# MAVEN BUILD TOOL

DevOps Training

# INTRODUCTION

1. Maven, a Yiddish word meaning accumulator of knowledge, was originally started as an attempt to simplify the build processes in the Jakarta Turbine project. a standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information and a way to share JARs across several projects.

# WHAT IS MAVEN?

# MAVEN AS A TOOL


Maven is a project management and comprehension tool. Maven provides developers a complete build lifecycle framework. 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 VS ANT

- Java projects originally managed with **Ant**
  - Manual build process
  - **Ant** scripts to compile, generate IDL, jar
  - Low-level scripting
  - No dependency management
  - No version management
- **Maven** provides high-level features for build process

# MAVEN VS ANT BASIC DIFF

| Ant | Maven |
| --- | --- |
| • Procedural<br>• Highly flexible, but complex<br>• Transitive task dependencies<br>• Each build script is very different<br>• Extensible via Ant tasks | • Declarative<br>• Convention over Configuration<br>• Well-defined build lifecycle<br>• Build scripts are standardized<br>• Extensible via plugins |

# WHAT MAVEN PROVIDES

Maven provides developers ways to manage following:

- Builds
- Documentation
- Reporting
- Dependencies

# WHAT MAVEN PROVIDES (CONTINUE)

Maven provides developers ways to manage following:

- SCMs
- Releases
- Distribution
- mailing list

# WHAT MAVEN PROVIDES SUMMARY

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of build related tasks.

# MAVEN'S OBJECTIVES

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with.

# MAKING THE BUILD PROCESS EASY

# PROVIDING A UNIFORM BUILD SYSTEM


**Providing a uniform build system**

Maven allows a project to build using its project object model (POM) and a set of plugins that are shared by all projects using Maven, providing a uniform build system. Once you familiarize yourself with how one Maven project builds you automatically know how all Maven projects build saving you immense amounts of time when trying to navigate many projects.

# PROVIDING QUALITY PROJECT INFORMATION


Maven provides plenty of useful project information that is in part taken from your POM and in part generated from your project's sources. For example, Maven can provide: ☐ Change log document created directly from source control ☐ Cross referenced sources ☐ Mailing lists ☐ Dependency list ☐ Unit test reports including coverage

# PROVIDING QUALITY PROJECT INFORMATION(CTN)

As Maven improves the information set provided will improve, all of which will be transparent to users of Maven. Other products can also provide Maven plugins to allow their set of project information alongside some of the standard information given by Maven, all still based on the POM.

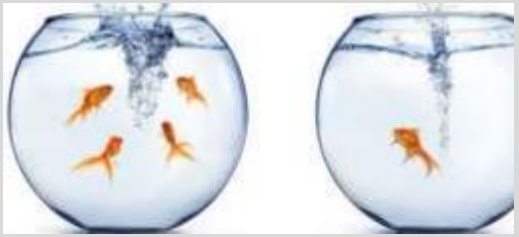
# PROVIDING GUIDELINES FOR BEST PRACTICES DEVELOPMENT

Maven aims to gather current principles for best practices development, and make it easy to guide a project in that direction. For example, specification, execution, and reporting of unit tests are part of the normal build cycle using Maven. Current unit testing best practices were used as guidelines: ☐ Keeping your test source code in a separate, but parallel source tree ☐ Using test case naming conventions to locate and execute tests ☐ Have test cases setup their environment and don't rely on customizing the build for test preparation.

# PROJECT WORKFLOW ASSISTANCE

Maven also aims to assist in project workflow such as release management and issue tracking. Maven also suggests some guidelines on how to layout your project's directory structure so that once you learn the layout you can easily navigate any other project that uses Maven and the same defaults.

# EASY MIGRATION AND TRANSPARENT



Maven provides an easy way for Maven clients to update their installations so that they can take advantage of any changes that been made to Maven itself. Installation of new or updated plugins from third parties or Maven itself has been made trivial for this reason.

# MAVEN CONVENTIONS

| 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 |

# SYSTEM REQUIRMENT

| System Requirement | |
|---|---|
| **JDK** | 1.7 or above. |
| **Memory** | No minimum requirement. |
| **Disk Space** | No minimum requirement. |
| **Operating System** | No minimum requirement. |

# SET JAVA PATH

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example:

| OS | Output |
|---|---|
| Windows | Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.7.0_60 |
| Linux | export JAVA_HOME=/usr/local/java-current |
| Mac | export JAVA_HOME=/Library/Java/Home |

Append Java compiler location to System Path.

| OS | Output |
|---|---|
| Windows | Append the string ";C:\Program Files\Java\jdk1.7.0.60\bin" to the end of the system variable, Path. |
| Linux | export PATH=$PATH:$JAVA_HOME/bin/ |
| Mac | not required |

# DOWNLOAD MAVEN ARCHIVE

Download Maven 2.2.1 from
http://maven.apache.org/download.html.

| | |
|---|---|
| Linux | apache-maven-3.3.1-bin.tar.gz |
| Mac | apache-maven-3.3.1-bin.tar.gz |

# EXTRACT MAVEN ARCHIVE

## Step 4: Extract the Maven Archive

Extract the archive, to the directory you wish to install Maven 3.3.1. The subdirectory apache-maven-3.3.1 will be created from the archive.

| OS | Location (can be different based on your installation) |
|---|---|
| Windows | C:\Program Files\Apache Software Foundation\apache-maven-3.3.1 |
| Linux | /usr/local/apache-maven |
| Mac | /usr/local/apache-maven |

# SET ENV MAVEN

## Step 5: Set Maven Environment Variables

Add M2_HOME, M2, MAVEN_OPTS to environment variables.

| OS | Output |
|---|---|
| Windows | Set the environment variables using system properties.<br>M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.3.1<br>M2=%M2_HOME%\bin<br>MAVEN_OPTS=-Xms256m -Xmx512m |
| Linux | Open command terminal and set environment variables.<br>export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.1<br>export M2=$M2_HOME/bin<br>export MAVEN_OPTS=-Xms256m -Xmx512m |
| Mac | Open command terminal and set environment variables.<br>export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.1<br>export M2=$M2_HOME/bin<br>export MAVEN_OPTS=-Xms256m -Xmx512m |

# ADD MAVEN BIN DIRECTORY LOCATION TO SYSTEM PATH

Now append M2 variable to System Path.

| OS | Output |
|---|---|
| Windows | Append the string ;%M2% to the end of the system variable, Path. |
| Linux | export PATH=$M2:$PATH |
| Mac | export PATH=$M2:$PATH |

# VERIFY INSTALL

Verify

.

## Step 7: Verify Maven Installation

Now open console and execute the following **mvn** command.

| OS | Task | Command |
|---|---|---|
| Windows | Open Command Console | c:\> mvn --version |
| Linux | Open Command Terminal | $ mvn --version |
| Mac | Open Terminal | machine:~ joseph$ mvn --version |

# POM EXAMPLE

POM

**POM Example**

```
<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.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>

</project>
```

# BUILD LIFE CYCLE

## What is Build Lifecycle?

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle.
As an example, a typical **Maven Build Lifecycle** consists of the following sequence of phases.

| Phase | Handles | Description |
|---|---|---|
| prepare-resources | resource copying | Resource copying can be customized in this phase. |
| validate | Validating the information | Validates if the project is correct and if all necessary information is available. |
| compile | compilation | Source code compilation is done in this phase. |
| Test | Testing | Tests the compiled source code suitable for testing framework. |
| package | packaging | This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml. |
| install | installation | This phase installs the package in local/remote maven repository. |
| Deploy | Deploying | Copies the final package to the remote repository. |

# WHAT IS A MAVEN GOAL

A **goal** represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases. A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.

The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below. The **clean** and **package** arguments are build phases while the **dependency:copy-dependencies** is a goal.

```
mvn clean dependency:copy-dependencies package
```

Here the *clean* phase will be executed first, followed by the **dependency:copy-dependencies goal**, and finally *package* phase will be executed.

# CLEAN LIFE CYCLE

## Clean Lifecycle

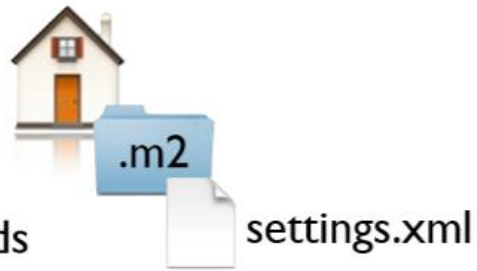
When we execute *mvn post-clean* command, Maven invokes the clean lifecycle consisting of the following phases.

- pre-clean
- clean
- post-clean

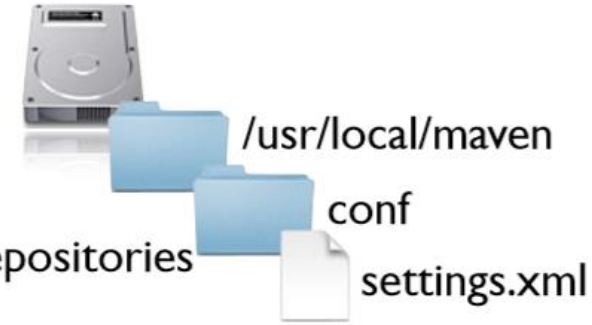
Maven clean goal (clean:clean) is bound to the *clean* phase in the clean lifecycle. Its **clean:cleangoal** deletes the output of a build by deleting the build directory. Thus, when *mvn clean* command executes, Maven deletes the build directory.

# MAVEN CONFIG DETAILS

- Maven configuration goes in the `settings.xml` file(s)
- There are two `settings.xml`
  - Local `settings.xml` file
    - `~/.m2/settings.xml`
    - Useful for private data such as usernames and passwords




  - Global `settings.xml` file
    - Located in a centrally installed copy of Maven
      - `${M2_HOME}/conf/settings.xml`
    - Useful for organizational settings such as enterprise repositories
    - All developers use this version of Maven



# MAVEN SETTINGS FILE

- A very simple `settings.xml` file:

```
<settings>
    <servers>
        <server>
            <id>acme-repo</id>
            <username>john</username>
            <password>t0ps3cr3t</password>
        </server>
    </servers>
</settings>
```

Specify authentication information to use when connection to some server

- A slightly more complicated `settings.xml` file:

```
<settings>
    <profiles>
        <profile>
            <id>myprofile</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <properties>
                <jdbc.user>john</jdbc.user>
                <jdbc.password>t0ps3cr3t</jdbc.password>
            </properties>
        </profile>
    </profiles>
</settings>
```

Define some properties in a build profile

# MAVEN SETTINGS FILE USAGE

The settings.xml files are useful for:

Security credentials for accessing servers

Configurations for mirror and proxy repositories

# MAVEN LOCAL REPO

- Maven stores downloaded libraries locally
  - By default, downloaded artifacts are placed in ~/.m2/repository
  - You can override this in the settings.xml file:

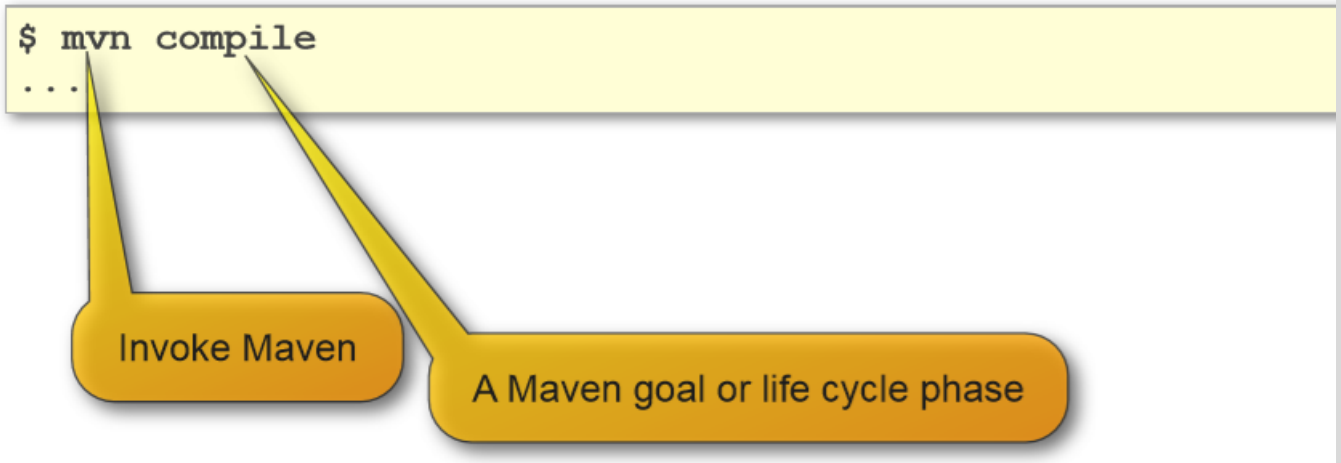
```
<settings>
    <localRepository>C:/repos</localRepository>
</settings>
```

Store the local repository in C:\repos


# MAVEN EXECUTION

## Executing Maven

- Running Maven from the command line



# MAVEN COMMON OPTIONS

-o (offline) Used to tell Maven not to go online for downloads -X (debug) Enable Maven's debug output for troubleshooting -D<property> -Dmaven.test.skip=true -e (errors) Output errors during the build process -P <profile> Activate a Build Profile

# SOURCES

Sources:

https://maven.apache.org

https://www.tutorialspoint.com/maven/maven_tutorial.pdf


# THANK YOU!

training@laksans.com