Jenkins

Hadrien Wilbert

• Résumé.

Jenkins est un logiciel de Devops qui permet de réaliser de tester et de déployer des logiciels grâce a des pipelines, le tout de façon automatique ou manuelle. On peut définir l'environnement dans lequel ce pipeline est exécuté. Le pipeline est créé avec plusieurs étapes dans lesquels ont peut lancer des commandes.

• Procédure d'installation (url, etc...).

Avant de pouvoir installer Jenkins il y a un nombre de prérequis a satisfaire :

- Avoir installer Java 11 ou Java 17

- Installer Docker (https://docs.docker.com/get-docker/)

Après cela on peut télécharger un installer Jenkins à proprement parler.

L'url ou les fichiers sont disponibles : https://www.jenkins.io/download/

On choisira Generic Java package (.war) .

Une fois le fichier choisi, on lance java-jar jenkins.war--httpPort=8080

Ce qui nous permet, en nous connectant a http://localhost:8080 de permettrer Jenkins en local.

Jenkins demandera ensuite un compte utilisateur pour le server local.

Après cela, Jenkins proposera de télécharger des plugins parmi une liste de plugins disponibles.

Une fois les plugins télécharger, Jenkins est prêt à être utiliser.

• Présentation des fonctionnalités.

L'utilisation principal de Jenkins est pour créer des pipelines. On peut créer 2 types de pipelines :

- Pipeline "locale" ; un pipeline lancé manuellement par l'utilisateur.

- Pipeline "online" ; un pipeline lancé automatiquement à chaque push vers un projet git.

Les pipelines Jenkins utilisent des fichiers des configurations capable de compiler dans les langages suivants :

- Java app grâce à Maven
- Node.js et React app avec npm
- Python avec PyInstaller

On peut définir un pipeline avec :

```
pipeline {
   /* insert Declarative Pipeline here */
}
```

Le pipeline sera soit défini dans un fichier de configuration JenkinsFile soit dans un script inclus dans le pipeline.

Afin de paramétrer le pipeline et exécuter des actions, il y a plusieurs stage a créer dans le fichier de configuration :

L'agent, définit l'environnement d'exécution. Il peut être défini de façon globale ou pour chaque étape. Il peut prendre les valeurs suivantes :

any

Exécute le pipeline avec n'importe qu'elle environnement disponible.

none

Il s'applique au pipeline global, l'agent devra être défini a chaque étape.

label

Exécute le pipeline dans l'environnement avec le label correspondant

```
agent { label 'my-defined-label' }
```

node

Fonctionne comme label mais permet a node de prendre des paramètres supplémentaires

```
agent { node { label 'labelName' } }
```

docker

Exécute le pipeline dans un conteneur docker préconfiguré.

```
agent {
  docker {
    image 'maven:3.9.0-eclipse-temurin-11'
    label 'my-defined-label'
    args  '-v /tmp:/tmp'
  }
}
```

Kubernetes
Exécute le pipeline dans un cluster Kubernetes. Pour que cela fonctionne il faut être dans un **Multibranch Pipeline** ou dans un **Pipeline créer depuis SCM**.

On peut ensuite définir des **stage** et des **step,** par exemple :

```
pipeline {
    agent none
    stages {
        stage('Example Build') {
            agent { docker 'maven:3.9.0-eclipse-temurin-11' }
            steps {
                echo 'Hello, Maven'
```

```
                sh 'mvn --version'
            }
        }
        stage('Example Test') {
            agent { docker 'openjdk:8-jre' }
            steps {
                echo 'Hello, JDK'
                sh 'java -version'
            }
        }
    }
}
```

On a ici un pipeline qui créer deux conteneurs en utilisant deux images différentes.

Il également est possible d'ajouter des conditions sur le résultat d'une step, par exemple si on réussit ou on échoue.

• Mini-guide d'utilisation sur les principales fonctionnalités dans le cadre de cette étude de cas.

Dans le cadre de notre projet, nous avons créé un pipeline local.

Nous avons d'abord essayé de compiler le projet avec le makefile mais le pipeline ne peut pas exécuter un fichier makefile comme on peut le voir dans le rapport du build suivant :

```
Démarré par l'utilisateur admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /home/local.isima.fr/hawilbert/.jenkins/workspace/testpipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (build)
[Pipeline] sh
+ cd /home/local.isima.fr/hawilbert/shared/projet-fourmi/projet-fourmi/Code_source
[Pipeline] sh
+ ./../../../shared/projet-fourmi/projet-fourmi/Code_source/makefile
/home/local.isima.fr/hawilbert/.jenkins/workspace/testpipeline@tmp/durable-2e0d2316/script.sh: 1: ./../../../shared/projet-fourmi/projet-
fourmi/Code_source/makefile: Permission denied
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 126
Finished: FAILURE
```

Nous nous contenterons donc de simplement lancer le .exe pour vérifier que le .exe se lance correctement. Nous avons fait cela avec le script suivant :

```
pipeline {
    agent any
    stages {
        stage('build') {
            steps {
                echo 'debut de la compilation'
                sh './../../../shared/projet-fourmi/projet-
                fourmi/Code_source/exeProjetFourmi'
            }
        }
    }
```

• Une synthèse des points clés du logiciel

Créer un pipeline git ou local.

Plusieurs étapes successives dans lesquels on lance des commandes.

Possibilité de définir un environnement d'exécution pour chaque étape.