
Maven™

Prérequis

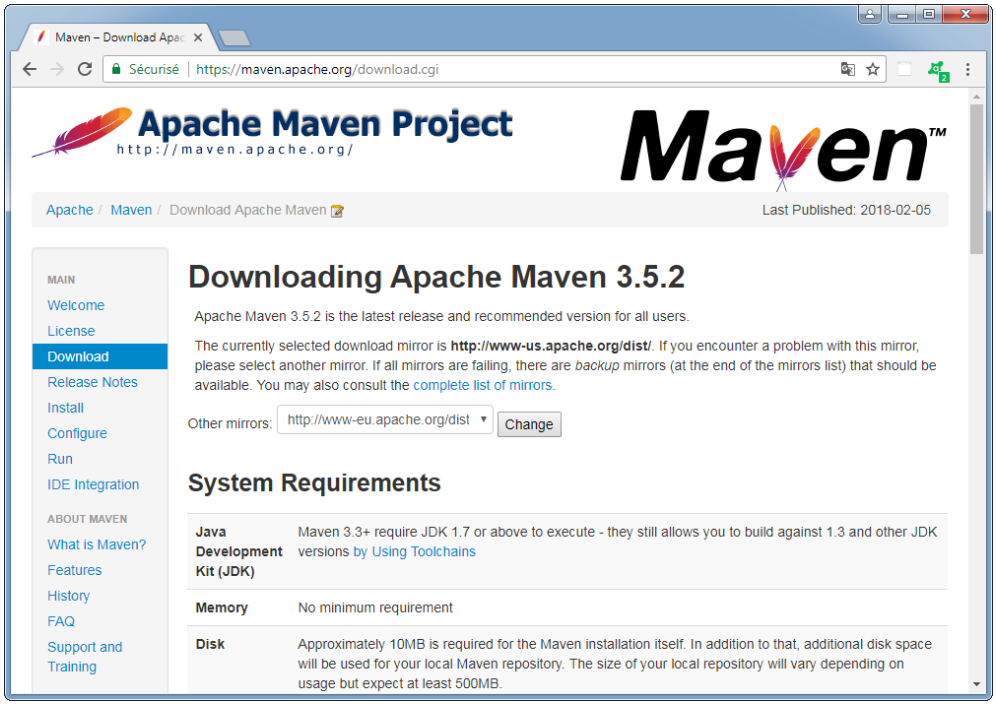
- Java : JDK 1.7 ou plus récent.
- RAM : aucun prérequis.
    - 1 Go de RAM libre est conseillé pour des projets de taille moyenne.
- Espace disque : aucun prérequis.
- Système d'exploitation : Aucun prérequis.
    - Disponible sous Windows, Linux, MAC, etc…

Installation de l'outil Maven

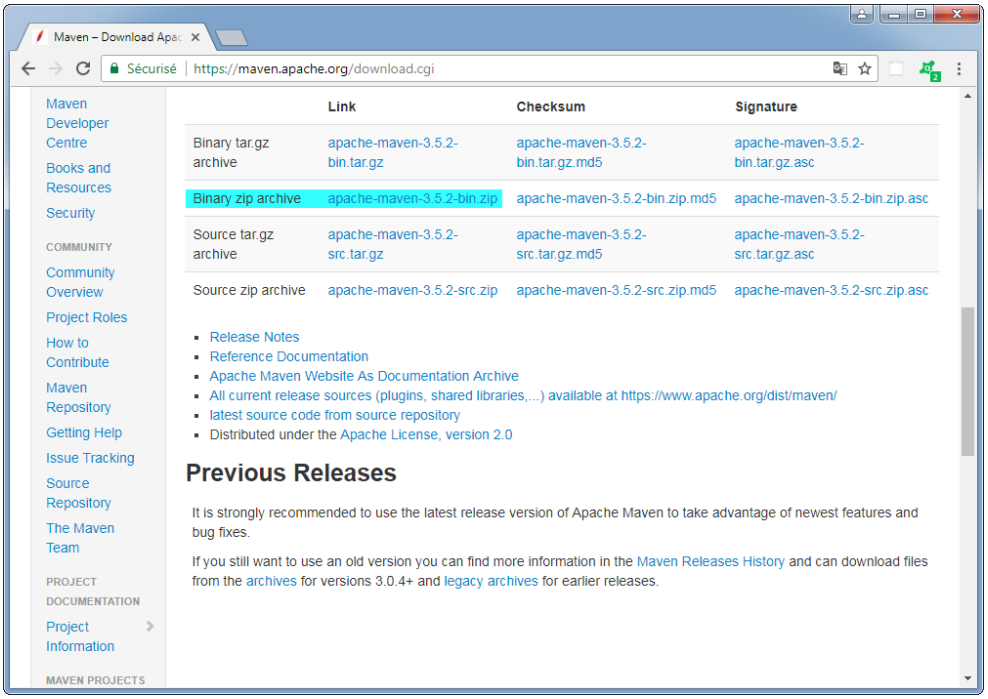
I. Sous Windows

Etape 1: Téléchargement de Maven

- Télécharger Maven depuis son site web officiel https://maven.apache.org/download.cgi/



- Choisir une version adaptée au système d'exploitation hôte.



Etape 2: Installation de Maven

- Ouvrez l'invite de commande. À partir de l'invite de commande, accédez au répertoire où le fichier apache-maven-x.x.x-bin.zip est enregistré et décompressez l'archive sous le chemin :

```
C:\integ_continue\maven\apache-maven-x.x.x
```

- Exécutez les commandes suivantes pour paramétrer les variables d'environnement requises pour le fonctionnement de Maven:
  - Définir la variable d'environnement JAVA_HOME :

```
SET JAVA_HOME="C:\Program Files\Java\jdk1.8.0_xxx"
```

  - Définir la variable d'environnement M2_HOME :

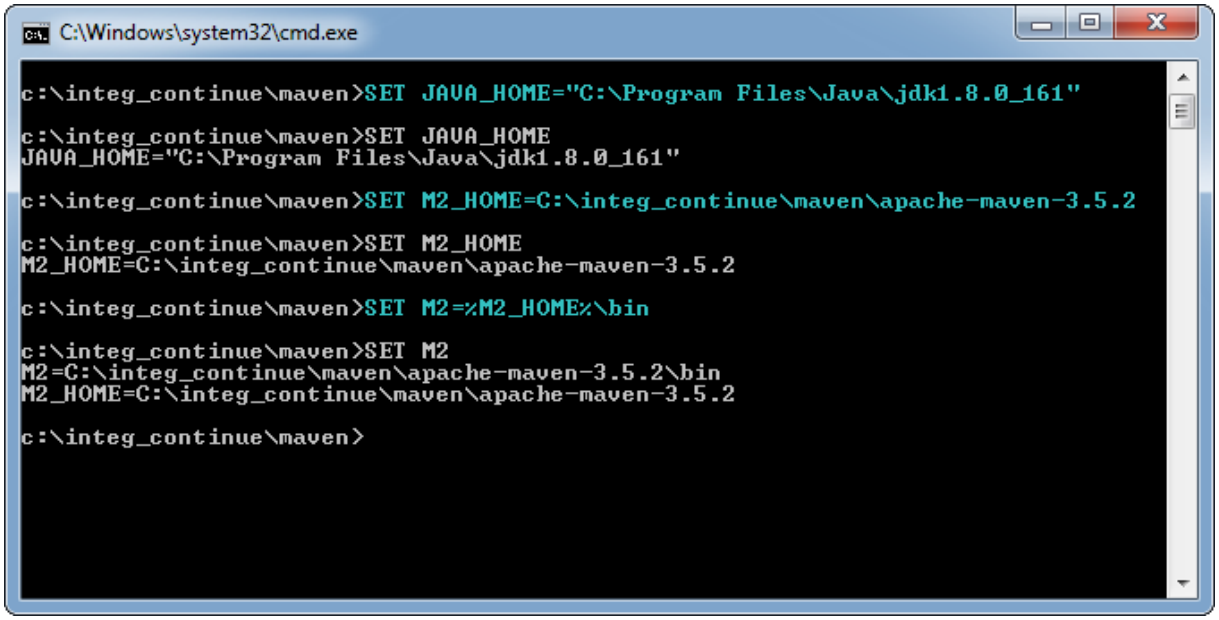
```
SET M2_HOME=C:\integ_continue\maven\apache-maven-x.x.x
```

  - Définir la variable d'environnement M2 :

```
SET M2=%M2_HOME%\bin
```

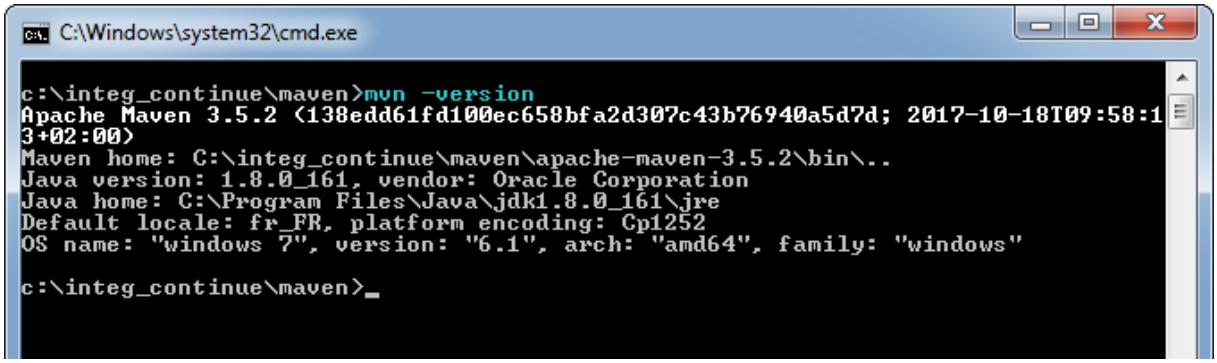
  - Ajouter les commandes Maven dans le PATH du système d'exploitation :

```
SET PATH=%PATH%;%M2%
```



  - Testez l'installation de Maven en exécutant la commande:

```
mvn -version
```



  - Exécutez la commande suivante pour afficher l'aide Maven:

```
mvn -help
```

II. Sous Linux (Centos 7)

Etape 1 : Installation du Java

1. Nous utiliserons open java pour notre démo, Obtenez la dernière version de http://openjdk.java.net/install/

```
yum install java-1.8*
#yum -y install java-1.8.0-openjdk
```

2. Configuration de la variable d'environnement JAVA_HOME

   1. Déterminez le bon emplacement de la version JAVA8.

```
find /usr/lib/jvm/java-1.8* | head -n 3
```

   2. Créez un fichier appelé java.sh dans le répertoire /etc/profile.d/

```
vi /etc/profile.d/java.sh
```

   3. Ajoutez le contenu suivant

```
#!/bin/bash
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.302.b08-0.el7_9.x86_64
export PATH=$PATH:$JAVA_HOME
```

NB: vous devez changer le chemin du java /usr/lib/jvm/java-1.8.0-**x86_64. Par le chemin généré dans la commande précédente.***

   4. Enregistrez et fermez le fichier. Rendez-le exécutable à l'aide de la commande suivante.

```
chmod +x /etc/profile.d/java.sh
```

   5. Ensuite, définissez les variables d'environnement de manière permanente en exécutant la commande suivante :

```
source /etc/profile.d/java.sh
```

   6. Maintenant, vérifiez la version de fourmi en utilisant la commande :

```
java -version
```

Etape 2 : Installation du Maven

- Télécharger Maven depuis son site web officiel https://maven.apache.org/download.cgi/
- Choisir et télécharger une version adaptée au système d'exploitation hôte.

```
# wget https://downloads.apache.org/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz
```

- Extraire l'archive de distribution dans le dossier /opt

```
tar xzvf apache-maven-* -C /opt
```

- Créez un fichier appelé maven.sh dans le répertoire /etc/profile.d/

```
vi /etc/profile.d/maven.sh
```

- Ajoutez le contenu suivant

```
#!/bin/bash

export M2_HOME=/opt/apache-maven-3.6.3
export M2=$M2_HOME/bin
export PATH=$PATH:$M2
```

NB: vous devez changer le chemin de l'outil maven par celui installé dans votre instance.

- Enregistrez et fermez le fichier. Rendez-le exécutable à l'aide de la commande suivante.

```
chmod +x /etc/profile.d/maven.sh
```

- Ensuite, définissez les variables d'environnement de manière permanente en exécutant la commande suivante :

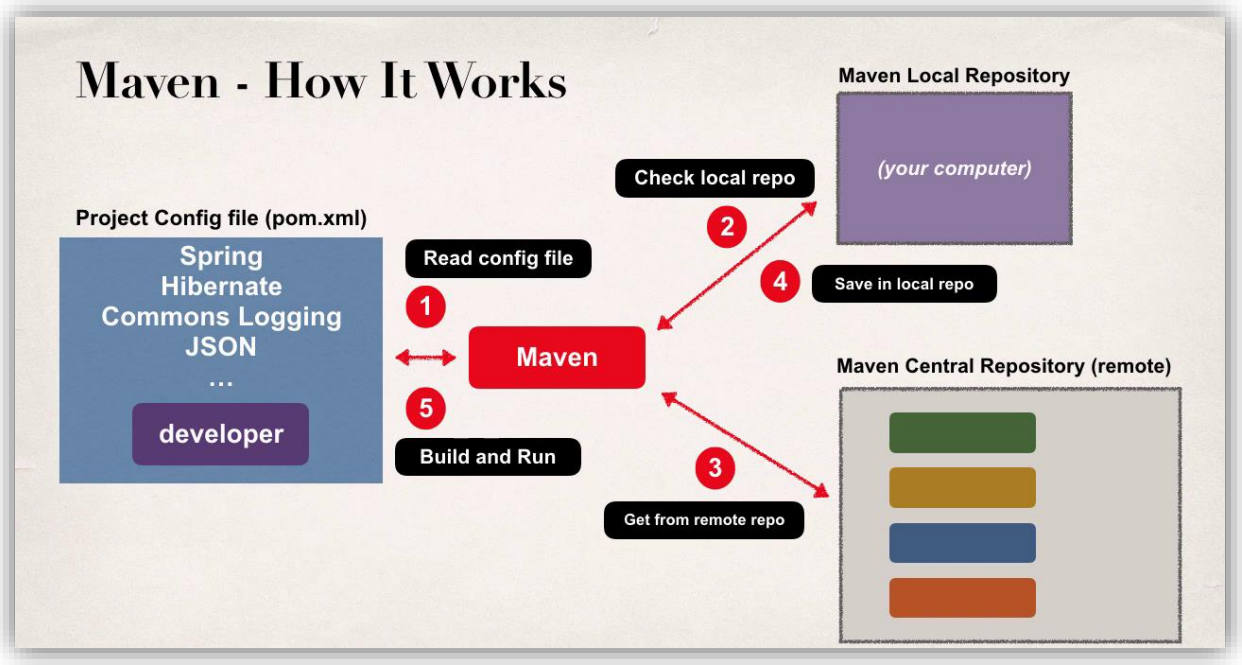
```
source /etc/profile.d/maven.sh
```

- Maintenant, vérifiez la version de fourni en utilisant la commande :

```
mvn -version
```

III. Génération d'un projet Maven

Maven repository



Project Object Model « pom.xml »

POM signifie "Project Object Model". Il s'agit d'une représentation XML d'un projet Maven contenu dans un fichier nommé pom.xml.

Exemple

```
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<packaging>jar</packaging>

<groupId>org.codehaus.mojo</groupId>
<artifactId>my-project</artifactId>
<version>1.0</version>

<!-- Build Settings -->
  <build>...</build>
  <reporting>...</reporting>
</project>
```

Générer une Application java de type Maven

- Générer l'architecture du projet

```
mvn archetype:generate
```

- Choisir le code par défaut (1723)

```
Choose a number or apply filter (format: [groupId:]artifactId, case
sensitive contains): 1723:
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
7: 1.3
8: 1.4
Choose a number: 8:
Define value for property 'groupId': org.gk.cusrsusdevops
Define value for property 'artifactId': AppJavaMaven
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' org.gk.cusrsusdevops: :
Confirm properties configuration:
groupId: org.gk.cusrsusdevops
artifactId: AppJavaMaven
version: 1.0-SNAPSHOT
package: org.gk.cusrsusdevops
 Y: : y
```

- Vérifier l'architecture du projet

```
[root@worker-node1 ~]# tree AppJavaMaven/
AppJavaMaven/
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── org
    │           └── gk
    │               └── cusrsusdevops
    │                   └── App.java
    └── test
        └── java
            └── org
                └── gk
                    └── cusrsusdevops
                        └── AppTest.java
```

- Compiler le projet

```
[root@worker-node1 AppJavaMaven]# mvn compile
```

- Créer un package

```
[root@worker-node1 AppJavaMaven]# mvn package
```

- Vérifier le résultat

```
[root@worker-node1 AppJavaMaven]# tree
.
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       └── org
│   │           └── gk
│   │               └── cusrsusdevops
│   │                   └── App.java
│   └── test
│       └── java
│           └── org
│               └── gk
│                   └── cusrsusdevops
│                       └── AppTest.java
└── target
    ├── AppJavaMaven-1.0-SNAPSHOT.jar
    ├── classes
    │   └── org
    │       └── gk
    │           └── cusrsusdevops
    │               └── App.class
    ├── generated-sources
    │   └── annotations
    ├── generated-test-sources
    │   └── test-annotations
    ├── maven-archiver
    │   └── pom.properties
    ├── maven-status
    │   └── maven-compiler-plugin
    │       ├── compile
    │       │   └── default-compile
    │       │       ├── createdFiles.lst
    │       │       └── inputFiles.lst
    │       └── testCompile
    │           └── default-testCompile
    │               ├── createdFiles.lst
    │               └── inputFiles.lst
    ├── surefire-reports
    │   ├── org.gk.cusrsusdevops.AppTest.txt
    │   └── TEST-org.gk.cusrsusdevops.AppTest.xml
    └── test-classes
```

```
        └── org
            └── gk
                └── cusrsusdevops
                    └── AppTest.class

32 directories, 13 files
```

- Tester le projet

```
[root@worker-node1 AppJavaMaven]# java -cp target/AppJavaMaven-1.0-
SNAPSHOT.jar org.gk.cusrsusdevops.App
Hello World!
```

- Créer la documentation de l'application

```
[root@worker-node1 AppJavaMaven]# mvn site
[root@worker-node1 AppJavaMaven]# tree target/site/
target/site/
├── css
│   ├── maven-base.css
│   ├── maven-theme.css
│   ├── print.css
│   └── site.css
├── dependencies.html
├── dependency-info.html
├── images
│   ├── close.gif
│   ├── collapsed.gif
│   ├── expanded.gif
│   ├── external.png
│   ├── icon_error_sml.gif
│   ├── icon_info_sml.gif
│   ├── icon_success_sml.gif
│   ├── icon_warning_sml.gif
│   ├── logos
│   │   ├── build-by-maven-black.png
│   │   ├── build-by-maven-white.png
│   │   └── maven-feather.png
│   └── newwindow.png
├── index.html
├── plugin-management.html
├── plugins.html
├── project-info.html
└── summary.html
```

- Nettoyer le projet

```
[root@worker-node1 AppJavaMaven]# mvn clean
```

- Testez la construction du projet `AppJavaMaven` en exécutant la commande

```
[root@worker-node1 AppJavaMaven]# mvn clean install
```

Gérer les dépendances d'un projet automatiquement avec Maven

- Modifier le code source de classe java principale **App.java**

```
package org.gk.cusrsusdevops;
import org.slf4j.*;
public class App{
    public static void main( String[] args ){
        System.out.println( "Hello World!" );
        Logger  logger = LoggerFactory.getLogger(App.class);
        logger.info("Hello world");
    }
}
```

- Compiler le projet et analyser le résultat
- Ajouter les dépendances du package slf4j (https://mvnrepository.com/)

```
...
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.7.30</version>
    </dependency>
...
```

- Compiler à nouveau le projet et analyser le résultat

Gérer les plugins d'un projet automatiquement avec Maven

Le plugin « maven-compiler-plugin » est ajouté automatiquement lors de la génération du projet et utiliser par Maven pour compiler les projets.

Si n'est pas spécifier, le projet sera compilé par la dernière version disponible dans le serveur.

- Modifier la configuration du Plugin

```
...
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
          <configuration>
                <source>1.4</source>
                <target>1.4</target>
          </configuration>
        </plugin>
...
```

- Ajouter une classe de test ListeVoitures.java

```
package org.gk.cusrsusdevops;
import java.util.*;
public class ListeVoitures
{
    public static void main( String[] args )
    {
        List<String> listVoitures = new ArrayList<String>();
        listVoitures.add("Peugeot");
        listVoitures.add("Citroen");
    }
}
```

- Compiler le projet et analyser le résultat
- Modifier la configuration du Plugin pour le changer à la version 1.5
- Compiler à nouveau le projet et analyser le résultat
- Supprimer le plugin «maven-compiler-plugin » puis Compiler à nouveau le projet et analyser le résultat

Générer une Application Web

- Générer l'architecture du projet webapp

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes
-DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.4
```

- Vérifier l'architecture du projet

```
[root@worker-node1 ~]# tree webapp/
webapp/
├── pom.xml
└── src
    └── main
        └── webapp
            ├── index.jsp
            └── WEB-INF
                └── web.xml
4 directories, 3 files
```

- Ajout du plugin Jetty pour tester rapidement l'application web créée. Modifier le fichier pom.xml et ajouter la définition du plugin

```
...
<plugin>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>maven-jetty-plugin</artifactId>
    <version>6.1.26</version>
</plugin>
...
```

- Exécuter le plugin Jetty

```
mvn jetty:run
```

- Tester l'application à partir d'un navigateur web sur le port 8080
- Modifier le code de la page index.jsp et vérifier les mises à jour