APACHE MAVEN

A QUOI SERT MAVEN

GÉRER LE CYCLE DE VIE DE L'APPLICATION

- Compilation
- Résolution des dépendances
- Documentation
- Déploiement
- Test

APACHE MAVEN 3

Compatible avec Maven 2

- 2.0 sortie en 2005
- 3.0 soitie en 2011

Simple à installer

Indispensable

CONVENTION PLUTÔT QUE CONFIGURATION

Maven impose une struture de fichier

```
.
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   └── test
│       ├── java
│       └── resources
└── target
```

COMMENT ÇA FONCTIONNE

Un fichier descriptif à renseigner : **pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.or
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.or
  <modelVersion>4.0.0</modelVersion>

  <groupId>fr.cpe</groupId>
  <artifactId>firstMavenProject</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  ...
</project>
```

COMMENT ÇA FONCTIONNE

CYCLE DE VIE

```
validate
└── compile
    └── test
        └── package
            └── verify
                └── install
                    └── deploy
```

`clean` permet de revenir au début du cycle

COMMENT ÇA FONCTIONNE

LANCEMENT DE MAVEN EN LIGNE DE COMMANDE

`mvn install`

Exécute toutes les phases du build jusqu'à `install`

`mvn clean test`

Effectue le `clean`, puis exécute les phases du build jusqu'à `test`

COMMENT ÇA FONCTIONNE

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy

DÉFINITION D'UN PROJET

Chaque projet/librairie est identifié par un triplet unique

```
<groupId>org.apache.commons</groupId>
<artifactId>commons-lang3</artifactId>
<version>3.8.1</version>
```

VERSIONS

Les versions utilisent un schéma classique

MAJEUR.MINEUR.CORRECTIF

Ex : 2.1.10

On distingue également les versions de développement des versions stables en ajoutant le tag **-SNAPSHOT**

Ex : 2.1.10-SNAPSHOT

INFORMATIONS DU PROJET

Beaucoup d’informations peuvent être précisées

```
<name>Apache Commons Lang</name>
<packaging>jar</packaging>
<description>
    Apache Commons Lang, a package of Java utility classes for the
    classes that are in java.lang's hierarchy, or are considered to be so
    standard as to justify existence in java.lang.
</description>
<scm>
    <connection>
        scm:git:http://git-wip-us.apache.org/repos/asf/commons-lang.git
    </connection>
</scm>
```

GESTION DES DÉPENDANCES

Besoin d'une librairie ?

- Indiquez-le dans le `pom.xml` de votre projet
- Maven regarde si la dépendance existe localement
- Sinon il la récupère depuis un repository public
  - http://search.maven.org
  - http://mvnrepository.com
- La librairie est disponible pour votre application

GESTION DES DÉPENDANCES

```
<dependencies>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.8.1</version>
    </dependency>

    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-core</artifactId>
        <version>${camel-version}</version>
    </dependency>

    ...
</dependencies>
```

${CAMEL-VERSION} ???

UTILISATION DES PROPERTIES

Il est possible de définir des propriétés dans les pom.xml

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <camel-version>2.9.0</camel-version>
</properties>
```

SCOPES

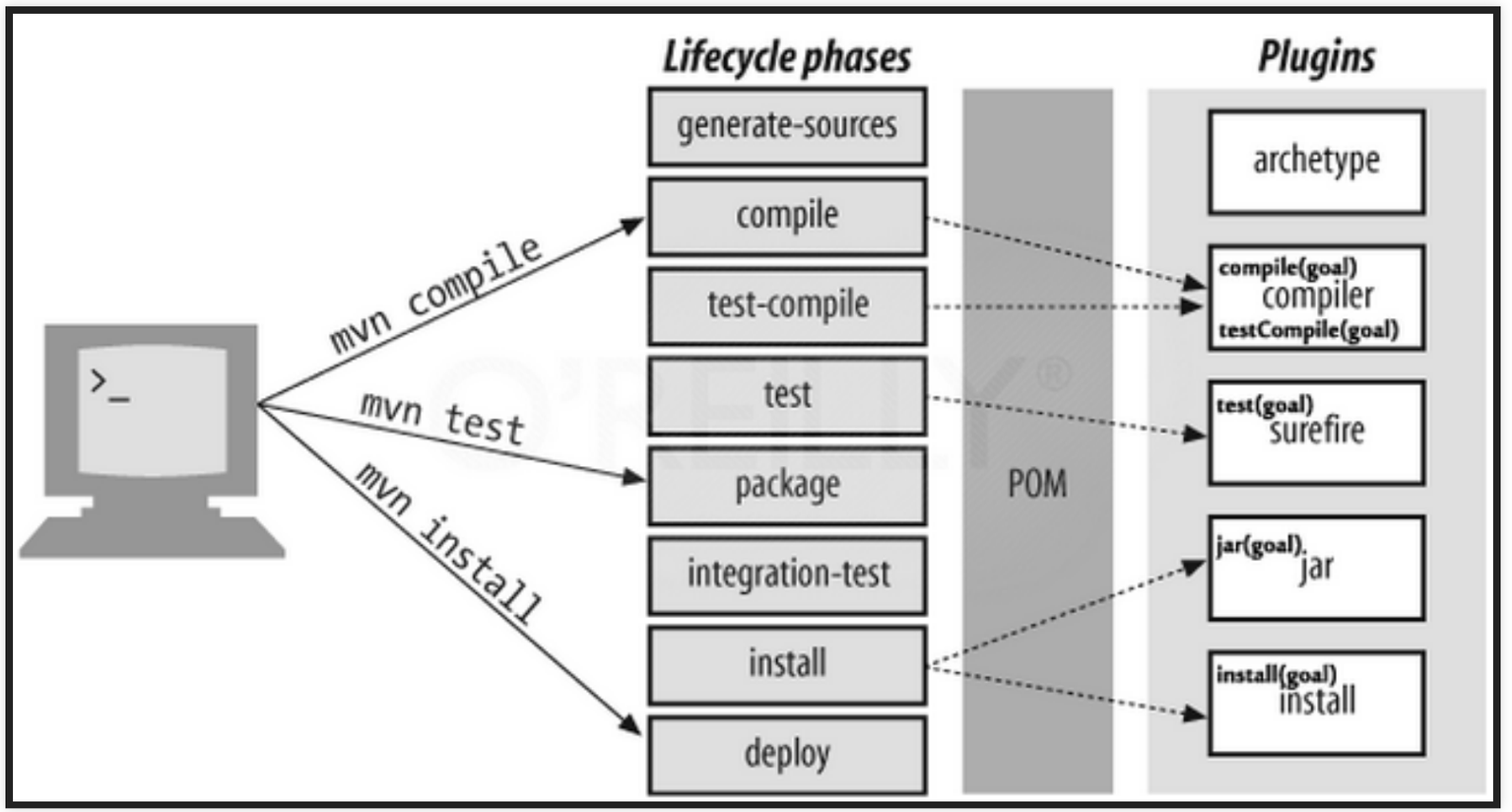
Les dépendances ne sont pas toutes nécessaires (Certaines juste pour les tests par exemple)

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>${mockito-version}</version>
    <scope>test</scope>
</dependency>
```

SCOPES POSSIBLES

- compile
- provided
- runtime
- test
- system
- import

PHASES DE BUILD - PLUGINS - GOALS



PLUGINS

Utile pour faire des actions spéciales pendant le build

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
    </plugins>
</build>
```

PLUGINS - TESTS

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.10</version>
    <configuration>
        <excludes>
            <exclude>**/TestIntegration.java</exclude>
        </excludes>
    </configuration>
</plugin>
```

PLUGINS - JETTY

```
<plugin>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>${jetty-version}</version>
    <configuration>
        <webAppConfig>
            <contextPath>/</contextPath>
        </webAppConfig>
        <scanIntervalSeconds>10</scanIntervalSeconds>
    </configuration>
</plugin>
```

LANCER UN GOAL

```
mvn compiler:compile
```

```
mvn jetty:run
```

Si le plugin n’est pas déclaré dans le pom, il est quand même possible de l’appeler, mais avec son nom complet

```
mvn org.mortbay.jetty:jetty-maven-plugin:run
```

```
mvn org.owasp:dependency-checken:check
```

PROFILES

Les profils permettent de définir des propriétés ou plugins disponibles pour certains profils uniquement

```
<profile>
    <id>release</id>
    <properties>
        <skipTests>true</skipTests>
    </properties>
</profile>
```

PROFILES

Pour executer une commande maven avec un profil, on le spécifie avec l’option P

```
mvn package -Prelease
```

PROPRIÉTÉS

Pour executer un une commande maven ponctuellement avec une propriété, on la spécifie avec l'option D

```
mvn package -DskipTests
```

REPOSITORIES

Certaines librairies ne sont pas disponibles sur les repos centraux, il faut les ajouter manuellement

```
<repositories>
    <repository>
        <id>apache.snapshots</id>
        <name>Apache Snapshot Repository</name>
        <url>https://repository.apache.org/content/repositories/snapshots/</ur
        <releases>
            <enabled>false</enabled>
        </releases>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
</repositories>
```

ARCHETYPES

Créer un projet rapidement à partir d'un squelette

```
mvn archetype:generate
```

APACHE MAVEN

TP

INSTALLER MAVEN

Suivre la procédure sur le site officiel

http://maven.apache.org/download.html

TESTER L'INSTALLATION

En ligne de commande :

```
mvn --version
```

CRÉER UN PROJET

En ligne de commande :

```
mvn archetype:generate -DarchetypeCatalog=internal
```

Archetype: `maven-archetype-quickstart`

GAV: `fr.cpe:td-maven:1.0-SNAPSHOT`

IMPORTER LE PROJET DANS L'IDE

Dans Eclipse, importer le projet :

Import → Import… → Existing Maven Project

MODIFIER UNE LIBRAIRIE

Dans le fichier pom.xml, changer la version de Junit pour en faire une propriété.

Pour vérifier que le projet compile toujours :

```
mvn compile
```

Mettre à jour la dépendence avec la dernière version :

```
mvn versions:display-dependency-updates
```

AJOUTER UNE LIBRAIRIE

Dans le fichier App.java, ajouter

```
DateTime now = new DateTime();
```

Il faut maintenant ajouter la librairie **joda-time** manquante

DÉPENDANCES

Faire une analyse de dépendances :

```
mvn dependency:analyse
```

Voir l’arbre de dépendances :

```
mvn dependency:tree
```

CRÉER UN PROFIL

Créer un profil **release**, qui exclu la phase de test.