Cahier de TPs

Jenkins Administration

Pré-requis :
Poste développeur avec accès réseau Internet libre
Linux (Recommandé) ou Windows 10
Pré-installation de :

- Git
- JDK11
- Docker
- Editeur fichier .yaml (Exemple VisualStudio Code)

Atelier 1 : Installation, Configuration

Objectifs

- Installation de Jenkins en service, configuration du service
- Configurer le système
- Configurer les outils de base
- Créer un premier job freestyle

1. 1 Installation

1.1.a : Installation en service

En fonction de votre OS, exécuter la procédure d'installation en service adéquat.

Avec un navigateur, accéder à localhost:8080 et continuer la procédure **sans** installer de plugin

Éditer le fichier */etc/default/jenkins* :

- modifier les paramètres de la JVM

Effectuer un redémarrage du service :

```
sudo service jenkins restart
```

Visualisez l'arborescence de JENKINS_HOME

1.1.b : Script de démarrage

Télécharger la distribution générique de Jenkins et écrire un script de démarrage qui positionne le JENKINS_HOME, la mémoire de la JVM et le port d'écoute de jenkins

Visualiser l'arborescence de JENKINS_HOME

1.2 Configuration générale et outils

1.2.1 Configuration serveur de mail

Si nécessaire installer le plugin ***mailer***

Ensuite aller dans la configuration système « *Administrer Jenkins* → *Configurer le système*» et renseigner les paramètres du serveur smtp

login : stageojen@plbformation.com

password : stageojen

smtp : smtp.plbformation.com

(port smtp 587)

Tester l'envoi de mail

1.2.2 Configuration des outils (JDK, Maven, Git)

1. Cliquer sur « *Administrer Jenkins* → *Configuration Globale des outils* »
2. Dans la section JDK, indiquer soit un JAVA_HOME pré-installé soit un installateur automatique
3. Dans la section Maven, utiliser l'installation automatique
4. Dans la section Git, indiquer votre installation de Git
   (Si la section Git n'est pas présente installer le plugin ***Git Client)***
5. Enregistrer vos modifications

1.2.3 Configuration As Code

1. Installer le plugin « *Configuration As code* »
2. Visualiser le fichier ***jenkins.yaml*** de configuration
3. Editer le et ajouter un deuxième JDK
4. Appliquer les modifications

Atelier 2 : Jobs

2.1 Job freestyle

Mise en place dépôt Git

Vérifier l'installation de git sur votre machine
Installer le plugin Jenkins « ***git*** »

Reprendre les source du projet et les décompresser.
Initialiser le dépôt et committer les fichiers sources :
git init
git add .
git commit -m 'Initial commit'

Création job et test

1. De retour sur la page d'accueil, Activer le lien « Créer un nouveau job »
2. Donner un nom et choisir job freestyle
3. Configurer le SCM afin qu'il point vers un dépôt git local
4. Indiquer que le build est déclenché à chaque changement dans GIT et que le repository est interrogé toutes les 5 mns. (*/5 * * * *)
5. Ajouter une étape de build affichant toutes les variables d'environnement disponibles
6. Ajouter ensuite une étape de build qui invoque la cible Maven « ***clean package*** », indiquer le chemin vers le *pom.xml*
7. Ajouter une étape « post-build » permettant d'afficher les résultats des tests : « Publier les rapports *JUnit* »
   Nécessite le plugin ***junit***
8. Ajouter une autre étape « post-build » pour archiver tous les jars produits
9. Lancer le build manuellement et observer la page d'accueil du projet

Déclenchement de job

1. Modifier un fichier du projet provoquant une erreur dans les tests, committer le changement dans le repository
   Voir fichier *library/src/test/java/hello/service/MyServiceTest.java*
2. Attendre le déclenchement du job
3. Observer la page d'accueil qui doit afficher un graphique de tendance sur l'exécution des tests.
4. Ensuite restaurer votre modification

Plugin Maven (Optionnel)

1. Installer le plugin ***Maven Integration***
2. Créer un job Maven nommé ***1_package*** effectuant les cibles ***clean package***
3. Exécuter manuellement le build et observer les résultats
4. Observer l'archivage automatique et les artefacts archivés, la configuration multi-modules

2.2 Fonctionnalités des Jobs

Job paramétré

1. Installer le plugin ***Git Parameter Plugin***
2. Définir un paramètre pour le job ***1_package*** dont la liste des valeurs est les différents hash de commit du repository Git
3. Démarrer le job manuellement

Job multi-configuration

Nécessite le plugin ***matrix-project***

Définir un deuxième JDK (il peut pointer sur le même JAVA_HOME) et définir un premier axe avec le JDK

Créer un second axe BD avec comme valeurs possibles :

- H2
- MySQL
- Postgres

Créer un nouveau projet multi-configuration ***1_package_multi*** avec les 2 axes précédents à partir du job ***1_package***

2.3 Exemple de pipeline

Pré-requis : Sonarqube
Démarrer Sonarqube via docker :

```
docker run -d --name sonarqube -p 9000:9000 sonarqube
```

Accéder à localhost:9000 , se connecter avec admin/admin et définir ***admin123*** comme nouveau mot de passe

Multi-branche pipeline

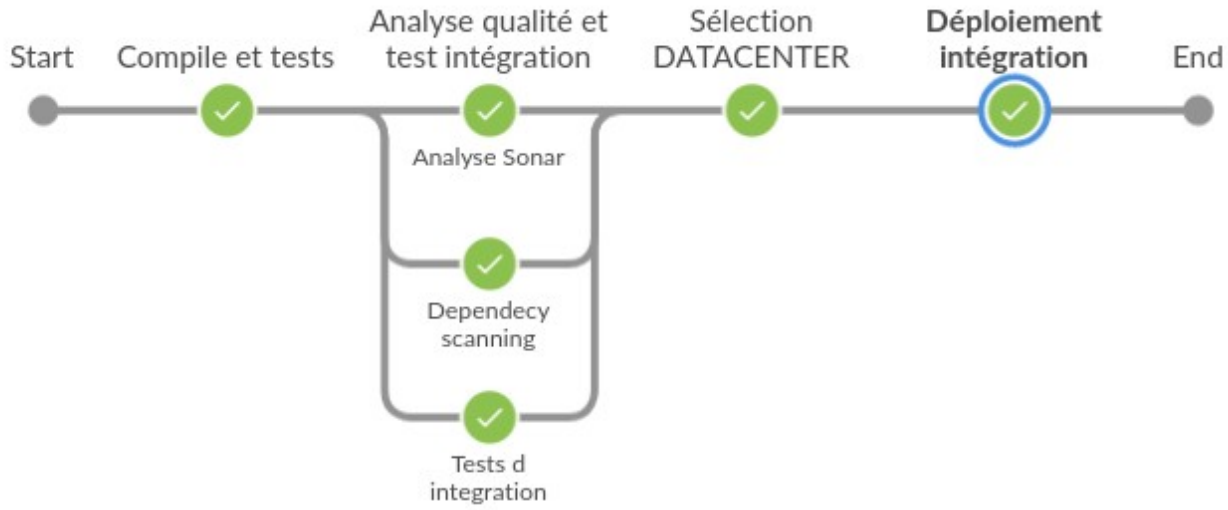
Installer le plugin « Blue Ocean »

Dans Jenkins, créer un Job « ***Multi-branche Pipeline*** » et indiquer le répertoire de travail précédent comme adresse de dépôt de Git

Provoquer le scan de répertoire et observer l'exécution de la pipeline.

Faire en sorte que la pipeline s'exécute avec succès

La pipeline dans Blue Ocean doit s'afficher comme ceci :



Atelier 3 : Architecture maître/esclave

Dans ce TP, nous mettons en place une architecture maître/esclaves permettant de distribuer la charge

.

Objectifs

- Mettre en place 1 nœud esclave
- Donner des étiquettes aux nœuds

Mise en place Agent SSH

Installation le plugin « ***SSH Build Agents*** »

Installation de OpenSSH Servers, (**openssh-server**) vérifier que le serveur sshd démarre

Vérifier l'accès avec *ssh localhost*

Dans l'interface d'administration Jenkins ; créer un nœud démarré en ssh, lui affecter 4 exécuteurs et des labels

Arrangez vous pour que ce nœud exécute un job. Visualisez ensuite le répertoire de travail du nœud esclave.

Atelier 4 : Pipeline docker

Objectifs

- Utiliser une image docker pour effectuer la partie Maven de votre build
- Utiliser les volumes et le cache pour ne pas repartir à zéro
- Construire une image Docker lors d'une étape de build
- Publier l'image vers un registre Docker

Vérifier l'installation du plugin ***Docker pipeline***

8.1 Compte DockerHub et déclaration crédentiel dans Jenkins

Créer un compte Dockerhub

Déclarer le user/password dans les crédentiels de Jenkins

8.2 Exécution pipeline

Créer une branche docker dans le repository Git de travail
Reprendre les fichiers ***Dockerfile*** et ***Jenkinsfile*** fournis
Modifier ces fichiers en fonction de votre compte Dockerhub (l'image doit être préfixée par votre compte) et du nom de votre crédentiel dans Jenkins

Exécuter la pipeline x

Atelier 5 : Kubernetes

Objectifs

- Installer Jenkins dans un cluster Kubernetes
- Utiliser le cluster kubernetes pour exécuter les builds

9.0 Installation Kubernetes

kubectl

- minikube
- kind

Helm

9.1 Installation Jenkins dans le cluster

Référence : https://www.jenkins.io/doc/book/installing/kubernetes/

Configurer Helm

```
helm repo add jenkinsci https://charts.jenkins.io
helm repo update
helm search repo jenkinsci
```

Créer un namespace ***jenkins***

```
kubectl create namespace jenkins
```

Créer un volume persistant :

```
kubectl apply -f
https://raw.githubusercontent.com/jenkins-infra/jenkins.io/master/
content/doc/tutorials/kubernetes/installing-jenkins-on-
kubernetes/jenkins-volume.yaml
```

Créer le compte service jenkins

```
kubectl apply -f
https://raw.githubusercontent.com/jenkins-infra/jenkins.io/master/
content/doc/tutorials/kubernetes/installing-jenkins-on-
kubernetes/jenkins-sa.yaml
```

Récupérer le fichier ***jenkins-values.yaml*** permettant de personnaliser l'installation helm et exécuter dans le même terminal :

```
chart=jenkinsci/jenkins
helm install jenkins -n jenkins -f jenkins-values.yaml $chart
```

Récupérer le mot de passe administrateur avec :

```
jsonpath="{.data.jenkins-admin-password}"
secret=$(kubectl get secret -n jenkins jenkins -o
jsonpath=$jsonpath)
echo $(echo $secret | base64 –decode)
```

Récupérer l'URL jenkins avec :

```
jsonpath="{.spec.ports[0].nodePort}"
NODE_PORT=$(kubectl get -n jenkins -o jsonpath=$jsonpath services
jenkins)
jsonpath="{.items[0].status.addresses[0].address}"
NODE_IP=$(kubectl get nodes -n jenkins -o jsonpath=$jsonpath)
echo http://$NODE_IP:$NODE_PORT/login
```

Se logger avec admin et le mot de passe précédent.

9.2 Pipeline avec un agent Kubernetes

Sur le projet multi-module, récupérer le fichier ***kubernetesPod.yaml*** et le visualiser.

Utiliser ce fichier yaml dans la pipeline Jenkins du projet multi-modules

Pousser le projet sur une URL publique (github.com ou gitlab.com)

Créer un Multi-branch pipeline pointant sur le dépôt public et tester

TP10 : Sécurité

.

Objectifs

- Mise en place de la sécurité
- Définition des autorisations par rôle, surcharge au niveau projet
- Autorisation dans l'étape manuelle d'une pipeline

Sécurité globale via des rôles

Activer la sécurité dans Jenkins
Installer le plugin *Role-based Authorization Strategy*
Définir un rôle admin, pouvant tout faire au niveau global
Définir un rôle ***marketing*** avec un accès en lecture uniquement
Créer un nouvel utilisateur et lui assigner le rôle ***marketing***

Se logger avec le nouvel utilisateur et vérifier votre configuration.

Sécurité projet

Créer un *Folder Legacy* et y déplacer les projets freestyle des premiers TPs
Définir un nouveau rôle projet avec comme expression régulière Legacy.* et ayant tous les droits sur les jobs (Create, Build, Configure, ...)
Affecter ce rôle à l'utilisateur

Se logger avec le nouvel utilisateur et vérifier qu'il peut démarrer les jobs legacy.

Pipeline avec approbation manuelle

Dans la pipeline avec approbation manuelle, ajouter des permissions sur l'action de déploiement afin que l'utilisateur et l'admin puisse activer le bouton

TP11 : Exploitation, Jenkins CLI, API Rest

Dans ce TP, nous abordons les aspects d'exploitation

.

Objectifs

- Découvrir les plugins d'exploitation et de surveillance
- Effectuer des sauvegardes et migration
- Automatiser le backup et restore d'un jobs
- Déclenchement de build via API Rest

Monitoring

Installer les plugins *Disk Usage* et *Monitoring*

Visualiser les vues associées

Backup and Restore

Démarrer un second serveur Jenkins (autre port http)

Effectuer un backup de la première instance et la restaurer dans la seconde instance

Comparer avec un déplacement du répertoire JENKINS_HOME

Utiliser le *Thin Backup* Plugin, effectuer des backup de la configuration toutes les 5 min

Jenkins CLI

Créer un token pour l'utilisateur *admin*.

Télécharcher jenkins-cli.jar via la console d'administration

Utiliser le token pour afficher l'aide :

java -jar jenkins-cli.jar -s http://localhost:8080 -auth admin:<token> list-jobs > jobs.txt

Via le cli récupérer la liste des jobs et leur configuration au format XML

Rest API

Déclencher un build paramétré via une requête htt