# Maven

## Automated build management.

# Manual build management (life before maven)

- Download the dependencies to compile the source code.
- Run the ***javac*** command to compile the source code.
- Run the ***java*** commands to test the source code.
- Run the ***jar*** command to package the source code.
- Run the ***javadoc*** command to create the documentation.

# Hey, but..

Anything, which is not the part of business logic, must be automated!

That’s correct! Maven is at our rescue.

# What is Maven?

- At its simplest, build automation tool
  - Produces deployable artifact (jar, war etc) from source code.
  - Helps us manage the dependencies.
  - Has built in **build lifecycle.**

- Can also be used as a project management tool
  - Can produce the project documentation
  - Supports plugins to provide various report.
  - Supports versioning and releasing the project.

# Maven in action!

Lets create a simple maven based project.

# Creating a simple java application

- ***mvn archetype:generate***
  - Creates an application in interactive mode; will prompt for
    - archetype - default 414 (maven-archetype-quickstart)
    - archetype version - default 1.1
    - groupId - com.hashedin.hu
    - artifactId - HelloWorldMaven
    - version - default 1.0-SNAPSHOT
    - package - default *groupId*

- ***mvn archetype:generate -DgroupId=com.hashedin.hu -DartifactId=HelloWorldMaven -DinteractiveMode=false -DarchetypeArtifactId=maven-archetype-quickstart***

# Directory Structure

```
rahul@rahul-2520:~/workspace/HelloWorldMaven$ tree
.
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- hashedin
    |               `-- hu
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- hashedin
                    `-- hu
                        `-- AppTest.java

11 directories, 3 files
```

- src/main/java
  - The source code directory.
  - com.hashedin.hu package created automatically by archetype.
- src/test/java
  - The test directory.
  - The test cases are also placed inside the same default package.
- pom.xml

# Contents of 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>com.hashedin.hu</groupId>
  <artifactId>HelloWorldMaven</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>HelloWorldMaven</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
~
```

# Goals - Automated build commands

- ***mvn compile***

  Compiles the source code, generates any files, copies resources to our target directory.
- ***mvn clean***

  Deletes the target directory and any generated resources.
- ***mvn package***

  Runs the compile command first, run test cases and packages the app based on its packaging type.
- ***mvn install***

  Runs the package command then installs it in local repo.

# The ‘target’ directory

- Automatically created by maven.
- This is where everything gets compiled to.
- This is also the place where test cases run from.
- Contents in this directory gets packaged into jar, war etc.

# The pom.xml

- Holds the meta information about the project.
- Project Information
  - groupId
  - artifactId
  - version
  - packaging
- Dependencies - Direct dependencies used in our application.
- Build - Plugins, Directory Structure etc.
- Repositories - Holds all the dependencies and artifacts.

# Creating a web application

mvn archetype:generate -DgroupId=com.hashedin
-DartifactId=hu-webapp
-DarchetypeArtifactId=maven-archetype-webapp

```
rahul@rahul-2520:~/workspace/hu-webapp$ tree
.
|-- pom.xml
`-- src
    `-- main
        |-- resources
        `-- webapp
            |-- index.jsp
            `-- WEB-INF
                `-- web.xml
```

- src/main/webapp
  - The base directory containing all the assets like JSP, Images etc.
- src/webapp/WEB-INF
  - web.xml - Deployment Desc.
- src/main/java
  - Standard java classes: Servlets, POJOs etc.
- pom.xml

# Dependencies

# Dependencies

- Resources that our project relies upon to function.
- To include, just list the dependency in pom.xml
  - Transitive dependencies will be pulled in by maven.
- Dependencies are identified by their coordinates
  - groupId
  - artifactId
  - version

Example:

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.3.5.Final</version>
</dependency>
```

# Transitive Dependencies

- Our project dependencies's dependencies.
  - Eg: Hibernate has dependency on 8 other resources:
    - hibernate-core
    - hibernate-commons-annotations
    - jboss-logging etc
- All the transitive dependencies are automatically downloaded and added by maven.
- The main reason people start off with maven.

# Dependencies Scope

- Not all resources are needed at all times!
- There are 6 available scopes:
  - compile - default scope, artifacts available everywhere.
  - provided - like compile, but not packaged into an artifact. It will be available where artifact is deployed. (servlets-api)
  - runtime - needed only for execution not for compiling.
  - test - available for test-compilation and execution phase.
  - system - provided, you specify the path to the jar on file system.
  - import - deals with dependencyManagement and multi modules.
- Lets add few dependencies to our simple application.

# Repositories

# Local Repository

- Where maven caches everything it downloads
  - /home/<username>/.m2/repository.
- When maven needs to resolve a dependency, it first looks into the local repository and then goes about downloading it.
- Stores the artifacts using the information we provide for artifactId, groupId and version.
  - ~/.m2/repository/com.hashedin/HelloWorldMaven/1.0-SNAPHOST/HelloWorldMaven-1.0-SNAPSHOT.jar
- No need to copy the artifact into every project and store it in the SCM (git etc)

# Remote Repository

- Just a network accessible location that maven downloads dependencies from.
- Default location: http://repo.maven.apache.org/maven2/
- We can add multiple repositories.
- All the artifacts that remote repository contains are open source. Dont add any private artifact of the organization.

# Adding a new remote repository

We can add a remote repository to search for dependencies if these dependencies aren't available on default location.

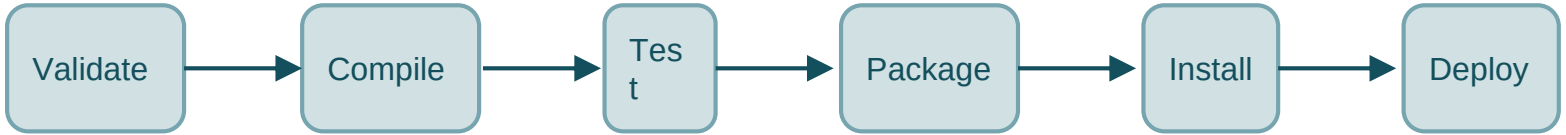
```
<repositories>
    <repository>
        <id>spring-snapshot</id>
        <name>Spring Maven SNAPSHOT Repository</name>
        <url>http://repo.springsource.org/libs-snapshot</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
        <releases>
            <enabled>false</enabled>
        </releases>
    </repository>
</repositories>
```

# Build lifecycle and plugins

# Build Lifecycle

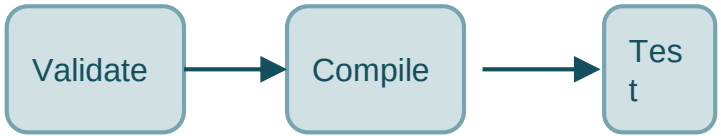
- Software projects undertake various steps before they are actually distributed. (eg. compile, test, package etc).
- Build lifecycle defines the process of building and distributing the artifacts.
- There are three built-in build lifecycles:
  - Default/Build - Handles project building and deployment.
  - Clean - Handles project cleaning
  - Site - Handles project's site generation.
- Each build lifecycle is made up of phases.

# Build / Default lifecycle




- You can invoke a specific phase on the lifecycle
  - Example: ***mvn test***

- When you invoke a specific phase, every previous phase run, including the one you specified. If we run, ***mvn test***



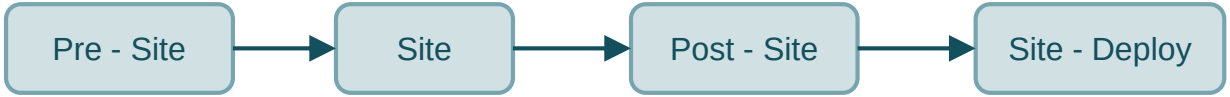

Refer: All lifecycle phases

# Clean and Site lifecycle phases

- Clean


  - Handles project cleaning.
  - Deletes target directory and any generated files.

- Site




  - Generates project documentation.
  - Used to generate various kind of reports.

# Maven Architecture

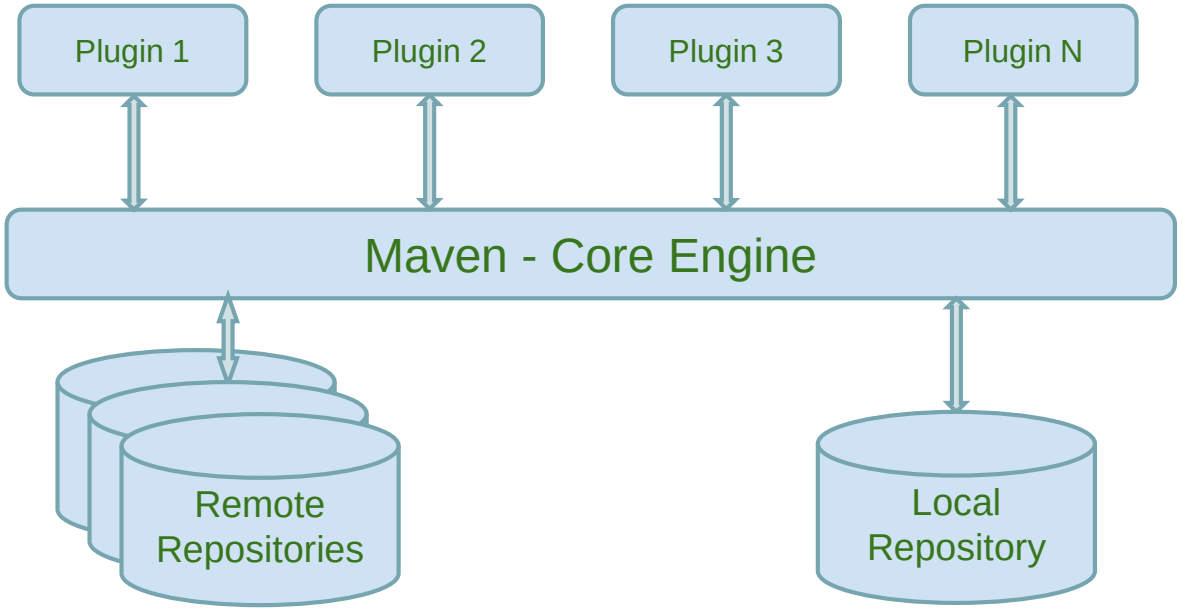



Fig: Maven's plugin based architecture.

# Maven Architecture - Core Engine

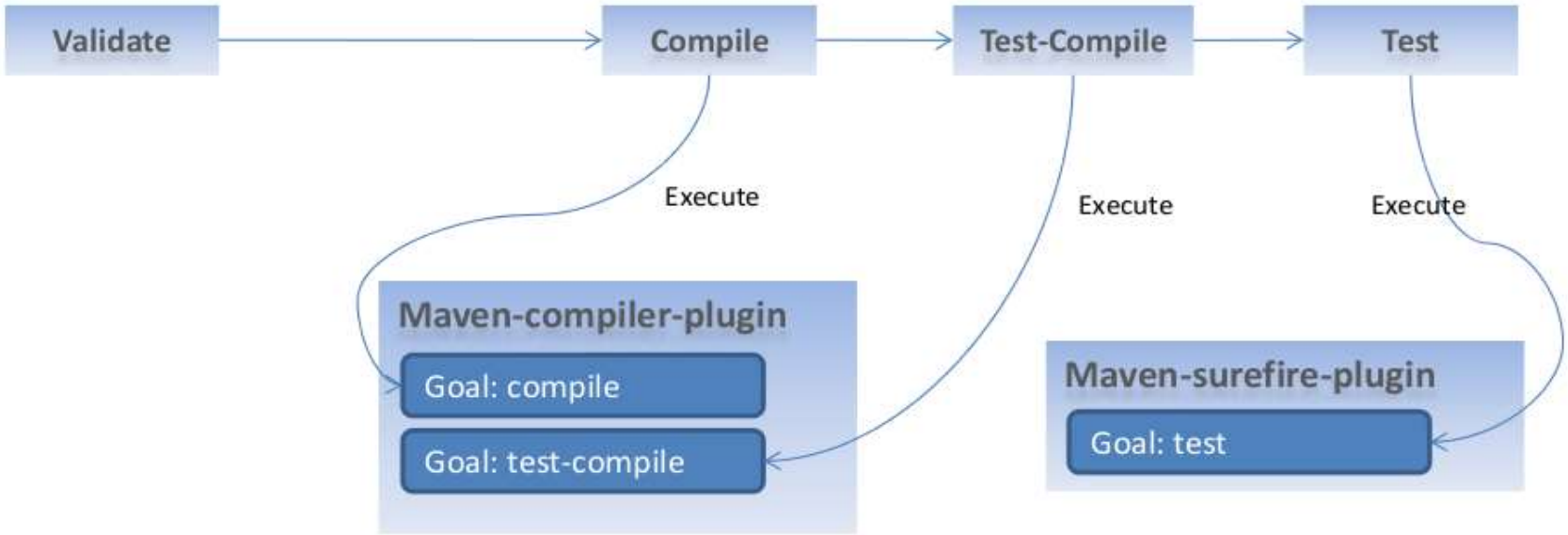
- Project processing
  - Read the project configuration file and configure the project accordingly.
- Build lifecycle management
  - Run a series of phases (as seen before).
- Framework for plug-ins

# Maven Architecture - Plugins

- Provides the core operations to build your project.
  - For example, to create a jar the maven jar plugin will do the job, not maven itself.
- Plug-ins provides one or more “Goals”
- A Goal perform some operation on the project.
  - Ex: compile, create a Jar, deploy to Jboss, etc.
- Goals can be bound to build lifecycle phases

# Maven lifecycle, plugins, and goals

- Goals can be bound to build-lifecycle phases
  - Eg: $ mvn test



# Maven lifecycle, plugins, and goals

We can bind a plugin's goal to a lifecycle phase

```
<plugin>
    <groupId>com.mycompany.example</groupId>
    <artifactId>some-maven-plugin</artifactId>
    <version>1.0</version>
    <executions>
        <execution>
            <phase>compile</phase>
            <goals>
                <goal>myGoal</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

# Maven lifecycle, plugins, and goals

- Putting lifecycle phases plug-ins and goals together
  - A lifecycle is a series of phases
  - A phase is made of goals
  - Goals are provided by plug ins

- Each phase have default bindings

# Jetty

## The web interface for Maven.

# Jetty - The Web Container

We can use the jetty plugin to run our web application

Plugin configuration:

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>jetty-maven-plugin</artifactId></plugin>
```

Usage:

Start: ***$mvn jetty:run***

Stop: <ctrl-c>, ***$mvn jetty:stop***

For jetty to function without any configuration, keep:

- **resources** in *${basedir}/src/main/webapp*
- **classes** in *${project.build.outputDirectory}*
- **web.xml** in *${basedir}/src/main/webapp/WEB-INF/*

# Summary - Benefits of maven

- Declarative way of defining the dependency.
- Transitive dependency management.
- Keeps the code organized.
- Various plugins available for almost any task we would need to undertake.
- Reduces build size by eliminating the need to check in jar files into source code repository.

# References

Apache Maven Guides

Maven by example

Apache Maven tutorial

# That’s It!

## Questions, Suggestions Or Feedback?