

**Automation Factory**

# **Maven Build Automation**

**Installation and user guide**

**Ganesh Palnitkar**

## Maven as Project management and Build automation tool:

Using Maven, development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage following:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- 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	<code>\${basedir}/src/main/java</code>
resources	<code>\${basedir}/src/main/resources</code>
Tests	<code>\${basedir}/src/test</code>
distributable JAR	<code>\${basedir}/target</code>
Compiled byte code	<code>\${basedir}/target/classes</code>

In order to build the project, Maven provides developers options to mention life-cycle goals and project dependencies (that rely on Maven plugging capabilities and on its default conventions). Much of the project management and build related tasks are maintained by Maven plugins.

Developers can build any given Maven project without need to understand how the individual plugins work. Refer to [Maven Plug-ins](#) section for more detail.

## Installing Apache Maven

The installation of Apache Maven is a simple process of extracting the archive and adding the `bin` folder with the `mvn` command to the `PATH`.

Detailed steps are:

Ensure `JAVA_HOME` environment variable is set and points to your JDK installation

Extract distribution archive in any directory

```
unzip apache-maven-3.3.9-bin.zip
```

or,

```
tar xzvf apache-maven-3.3.9-bin.tar.gz
```

Alternatively use your preferred archive extraction tool.

Add the `bin` directory of the created directory `apache-maven-3.3.9` to the `PATH` environment variable

Confirm with `mvn -v` in a new shell. The result should look similar to

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5;
2015-11-10T22:11:47+05:30)

Maven home: C:\apache-maven-3.3.9\bin\..
Java version: 1.8.0_112, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_112\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family:
"dos"
```

## Configure Maven on Windows

Check environment variable value e.g.

1. `echo %JAVA_HOME%`
2. `C:\Program Files\Java\jdk1.7.0_51`

Adding to PATH: Add the unpacked distribution's bin directory to your user PATH environment variable by opening up the system properties (WinKey + Pause), selecting the "Advanced" tab, and the "Environment Variables" button, then adding or selecting the PATH variable in the user variables with the value `C:\Program Files\apache-maven-3.3.9\bin`.

The same dialog can be used to set `JAVA_HOME` to the location of your JDK, e.g. `C:\Program Files\Java\jdk1.7.0_51`

Open a command window and run `mvn -v` to check if the Maven installation is successful.

## Configure Maven on Linux systems

Check environment variable value

```
echo $JAVA_HOME
/Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home
```

Adding to PATH

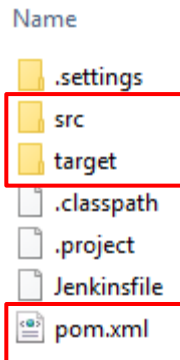
```
export PATH=/opt/apache-maven-3.3.9/bin:$PATH
```

## Using Maven as a build tool:

Maven follows certain folder structure while running build. The build configuration is done using `POM.xml` file.

# Automation factory

---



Using the POM.xml (project object model) file the developer / build masters perform the build action.

Maven follows the build life cycle which starts with validation → compile → test → package → integration test → verify → install → and Deploy.. Every Maven build follows this flow which sometimes can be time taking.

Maven Build lifecycle,

1. Validate
2. Compile
3. Test
4. Package
5. Integration-test
6. Verify
7. Install
8. Deploy

- **validate** - validate the project is correct and all necessary information is available
- **compile** - compile the source code of the project
- **test** - 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 on results of integration tests to ensure quality criteria are met

# Automation factory

- **install** - install the package into the local repository, for use as a dependency in other projects locally
- **deploy** - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

All dependencies in the software are resolved by Maven as mentioned in the POM.xml file.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Maven relies on the plug-ins to perform the build actions as mentioned in the Maven build lifecycle.

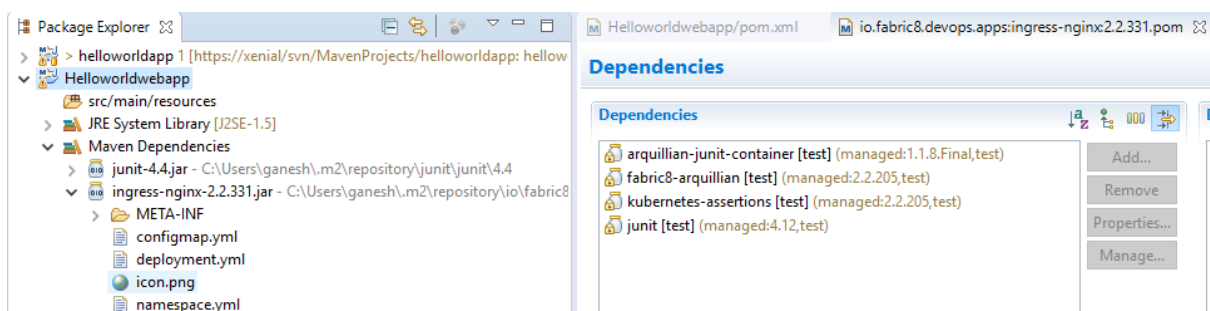
Maven connects to the <http://maven.apache.org/> and downloads all plug-ins to carry the build actions, like compile, package, validate, etc.

## Managing Maven Transitive Dependencies

If we add a dependency in the POM.xml file, Maven will go ahead download the dependency for us from the Maven Repository.

There can be some dependencies that the dependency itself needs to work satisfactorily. These are called as **transitive dependencies**.

Maven will identify such transitive dependencies and download the same for us as well.



Above image shows transitive dependencies for the 'ingress-nginx .jar'. 'nginx' is added as a dependency in the project.