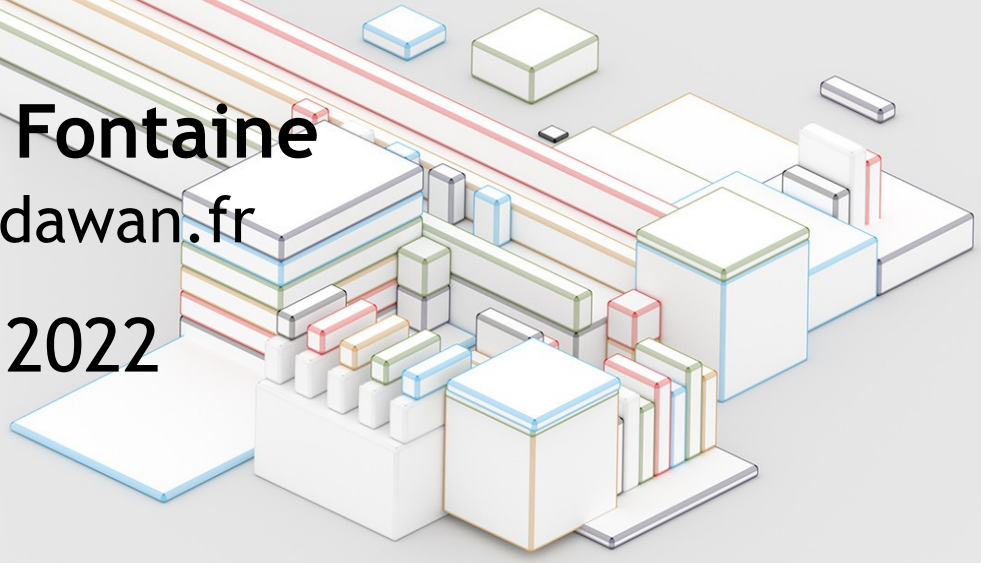
Dawan
Training is our way

Apache Maven



Christophe Fontaine
cfontaine@dawan.fr

23/11/2022

Objectifs

- Connaître les enjeux et possibilités de Maven dans une équipe de développement

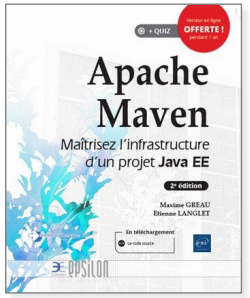
- Être capable de mettre en place et maintenir de multiples projets avec Maven

Durée : 2 jours

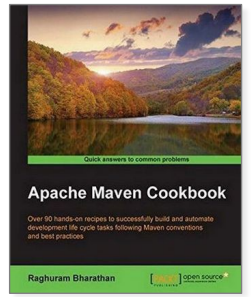
Pré-requis : Connaissance de Java

Bibliographie

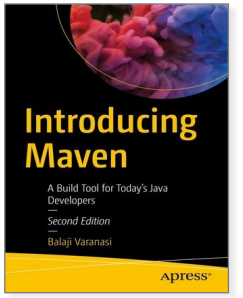
- **Apache Maven**
  Maîtrisez l'infrastructure d'un projet Java EE
  Étienne Langlet, Maxime Gréau
  Éditions ENI - 2ème édition - Juin 2019



- **Apache Maven Cookbook**
  Raghuram Bharathan
  Packt Publishing - Avril 2015



- **Introducing Maven**
  Balaji Varanasi
  Apress - 2nd edition - Novembre 2019



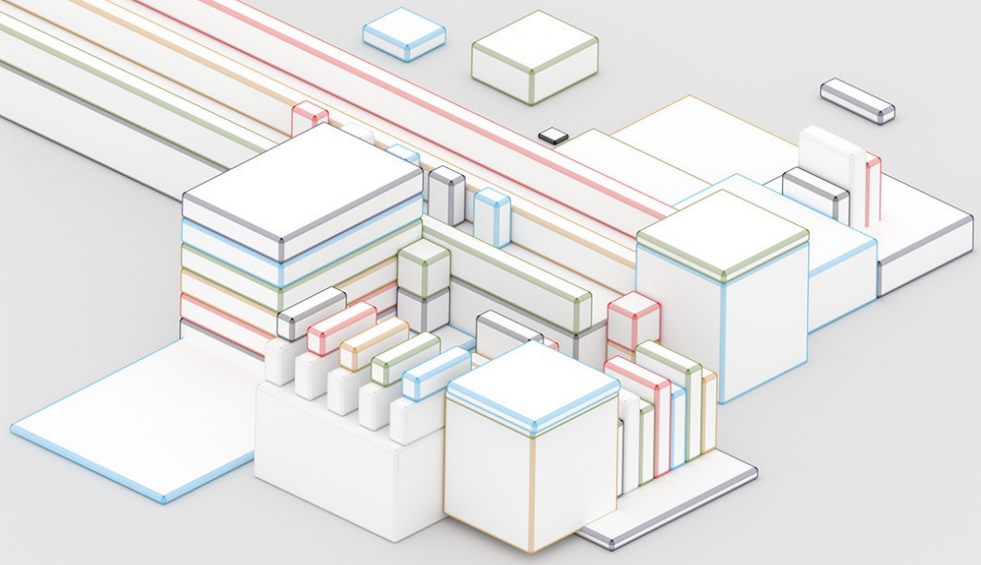
- **Maven: The Complete Reference** (en ligne, pdf)
- **Maven by Example** (en ligne, pdf)
- **Site officiel de Maven** (http://maven.apache.org/)

Plan

- Découvrir Maven
- Construire un projet avec Maven
- Gérer les dépendances du projet
- Générer le site web et des rapports du projet
- Synthèse et Bonnes pratiques
- Annexe

Dawan
Training is our way

Découvrir Maven



Présentation

- Maven est une outil de gestion de projets regroupant :
  - Un ensemble de standards
  - Un « repository » d'un format particulier
  - Un outil pour gérer et décrire un projet
- Il fournit un cycle de vie standard pour Construire, Tester et Déployer des projets selon une logique commune
- Il s'articule autour d'une déclaration commune de projet que l'on appelle le POM (**P**roject **O**bject **M**odel)
- Maven, c'est une façon de voir un produit comme une collection de composants inter-dépendants qui peuvent être décrits sous un format standard

Origines

- Différentes approches dans le processus de construction de projets
- Manque de vue globale (copier/coller) pour construire un projet à partir d'un autre
  ↳ Génère rapidement des incohérences
- **ANT**
  - Apache Ant est un outil open source de build
  - Avec Ant, il faut définir des tâches (task) et des cibles (target)
  - Une tâche Ant peut être :
    - la création d'un répertoire
    - l'exécution d'un test
    - la compilation du code source
    - création d'un fichier d'archive (JAR, WAR …)
    - …

Origines

- Une cible est un ensemble de tache

```
<project name="Sample Build File" default="compile" basedir=".">
    <!-- une cible -->
    <target name="compile" description="Compile Source Code">
        <!-- 3 taches -->
        <echo message="Starting Code Compilation"/>
        <javac srcdir="src" destdir="dist"/>
        <echo message="Completed Code Compilation"/>
    </target>
</project>
```

- Ant n'impose pas de convention ou de restriction
  ↪ cela peut entrainer un fichier de configuration (build.xml) complexe, difficile à comprendre et à maintenir

Principes

- Maven fournit un modèle détaillé applicable à n'importe quel projet : langage commun, interface d'abstraction
- La structure d'un projet Maven et son contenu sont déclarés dans un POM
- Le POM est purement déclaratif :
  - Le développeur va y déclarer des objectifs et des dépendances
  - L'orchestration des tâches qui suivront (compilation, test, assemblage, installation) sera gérée par des plugins appropriés
- Apports : Cohérence, réutilisabilité, agilité, maintenabilité

Principes

- Éléments importants :
  - Répertoires standards pour les projets pour facilite la lecture de tout nouveau projet (structure connue) Structure modifiable mais impacte la complexité du POM
  - Sortie unique (output) ; par contre, Maven favorise le découpage en sous-projets (Separation of Concern)
  - Une convention de nommage standard : Directories, fichiers

Fonctions

- Gestion des dépendances
- Construction multi-modules
- Normalisation de la structure de projets
- Modèle de construction cohérent
- Système orientée plugin
- Génération de rapports pour le projet

Historique

2003 **Maven 1**

- verbeux

2005 **Maven 2**

- ré-écriture complète
- pas de rétrocompatibilité

2010 **Maven 3**

- amélioration de Maven 2 (plus stable)

? **Maven 4**

- 2022-10-24 version: 4.0.0-alpha-2

Installation du JDK

- **Téléchargement**
  - Oracle JDK : https://www.oracle.com/java/technologies/downloads/
  - OpenJDK : https://adoptium.net/
- **Paramétrages des variables d'environnement**

  Dans le menu Windows, taper : **env**

  Choisir → Modifier les variables d'environnement **du système**

  ↳ Variable d'environnement …

  Dans **variables système**
  - Créer une variable **`JAVA_HOME`** qui contient le chemin vers le dossier contenant le JDK
  - Modifier la variable **`PATH`** en ajoutant **`%JAVA_HOME%\bin`**
- **Vérification**

  `> javac -version` → doit afficher la version de java

Installation de Maven

- **Téléchargement**
  https://maven.apache.org/download.cgi → Binary zip archive
  ↳ décompresser le fichier dans un répertoire
- **Paramétrages des variables d'environnement**
  Dans le menu Windows, taper : **env**

  Choisir → Modifier les variables d'environnement du système
  ↳ Variable d'environnement …

  Dans **variables système**
  - Créer une variable **`M2_HOME`** qui contient le chemin vers le dossier de maven
  - Modifier la variable **`PATH`** en ajoutant **`%M2_HOME%\bin`**
- **Vérification**

  `> mvn -version` → doit afficher la version de maven

POM (Project Object Model)

- Descripteur d'un projet Apache Maven, au format XML
- Indique à Maven quel type de projet il va devoir traiter et comment il va devoir s'adapter pour transformer les sources et produire le résultat attendu en définissant plusieurs goals (tâches)
- Ces tâches ou goals, utilisent le POM pour s'exécuter correctement. Des plugins peuvent être développés et utilisés dans de multiples projets de la même manière que les tâches pré-construites
- Description complète :

  https://maven.apache.org/ref/3.8.6//maven-model/maven.html

Goals

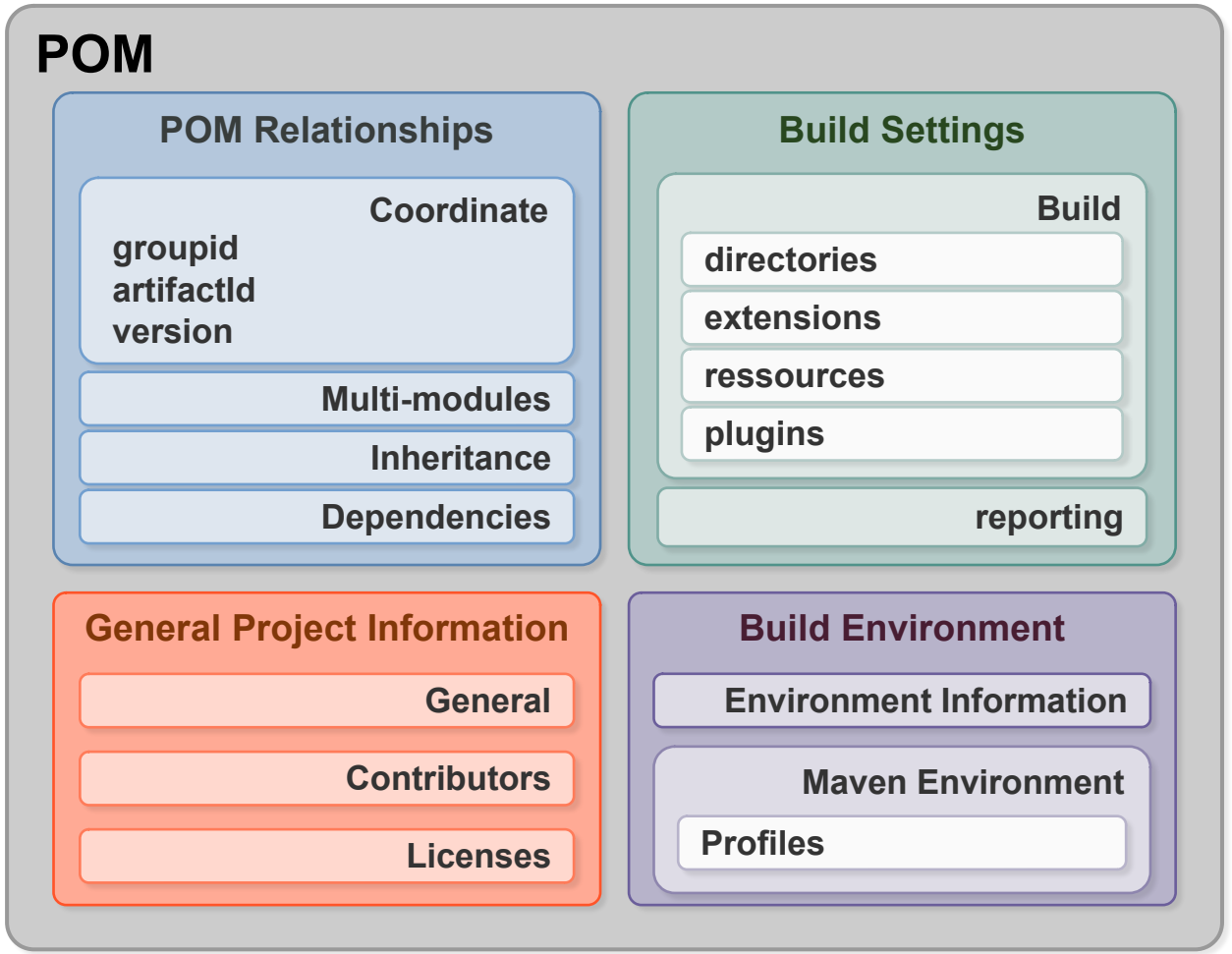
- En introduisant un descripteur de projet, Maven nécessite qu'un développeur fournisse des informations **sur ce qui est construit**
- Cela diffère des systèmes de construction traditionnels où les développeurs informent le système sur **la manière de construire**
- Les développeurs peuvent se concentrer sur la logique de développement

Goals

- Goals Communs : l'introduction des cycles de vies dans Maven a considérablement réduit le nombre de goals qu'un développeur doit absolument savoir maîtriser
- Les goals suivants seront toujours considérés comme utiles (et peuvent être utilisés dès que le descripteur de projet a été généré) :
  - **clean:clean** → Supprime tous les artéfacts et les fichiers intermédiaires crées par Maven
  - **eclipse:eclipse** → Génère des fichiers projets pour Eclipse
  - **javadoc:javadoc** → Génère la documentation Java pour le projet
  - **antrun:run** → Exécute une cible ANT
  - **clover:check** → Génère un rapport d'analyse du code
  - **checkstyle:checkstyle** → Génère un rapport sur le style de codage du projet
  - **site:site** → Crée un site web de documentation pour le projet
    Il inclura beaucoup de rapports d'informations concernant le projet

Structure du POM

Le POM se compose de 4 catégories de description et de configuration



Entête d'un POM (GAV)

- Maven identifie de manière unique un projet à l'aide de :
  - **groupID** : identifiant arbitraire du groupe de projet habituellement basé sur le package Java (sans espace, ni :)
  - **artfiactId** : nom arbitraire du projet (sans espaces, ni :)
  - **version** : Version du projet
    - ↪ format **{Major}.{Minor}.{Maintanence}**
      ajouter **-SNAPSHOT** si en développement

Syntaxe du GAV → **groupId:artifactId:version**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.dawan.training</groupId>
    <artifactId>maven-training</artifactId>
    <version>1.0</version>
</project>
```

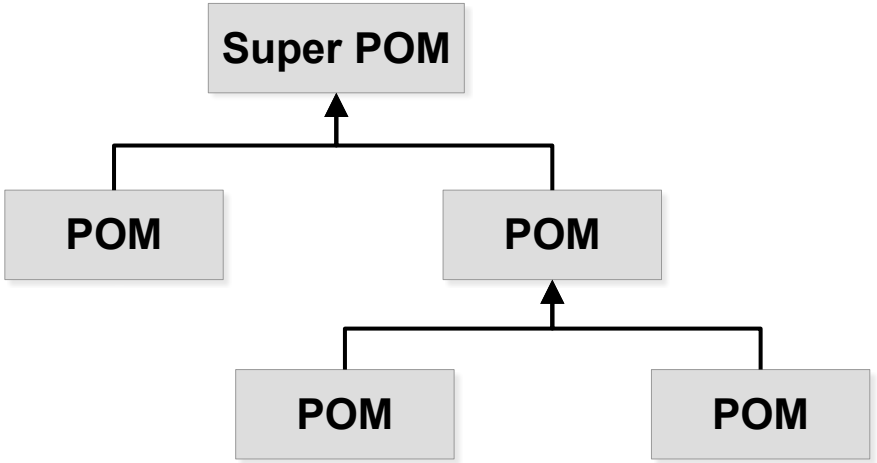
Packaging

- Identifier le type de paquet à produire :
  - pom, jar, war, ear, rar, par, custom
  - jar est la valeur par défaut

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.dawan.training</groupId>
    <artifactId>maven-training</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
</project>
```

Super POM

- Tous les POM étendent le Super POM
  Le super POM est à Maven ce que Object est à JAVA
- Il est une partie de l'installation Maven
  Fichier : **pom-4.0.0.xml** contenu :
  dans le dossier **`org\apache\maven\model`**
  du jar `\lib\maven-model-builder-3.8.3.jar`
- Le Super POM définit les valeurs par défaut
  https://maven.apache.org/ref/3.8.6/maven-model-builder/super-pom.html



Héritage

- Les fichiers POM peuvent hériter d'un configuration : groupId, version, configuration, dépendances …

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <parent>
        <artifactId>maven-training-parent</artifactId>
        <groupId>com.dawan.training</groupId>
        <version>1.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-training</artifactId>
    <packaging>jar</packaging>
</project>
```

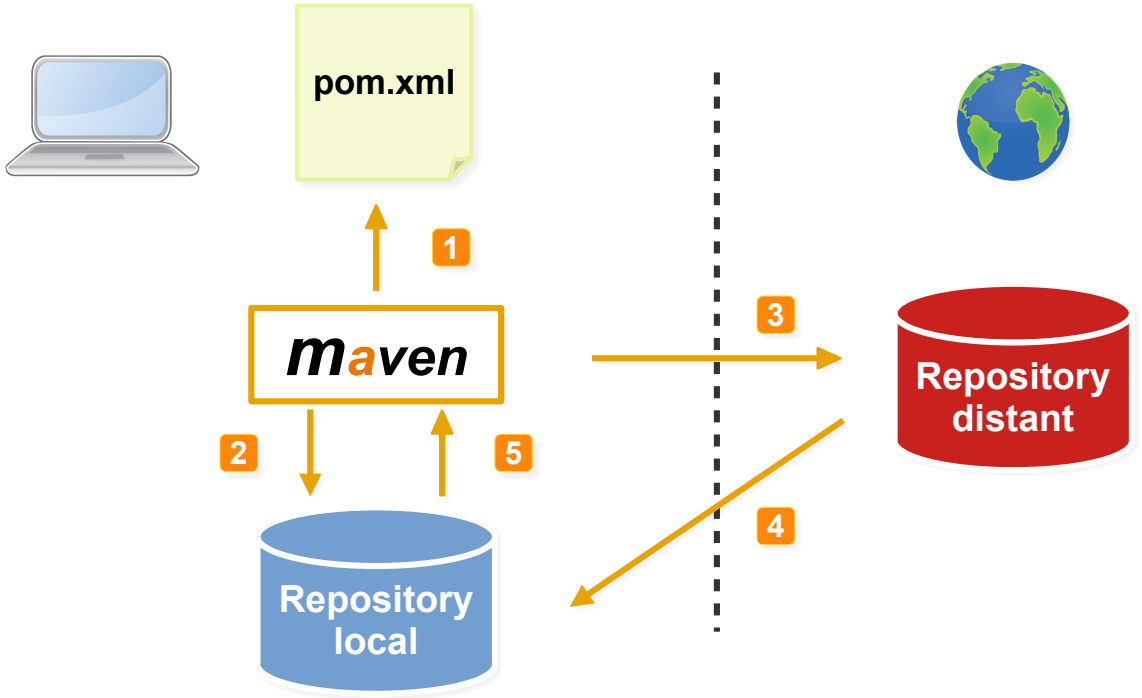
Projets multi-modules

- Maven permet un support multi-modules
- Chaque projet maven crée un artefact primaire
- Une POM parent est utilisé par des modules de groupe

```
<project>
    ...
    <packaging>pom</packaging>
    <modules>
        <module>maven-training</module>
        <module>maven-training-web</module>
    </modules>
</project>
```

Référentiel (Repository)

- Emplacement des bibliothèques logicielles organisées selon une arborescence spécifique à Maven (metadata.xml)/
- 2 types de référentiels : local ou remote
  central Repository : https://search.maven.org/



Premières commandes

- Syntaxe : **`mvn [options] [goal(s)] [phase(s)]`**
- Aide : **`mvn --help`**
- Valider le POM : **mvn validate**

  Options : `-X` pour le mode debug, `-e` pour la trace Java
- Afficher le POM complet (avec les données héritées) :

  `mvn help:effective-pom`

  Option : `-Doutput` pour rediriger la sortie

  `mvn help:effective-pom -Doutput=pom-complet.xml`

Fichier settings.xml

- Le fichier settings.xml,permet d'ajouter une configuration spécifique à l'utilisateur

  http://maven.apache.org/xsd/settings-1.0.0.xsd
- Maven recherche le settings.xml à 2 emplacements :
  - dans le dossier **conf** du dossier d'installation de maven
    ↳ configuration globale
  - dans le dossier **.m2** qui se trouve dans le dossier utilisateur
    ↳ configuration utilisateur
- Si les 2 fichiers existent maven va fusionner leur contenu et la configuration utilisateur est prioritaire
- **<localRepository>**

  Configuration de l'emplacement du local Repository

```
<localRepository>c:\mavenrepo</localRepository>
```

Fichier settings.xml

- **<offline>false</offline>**

  indique à Maven de ne pas se connecter au réseau et de fonctionner en mode hors ligne

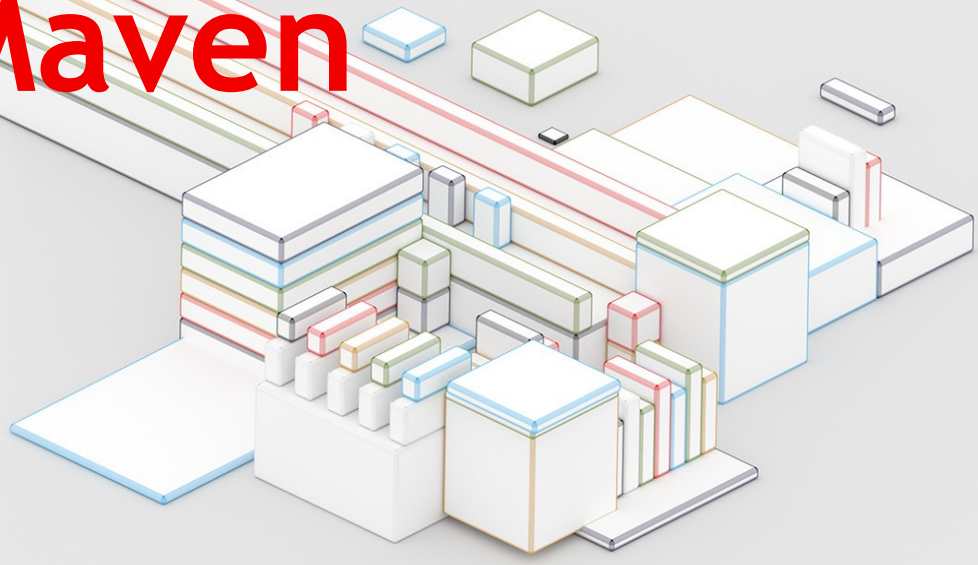
- **<proxy>**

  Configuration du proxy

```
<proxies>
  <proxy>
     <id>example-proxy</id>
     <active>true</active>
     <protocol>http</protocol>
     <host>proxy.example.com</host>
     <port>8080</port>
     <username>proxyuser</username>
     <password>password</password>
     <nonProxyHosts>www.google.com|*.example.com</nonProxyHosts>
   </proxy>
</proxies>
```

Dawan
Training is our way

Construire un projet avec Maven



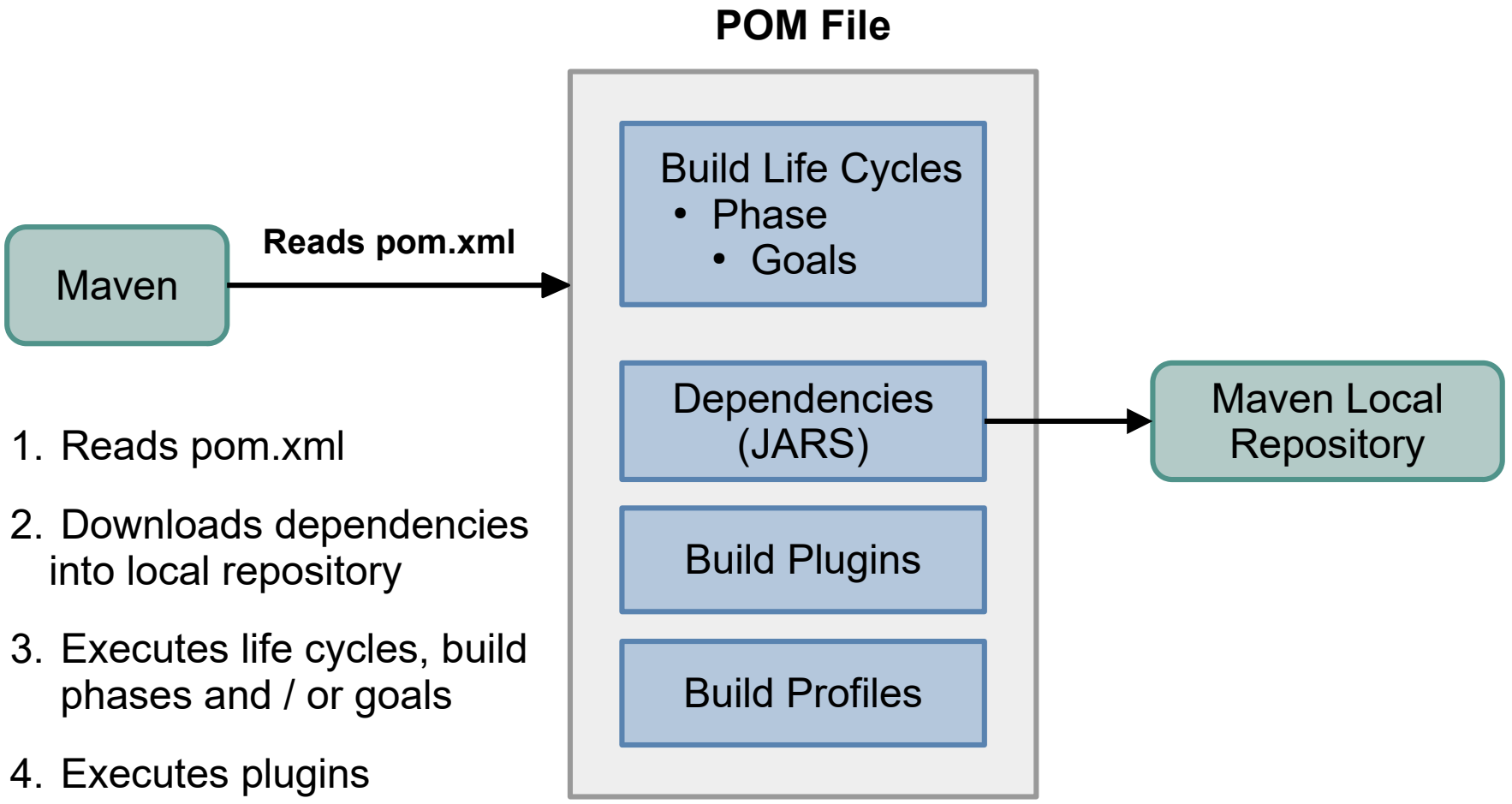
Plugins

https://maven.apache.org/plugins/index.html

Principaux plugins

- **Antrun** : Lance un ensemble de tâches Ant à partir d'une phase de la construction
- **Clean** : Nettoie après la construction
- **Compiler** : compile des sources Java
- **Deploy** : Construit et Déploie les artéfacts dans un repository
- **Ear** : Génère un fichier EAR à partir du projet
- **Eclipse** : Génère un fichier projet Eclipse du projet courant
- **Install** : Installe les artefacts construits dans un repository
- **Jar** : Génère un fichier Jar à partir du projet
- **Javadoc** : Crée la documentation Java du projet
- **Projecthelp** : fournit des informations sur l'environnement de travail du projet
- **Project-infos-reports** : Génère un rapport standard de projet
- **Rar** : construit un RAR à partir du projet
- **Release** : Libère le projet en cours – mise à jour du POM et étiquetage dans le SCM (Source Control Management)
- **Resources** : Copie les fichiers sous le répertoire "resources" dans un fichier JAR
- **Site** : Génère un site pour le projet courant.
- **Source** : Génère un JAR contenant les sources du projet.
- **Surefire** : Exécute les jeux de test.
- **War** : construit un fichier WAR du projet courant

Processus global



1. Reads pom.xml
2. Downloads dependencies into local repository
3. Executes life cycles, build phases and / or goals
4. Executes plugins

All executed according to selected build profile

Gestion de projet

- Difficulté : Créer « l'infrastructure de développement »
  ↳ En utilisant Maven, il est possible d'accélérer ce processus en générant un squelette de projet qui peut être utilisé comme modèle
- **Quelques questions qu'il faut poser avant de commencer**:
  - Quels sont les attributs de mon projet (Nom, Version, Equipe…) ?
  - Quelle est la structure de fichiers de mon projet ?
  - Quels sont les composants, objets et artefacts de mon projet ?
  - Quelles sont les dépendances de mon projet ?
  - De quoi a besoin mon projet pour se construire ?
  - De quoi a besoin mon projet pour fonctionner ?
  - De quoi a besoin mon projet pour être testé ?

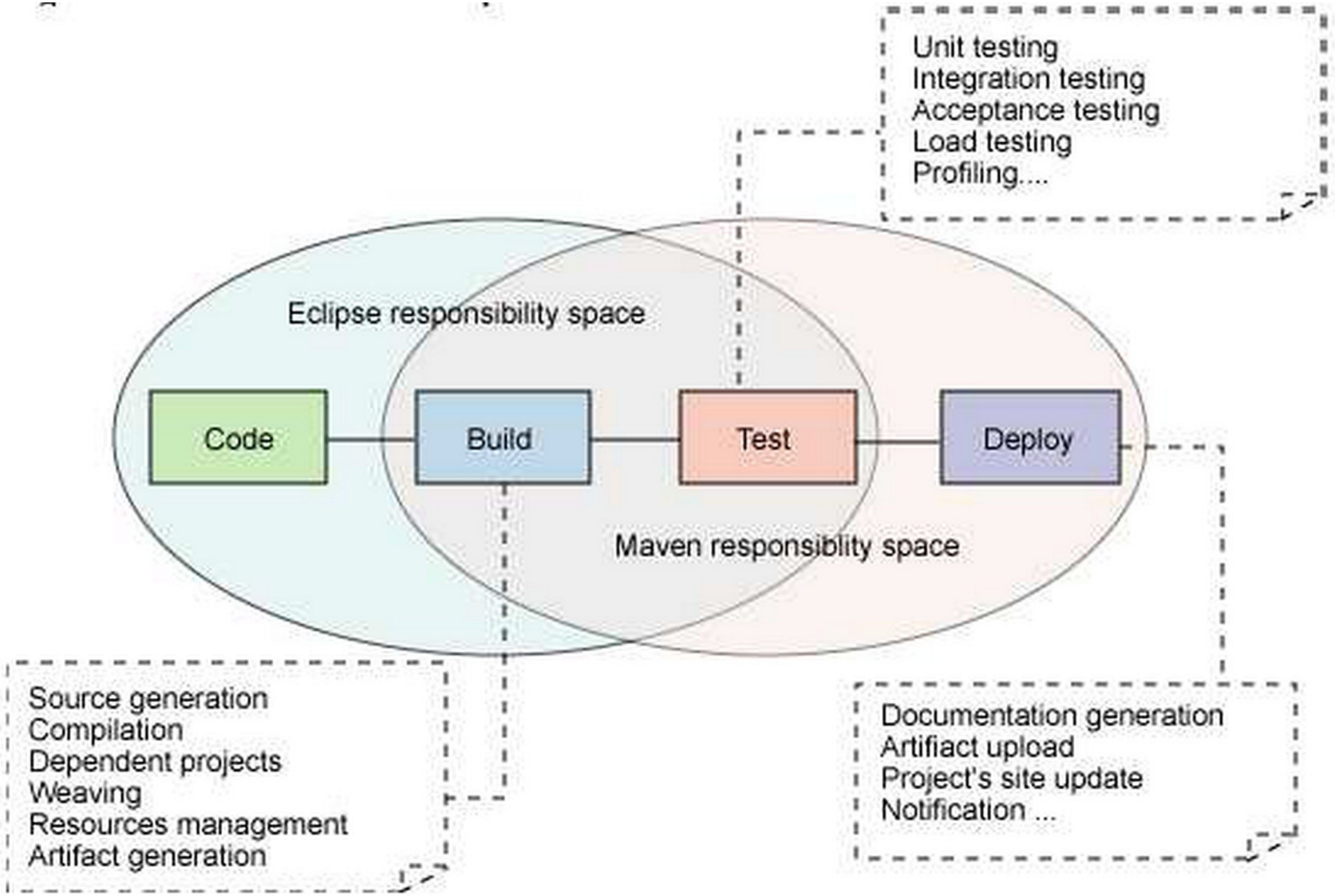
Gestion de projet

- De quels plug-ins aura besoin mon projet ?
- Quels rapports ai-je envie de générer pour mon projet en construisant mon site Web ?

- **Après avoir répondu aux questions, il est possible de réaliser les tâches suivantes:**

1. Faire le design de la structure de fichiers du projet.
2. Commencer à créer/modifier la structure du principal :
   ↳ pom.xml
3. Définir les dépendances du projet

Maven et Eclipse



Structure d'un projet (Archetype)

- Un archetype est un modèle (pattern), constitué d'un descripteur (archetype.xml), des fichiers qui seront copiés par l'archetype, du POM
- Il existe des archetypes déjà construits
  - https://maven.apache.org/guides/introduction/introduction-to-archetypes.html
  - https://maven.apache.org/guides/introduction/introduction-to-archetypes.html
- Le type d'archetype à créer sera à passer dans le sélecteur

  -DarchetypeArtifactId de la commande mvn
- On peut définir des nouveaux archetypes :
  - https://maven.apache.org/guides/mini/guide-creating-archetypes.html

Structure d'un projet

Chemin	Description
`target`	default work directory
`src`	l'ensemble des sources
`src/main`	sources du primary artefact
`src/test`	sources de test
`src/main/java`	fichiers sources java
`src/main/webapp`	sources web
`src/main/resources`	fichiers non compilables
`src/test/java`	sources de tests Java
`src/test/resources`	fichiers de tests non compilables

Création d'un projet

- **Création** : `mvn archetype:create -DgroupId=com.dawan.app1 -DartifactId=my-app1`
- **Compilation** : `mvn compile`
- **Tests** : `mvn test`
  Option : `-Dtest=NomClasseDeTest`
- **Packaging** : `mvn package`
- **Ressources** : `mvn process-resources`
- **Site** : `mvn site`
- **Nettoyage** : `mvn clean`
- **Création d'un projet Eclipse (.projet)** : `mvn eclipse:eclipse`
  Options : `-DdownloadJavadocs=true, -DdownloadSources=true`
- **Téléchargement de nouveaux plugins** :
  `mvn help:describe -DgroupId=org.apache.maven.plugins-DartifactId=mavencompiler-plugin -Dfull=true|more`

Enregistrement d'une dépendance dans le repository

- Pouvoir ajouter un projet au repository en tant que dépendance réutilisable
- **`mvn install`** génère le descripteur et le meta-data de la dépendance et réalise la copie dans le local repository ou remote
- **`mvn install:install-file`** pour installer un jar dans le repository
- Exemple du driver Oracle :

```
mvn install:install-file  -DgroupId=com.oracle
                          -DartifactId=ojdbc6
                          -Dversion=11.2.0.3
                          -Dpackaging=jar
                          -Dfile=ojdbc6.jar
                          -DgeneratePom=true
```

Cycle de vie d'un projet

- Pour construire une application, Maven s'appuie sur un cycle de construction, constitué de plusieurs phases
- Si vous demandez d'exécuter une phase, alors toutes les phases précédentes seront forcément d'abord appelées, mais on peut changer ce mode grâce aux conventions

  https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
- Si vous devez ajouter des phases, vous passez par un plugin, soit livré, soit que vous développez

Plugins

- Plugins : module réalisant un traitement liés aux goals ou **Mojos** (**M**aven **O**ld **J**ava **O**bject) issus de plugins
- Résolution des plugins :
  - Fichier de configuration interne components.xml (inclus dans lib/maven-core.jar)
  - Éléments du POM (effective POM) : <plugins>, <pluginManagement>
- Référentiels pour les plugins à définir dans settings.xml ou dans le POM : <pluginRepositories>
- Possibilité de configurer les plugins directement : maven-jar-plugin, maven-compiler-plugin

Encodage

- Encodage des caractères pour les fichiers Java dans

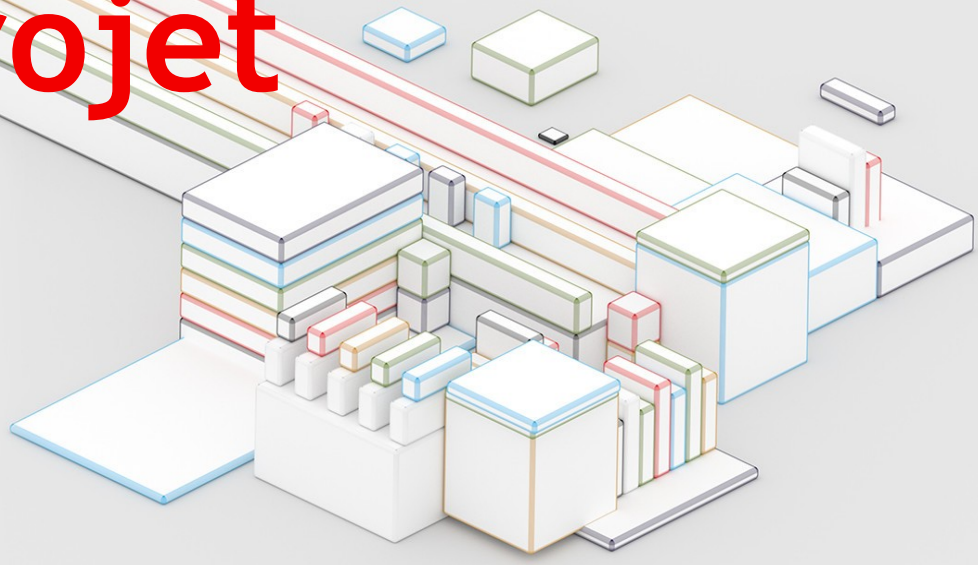
```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

- Si fichiers de propriétés en ISO, ajouter une configuration dans maven-resources-plugin :

```
<plugin>
    <artifactId>maven-resources-plugin</artifactId>
        <version>2.6</version>
        <configuration>
        <nonFilteredFileExtensions>
        <nonFilteredFileExtension>xml</nonFilteredFileExtension>
        </nonFilteredFileExtensions>
        <encoding>ISO-8859-1</encoding>
    </configuration>
</plugin>
```

Dawan
Training is our way

Gérer les dépendances du projet



Gestion des dépendances

- Maven a révolutionné la gestion des dépendances Java
- Plus besoin de vérifier les bibliothèques dans le système de gestion de versions grâce au référentiel

  Maven (Maven Central Repository) :

  https://repo.maven.apache.org/maven2/

  https://search.maven.org/
- Crée un fichier module de métadonnées (POM)
- Concept introduit de dépendance transitive.
- On peut inclure les sources et la javadoc.
- Affichage : **`mvn dependency:tree`**

Ajout d'une dépendance

- Une dépendance est composée de :
  - GAV
  - Scope: compile, test, provided (default=compile)
  - Type: jar, pom, war, ear, zip (default=jar)

```
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>
</dependencies>
```

Portée des dépendances <scope>

- **Compile** : indique que la dépendance est nécessaire pour la compilation et l'exécution. Ces dépendances seront souvent fournies avec les distributions et empaquetées dans les déploiements
- **Provided** : indique que la dépendance est nécessaire pour la compilation mais ne devrait pas être empaquetée dans les déploiements et distributions. Ces dépendances devraient être fournies par une source externe (typiquement un container) durant le runtime
- **Runtime** : indique qu'une dépendance n'est pas nécessaire pour la compilation, mais l'est pour le runtime. Souvent les dépendances de runtime sont des implémentations d'API externes et sont injectées au moment de l'exécution. Un exemple de dépendance d'exécution est le pilote JDBC
- **Test** : ces dépendances sont requises pour l'exécution des tests unitaires et ne sont pas rendues disponibles dans les distributions et les déploiements
- **System** : permet au projet de dépendre d'une bibliothèque non présente dans le référentiel mais dans le système de fichiers <systemPath> (portée déconseillée)
- **Import** : indique des dépendances incluses dans un autre POM

Résolution des conflits de dépendance et utilisation des versions

La recherche « la plus proche », utilisable avec des patterns

- (,1.0] Inférieur ou égal à 1
- [1.2,1.3] Entre 1.2 et 1.3 (inclusif)
- [1.0,2.0) Supérieur ou égal à 1.0, mais inférieur à 2.0
- [1.5,) Supérieur ou égal à 1.5
- (,1.1),(1.1,) Toute version, excepté 1.1

```
<dependency>
 <groupId>org.codehaus.plexus</groupId>
 <artifactId>plexus-utils</artifactId>
 <version>[1.1,)</version>
</dependency>
```

Par défaut, la repository est contrôlée une fois par jour pour MAJ des versions des artefacts utilisés, mais ceci peut être configuré dans le POM avec :

```
...
<repository>
...
 <releases>
  <updatePolicy>interval:60</updatePolicy>
 </releases>
</repository>
```

Stockage des dépendances

- Les dépendances sont téléchargés depuis les repositories via http
- Version en cache dans le local repository

  `${user.home}/.m2/repository`

- Structure des répertoires :

  `{groupId}/{artifactId}/{version}/{artifactId}-{version}.jar`

  `groupId '.' is replaced with '/'`

- Maven Central est le repository communautaire primaire : https://repo1.maven.org/maven2/

Proxy Repositories

- Utiliser un proxy de repositories est utile :
  - avoir un cache organisationnel d'artefacts
  - effectuer un contrôle sur les dépendances
  - combiner les référentiels
- Nexus repository manager :
  - https://www.sonatype.org/nexus/
  - https://books.sonatype.com/mcookbook/reference/repoman-sect-proxy-repo.htm

Définition de repositories

- Définition dans le pom
- Repositories peuvent être hérités du parent
- Le téléchargement de snapshots peut être contrôlé

```
<project>
    …
    <repositories>
        <repository>
            <id>dawan-mainRep</id>
            <name>Dawan Main Repository</name>
            <url>http://dev.dawan.fr/nexus/content/groups/main-repo</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
</project>
```

Dépendances transitives

- Une dépendance qui devrait être inclus lors de la déclaration du projet lui-même est une dépendance

  ProjetA dépend du ProjetB

  Si ProjetC dépend ProjetA alors ProjetB est automatiquement inclus

- Seules les dépendances avec un portée (`<scope>`) “compile ou runtime” sont transitives

- Les dépendances transitives sont contrôlés à l'aide:
  - d'exclusions
  - de déclarations facultatives

Exclusion d'un dépendance transitive


```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.0.5.RELEASE</version>
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
</project>
```

Dépendances optionnelles

If project Y depends on project Z, the owner of project Y can mark project Z as an optional dependency, using the “optional” element. When project X depends on project Y, X will depend only on Y and not on Y’s optional dependency Z

The owner of project X may then explicitly add a dependency on Z, at her option (It may be helpful to think of optional dependencies as “excluded by default.”)

Maven Book

- Une dépendance est marquée comme optionnelle car une partie de notre livrable qui l’utilise n’est pas indispensable pour l’utilisateur

  De ce fait, au lieu d’imposer à ce dernier de récupérer toutes les librairies qu’il n’utilisera peut-être pas, elles sont marquées optionnelles, et cela sera du ressort de l’utilisateur de rajouter dans son propre projet la dépendance manquante si il vient à l’utiliser

Dépendances optionnelles

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.0.5.RELEASE</version>
            <optional>true</optional>
        </dependency>
    </dependencies>
</project>
```

Dependency Management

2 types d'entrée jouant un rôle au niveau des dépendances dans le POM :

- Tout d'abord, l'entrée `<dependencyManagement>` n'interfère pas dans le graphe de dépendances, mais joue le rôle de préférences appliquées aux entrées (exemple la version)
- Par contre, l'entrée `<dependencies>` constitue le graphe des dépendances, où l'héritage va jouer un rôle essentiel

https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Importing_Dependencies

Propriétés

- Utilisation des propriétés existantes déclarées dans `<properties>` : `${project.*}`, `${settings.*}`, `${env.*}`

  https://books.sonatype.com/mvnref-book/reference/resource-filtering-sect-properties.html

- Création de nouvelles propriétés

```
<!-- déclaration d'une propriété, utilisable avec :
                                                ${dawan.repository} -->
<dawan.repository>http://dev.dawan.fr/repository</dawan.repository>
```

- Propriétés des plugins :

```
<!-- propriétés de plugins : raccourcis à la syntaxe interne du
                                                            plugin -->
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
<maven.compiler.optimize>true</maven.compiler.optimize>
```

Resource Filtering

- On peut utiliser des propriétés du projet dans un fichier de ressource

```
<build>
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <filtering>true</filtering>
        </resource>
    </resources>
</build>
```

https://books.sonatype.com/mvnref-book/reference/resource-filtering-sect-description.html

Profils

- Les « profiles » permettent de créer des variations dans le cycle de construction, afin de s'adapter à des particularités :
  - des constructions pour des plateformes différentes (OS)
  - tester sur différentes BDD
  - référencer un système de fichiers local

    …
- Les profils sont déclarés dans des entrées du POM
- Ils peuvent être « activés » de différentes manières
- Ils modifient le POM au moment du build, prenant en considération les paramètres passés

  https://books.sonatype.com/mvnref-book/reference/resource-filtering-sect-description.html

Profils

- Les profils pourront être définis à trois endroits :
  - settings.xml de la directory .m2 (repository)
  - un fichier profiles.xml dans la même directory que le POM
  - le POM lui-même
- Dans un de ces fichiers, vous pouvez définir les éléments suivants : Repositories, pluginRepositories, dependencies, plugins,modules, reporting, dependencyManagement, distributionManagement
- Activer un profile : sélecteur `-P` de la commande `mvn` qui prendra en argument la liste des ids de profil :

```
mvn -P profile1, profile2 install
```

Plus d'informations sur http://www.dawan.fr

Contactez notre service commercial au
09.72.37.73.73 (prix d'un appel local)