

sonarqube 



Apache Maven Project

<http://maven.apache.org/>

Prérequis Logiciels

- Java : JDK 11 ou plus récent.
- Une instance SonarQube préconfigurée
- Git / GitBash

1. Rappel sur Maven

Apache Maven est un outil permettant d'automatiser la gestion d'un projet Java. Il offre entre autres les fonctionnalités suivantes :

- Compilation et déploiement des applications Java (JAR, WAR)
- Gestion des librairies requises par l'application en utilisant des dépôts (*repository*) local ou distant
- Exécution des tests unitaires
- Génération des documentations du projet (site web, pdf, Latex)
- Intégration dans différents IDE (Eclipse, NetBeans, etc.)

Ce tutoriel va vous apprendre à utiliser Maven dans tout projet de développement utilisant Java. Après avoir terminé ce tutoriel, vous vous retrouvez avec un niveau d'expertise modéré dans l'utilisation d'Apache Maven.

Pour résumer, Maven simplifie et standardise le processus de construction d'un projet Java. Il gère la compilation, la distribution, la documentation, la collaboration en équipe et d'autres tâches de façon transparente. Maven augmente la réutilisabilité et prend en charge la plupart des tâches liées à la construction.

2. Prérequis Maven

- Java : JDK 1.8 ou plus récent.
- RAM : Pas de minimum requis
- Espace disque :
 - Environ 10 Mo sont nécessaires pour l'installation de Maven.
 - Un espace disque supplémentaire sera utilisé pour le dépôt local de Maven dont La taille varie en fonction de l'utilisation, mais il faut prévoir au moins 500 Mo.
- Système d'exploitation : Aucun prérequis.
 - Disponible sous Windows, Linux, MAC, etc...

3. Installation d'Apache Maven

Etape 1: Vérifier l'installation de Java sur votre machine

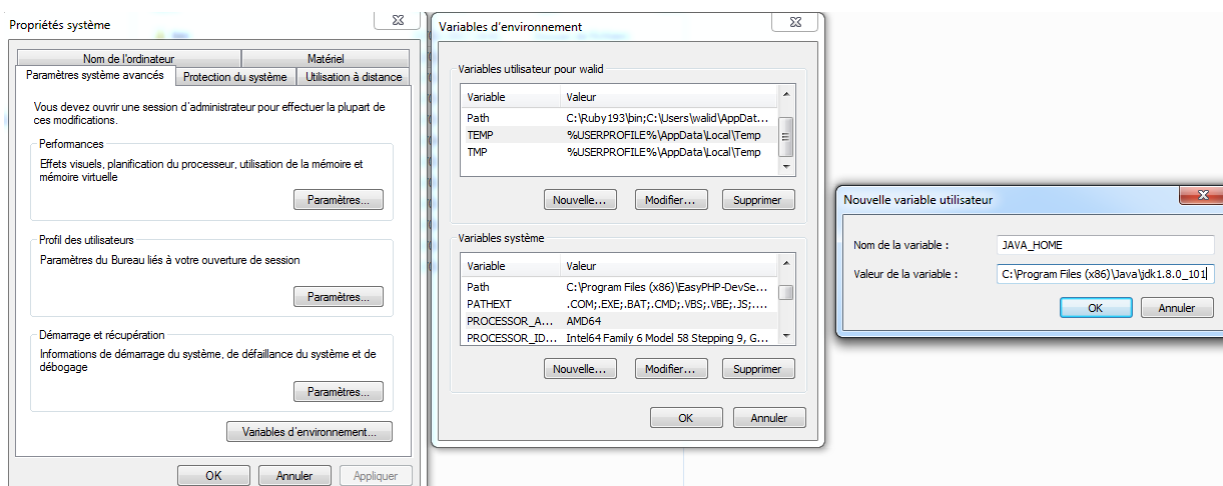
```
C:\Users\walid>java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
```

Si Java n'est pas installé, téléchargez et installez le Kit de Développement Java (SDK) depuis le site officiel :

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>.

Etape 2: Définir l'environnement JAVA

Définissez la variable d'environnement **JAVA_HOME** pour qu'elle pointe vers l'emplacement du répertoire de base où Java est installée sur votre machine. Par exemple :



Etape 3: Télécharger l'Archive Maven

- Télécharger Maven depuis son site web officiel <https://maven.apache.org/download.cgi>
- Pour Windows : <http://mirrors.ircam.fr/pub/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.zip>



The screenshot shows the Apache Maven Project website. The main heading is 'Downloading Apache Maven 3.5.3'. Below this, it states that Apache Maven 3.5.3 is the latest release. It also provides a list of download mirrors, with the currently selected one being <http://www-us.apache.org/dist/>. A section titled 'System Requirements' lists the following:

Requirement	Details
Java Development Kit (JDK)	Maven 3.3+ require JDK 1.7 or above to execute - they still allows you to build against 1.3 and other JDK versions by Using Toolchains
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, additional disk space will be used for your local Maven repository. The size of your local repository will vary depending on usage but expect at least 500MB.
Operating	No minimum requirement. Start up scripts are included as shell scripts and Windows batch files.

Etape 4 : Extraire l'archive de Maven

Extraire l'archive dans le répertoire dans lequel vous souhaitez installer Maven 3.x.x. Le sous-répertoire apache-maven-3.x.x sera créé à partir de l'archive. Par exemple : C:\apache-maven-3.x.x

Etape 5 : Définir les variables d'environnement Maven

- Ajoutez M2_HOME, M2, MAVEN_OPTS aux variables d'environnement.
 - M2_HOME → C:\Program Files (x86)\apache-maven-3.5.3
 - M2 → %M2_HOME%\bin
 - MAVEN_OPTS → -Xms256m -Xmx512m

Etape 6 : Ajouter l'emplacement du répertoire bin du Maven au chemin du système

- Ajoutez la variable M2 (%M2%) au à la variable PATH

Etape 7 : Vérifier l'installation

```
C:\Users\hp>mvn --version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\apache-maven-3.6.3-bin\apache-maven-3.6.3\bin\..
Java version: 11.0.5, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-11.0.5
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

4. Mon premier projet Maven

On veut créer notre premier projet Maven. Pour cela, nous allons utiliser le mécanisme d'Archetype de Maven. Un Archetype est défini comme un modèle à partir duquel notre projet sera créé. Dans Maven, un Archetype pourrait être combiné avec une entrée de l'utilisateur pour produire un projet adapté aux besoins de l'utilisateur.

QUALITE DE CODE AVEC SONARQUBE

Pour créer un nouveau projet Maven, créer un dossier *maven-lab* et exécuter ce qui suit à partir de la ligne de commande :

```
$ mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.mycompany.app -DartifactId=training-app
```

```
C:\Users\hp\maven-lab>mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.mycompany.app -DartifactId=training-app
[INFO] Scanning for projects...
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
```

```
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Users\hp\maven-lab
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: training-app
[INFO] Parameter: packageName, Value: com.mycompany.app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\hp\maven-lab\training-app
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 03:00 min
[INFO] Finished at: 2019-12-02T21:47:34+01:00
[INFO] -----
```

Une fois que vous avez exécuté cette commande, vous remarquerez que certaines choses se sont produites. Tout d'abord, vous remarquerez qu'un répertoire nommé *training-app* a été créé pour le nouveau projet, et ce répertoire contient un fichier nommé *pom.xml*. La structure du projet devrait ressembler à ceci :

Comme vous pouvez le voir, le projet créé à partir de l'Archetype a un dossier *main* pour les sources de votre application et un dossier *test* pour vos sources de test. C'est la disposition standard pour les projets Maven (les sources d'application résident dans *\${basedir}/src/main/java* et les sources de test résident dans *\${basedir}/src/test/java*, où *\${basedir}* représente le répertoire contenant le fichier *pom.xml*).

The screenshot shows an IDE with two panels. The left panel, titled 'FOLDERS', displays the project structure:

- training-app
 - src
 - main
 - java
 - com
 - mycompany
 - app
 - App.java
 - test
 - java
 - com
 - mycompany
 - app
 - AppTest.java

The right panel shows the content of the *pom.xml* file:

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.mycompany.app</groupId>
6   <artifactId>training-app</artifactId>
7   <packaging>jar</packaging>
8   <version>1.0-SNAPSHOT</version>
9   <name>training-app</name>
10  <url>http://maven.apache.org</url>
11  <dependencies>
12    <dependency>
13      <groupId>junit</groupId>
14      <artifactId>junit</artifactId>
15      <version>3.8.1</version>
16      <scope>test</scope>
17    </dependency>
18  </dependencies>
19 </project>
```

Aussi, le projet créé possède un POM. Le fichier **pom.xml** contient le modèle d'objet de projet (POM Project Object Model) pour ce projet. Maven est centré autour de la notion de projet, et le POM présente l'unité de travail de base. En bref, le POM contient toutes les informations importantes sur votre projet.

Le fichier **pom.xml** s'agit d'un POM très simple, et qui affiche toujours les éléments clés de chaque POM. Parcourons chacun d'entre eux pour vous familiariser avec les éléments essentiels de POM:

Eléments POM	Description
project	C'est l'élément de premier niveau dans tous les fichiers pom.xml Maven.
modelVersion	Cet élément indique quelle version du modèle d'objet utilise ce POM. La version du modèle elle-même change très rarement (4.0.0 est la version actuelle)
groupId	Cet élément indique l'identifiant unique de l'organisation ou du groupe qui a créé le projet. Le groupId est l'un des identifiants de clé d'un projet et est généralement basé sur le nom de domaine complet de votre organisation. Par exemple org.apache.maven.plugins est le groupId désigné pour tous les plugins Maven.
artifactId	Cet élément indique le nom de base unique de l'artefact primaire généré par ce projet. L'artefact principal d'un projet est généralement un fichier JAR. Les artefacts secondaires tels que les bundles sources utilisent également l'artifactId comme partie de leur nom final. Un artefact typique produit par Maven aurait la forme <artifactId> - <version>. <Extension> (par exemple, trainingapp-1.0.jar).
packaging	Cet élément indique le type de package à utiliser par cet artefact (par exemple, JAR, WAR, EAR, etc.). Cela ne signifie pas seulement que l'artefact produit est JAR, WAR ou EAR mais peut également indiquer un cycle de vie spécifique à utiliser dans le cadre du processus de construction. La valeur par défaut de l'élément de packaging est JAR. Vous n'avez donc pas besoin de spécifier cela pour la plupart des projets.
version	Cet élément indique la version de l'artefact généré par le projet. La gestion des versions dans Maven est basée sur le concept de SNAPSHOT. Dans une version, un SNAPSHOT indique qu'un projet est en cours de développement.
name	Cet élément indique le nom d'affichage utilisé pour le projet. Ceci est souvent utilisé dans la documentation générée par Maven.
url	Cet élément indique où le site du projet peut être trouvé. Ceci est souvent utilisé dans la documentation générée par Maven.
description	Cet élément fournit une description de base de votre projet. Ceci est souvent utilisé dans la documentation générée par Maven.

5. Cycle de vie Maven (Build Life Cycle)

1. Cycle de vie (Définition)

Maven est basé sur le concept de base d'un cycle de vie de construction de projets. Cela signifie que le processus de construction, de distribution et de déploiement d'un projet (artefact particulier) est clairement défini.

La construction d'un projet est définie par un ensemble de commandes, et le POM s'assurera d'obtenir les résultats souhaités.

Il existe **trois cycles de vie** de construction intégrés : *default*, *clean* et *site*.

- Le cycle de vie *default* (*Build*) gère le déploiement de votre projet
- Le cycle de vie *clean* gère le nettoyage du projet
- Le cycle de vie du *site* gère la création de la documentation du site de votre projet.

2. Les Phases

Un cycle de vie de construction (Build Life Cycle) est **une séquence de phases** (appelées aussi stages) qui définit l'ordre dans lequel les **objectifs** (*goals*) qui doivent être exécutés. Ici, la phase représente une étape du cycle de vie. À titre d'exemple, un cycle de vie Maven Build typique (càd *default*) comprend la séquence de phases suivante :

- **Validate** : Valide que le projet est correctement défini
- **Compile** : Compile les sources
- **Test** : Lance les tests unitaires
- **Package** : Prépare la distribution du projet. (Archives Jar, War, Ear...)
- **integration-test** : Lance les tests d'intégration
- **verify** : Lance des tests de validation du package créé.
- **Install** : Installe-le package **en local** sur la machine pour pouvoir être réutilisé comme dépendance.
- **Deploy** : Déploie-le package **sur un serveur** pour qu'il puisse être **réutilisé** par tout le monde.

3. Les Objectifs (Goals)

On fournit à Maven une liste de goals à exécuter. Un goal est **une tâche** précise que Maven est en mesure de réaliser à partir des informations qu'il pourra trouver dans le fichier pom.xml.

- Tous les goals se trouvent dans **des plugins** Maven.
- Pour exécuter un goal, Maven va donc commencer par résoudre le nom du goal pour en déduire le plugin dans lequel il se trouve et le télécharger.

- o Commande : `mvn <nom du plugin>:<nom du goal>` ou bien directement `mvn <nom du goal>`

4. Les plug-ins Maven

Maven est en fait un framework d'exécution de plugin où chaque tâche est en fait effectuée par des plugins. Les Plugins Maven sont généralement utilisés pour :

- **Jar** : créer un fichier jar depuis le projet
- **War** : créer un fichier war depuis le projet
- **Compiler** : compiler les fichiers JAVA
- **surefire** : lancer des tests unitaires avec JUnit et créer les rapports de test
- **javadoc** : créer la documentation du projet

6. Manipulation des phase Maven

Ce que nous avons appris dans la section précédente est comment créer une application Java avec Maven et la notion du cycle de vie. Nous verrons maintenant comment lancer les différentes phases d'un cycle de vie afin de tester notre application créée dans la section précédente.

6.1. Phase Compile (Build)

Lancer la commande **mvn** avec l'objectif **compile**.

```
[INFO] -----< com.mycompany.app:training-app >-----
[INFO] Building training-app 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ training-app ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\hp\maven-lab\training-app\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ training-app ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\hp\maven-lab\training-app\target\classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.633 s
[INFO] Finished at: 2019-12-02T22:35:38+01:00
[INFO] -----
```

- o Les classes compilées ont été placées dans `${basedir}/target/classes`, ce qui est une autre convention standard de Maven.
- o On remarque que Maven a exécuté dans l'ordre la phase de validation (récupération des ressources et téléchargement des plugins nécessaires) et de compilation sans passer automatiquement aux phases suivantes (*test*, *package*, *install*, *deploy*, etc.) du cycle de vie.

On veut maintenant tester le fichier JAVA compilé :

```
C:\Maven\training-app\target\classes>java com.mycompany.app.App
Hello World!
C:\Maven\training-app\target\classes>
```

6.2. Phase Test (Tests Unitaires)

Dans le fichier pom.xml, Maven a déjà ajouté le plugin *Junit* comme outil de test unitaire. Dans notre projet, Maven ajoute par défaut un fichier source *App.java* et un fichier de test *AppTest.java*, comme indiqué dans la section 4.

On vous demande maintenant de lancer la commande **mvn test** pour lancer les tests unitaires.

```
-----
T E S T S
-----
Running com.mycompany.app.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.009 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 17.528 s
[INFO] Finished at: 2019-12-02T22:41:59+01:00
[INFO] -----
```

A partir des traces d'exécution, quelles étapes ont été exécutées par Maven ?

- Maven télécharge plus de dépendances cette fois-ci. Ce sont les dépendances et les plugins nécessaires à l'exécution des tests (il a déjà les dépendances dont il a besoin pour la compilation et ne les téléchargera plus).
- Avant de compiler et d'exécuter les tests, Maven compile le code principal (pas de changement ici).
- Le rapport de Test reports est disponible dans le dossier \${basedir}/target/surefire-reports
- Les fichiers de test compilés se trouvent dans le dossier \${basedir}/target/test-classes
- Pour compiler uniquement les fichiers test unitaire et sans les exécuter, on lance **mvn test-compile**

6.3. Phase Package (génération du fichier JAR)

Dans le fichier POM pom.xml, le packaging par défaut est configuré à jar. C'est ainsi que Maven sait produire un fichier JAR à partir de la commande ***mvn package***.

On vous demande maintenant de la lancer la commande ***mvn package*** pour créer une archive jar du notre projet.

```
[INFO] Building jar: C:\Users\hp\maven-lab\training-app\target\training-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 47.764 s
[INFO] Finished at: 2019-12-02T22:44:15+01:00
[INFO] -----
```

- Le fichier jar est généré dans le répertoire `${basedir}/target`
- On peut mettre le fichier jar dans un repository local (dans `${user.home}/.m2/repository`) ou bien dans un serveur distant.

Essayer d'exécuter le fichier jar avec la commande suivante :

java -jar target\training-app-1.0-SNAPSHOT.jar

```
C:\Users\hp\maven-lab\training-app>java -jar target\training-app-1.0-SNAPSHOT.jar
no main manifest attribute, in target\training-app-1.0-SNAPSHOT.jar
```

On remarque ici que notre fichier jar n'est pas exécutable. En effet, le fait de préciser `<packaging>jar</packaging>` n'indique pas que Maven va créer un jar exécutable avec toutes ses dépendances. Ce jar ne contiendra que les classes compilées de notre projet et ne sera pas exécutable. Pour créer un fichier exécutable, on doit ajouter le plugin ***maven-jar-plugin*** à notre projet.

Ajouter la section suivante dans votre fichier POM :

```
<build>

  <plugins>

    <plugin>

      <!-- Build an executable JAR -->

      <groupId>org.apache.maven.plugins</groupId>

      <artifactId>maven-jar-plugin</artifactId>

      <version>3.0.2</version>

      <configuration>

        <archive>

          <manifest>

            <addClasspath>true</addClasspath>

            <classpathPrefix>lib/</classpathPrefix>

            <mainClass>com.mycompany.app.App</mainClass>

          </manifest>

        </archive>

      </configuration>

    </plugin>

  </plugins>

</build>
```

On indique donc ici la classe main (<mainClass>), que notre jar sera créé lors de la phase « packaging » de Maven

Exécuter de nouveau la commande ***mvn package*** et lancer le fichier jar.

```
[INFO] Building jar: C:\Users\hp\maven-lab\training-app\target\training-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:04 min
[INFO] Finished at: 2019-12-02T22:55:00+01:00
[INFO] -----

C:\Users\hp\maven-lab\training-app>java -jar target\training-app-1.0-SNAPSHOT.jar
Hello World!
```

6.4. Phase Install

Dans un environnement de développement, on utilise la commande ***mvn install*** pour générer et installer les artefacts **dans le référentiel (repository) local**.

- Cette commande exécute chaque phase du cycle de vie par défaut dans l'ordre (validation, compilation, package, etc.) avant d'exécuter l'installation.
- On aura besoin uniquement d'appeler la dernière phase de construction à exécuter, dans ce cas, lancez :

```
[INFO] Installing C:\Users\hp\maven-lab\training-app\target\training-app-1.0-SNAPSHOT.jar to C:\Users\hp\.m2\repository\com\mycompany\app\training-app\1.0-SNAPSHOT\training-app-1.0-SNAPSHOT.jar
[INFO] Installing C:\Users\hp\maven-lab\training-app\pom.xml to C:\Users\hp\.m2\repository\com\mycompany\app\training-app\1.0-SNAPSHOT\training-app-1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:00 min
[INFO] Finished at: 2019-12-02T23:01:12+01:00
[INFO] -----
```

Vérifier que le jar est bien installé dans votre local repository (`${user.home}/.m2/repository`).

7. Gestion de dépendances du projet

Pour en revenir aux principes de Maven :

- Le répertoire *src* ne doit contenir que des fichiers sources apportés au projet
- Les bibliothèques externes (appelées dépendances) utilisées par le projet ne doivent être que des liens vers d'autres artefacts Maven et surtout pas copiées dans le répertoire *src* du projet.
- Un grand nombre d'artefacts jars est disponible sur les entrepôts officiels de Maven : <http://www.mvnrepository.com/>

Maven propose de définir toutes les dépendances par configuration dans le fichier *pom.xml*. C'est ensuite le plugin Maven de gestion de dépendances qui ira télécharger sur les repositories distants les fichiers jar indiqués comme dépendances, s'ils ne se trouvent pas dans le repository local.

On veut maintenant modifier notre projet afin de gérer une liste des sessions de formations. Les informations qu'on voudrait afficher sont stockées dans une base de données MYSQL (dans la table *session*). Pour cela, faites les modifications suivantes :

- Modifier la classe *App.java* (le remplacer par le contenu du fichier *Training_Data_Service.java*)
- Installer un serveur XAMP, accéder au portail PhpMyAdmin (<http://localhost/phpmyadmin>) et créer une nouvelle base données *training*
- Importer dans PhpMyAdmin le fichier *training.sql* qui permet de créer la table *Session* (contient deux enregistrements par défaut).
- Ajouter la dépendance *mysql-connector-java* dans le fichier *pom.xml*

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.35</version>
</dependency>
```

- Compiler votre projet et tester le fichier jar *mvn clean install && java -jar target\training-app-1.0-SNAPSHOT.jar*
- Faire les modifications nécessaires dans votre fichier POM afin de rendre le plugin *mysql-connector-java* disponible à l'exécution du fichier jar. Assembler le fichier jar de la dépendance avec le fichier jar final de l'application en utilisant le plugin ***maven-assembly-plugin***

```
<plugin>

  <artifactId>maven-assembly-plugin</artifactId>

  <configuration>

    <archive>

      <manifest>

        <mainClass>com.mycompany.app.App</mainClass>

      </manifest>

    </archive>

    <descriptorRefs>

      <descriptorRef>jar-with-dependencies</descriptorRef>

    </descriptorRefs>

  </configuration>

  <executions>

    <execution>

      <id>make-assembly</id> <!-- this is used for inheritance merges -->

      <phase>package</phase> <!-- bind to the packaging phase -->

      <goals>

        <goal>single</goal>

      </goals>

    </execution>

  </executions>

</plugin>
```

```

[INFO] Installing C:\Maven\training-app\target\training-app-1.0-SNAPSHOT.jar to
C:\Users\walid\.m2\repository\com\mycompany\app\training-app\1.0-SNAPSHOT\traini
ng-app-1.0-SNAPSHOT.jar
[INFO] Installing C:\Maven\training-app\pom.xml to C:\Users\walid\.m2\repository
\com\mycompany\app\training-app\1.0-SNAPSHOT\training-app-1.0-SNAPSHOT.pom
[INFO] Installing C:\Maven\training-app\target\training-app-1.0-SNAPSHOT-jar-wit
h-dependencies.jar to C:\Users\walid\.m2\repository\com\mycompany\app\training-a
pp\1.0-SNAPSHOT\training-app-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.722 s
[INFO] Finished at: 2018-06-21T17:50:44+02:00
[INFO] -----

C:\Maven\training-app>copy target\training-app-1.0-SNAPSHOT-jar-with-dependencie
s.jar ..
1 fichier(s) copi  (s).

C:\Maven\training-app>cd ..

C:\Maven>java -jar training-app-1.0-SNAPSHOT-jar-with-dependencies.jar
----- Connexion au serveur de donn  es MYSQL -----
Le driver JDBC pour MySQL est disponible.
Connexion    la base de donn  es      t     table avec succ  s.
----- Afficher toutes les sessions de formations -----
Formation Integration Continue, Maven, Toulouse, 2018-06-25, 10, 1
Formation Integration Continue, Jenkins, Toulouse, 2018-06-27, 10, 1

C:\Maven>

```

8. Configuration du Maven avec SonarQube

8.1. Configuration du Maven

Etape 1: Fichier Sesstings.xml (configurer le serveur Sonar)

- Dans la section <pluginGroups>, ajouter la configuration suivante :

```

<pluginGroups>

    <pluginGroup>org.sonarsource.scanner.maven</pluginGroup>

</pluginGroups>

```

- Dans la section <profiles>, ajouter la configuration suivante :

```

<profile>

    <id>sonar</id>

    <activation>

        <activeByDefault>true</activeByDefault>

    </activation>

    <properties>

        <!-- Optional URL to server. Default value is http://localhost:9000 -->

        <sonar.host.url>

```

```
http://localhost:9000  
</sonar.host.url>  
</properties>  
</profile>
```

- Dans la section < activeProfiles>, activer le profile Sonar :

```
<activeProfiles>  
  <activeProfile>sonar</activeProfile>  
</activeProfiles>
```

- Dans la section < servers>, ajouter les paramètres d'authentications au serveur Sonar :

```
<server>  
  <id>sonar</id>  
  <username>admin</username>  
  <password>admin</password>  
</server>
```

Etape 2 : Fichier pom.xml (ajouter le plugin sonar)

```
<plugin>  
  <groupId>org.sonarsource.scanner.maven</groupId>  
  <artifactId>sonar-maven-plugin</artifactId>  
  <version>3.4.0.905</version>  
</plugin>
```

8.2. Lancer le test de la qualité de code

- Lancer la commande *mvn clean verify sonar:sonar*


```
Downloaded from central: https://repo.maven.apache.org/maven2/org/sonarsource/scanner/api/sonar-scanner-api/2.10.0.1189/sonar-scanner-api-2.10.0.1189.jar (582 kB at 74 kB/s)
[INFO] User cache: C:\Users\hp\.sonar\cache
[INFO] SonarQube version: 8.0.0
[INFO] Default locale: "fr_FR", source code encoding: "UTF-8"
[INFO] Load global settings
[INFO] Load global settings (done) | time=113ms
[INFO] Server id: 86E1FA4D-AW6u2J_xPDJBDxpROQ5y
[INFO] User cache: C:\Users\hp\.sonar\cache
[INFO] Load/download plugins
[INFO] Load plugins index
[INFO] Load plugins index (done) | time=64ms
[INFO] Load/download plugins (done) | time=464ms
[INFO] Process project properties
[INFO] Process project properties (done) | time=16ms
[INFO] Execute project builders
[INFO] Execute project builders (done) | time=6ms
[INFO] Project key: com.mycompany.app:training-app
[INFO] Base dir: C:\Users\hp\maven-lab\training-app
[INFO] Working dir: C:\Users\hp\maven-lab\training-app\target\sonar
[INFO] Load project settings for component key: 'com.mycompany.app:training-app'
[INFO] Load quality profiles
[INFO] Load quality profiles (done) | time=149ms
[INFO] Load active rules
[INFO] Load active rules (done) | time=859ms
[WARNING] SCM provider autodetection failed. Please use "sonar.scm.provider" to define SCM of your project, or disable the SCM Sensor in the project settings.
```

```
[INFO] Sensor JavaXmlSensor [java] (done) | time=164ms
[INFO] 1/1 source files have been analyzed
[INFO] Sensor HTML [web]
[INFO] Sensor HTML [web] (done) | time=7ms
[INFO] Sensor XML Sensor [xml]
[INFO] 1 source files to be analyzed
[INFO] Sensor XML Sensor [xml] (done) | time=160ms
[INFO] 1/1 source files have been analyzed
[INFO] ----- Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=23ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=20ms
[INFO] SCM Publisher No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
[INFO] CPD Executor Calculating CPD for 1 file
[INFO] CPD Executor CPD calculation finished (done) | time=7ms
[INFO] Analysis report generated in 195ms, dir size=87 KB
[INFO] Analysis report compressed in 34ms, zip size=17 KB
[INFO] Analysis report uploaded in 798ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=com.mycompany.app%3Atraining-app
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AW7IuErIPDJBDxpROTpE
[INFO] Analysis total time: 7.560 s
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 51.960 s
[INFO] Finished at: 2019-12-02T23:25:51+01:00
```

- Vérifier sur le serveur <http://localhost:9000> le résultat de test du votre projet (Bugs, risques, etc.)

The screenshot shows the SonarQube web interface. The top navigation bar includes 'sonarqube', 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar is on the right. Below the navigation bar, there's a 'Perspective' dropdown set to 'Overall Status' and a 'Sort by' dropdown set to 'Name'. A search bar for projects is also present. The main content area shows the project 'training-app' with a green 'Passed' status. Below this, there are metrics: 2 Bugs (red E), 5 Vulnerabilities (yellow B), 6 Code Smells (green A), 0.0% Coverage (red circle), and 0.0% Duplications (green circle). The last analysis was on June 28, 2018, at 11:53 PM. The bottom of the interface shows '169' issues and 'XML, Java' as the analyzed languages. The bottom status bar indicates '1 of 1 shown'.

QUALITE DE CODE AVEC SONARQUBE

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration ? Search for projects, sub-projects and files...

training-app master June 28, 2018, 11:53 PM Version 1.0

Overview Issues Measures Code Activity Administration

Quality Gate **Passed**

Bugs Vulnerabilities

2 Bugs 5 Vulnerabilities

Code Smells

38min Debt 6 Code Smells

Coverage

0.0% Coverage 1 Unit Tests

About This Project

No tags

169 Lines of Code XML 87 Java 82

Project Activity

June 28, 2018 1.0 Show More

Quality Gate (Default) Sonar way

Quality Profiles (Java) Sonar way (XML) Sonar way

External Links Project's Website

training-app master June 28, 2018, 11:53 PM Version 1.0

Overview Issues Measures Code Activity Administration

My Issues All

Filters Clear All Filters

Display Mode Issues Effort

Type Bug 2 Vulnerability 5 Code Smell 6

Bulk Change

src/main/java/com/mycompany/app/App.java

Change this condition so that it does not always evaluate to "true" 7 minutes ago L26 3 cert, cwe, misra, pitfall, unused

Use try-with-resources or close this "ResultSet" in a "finally" clause. 7 minutes ago L69 cert, cwe, denial-of-service, leak

2 of 2 shown

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration ? Search for projects, sub-projects and files... A

training-app master June 28, 2018, 11:53 PM Version 1.0

Overview Issues Measures Code Activity Administration

My Issues All

Filters Clear All Filters

Display Mode Issues Effort

Type Bug 20min Vulnerability 50min Code Smell 38min

Severity Blocker 0 Critical 0 Minor 3min Major 35min

Resolution Status Creation Date

Bulk Change

src/main/java/com/mycompany/app/App.java

Remove this redundant jump. 8 minutes ago L34 clumsy, redundant

Remove this unused private "addDataToDB" method. 8 minutes ago L39 unused

Rename this local variable to match the regular expression "^[a-zA-Z0-9]*\$". 8 minutes ago L77 convention

Replace this use of System.out or System.err by a logger. 8 minutes ago L82 bad-practice, cert

%n should be used in place of \n to produce the platform-specific line separator. 8 minutes ago L82 cert, confusing

Replace this use of System.out or System.err by a logger. 8 minutes ago L94 bad-practice, cert

8.1. Mesurer la couverture des tests unitaires et importer le rapport d'analyse avec le plugin JaCoCo

8.1.1. Projet Basique

- Installer Git et Cloner le dépôt <https://github.com/SonarSource/sonar-scanning-examples>

```
hp@DESKTOP-L47VJUF MINGW64 ~/maven-lab
$ git clone https://github.com/SonarSource/sonar-scanning-examples.git
Cloning into 'sonar-scanning-examples'...
remote: Enumerating objects: 604, done.
remote: Total 604 (delta 0), reused 0 (delta 0), pack-reused 604
Receiving objects: 100% (604/604), 323.40 KiB | 177.00 KiB/s, done.
Resolving deltas: 100% (183/183), done.

hp@DESKTOP-L47VJUF MINGW64 ~/maven-lab
$ cd sonar-scanning-examples/sonarqube-scanner-maven/maven-basic/

hp@DESKTOP-L47VJUF MINGW64 ~/maven-lab/sonar-scanning-examples/sonarqube-scanner-maven/maven-basic (master)
```

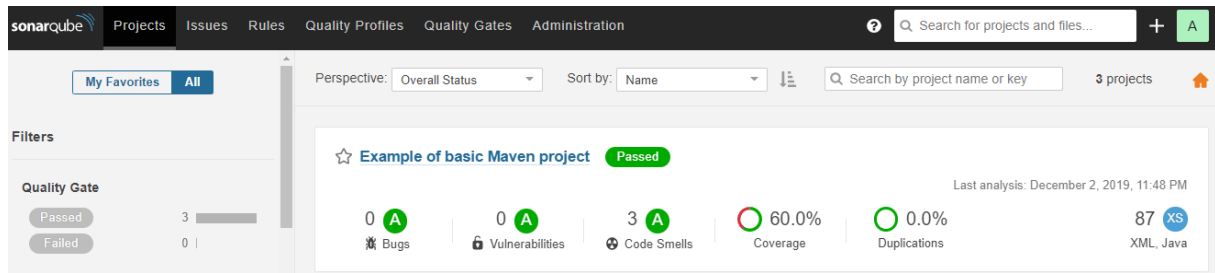
- Lancer le scan (mvn clean verify sonar:sonar) :

```
[INFO] ----- Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=1ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=10ms
[INFO] SCM Publisher SCM provider for this project is: git
[INFO] SCM Publisher 3 source files to be analyzed
[INFO] SCM Publisher 3/3 source files have been analyzed (done) | time=185ms
[INFO] CPD Executor 1 file had no CPD blocks
[INFO] CPD Executor Calculating CPD for 0 files
[INFO] CPD Executor CPD calculation finished (done) | time=0ms
[INFO] Analysis report generated in 142ms, dir size=85 KB
[INFO] Analysis report compressed in 29ms, zip size=15 KB
[INFO] Analysis report uploaded in 561ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=org.sonarqube%3Asonar-scanner-maven-basic
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AW7IzSiEPDjBDxpROTpL
[INFO] Analysis total time: 5.785 s
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 22.248 s
[INFO] Finished at: 2019-12-02T23:48:37+01:00
[INFO] -----

hp@DESKTOP-L47VJUF MINGW64 ~/maven-lab/sonar-scanning-examples/sonarqube-scanner-maven/maven-basic (master)
$ mvn clean verify sonar:sonar
```

- Visualiser le résultat sur SonarQube (on vérifie bien qu'on a 60% le taux du couverture). Ceci est expliqué par le fait que les tests unitaires dans src/Test couvre la fonction *sayHello()* uniquement :

QUALITE DE CODE AVEC SONARQUBE



The SonarQube Projects Overview page shows a list of projects. The selected project is 'Example of basic Maven project', which has a 'Passed' status. The page displays various quality metrics: 0 Bugs, 0 Vulnerabilities, 3 Code Smells, 60.0% Coverage, and 0.0% Duplications. The last analysis was performed on December 2, 2019, at 11:48 PM. The project has 87 lines of code (XS) in XML and Java.

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects and files...

Perspective: Overall Status Sort by: Name

Search by project name or key 3 projects

My Favorites All

Filters

Quality Gate

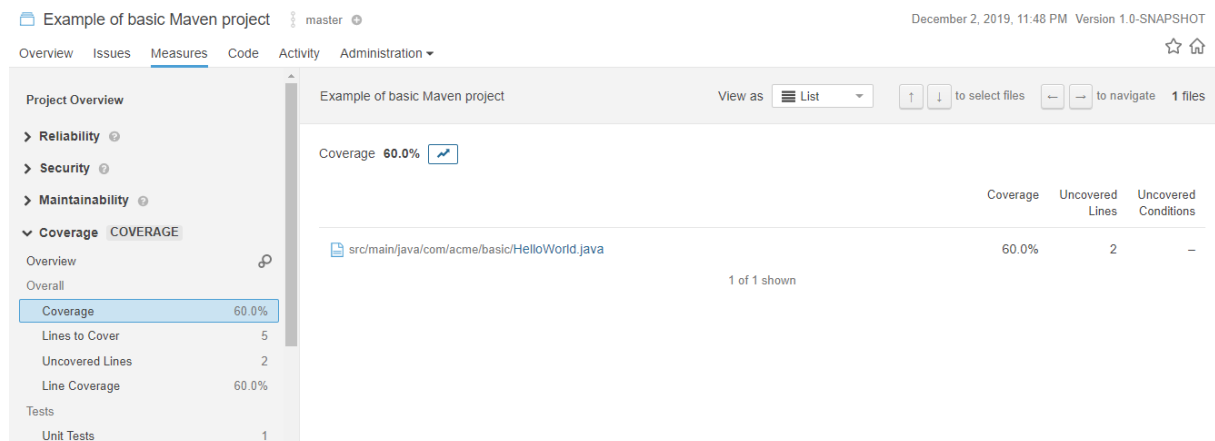
Passed 3 Failed 0

Example of basic Maven project Passed

Last analysis: December 2, 2019, 11:48 PM

0 Bugs 0 Vulnerabilities 3 Code Smells 60.0% Coverage 0.0% Duplications 87 XS XML, Java

```
1 package com.acme.basic;
2
3 public class HelloWorld {
4
5     void sayHello() {
6         System.out.println("Hello World!");
7     }
8
9     void notCovered() {
10        System.out.println("This method is not covered by unit tests");
11    }
12
13 }
```



The Measures page for the 'Example of basic Maven project' shows the 'Coverage' measure selected. The overall coverage is 60.0%. The table below shows the coverage for the file 'src/main/java/com/acme/basic/HelloWorld.java'.

Example of basic Maven project master December 2, 2019, 11:48 PM Version 1.0-SNAPSHOT

Overview Issues Measures Code Activity Administration

Project Overview

Reliability Security Maintainability Coverage COVERAGE

Overview Overall Coverage 60.0% Lines to Cover 5 Uncovered Lines 2 Line Coverage 60.0% Tests Unit Tests 1

Example of basic Maven project View as List to select files to navigate 1 files

Coverage 60.0%

	Coverage	Uncovered Lines	Uncovered Conditions
src/main/java/com/acme/basic/HelloWorld.java	60.0%	2	-

1 of 1 shown

8.1.2. Projet Multi-module

- Cloner le dépôt <https://github.com/SonarSource/sonar-scanning-examples>

```
$ mvn clean verify sonar:sonar
[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
[INFO] Example of multi-module Maven project [pom]
[INFO] Module 1 [jar]
[INFO] Module 2 [jar]
[INFO] Tests [jar]
[INFO]
[INFO] -----< org.sonarqube:sonarscanner-maven-aggregate >-----
[INFO] Building Example of multi-module Maven project 1.0-SNAPSHOT [1/4]
[INFO] -----[ pom ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ sonarscanner-maven-aggregate ---
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.4:prepare-agent (prepare-agent) @ sonarscanner-maven-aggregate ---

[INFO] SCM Publisher 9/9 source files have been analyzed (done) | time=304ms
[INFO] CPD Executor 2 files had no CPD blocks
[INFO] CPD Executor Calculating CPD for 0 files
[INFO] CPD Executor CPD calculation finished (done) | time=0ms
[INFO] Analysis report generated in 295ms, dir size=104 KB
[INFO] Analysis report compressed in 52ms, zip size=25 KB
[INFO] Analysis report uploaded in 537ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=org.sonarqube%3Asonarscanner-maven-aggregate
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AW7I45XFPDJBdXpR0TpZ
[INFO] Analysis total time: 14.364 s
[INFO] -----
[INFO] Reactor Summary for Example of multi-module Maven project 1.0-SNAPSHOT:
[INFO]
[INFO] Example of multi-module Maven project ..... SUCCESS [ 19.637 s]
[INFO] Module 1 ..... SUCCESS [ 5.393 s]
[INFO] Module 2 ..... SUCCESS [ 2.495 s]
[INFO] Tests ..... SUCCESS [ 2.362 s]
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 34.043 s
[INFO] Finished at: 2019-12-03T00:13:08+01:00
[INFO] -----
```

