# TP1: JENKINS

### 1- Installation de Jenkins

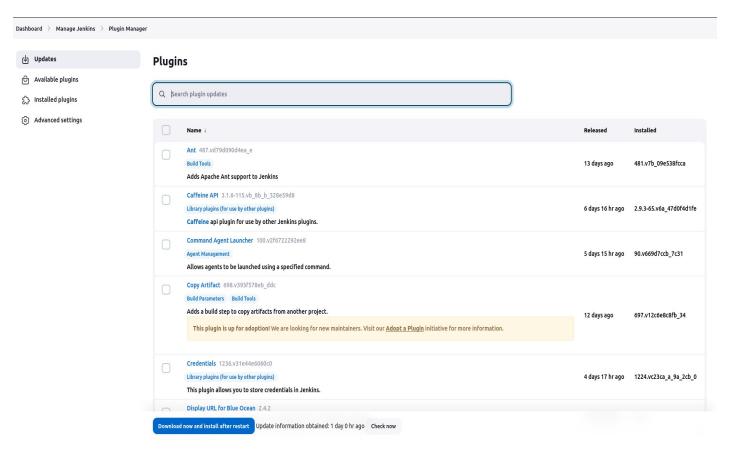
Suivre le tutorial : Installing and Running Jenkins as a Standalone Application

### 2- Installation de plugins

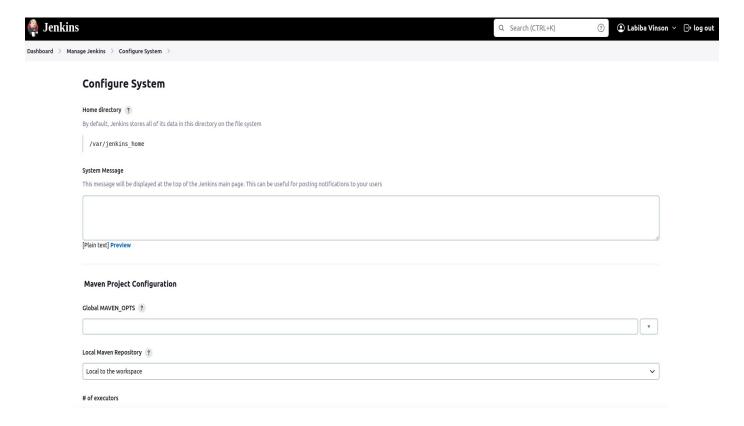
Une grande richesse de Jenkins est son système de plugins, ainsi que le grand nombre de plugins existant.

Pour accéder aux écrans de gestion des devrait vous mener à <a href="http://localhost:8080/pluginManager/">http://localhost:8080/pluginManager/</a>.

Nous allons choisir les plugins dont nous avons besoin pour faire fonctionner nos projets notamment : maven, Unit Test, Docker, ...



Maintenant que notre plate-forme d'Intégration Continue Jenkins est installée, avec l'ensemble des plugins dont nous aurons besoin pour nos projets java, il ne nous reste plus qu'à la configurer, avant de pouvoir passer à la mise en IC d'un projet java. Configuration du système Deux étapes au minimum : Informer Jenkins de votre installation de Maven. Informer Jenkins de votre installation du Jdk. Survolez et cliquez sur le lien Jenkins > Administrer Jenkins > Configurer le système ; qui vous permettra de configurer les paramètres généraux de la plate-forme .



## 2- Configuration du GitHub - Jenkins

**Suivre le tutoriel :** Integrating GitHub Webhooks with Jenkins to automate unit and integration test after GitHub events for CI/CD

### 3- Creer un Job

- Dans Jenkins Cliquez sur [Nouveau Item] puis sur Pipeline et saisissez un nom pour votre Pipeline.
- Entrez l'url de votre projet au format https.
- Créez le crédential avec votre utilisateur + PAT
- Lancement du Pipeline

Une fois sauvegardé, il suffit de cliquer sur [Lancer un build]. Si vous cliquez ensuite sur le job, vous devriez voir les traces de son exécution.

## 4- Jenkins pipeline

Un fichier Jenkinsfile utilise une syntaxe appelée Jenkins Job DSL qui est fournie avec le plugin Pipeline. Voici un exemple de Jenkinsfile :

```
pipeline {
              agent { docker { image 'python:3.7.2' } }
              stages {
               stage('build') {
                 steps {
                  sh 'pip install -r requirements.txt'
                 }
                }
               stage('test') {
                 steps {
                  sh 'python test.py'
                 }
                 post {
                  always {
                   junit 'test-reports/*.xml'
                  }
                 }
               }
              }
              }
```

### Rappel:

#### Le bloc pipeline

Tous les pipelines possèdent un premier bloc de type pipeline.

C'est ce bloc qui va contenir des directives, qui sont pour les principales de type agent, tools, options, environment, post et stages. Nous verrons dans un autre billet le reste des directives.

#### La directive Agent (requis)

Dans le bloc agent où nous définissons sur quel agent va tourner notre pipeline. On peut ainsi indiquer n'importe quel agent avec la balise any, ou des spécifiques en

#### La directive stages (requise)

Cette section va permettre de définir tous les stages d'un pipeline. Il en faut au minimum un. indiquant un node portant un label.

Chaque stage va contenir des steps qui doivent contenir au minimum un step.

#### Le code des steps

Composantes des stages, les steps permettent de lancer des fonctions, appelés step. Ces steps sont composés de celles fournies nativement avec Jenkins, appelés basic steps.

#### La directive post (optionnel)

Cette directive permet de définir un ou plusieurs steps qui sont exécutés à la fin de l'exécution d'un pipeline. Post peut prendre des post-condition permettent l'exécution des steps à l'intérieur de chaque condition en fonction de l'état d'exécution du pipeline : always, changed, fixed; regression, aborted,...

#### 1- Pipeline avec paramètre:

Créer un job jenkins qui permet d'utiliser les paramètres de builds et d'afficher nos paramètres dans la console.( tester les différents types de paramètres)

#### 2- Pipeline avec variable d'environnement:

Créer une pipeline Jenkins qui permet d'utiliser les paramètres dans les steps comme exemple ci-après:

```
pipeline{
  agent any
  environment {
    VAR = "${params.VAR}"
  }
  stages {
     stage("Display param") {
       steps {
         script {
            if(VAR)
            {
             Echo "my value is true"
            }
            else if(VAR)
            {
               Echo "my value is false"
            }
          }
       }
     }
}
```

### 3- Pipeline avec conditions:

Créer une pipeline avec conditions comme dans l'exemple ci après :

```
1) When:
          when {
             expression {
               params.ENVIRONMENT == 'development'
             }
          }
2- IF-ELSE:
node {
  stage('Step1') {
    if (env.BRANCH NAME == 'main') {
       echo 'Hello from main branch'
     } else {
       sh "echo 'Hello from ${env.BRANCH_NAME} branch!'"
     }
  }
}
```