**Déploiement**

**4 composants GitLab :**

- Runner

- Référentiel Maven

- PipeLine

- Dépot Git

**Dépôt Git**

- Créez un projet gitlab vide par projet github

- Sur vos dépôts locaux, ajoutez le remote gitlab

- Poussez le code sur le remote créé précédemment

- Pour la suite une branche “dev” sera nécessaire, créez cette branche

- Placez vos dépôts locaux sur cette branche et poussez-la sur gitlab.

**Runner**

Se sont des machines fournies ou enregistrées par nous-mêmes pour l'exécution des pipelines CI/CD).

Les runners fournis par GitLab sont payants.

**Installation d’un Runner sous Windows**

dans un dossier

installer gitlab-runner.exe correspondant à son environnement

ouvrir un terminal en mode administrateur (cmd clique-droit administrateur)

gitlab-runner.exe register

https://gitlab.com

token : prendre celui founit sous GitLab/settings/CICD/Runners

nom runner

tags : enter

optionnal : enter

docker

openjdk:xx xx correspondant à l’environnement jdk de son projet (17, 19, )

normalement, il n’y a pas besoin de faire « *gitlab-runner.exe start »* pour le lancer.

Mêmle au redémarrage de la machines

**PipeLine**

Se sont des sortes de script.

Les pipelines de déploiement continu assurent l'installation de la machine cible

en termes de dépendances logicielles telles que MySQL, Apache HTTPD ou la JVM.

Ainsi que l'installation de la dernière version compilée de notre application dans la machine cible.

Le pipeline est exécuté dans le Runner GitLab et utilise une connexion SSH avec la machine cible pour réaliser lesdifférentes tâches.

**Créez un fichier nommé *.gitlab-ci.yml* dans son projet**

C’est le pipeline de son projet

**ce nom permet son exécution automatique quand on push**

Dans ce pipeline, il faut sélectionner une image docker correspondant à la version de java de son projet :

***image: maven:3.9.0-eclipse-temurin-17***

cette image qui fonctionne le 16/02/2023, 17 correspond à la version de jdk de son projet

**vérifier la présence de maven**

dans un terminal tapez mvn -version → si ce n’est pas le cas, installez maven

**installation de maven sous windows**

Télécharger l'archive sur le site https://maven.apache.org/download.html

Créer la variable d'environnement ***M2\_HOME*** qui pointe sur le répertoire contenant Maven

Ajouter le chemin ***%M2\_HOME%\bin*** à la variable **PATH** du système

**Créer un fichier settings.xml dans la racine du projet**

On pourrait compléter le fichier settings.xml contenu dans users/.m2, mais cela incluerait le token propre au projet nécessaire pour maven

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0

http://maven.apache.org/xsd/settings-1.0.0.xsd">

<servers>

<server>

<id>gitlab-maven</id>

<configuration>

<httpHeaders>

<property>

<name>Private-Token</name>

<value>glpat--6JvuBySjjxrYSb2DaUd</value>

</property>

</httpHeaders>

</configuration>

</server>

</servers>

</settings>

Le token se génère sour GitLab/settings/Access Tokens.

Le sauvegarder quelquepart

**Compléter le fichier pom.xml du projet**

<repositories>

<repository>

<id>gitlab-maven</id>

<url>https://gitlab.com/api/v4/projects/43544440/packages/maven</url>

</repository>

</repositories>

<distributionManagement>

<repository>

<id>gitlab-maven</id>

<url>https://gitlab.com/api/v4/projects/43544440/packages/maven</url>

</repository>

<snapshotRepository>

<id>gitlab-maven</id>

<url>https://gitlab.com/api/v4/projects/43544440/packages/maven</url>

</snapshotRepository>

</distributionManagement>

**Créer l’application java à l’aide maven**

dans le dossier du projet :

**mvn clean install** → création du .jar du projet

**mvn validate** → télécharge toutes les dépendances requises dans le .m2référentiel local.

**mvn compile** → compile le projet java

**mvn test** → La commande exécute le test unitaire qui fait partie du code. Vous pouvez tester les classes individuellement, les méthodes individuellement ou ajouter des modèles pour exécuter des tests sur toutes les méthodes qui correspondent au modèle.

**mvn package** → compile le code, le teste et enfin le conditione au format requis (jar,war,.)

**mvn deploy** -s settings.xml → -s settings.xml permet d’aller chercher le fichier settings.xml à la racine du projet, il contient le token nécessaire à Maven

→ commande, déploie le package dans le référentiel maven distant. Lorsque vous exécutez le déploiement, il exécute d'abord valider, compiler, tester, empaqueter, vérifier, installer, puis déploie enfin le paquet sur le référentiel maven distant.

**Exemple d’un pipeline « tout simple »**

*stages:*

*- build*

*- test*

*- deploy*

*image: maven:3.9.0-eclipse-temurin-17*

*build:*

*stage : build*

*script:*

*- mvn install*

*test:*

*stage: test*

*script:*

*- mvn test*

*deploy:*

*stage: deploy*

*script:*

*- mvn deploy -s settings.xml*

**Suite du module déploiement avec Bassem**

**Créer un repository propre au script d’installation d’Apache, JVM, Mysql sur son serveur.**

**Sur le serveur cible, créer une clef SSH**

ssh-keygen -t ed25519 -C "GitLab SSH key"

Rajouter la clef publique générée ed25519.pub dans le fichier .ssh/authorized\_keys

**Rajouter la clef privée (cat .ssh/ ed25519) dans GitLab via une variable d’environnement**

**Initialisation des variables d’environnement dans Gitlab**

Gitlab / Settings / CCID / Variables → Expand

Add variable

SSH\_PRIVATE\_KEY / -----BEGIN OPENSSH PRIVATE KEY----- b3BlbnNzaC1rdjE…

SSSERVER\_IPADDRESS / xxx.xxx.xxx.xxx

H\_USER / root

La variable sera utilisée dans le script d’un pipeline : $SSH\_PRIVATE\_KEY

**Script du pipeline**

dans le fichier .gitlab-ci.yml situé à la racine du projet java

check-ssh:

image: alpine:latest

before\_script:

- 'command -v ssh-agent >/dev/null || ( apk add --update openssh )'

- eval $(ssh-agent -s)

- echo "$SSH\_PRIVATE\_KEY" | tr -d '\r' | ssh-add -

- mkdir -p ~/.ssh

- chmod 700 ~/.ssh

- ssh-keyscan $SERVER\_IPADDRESS >> ~/.ssh/known\_hosts

- chmod 644 ~/.ssh/known\_hosts

script:

# Message de bienvenue si connexion ssh ok

- ssh $SSH\_USER@$SERVER\_IPADDRESS "hostname && echo 'Welcome!!!'"

#

# **mysql / mariadb**

#

# Download and Install the Latest Updates for the OS

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**apt-get update && apt-get upgrade -y**"

# Enable Ubuntu Firewall and allow SSH & MySQL Ports

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**ufw allow 22**" // le port 22 est celui utilisé par SSH

# Install essential packages

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**apt-get -y install zsh htop**"

# Install MySQL Server in a Non-Interactive mode. Default root password will be "root"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**apt-get -y install mariadb-server-10.3**" // mariadb : mysql open source

#

# **JVM / Jdk17**

#

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**sudo apt -y install wget curl**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**wget https://download.oracle.com/java/17/latest/jdk-17\_linux-x64\_bin.deb**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**sudo apt -y install ./jdk-17\_linux-x64\_bin.deb**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "s**udo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jdk-17/bin/java 1717**"

#

# **Apache2**

#

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**apt -y install apache2**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**sudo service apache2 start**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**systemctl status apache2**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**sudo ufw allow 80/tcp**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS "**ufw enable**"

- ssh $SSH\_USER@$SERVER\_IPADDRESS **"sudo ufw status**"