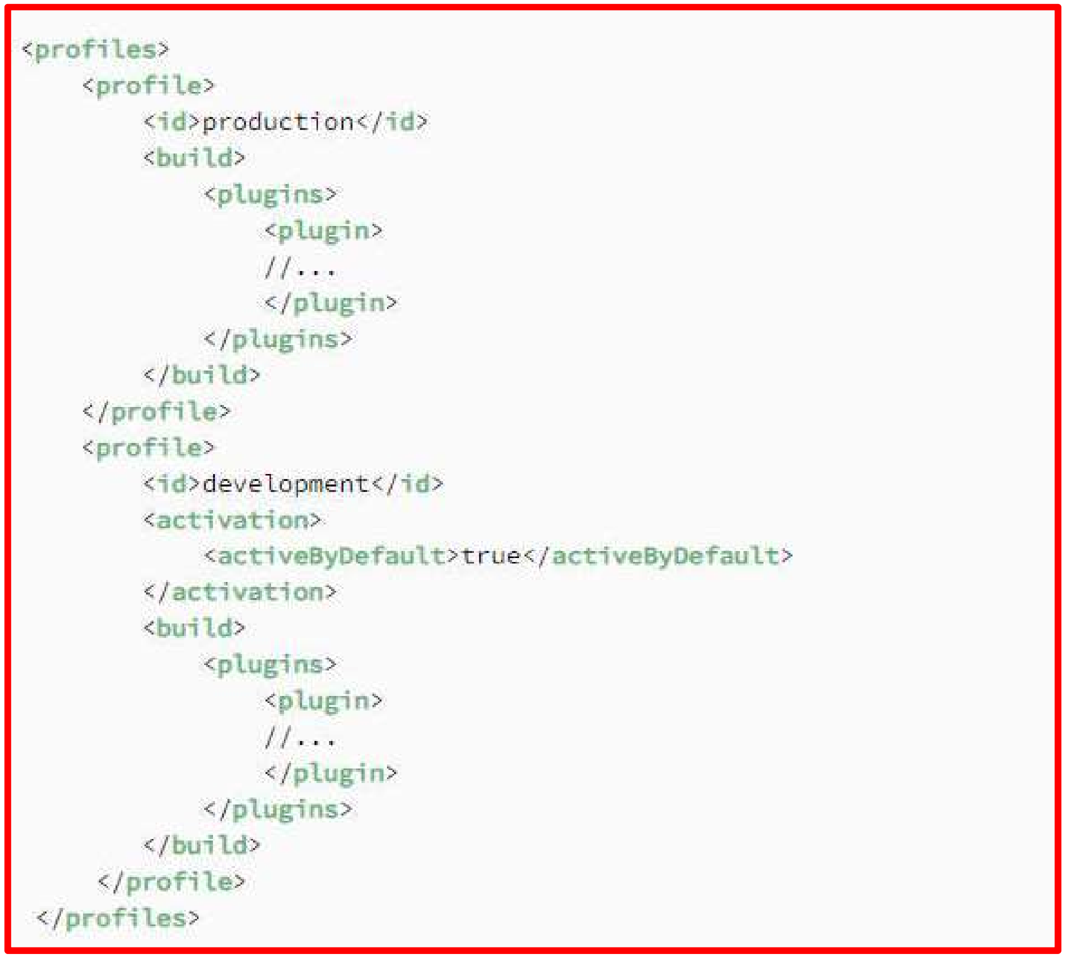
- This refers to the set of configuration values required to build a project using different configurations
- Different build profiles are added to the POM files when enabling different builds
- A build profile helps in customizing the build for different environments

Using *Profiles*

- Another important feature of Maven is its support for *profiles.* A *profile* is basically a set of configuration values. By using *profiles*, you can customize the build for different environments such as Production/Test/Development:
- The default profile is set to *development*. If you want to run the *production profile*, you can use the following Maven command:

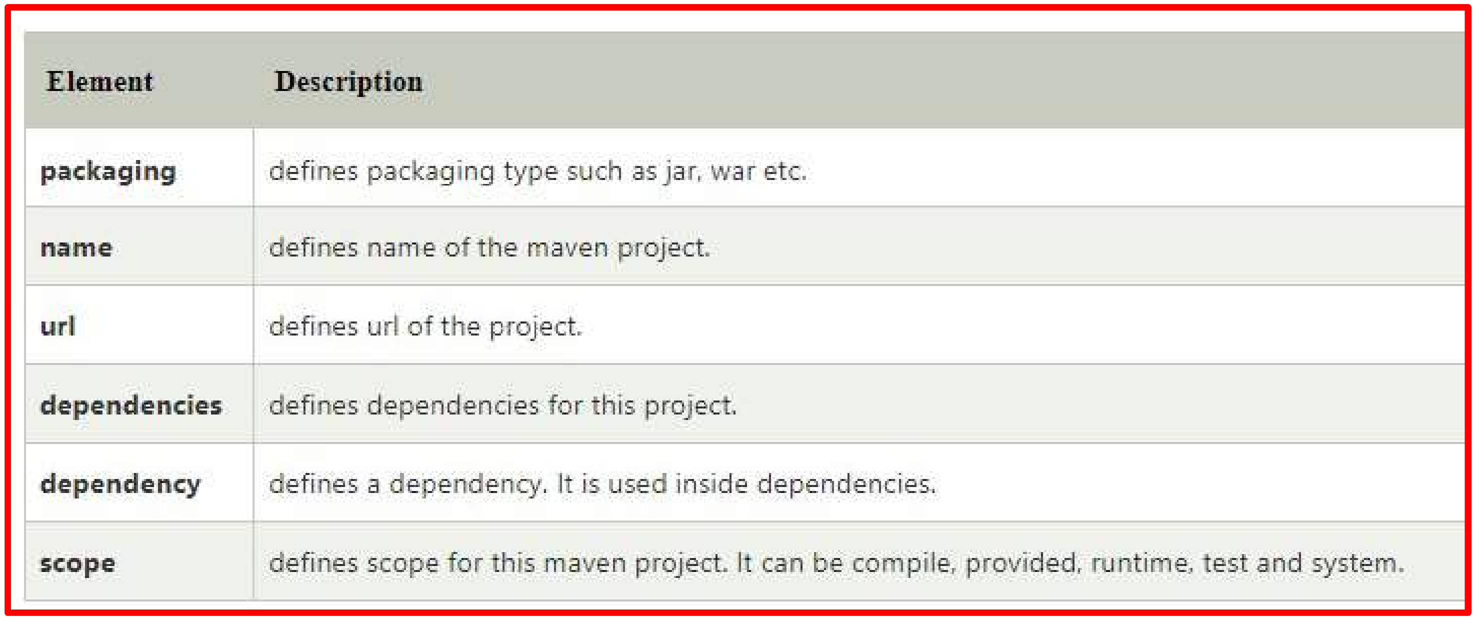


```
mvn clean install -Pproduction
```



```
<profiles>
    <profile>
        <id>production</id>
        <build>
            <plugins>
                <plugin>
                //...
                </plugin>
            </plugins>
        </build>
    </profile>
    <profile>
        <id>development</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <build>
            <plugins>
                <plugin>
                //...
                </plugin>
            </plugins>
        </build>
     </profile>
 </profiles>
```

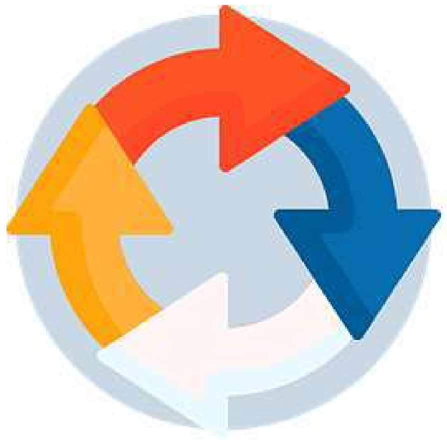
Maven pom.xml file with additional elements



Element	Description
packaging	defines packaging type such as jar, war etc.
name	defines name of the maven project.
url	defines url of the project.
dependencies	defines dependencies for this project.
dependency	defines a dependency. It is used inside dependencies.
scope	defines scope for this maven project. It can be compile, provided, runtime, test and system.

Build Life Cycles, Phases and Goals

- This consists of a sequence of build phases, and each build phase consists of a series of goals
- Each goal is responsible for a particular task
- When a process is executed, all purposes related to that phase and its plugins are also compiled

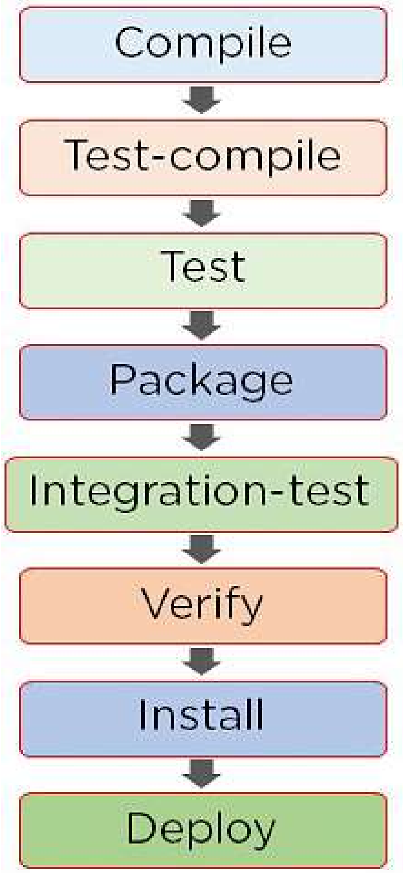


- Every Maven build follows a specified *lifecycle*. You can execute several build *lifecycle goals*, including the ones to *compile* the project’s code, create a *package,* and *install* the archive file in the local Maven dependency repository.

Lifecycle Phases

The following list shows the most important Maven *lifecycle* phases:

1. *validate* – checks the correctness of the project
2. *compile* – compiles the provided source code into binary artifacts
3. *test* – executes unit tests
4. *package* – packages compiled code into an archive file
5. *integration-test* – executes additional tests, which require the packaging
6. *verify* – checks if the package is valid
7. *install* – installs the package file into the local Maven repository
8. *deploy* – deploys the package file to a remote server or repository


Compile
Test-compile
Test
Package
Integration-test
Verify
Install
Deploy

There are three types of standard life cycles:

1. Default:

   The default or build life cycle is the primary cycle that is responsible for building the application. There are 21 different phases in this primary life cycle, starting from the process of validation to the step of deployment.

2. Clean:

   The clean life cycle handles project cleaning. The clean phase consists of the following three steps:

   1. Pre-clean
   2. Clean
   3. Post-clean

   The mvn post-clean command is used to invoke the clean lifestyle phases in Maven.

3. Site:

The Maven site plugin handles the project site's documentation, which is used to create reports, deploy sites, etc.

The phase has four different stages:

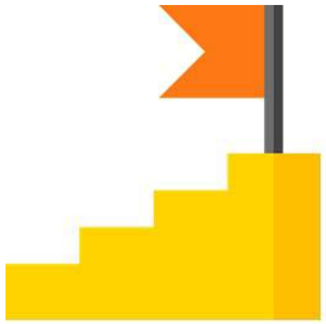
1. Pre-site
2. Site
3. Post-site
4. Site-deploy

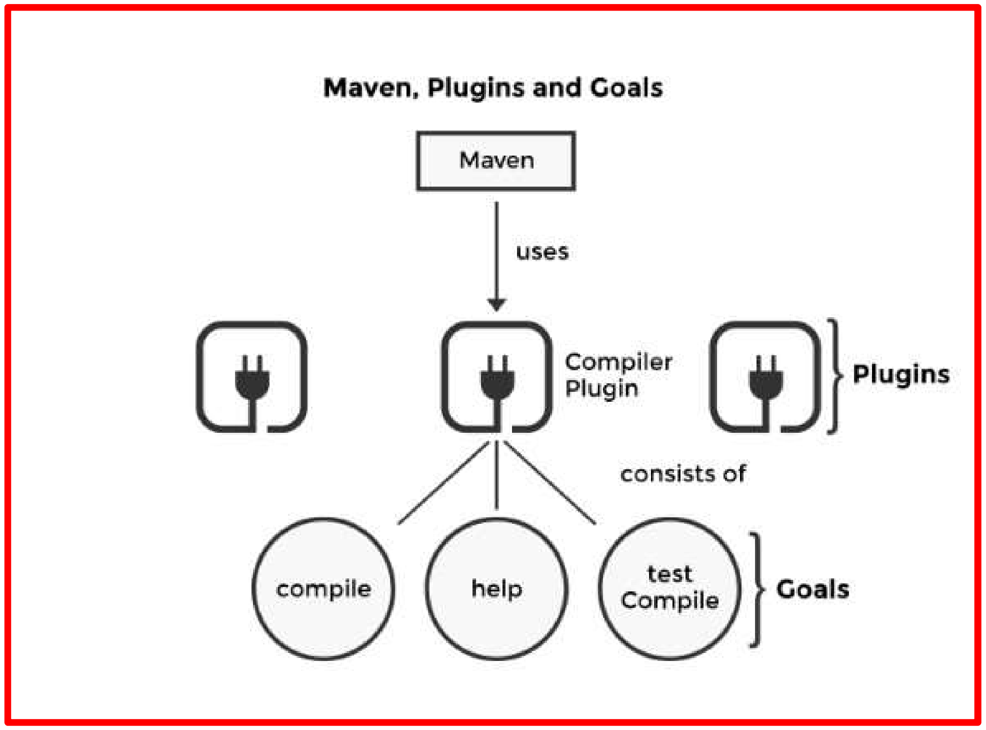
Ultimately, the site that is created contains the project report.

Plugins and *Goals*

- A Maven *plugin* is a collection of one or more *goals*. Goals are executed in phases, which helps to determine the order in which the *goals* are executed.
- The rich list of plugins that are officially supported by Maven is available here. [Maven – Available Plugins (apache.org)]

- The plugins are used to perform a specific goal
- Maven has its standard plugins that can be used. If desired, users can also implement their own in Java




Maven, Plugins and Goals
Maven
uses
Compiler Plugin
Plugins
consists of
compile
help
test Compile
Goals

- To go through any one of the above phases, we just have to call one command

```
mvn <phase>
```

- For example, *mvn clean install* will remove the previously created jar/war/zip files and compiled classes (*clean)* and execute all the phases necessary to install the new archive (*install)*.
- **Please note that *goals* provided by *plugins* can be associated with different phases of the *lifecycle*.**

Summary

In this lesson, you should have learned how to:

- An Introduction to Maven
- Installing Maven
- Core Concepts
- The Project Object Model (POM)
- Custom Builds