Apache Maven

Apache

Apache is a company well known for open source products.
Few products from apache

- Maven
- Ant

- Hadoop
- Tomcat
- Elastic search
- etc...

Apache Maven

Maven is a powerful **build & dependency** management tool for Java software projects. Maven is primarily known for build and dependency management however it can do more than that.

What is a build tool?

Build tools typically automates the following activities

- Organise project specific files
- Compile the source code
- Generates documentation for the source code
- Runs Junit and integration test cases
- Create software deployable package (war/ear/jar)

What is Junit?

Junit is a framework for automating unit testing, whenever we add new features to the software we need to retested with all functionalities, developers write Junit test cases.

What is integration testing?

Integration test cases are written by QA team, integrations testing is testing end to end flow QA teams use tools like selenium, QAT for automating integration testing. This can be integrated with build tool like maven.

What is war/ear/jar ?

War stands **W**eb**Ar**chive, it a format to package web applications
Ear stands for **E**nterprise **Ar**chive, this format is for EJB based applications
Jar stands for **J**ava **Ar**chive

What is dependency management?

Dependency:

We our project wants to use a framework, frameworks come as a jar file, this jar is our project dependency.

Transitive Dependency:

A dependency on which our dependency depends on

a.jar is our dependency and **b.jar** is our transitive dependency

Dependency Management:

Maven automatically manages dependencies and transitive dependencies by downloading them from maven repositories.

Maven repositories

Is the server where maven maintains all the dependencies with different versions of it.
Maven mainly deals with three different repositories

- **Central repository**, maintained over internet
    - Dependencies which are publicly accessible to everyone
- **Remote repository**, maintained within organization
    - Organization specific dependencies which should not be public is maintained here.
- **Local repository**, maintained in our local machine
    - When we run build first time maven downloads dependencies from central, remote and next time onwards picks dependencies from local.

Maven pom.xml

Is a configuration file used by maven to perform its tasks, Maven looks for pom.xml in the current directory when we run maven commands.
POM stands for Project Object Model

Sample pom.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/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>in.javahome</groupId>
        <artifactId>myweb</artifactId>
        <packaging>war</packaging>
        <version>0.7.0</version>
        <name>myweb Maven Webapp</name>
        <url>http://maven.apache.org</url>
        <!-- This is comment in XML-->
        <dependencies>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>3.0.1</version>
                </dependency>
                <dependency>
                   <groupId>org.springframework</groupId>
                   <artifactId>spring-core</artifactId>
                   <version>4.3.10.RELEASE</version>
                </dependency>
        </dependencies>
 </project>
```

Pom.xml attributes

groupId: This reflects the client information for whom we are developing this project. Technically we can put any value in this, but we follow the following convention.
By convention it should be company name or reverse domain name of the company as follows

```
        <groupId>in.javahome</groupId>
        <groupId>net.citi</groupId>
        <groupId>icici</groupId>
```

artifactId: This represents project name, some examples

```
       <artifactId>online-shopping</artifactId>
       <artifactId>online-banking</artifactId>
      <artifactId>order-tracking-system</artifactId>
```

packaging: This represents the software package format that needs to be created by maven, few examples

```
<packaging>war</packaging>

<packaging>jar</packaging>
<packaging>ear</packaging>
```

version: current version of the project

```
<version>1.7.2</version>
```

Version numbers also has conventions, in our example

1 is Major version
7 is Minor version
2 is a patch or bug fix

Maven plugins

Plugin gives additional functionalities to the tool

Maven plugin examples

- Compiler plugin (Specify a specific version of a compiler to use)
- Docker plugin (Used to create docker images)
- Jetty server plugin (configuring a web server to deploy over code on jetty web server)
- Maven sonatype nexus plugin (Nexus is a Maven remote repository)

Maven Build lifecycle (Interview Question)

- Validate, validates *pom.xml* and downloads dependencies
- Compile, Compiles the source code
- Test, runs Junit test cases
- Package, creates a software package (war, ear, jar)
- Verify, verifying integration test cases if configured
- Install, copy the package to the local repository
- Deploy, upload the package to remote repository

Note: Here validate, compile, test, package, verify, install, deploy are called as maven goals

Examples using maven commands

- Create a package using maven

1. Clone the project from git
   git clone https://github.com/javahometech/myweb
2. cd myweb
3. mvn package

- Install war file to the local repository
  - mvn install
- Install war file to the local & remote repository
  - mvn install deploy
- Maven target directory

  Target directory is created by maven to organise project out files

Installing and configuring Maven on Linux

Maven depends on java, make sure java is installed.

sudo wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo

sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo

sudo yum install -y apache-maven

Verifying maven installation

mvn --version

Skipping test cases in maven(Interview Question)

mvn package -DskipTests

Snapshot And Release versions

Snapshot Version

Snapshot version is an version which is currently under development.
If our project depends on snapshot version, every time we build maven get latest copy from central/remote repository.

Note: If a version is ending with -SNAPSHOT then it is snapshot version otherwise it is RELEASE version.
Example:

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>4.3.10-SNAPSHOT</version>
</dependency>
```

Release Version

A version whose development is completed is called as release version.
Note: If a version is not ending with -SNAPSHOT then it is RELEASE version.

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>4.3.10</version>
</dependency>
```

Maven settings.xml

This file contains certain information about maven settings, for example

- Local repository path
- Remote repository credentials
- Details about central repositories
- Proxy settings, etc

There are two locations where a `settings.xml` file may live:

- The Maven install: `${maven.home}/conf/settings.xml`
- A user's install: `${user.home}/.m2/settings.xml`