

Formation Maven 2

IUT Informatique



BONJOUR !

Je suis Laurent Michenaud

Je suis architecte solutions mobiles.

18 ans d'expériences.

lmichenaud@gmail.com



**Download
PDF**





PLAN

1. Introduction à Maven
2. Les plugins
3. Les dépendances
4. Les dépôts
5. Les modules

1

INTRODUCTION

Premier pas avec Maven.



Maven

- Projet de la fondation Apache Software.
- Open Source (Licence : Apache License 2.0).
- Outil multi plateformes : Windows, Linux, OSX.
- En ligne de commande ou intégré dans les IDE de développement.
- <https://maven.apache.org/>

Industrialisation des projets Java

Permet de gérer, d'automatiser et de standardiser la production d'un projet Java.





A partir des sources...

- Compiler le projet.
- Exécuter les tests unitaires.
- Contrôler la qualité du code source.
- Produire les livrables (jar, zip).
- Déployer les livrables sur des dépôts de packages.
- Générer la documentation du projet.
- ...



Avant Maven

- make
 - ▷ tâches décrites dans un fichier Makefile.
- ant
 - ▷ Pour les projets Java
 - ▷ tâches décrites dans un fichier xml

Le descripteur de projet : pom.xml

- Situé à la racine du projet.
- POM signifie : **P**rojet **O**bject **M**odel
- Décrit le projet :
 - ▷ Informations générales (nom, organisation, ...)
 - ▷ Ses dépendances.
 - ▷ Les instructions de construction.

Attributs du pom.xml

- **<groupId>** : L'identifiant de groupe (nom de package, ex: com.mycompany)
- **<artifactId>** : L'identifiant de projet
- **<version>** : version du projet.
- **<packaging>** : Le type de package produit (jar, war, ear, ...)
- **<name>** : Nom du projet.
- **<description>** : description longue.
- **<author>** : auteur du projet.
- **<licences>** : licence(s) du projet.
- **<scm>** : url de gestion de sources (GIT, SVN).
- **<dependencies>** : les dépendances.
- **<build>** : configuration du build

pom.xml minimal

```
<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.mycompany</groupId>
  <artifactId>myproject</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>My project</name>

</project>
```

Les dépôts Maven 2

- Contiennent des paquets.
- Un projet Maven rapatrie ses dépendances à partir des dépôts Maven 2.
- Un dépôt peut être local ou distant (<http://>)
- Un projet dépose ses paquets sur les dépôts Maven.

Au final

- Les build des différents projets sont standardisés.
- Les livrables (dont des librairies) sont centralisés et partagés sur des dépôts.
- Les dépendances entre projets sont simples à gérer.
- La qualité des build est sécurisée via des contrôles.

Structure par défaut d'un projet Maven

- pom.xml (*descripteur maven*)
- src
 - ▷ main
 - ▷ java (*classes Java*)
 - ▷ resources (*autres fichiers comme les fichiers .properties*)
 - ▷ test
 - ▷ java (*classes de tests*)
 - ▷ resources (*ressources de tests*)
- target (*répertoire de sortie, à ne pas commiter*)
 - ▷ classes (*contient les classes Java compilées, .class*)

Installation

- Pré-requis : JDK 1.7
- Définir la variable d'environnement JAVA_HOME (conseillé).
- Installation en fonction de l'OS : zip, exe, ...
- Définir la variable d'environnement M2_HOME (conseillé) : doit pointer vers le répertoire d'installation Maven.
- Ajouter à la variable d'environnement PATH :
 - ▷ \$M2_HOME/bin (Linux, OSX)
 - ▷ %M2_HOME%\bin (windows)

Configuration du proxy pour Maven

■ Dans le fichier ~/.m2/settings.xml

```
■ <settings>
■   <proxies>
■     <proxy>
■       <id>iut-proxy</id>
■       <protocol>http</protocol>
■       <host>proxy-etu</host>
■       <port>3128</port>
■     </proxy>
■   </proxies>
■ </settings>
```

Création d'un projet Maven 2

1. Avec la commande :
 - a. `mvn archetype:generate`
2. Affiche la liste des archetypes disponibles
 - a. archetype = template de projet.
3. Par défaut, utilise l'archetype : maven-archetype-quick-start
4. Saisir le group id, artifact id, la version et le package.

👍 Possibilité de créer ses propres archetypes

Import des projets dans les IDE

- Eclipse
 - ▷ Plugin Maven intégré par défaut dans les versions Eclipse Java Developer et JEE Developer
 - ▷ Via la fonction : “Import Maven project...”
- IntelliJ
 - ▷ Intégré de base dans la version community.

Exercice 1 : création du projet

- Installer Maven.
- Ouvrir une fenêtre shell et lancer la commande :
 - ▷ `mvn --version`
- Créer un projet Java avec les informations suivantes :
 - ▷ group id : `org.iut.nantes`
 - ▷ artifact id : `myproject`
 - ▷ package : `org.iut.nantes`
 - ▷ version : `1.0.0-SNAPSHOT`
- Importer le projet sous Eclipse.

Exercice 2 : compilation du projet

- Compléter le pom.xml avec :
 - ▷ la description, la licence, l'organisation, nom des développeurs.
- Ouvrir une console shell et se placer à l'intérieur du projet.
- Lancer la compilation avec la commande :
 - ▷ mvn compile
- Réaliser la même opération sous Eclipse ou IntelliJ
 - ▷ Eclipse: Bouton droit sur le projet, Run as, maven build...

Exercice 3 : implémentation d'une fonction métier

- Créer une classe Calculette dans le répertoire `src/main/java/org/iut/nantes` avec une méthode réalisant une addition de 2 nombres passés en paramètre.
- Dans la classe `App.java`, instancier votre calculette et faites lui réaliser une addition quelconque.

Exercice 4 : Ecriture d'un test unitaire

- Modifier le pom.xml et utiliser la version 4.12 de la dépendance Junit.
- Sous Eclipse, Bouton Droit sur le projet -> Maven ->Update Project pour prendre en compte la modification.
- Créer la classe de test CalculetteTest.java dans le répertoire src/test/java/org/iut/nantes
- Ajouter une méthode testAddition() pour tester la méthode addition() de la classe Calculette.

Exercice 5 : exécution des tests

- Exécuter les tests :
 - ▷ En ligne de commande : `mvn test`
 - ▷ Eclipse : run as, maven test
 - ▷ IntelliJ: panel “Maven Project” à droite de l'écran

Les plugins Maven

Les plugins

- Toutes les tâches exécutées sont fournies par des plugins :
 - ▷ mvn compile -> plugin maven-compile-plugin
 - ▷ mvn test -> plugin maven-surefire-plugin
 - ▷ mvn clean -> plugin maven-clean-plugin
- Les plugins sont téléchargés au moment de la première utilisation.
- Un plugin contient un ou plusieurs goals (tâches).
- Les plugins sont configurables dans le pom.xml.

 **Possibilité de créer ses propres plugins**

Core Plugins

- clean : nettoie le projet de tous les fichiers générés.
- compile : pour compiler un projet.
- resources : recopie les ressources nécessaires au projet (fichiers autre que .java)
- site : génère la documentation du projet.
- surefire : exécute les tests unitaires.

Packaging plugins

- jar : créé le jar à partir des classes compilées et des fichiers ressources.
- war : crée le war avec les classes compilées, les ressources web, le descripteur web.xml

Tools plugins

- assembly : création d'un package de distribution.
- dependency : gestion et analyse des dépendances du projet
- archetype : création un projet à partir de templates de projets.
- release : création de la release d'un projet (contrôles, build, création du tag, dépôt de la release).

Reporting plugins

- javadoc : génération de la javadoc du projet.
- checkstyle : contrôle de qualité du code source.
- findbugs : détection d'anomalies dans le code source.
- surefire-reports : génération du rapport de tests.

Exercice 1 : Compilation du projet en Java 8

■ Modifier le pom.xml ainsi :

```
■ <build>
■   <plugins>
■     <plugin>
■       <groupId>org.apache.maven.plugins </groupId>
■       <artifactId>maven-compiler-plugin </artifactId>
■       <version>3.7.0</version>
■       <configuration>
■         <source>1.8</source>
■         <target>1.8</target>
■       </configuration>
■     </plugin>
■   </plugins>
■ </build>
```

Exercice 2 : Encodage des sources

- Des développeurs travaillant sur des environnements différents (Windows, Linux, ...) peuvent se retrouver avec des caractères erronés dans le code source si le projet ne spécifie pas l'encodage des sources.

■

- `<properties>`

-

```
<project.build.sourceEncoding> UTF-8</project.build.sourceEncoding>
```

- `</properties>`

■

Exercice 3 : Qualité du code source

- Contrôler la qualité du code source en exécutant le plugin checkstyle :
 - ▷ `mvn checkstyle:check`
- Réaliser quelques corrections des anomalies remontées.

Exercice 4 : Génération de la javadoc

- Exécuter la commande
 - ▷ `mvn javadoc:javadoc`
- La javadoc est générée dans le répertoire `target/site/apidocs`

Exercice 5 : Génération de la documentation du projet

- Exécuter la commande :
 - ▷ *mvn site*
- La documentation est générée dans le répertoire target/site
- Pour la compatibilité avec Java 10 :
- `<plugins> <plugin>`
- `<artifactId>maven-site-plugin</artifactId>`
- `<version>3.7.1</version>`
- `</plugin> </plugins>`

Exercice 6 : Ajout de rapports dans la documentation projet

Ajout du rapport Javadoc dans la documentation projet :

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.0.0-M1</version>
    </plugin>
  </plugins>
</reporting>
```

Exercice 7 : Ajout du rapport checkstyle dans le documentation projet

```
<reporting>
  <plugins><plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.17</version>
    <reportSets>
      <reportSet><reports><report>checkstyle</report></reports></reportSet>
    </reportSets>
  </plugin></plugins>
</reporting>
```

Exercice 8 : Ajout du rapport des tests unitaire dans le documentation projet

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>2.20.1</version>
    </plugin>
  </plugins>
</reporting>
```

Les dépendances

Les dépendances

- Un projet Maven peut utiliser d'autres projets Maven via le système de dépendances.
- Les dépendances sont référencées via leur groupId, artifactId et numéro de version.
- Les dépendances sont transitives:
 - ▷ Si A utilise B et B utilise C
 - ▷ Alors les dépendances de A sont B & C.

Scope des dépendances

- Les dépendances ont un scope. Ces scopes sont :
 - ▷ **compile** (par défaut) : signifie que la dépendance est nécessaire à la compilation du projet.
 - ▷ **test** : signifie que la dépendance est nécessaire pour la compilation et l'exécution des tests unitaires.
 - ▷ **provided** : signifie que la dépendance est fourni par le conteneur d'exécution du projet (exemple: servlet.jar). Elle sert uniquement pour la compilation mais il est inutile de l'embarquer dans le packaging.
 - ▷ **runtime** : signifie que la dépendance est utile uniquement à l'exécution du projet et non pour la compilation.

Déclaration du scope

Exemple de déclaration du scope d'une dépendance :

```
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit</artifactId>  
  <version>4.12</version>  
  <scope>test</scope>  
</dependency>
```

Exclusion des dépendances

- Il est possible d'exclure une dépendance transitive :

- `<dependency>`
- `<groupId>junit</groupId>`
- `<artifactId>junit</artifactId>`
- `<version>4.12</version>`
- `<exclusions>`
- `<exclusion>`
- `<groupId>org.hamcrest</groupId>`
- `<artifactId>hamcrest-core</artifactId>`
- `</exclusion>`
- `</exclusions>`
- `</dependency>`

Exercice 1

- Importer la librairie :
 - ▷ `<groupId>org.springframework.ws</groupId>`
 - ▷ `<artifactId>spring-ws-core</artifactId>`
 - ▷ `<version>2.4.0.RELEASE</version>`
- Afficher l'arbre de dépendances et comprendre pourquoi la librairie aopalliance est incluse dans le projet.
 - ▷ `mvn dependency:tree`
 - ▷ Sous Eclipse, ouvrir le pom.xml puis l'onglet Dependency Hierarchy.
- Exclure cette dépendance transitive

Les dépôts Maven (repository)

Les dépôts

- Contiennent des paquets (dépendances, plugins, ...) nécessaires aux projets Maven.
- Pour un projet, Maven télécharge les dépendances à partir du dépôt.
- Maven les stocke dans un dépôt local pour ne pas avoir à les télécharger à chaque fois (~/.m2/repository). La première exécution est souvent longue.
- Maven utilise par défaut un dépôt central.
- Il est possible de créer des dépôts privés.

Les dépôts

- Un dépôt peut contenir :
 - ▷ Des dépendances en version finale (x.y.z).
 - ▷ Des dépendances en cours de développement (x.y.z-SNAPSHOT)
 - ▷ Des plugins en version finale (x.y.z).
 - ▷ Des plugins en cours de développement (x.y.z-SNAPSHOT)

Déclaration d'un dépôt

- Les dépôts sont déclarés :
 - ▷ Soit globalement à tous les projets dans le fichier `~/.m2/settings.xml`
 - ▷ Soit dans le `pom.xml` du projet
- Pour télécharger une dépendance, Maven cherche dans tous les dépôts configurés jusqu'à la trouver.
- Pour les dépendances SNAPSHOT, Maven vérifie aussi systématiquement si une nouvelle version est disponible.

Déclaration d'un dépôt

- Déclaration d'un dépôt pour les dépendances finales du projet :
- `<repositories>`
- `<repository>`
- `<releases><enabled>true</enabled></releases>`
- `<snapshots><enabled>>false</enabled></snapshots>`
- `<id>bintray-maven</id>`
- `<name>bintray</name>`
- `<url>https://dl.bintray.com/jaycroaker/maven</url>`
- `</repository>`
- `</repositories>`

Déclaration d'un dépôt

- Déclaration d'un dépôt pour les dépendances SNAPSHOT du projet :
- `<repositories>`
- `<repository>`
- `<releases><enabled>false</enabled></releases>`
- `<snapshots><enabled>true</enabled></snapshots>`
- `<id>bintray-maven</id>`
- `<name>bintray</name>`
- `<url>https://dl.bintray.com/jaycroaker/maven</url>`
- `</repository>`
- `</repositories>`

Déclaration d'un dépôt

- Déclaration d'un dépôt pour les plugins du projet :
- `<pluginRepositories>`
- `<pluginRepository>`
- `<releases><enabled>true</enabled></releases>`
- `<snapshots><enabled>>false</enabled></snapshots>`
- `<id>bintray-maven</id>`
- `<name>bintray</name>`
- `<url>https://dl.bintray.com/jaycroaker/maven</url>`
- `</pluginRepository>`
- `</pluginRepositories>`

Déclaration d'un dépôt

- Déclaration d'un dépôt pour les plugins SNAPSHOT du projet :
- `<pluginRepositories>`
- `<pluginRepository>`
- `<releases><enabled>false</enabled></releases>`
- `<snapshots><enabled>true</enabled></snapshots>`
- `<id>bintray-maven</id>`
- `<name>bintray</name>`
- `<url>https://dl.bintray.com/jaycroaker/maven</url>`
- `</pluginRepository>`
- `</pluginRepositories>`

Publication d'un paquet

- La publication d'un paquet est réalisé via le Maven Deploy Plugin.
 - ▷ *mvn deploy*
- 3 éléments peuvent être déployés :
 - ▷ Le paquet en version finale.
 - ▷ Le paquet en version développement (-SNAPSHOT).
 - ▷ Le site.

Publication d'un paquet

- L'élément `<distributionManagement>` doit être renseigné au préalable.
 - ▽ `<distributionManagement>`
 - ▽ `<repository>`
 - ▽ `<id>myrepository</id>`
 - ▽ `<url>file:///Users/lmichenaud/Desktop/IUT/maven/myrepo-releases</url>`
 - ▽ `</repository>`
 - ▽ `<snapshotRepository>`
 - ▽ `<id>mysnapshotrepository</id>`
 - ▽ `<url>file:///Users/lmichenaud/Desktop/IUT/maven/myrepo-snapshots</url>`
 - ▽ `</snapshotRepository>`
 - ▽ `</distributionManagement>`
- L'url peut être un répertoire local, un ftp, un lien SCP, HTTP, ...

Les gestionnaires de dépôts

- Des produits permettent de gérer simplement ses dépôts Maven :
 - ▷ JFrog Artifactory
 - ▷ Nexus
- Quelques fonctionnalités : création/modification/suppression des dépôts, gestion de la sécurité, gestion des répliquions entre dépôts,

Exercice 1

- Configurer le projet Maven pour qu'il déploie ses versions de développement dans un dépôt local.

Exercice 2

- Créer un deuxième projet avec Maven.
- Ajouter le dépôt de l'exercice 1 à ce 2eme projet.
- Ajouter la dépendance vers le projet myproject.
 - ▷ La dépendance doit être résolue avec succès.

Les modules

Les modules

- Un projet Maven 2 peut être découpé en modules.
- Intérêts :
 - ▷ Facilite la réutilisabilité du code.
 - ▷ Améliore la maintenabilité du code.
 - ▷ Permet d'éviter des dépendances illégales entre classes.

Structuration d'un projet multi-modules

- Chaque module est contenu dans un sous-répertoire.
- Un module a la même structure qu'un projet Maven 2.
- Le pom.xml racine a un packaging de type "pom".
- Les pom.xml des modules font référence au module parent.
- Le projet et ses modules partagent la même base pour le groupId et le même numéro de version.
- Un module peut avoir des dépendances vers les autres modules.

Exemple de structure d'un projet multi modules

- Exemple :
- ./pom.xml
 - ▷ ./module1
 - ▷ pom.xml
 - ▷ src
 - ▷ ./module2
 - ▷ pom.xml
 - ▷ src

Exemple de pom.xml racine

- <project
- <groupId>org.nantes-iut</groupId>
- <artifactId>myproject</artifactId>
- <version>1.0-SNAPSHOT</version>
- <packaging>pom</packaging>
- <modules>
- <module>mylib</module>
- <module>mywebapp</module>
- </modules>

Exemple de pom.xml d'un module

- <project>
- <parent>
- <groupId>org.nantes-iut</groupId>
- <artifactId>myproject</artifactId>
- <version>1.0-SNAPSHOT</version>
- </parent>
-
- <artifactId>mylib</artifactId>
- <packaging>jar</packaging>

Exercice 1

- Transformer votre projet Maven en projet multi-modules avec :
 - ▷ 1 module calcul contenant la classe Calculette.
 - ▷ 1 module app contenant la classe App.

Exercices complémentaires

Exercice 1

- Commiter le projet sous GitHub.
 - ▷ Attention à ne pas commiter les éléments générés :
 - ▷ target, .classpath, .project ...
- Demander à votre voisin de récupérer votre projet et de le compiler.
- Mettre en place le plugin release sur le projet.

Exercice 2

- Utiliser le plugin Assembly pour générer un zip de votre projet contenant le jar, les sources et la javadoc.