

MAVEN



P.Mathieu

IUT de Lille

<http://www.iut-a.univ-lille.fr>

prenom.nom@univ-lille.fr

MAVEN

- ▶ Outil de gestion et d'automatisation de production de projets Java en général et Java EE en particulier.
- ▶ Fourni par l'Apache Software Foundation.
- ▶ Objectifs
 - ▶ Fournir une arborescence standardisée
 - ▶ Assurer les tâches de fabrication
 - ▶ Gérer automatiquement les dépendances nécessaires
- ▶ Version actuelle 3.9.9
- ▶ Dans la lignée de Make, Ant, Gradle (pour Java) npm (pour Javascript), Composer (pour php), pip/venv (pour Python)

- ▶ **Installation** : <http://maven.apache.org/>
- ▶ **Java** : Maven utilise la variable `JAVA_HOME` pour déterminer le JRE à utiliser pour son usage (comme tomcat)
- ▶ **Test** : `mvn -v`
- ▶ **Aide en ligne** : dans <http://maven.apache.org/> aller sur `use` puis `user center` puis “Maven in 5 minutes” et “getting started”

Création d'un projet

```
mvn archetype:generate
```

4 informations importantes :

- ▶ l'archétype (le prototype) à utiliser
défaut : `maven-archetype-quickstart`
pour le web : `maven-archetype-webapp`
- ▶ le `groupId` (le nom de reference externe) : `fr.but3`
- ▶ l'`artifactId` (le nom du projet) : `tp503`
- ▶ le package principal : (par défaut, le `groupId`)

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart -DgroupId=fr.but3  
-DartifactId=tp503 -DinteractiveMode=false
```

Arborescence générée

L'arborescence obtenue dépend de l'archétype choisi

tp503

```
|-- pom.xml
|-- src
    |-- main
        |-- java
            |-- fr
                |-- but3
                    |-- App.java
    |-- test
        |-- java
            |-- fr
                |-- but3
                    |-- AppTest.java
```

Le POM : Project object Model

Chaque projet est configuré dans un fichier `pom.xml` placé à la racine du projet. `pom.xml` fournit des informations sur la version, la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.but3</groupId>
  <artifactId>tp503a</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>tp503a</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Plusieurs buts sont disponibles

- ▶ compile
- ▶ test
- ▶ package
- ▶ install - pour ajouter le nouvel archétype localement
- ▶ deploy - pour déployer l'application sur un serveur

D'autres buts sont exécutables en dehors du cycle de vie : `clean`, `site`, ...

Quel que soit le but, tous les buts en amont seront exécutés


```
mvn archetype:generate
    -DarchetypeArtifactId=maven-archetype-quickstart
    -DgroupId=fr.but3 -DartifactId=tp503 -DinteractiveMode=false
```

```
cd tp503
```

```
mvn compile
```

```
mvn test
```

```
mvn package
```

```
java -cp target/classes/:..... fr.but3.App
```

```
java -cp target/tp503-1.0-SNAPSHOT.jar fr.but3.App
```

```
mvn exec:java -Dexec.mainClass=fr.but3.App
```

```
mvn clean
```

Trois manières d'exécuter un projet

- ▶ `mvn exec:java -Dexec.mainClass=fr.but3.App`
Maven s'occupe du classpath en pointant vers son repository local (`~/.m2/repository` par défaut).
- ▶ `java -cp target/classes/:.... fr.but3.App`
On exécute un `.class` Il faut gérer sois-même son classpath, et donc avoir ses dépendances quelque part (voir le `maven-dependency-plugin`)
- ▶ `java -jar target/tp503.jar fr.but3.App`
On exécute l'archive `jar`. Les dépendances doivent être indiquées dans le `MANIFEST.MF` du jar (voir le `maven-jar-plugin`)

Remarque sur l'option -D

Toutes les options peuvent être passées ...

- ▶ soit à la commande, avec l'option `-D`

```
mvn exec:java -Dexec.mainClass=fr.but3.App
```

- ▶ soit dans l'entrée `properties` du pom

```
<exec.mainClass>${project.groupId}.App</exec.mainClass>
```

La propriété précédente permet donc de lancer l'exécution avec simplement


```
mvn exec:java
```

Référentiels de dépendances et plugins

Maven calcule et télécharge automatiquement les dépendances dans les référentiels déclarés (dépôts distants) et les place dans un répertoire local

- ▶ Local : `~/ .m2` (par défaut)
- ▶ Distant : `https://repo1.maven.org/maven2` (par défaut)
- ▶ Moteur de recherche : `https://www.mvnrepository.com/`

Il est possible d'ajouter d'autres “dépôts” via l'entrée `repositories` du pom.



Repository

- Central 2.3k
- Sonatype 1.3k
- Spring Plugins 647
- Spring Lib M 587
- JCenter 171
- Clojars 165
- Spring Lib Release 138
- IBiblio 102


Group

- com.github 296
- org.apache 192
- org.onosproject 66
- org.eclipse 59
- com.google 43
- org.glassfish 43
- org.mobicens 43
- org.springframework 39

Category

Found 3206 results

Sort: **relevance** | popular | newest



- Spring JDBC** 2,503 usages
[org.springframework](#) » [spring-jdbc](#) Apache
Spring JDBC
Last Release on Oct 29, 2018
- MongoDB Java Driver** 911 usages
[org.mongodb](#) » [mongo-java-driver](#) Apache
The MongoDB Java Driver uber-artifact, containing the legacy driver, the mongodb-driver, mongodb-driver-core, and bson
Last Release on Nov 7, 2018
- SQLite JDBC** 472 usages
[org.xerial](#) » [sqlite-jdbc](#) Apache
SQLite JDBC library
Last Release on Oct 1, 2018

Déclaration des dépendances

Un exemple ...

```
<dependencies>
  ...
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.4</version>
  </dependency>
  ...
</dependencies>
```

Dépendances les plus courantes : Log4J, JUnit, Apache commons, servlet-api, ...

Limiter la portée des dépendances

Le tag `scope` permet de limiter la portée des dépendances à ce qui est nécessaire

- ▶ `scope: compile` (valeur par défaut)
Nécessaire au compile et au runtime
Maven place ce jar dans la librairie `lib` du package et dans le `war`.
Typiquement c'est le cas des jars classiques
- ▶ `scope: provided`
Nécessaire au compile, mais pas au runtime
Typiquement c'est `servlet-api.jar` qui est fourni par le serveur web qui exécutera l'appli
- ▶ `test`
ne sert qu'à la compilation et exécution des tests

Export des dépendances

Pour un pom donné ...

- ▶ Lister les dépendances : `mvn dependency:list`
- ▶ Dépendances utilisées au runtime : `mvn dependency:tree -Dscope=runtime`
- ▶ Vérifier les versions :
`mvn versions:display-dependency-updates`
`mvn versions:display-plugin-updates`
- ▶ Exporter les dépendances :
`mvn dependency:copy-dependencies`
 - ▶ Par défaut exportées dans `/target/dependency`
 - ▶ Pour exporter ailleurs `-DoutputDirectory="/tomcat/lib"`

- ▶ Les ressources (données textuelles) sont à placer dans `src/main/resources`
- ▶ Maven les recopie automatiquement dans `target/classes`
- ▶ On accède impérativement aux ressources via le `ClassLoader`

```
InputStream is = App.class.getClassLoader().getResourceAsStream("fich.txt");  
BufferedReader br = new BufferedReader(new InputStreamReader(is));
```

Une appli web 1/3 : génération

Utiliser maven-archetype-webapp

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp -DgroupId=fr.but3  
-DartifactId=tp503 -DinteractiveMode=false
```

tp503

```
|-- pom.xml  
|-- src  
|   |-- main  
|       |-- resources  
|       |-- webapp  
|           |-- WEB-INF  
|               |-- web.xml  
|               |-- index.jsp
```

Une appli web 2/3 : compléments

- ▶ Créer `java/fr/but3` sous `main`
- ▶ Ajouter dans le `pom` la dépendance `servlet-api.jar` pour compiler les servlets
(`servlet-api.jar` pour `javax.*` ou `jakarta-servlet-api` pour `jakarta.*`)
- ▶ Vérifiez que le `web.xml` utilise bien l'api servlet 4.0 (voir `conf/web.xml` pour un exemple)
- ▶ `mvn compile` compile simplement les classes et les range dans `target/classes`
- ▶ `mvn package` crée une structure de webapp sous le rep `target` ainsi qu'un `war`

Une appli Web 3/3 : Arborescence

```
MonProj
|-- pom.xml
|-- src
|   |-- main
|       |-- java
|           |-- fr
|               |-- but3
|                   |-- MaServlet.java
|-- resources
|-- webapp
|   |-- WEB-INF
|       |-- web.xml
|       |-- index.jsp
|       |-- login.html
|       |-- style.css
```

```
target
|-- MonProj
|   |-- META-INF
|   |-- WEB-INF
|       |-- classes
|       |-- MaServlet.class
|       |-- lib
|           |-- postgresql-42.7.4.jar
|       |-- web.xml
|       |-- index.jsp
|       |-- mapage.html
|       |-- style.css
|-- Monproj-1.0-SNAPSHOT.jar
```

Les plugins permettent l'ajout de nouvelles fonctionnalités à Maven.

- ▶ `maven-dependency-plugin` (installé par défaut) (Avec but `"dependency:copy-dependencies"` pour exporter les dépendances du projet)
- ▶ `maven-jar-plugin` (pour définir un jar exécutable)
- ▶ `maven-war-plugin`
- ▶ `maven-assembly-plugin` (permet de mettre des jars dans des jars avec le but `"assembly-single"`)
- ▶ ... il en existe des dizaines ... SonarQube, checkstyle

Chaque plugin vient avec de nouveaux buts : `mvn plugin:but`

Plugins par défaut

- ▶ Maven n'est en fait qu'un manager de plugins ! Certains plugins fournis par **Apache** (`maven-compiler`, `maven-dependency`, `maven-jar`, `maven-clean`, ...), et plein d'autres.
- ▶ Chaque plugin possède ses propres options
- ▶ Si les options par défaut conviennent : rien à déclarer !
On lance via `mvn groupId:artifactId:but` et Maven se débrouille (Quand l'`artifactId` du plugin respecte la norme de nommage `maven-xxx-plugin` on peut omettre `-maven-plugin`.)
- ▶ Si on souhaite changer ces options, il faut alors "surcharger" le plugin dans le `pom`
- ▶ Les deux archétypes précédemment cités, ne surchargent par défaut aucun plugin, les poms sont donc "vides"

Pour voir le pom "effectif" avec l'ensemble des plugins hérités

```
mvn help:effective-pom
```

Le plugin “jar executable”

Surcharge du maven-jar-plugin

```
<build>
  <plugins>
    .....
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.4.2</version>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <classpathPrefix>dependency/</classpathPrefix>
            <mainClass>fr.but3.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
    .....
  </plugins>
</build>
```

Le plugin “serveur web”

Parmi les plugins possibles d'un projet Maven, il y a le serveur web lui-même. On peut donc au choix :

- ▶ Déployer le war obtenu dans un tomcat externe
- ▶ Créer un lien symbolique sur le répertoire `target/nomproj` (structure de webapp)
- ▶ Déclarer `<Context docBase="/chemin/vers/target/nomproj" reloadable="true" />` dans le `context.xml` de tomcat
- ▶ Intégrer un serveur dans Maven lui-même
 - ▶ Cargo de chez **CodeHaus** (enveloppe multi-serveurs)
 - ▶ tomcat, jetty, glassfish, Resin

Le plugin cargo

- ▶ Cargo est un wrapper léger qui permet de manipuler différents types de serveurs de manière standard
- ▶ On peut le faire tourner avec de nombreux serveurs JEE (jetty, tomcat, glassfish, ...)

```
<build>
  <plugins>
    .....
    <plugin>
      <groupId>org.codehaus.cargo</groupId>
      <artifactId>cargo-maven3-plugin</artifactId>
      <version>1.10.14</version>
    </plugin>
    .....
  </plugins>
</build>
```

Lancement :

```
mvn org.codehaus.cargo:cargo-maven3-plugin:run ou
cargo:run
```

Le but deploy

Maven permet de déployer automatiquement une application sur un serveur web distant

- 1 Activer la page d'administration (déploiement "à chaud") en donnant login/mdp
- 2 Dans `.m2/settings.xml` indiquer le login/mdp d'admin du serveur.

```
<servers>
  <server>
    <id>TomcatServer</id>
    <username>admin</username>
    <password>password</password>
  </server>
</servers>
```
- 3 Dans le `pom.xml` utiliser le plugin `tomcat7-maven-plugin` afin d'y mettre la référence au serveur (tags `url` et `server`) dans configuration
- 4 lancer le but `mvn tomcat:deploy`

Attention aux versions !

Serveur	Servlet	JSP	EL	WS	JASPIC
Tomcat 9.x	Servlet 4.0	JSP 2.3	EL 3.0	WS 1.1	JASPIC 1.0
Tomcat 10.0.x	Servlet 5.0	JSP 3.0	EL 4.0	WS 2.0	JASPIC 2.0
Tomcat 10.1.x	Servlet 6.0	JSP 3.1	EL 5.0	WS 2.1	JASPIC 3.0

(voir <https://tomcat.apache.org/whichversion.html>)

Les API servlets sont fournies par le serveur, elles doivent donc être indiqués

```
<scope>provided</scope>
```

De très nombreuses autre possibilités,

- ▶ Héritage entre POM
(Voir `Dependency Management` et `Plugin Management`)
- ▶ Gestion multi-modules
- ▶ Gestion de profils
- ▶ ...

... plein d'autres choses encore !

Voir aussi Gradle, sensiblement équivalent