**Déploiement**

**4 composants GitLab :**

- Runner

- Référentiel Maven

- PipeLine

- Dépot Git

**Dépôt Gitlab**

- 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.

Désintaller proprement **Apache2** avec un compte administrateur(root)

**netstat -an** → liste les applications installées

**cd /etc/init.d** → chemin qui mène au processus init qui démarre les processus interne

**./apache2 stop** → arrête le processus

**apt-get remove –purge apache2** → suppression de manière complète

**apt-get auto-remove apache2** → suppression de tous les packages associés

**ps -ef|grep apache2** → vérifie qu’il ne reste plus aucun processus d’ apache2

**rm -r apache2** → supprime tout ce qui est récursif à ce dossier.

Désinstaller proprement **mysql** avec un compte administrateur(root)

**apt purge mysql-server\* →** désinstallation complète des paquets liés à Mysql

**rm -r /etc/mysql /var/lib/mysql** → supprime les différents fichiers de configuration

**rm -r /var/log/mysql** → supprime le dossier des logs

**apt autoremove** → suppression de tous les packages qui ne sont plus nécessaires

Déinstaller proprement **Java** avec un compte administrateur(root)

**update-alternatives --display java** → liste les version de java installées

*java - auto mode*

*link best version is /usr/lib/jvm/****jdk-17****/bin/java*

*link currently points to /usr/lib/jvm/jdk-17/bin/java*

*link java is /usr/bin/java*

*slave java.1.gz: /usr/lib/jvm/java-11-openjdk-amd64/man/man1/java.1.gz*

***/usr/lib/jvm/jdk-17/bin/java*** *- priority 1717*

**sudo update-alternatives --remove** "**java**" \ "**/usr/lib/jvm/jdk-17/bin/java**"

**cd /usr/lib/jvm →** ne pas effacer les dossiers *usr*/lib pour ne pas détruire définitivement le serveur !

**rm -rf jdk-17** → supprime le dossier jdk-17 et tous ses sous-dossiers

**sudo update-alternatives --config java** → pour vérifier qu’il n’y a aucune alternative à java

**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.

**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 c:\gitlab-runner

installer gitlab-runner.exe correspondant à son environnement

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

verifier que docker est installé : **docker version**

créer le container :

**docker run**

-d

--name gitlab-runner

--restart always

-v C:\gitlab-runner\config:/etc/gitlab-runner

-v /var/run/docker.sock:/var/run/docker.sock

gitlab/gitlab-runner:latest

***docker run*** *-d --name gitlab-runner --restart always -v C:\gitlab-runner\config:/etc/gitlab-runner -v /var/run/docker.sock:/var/run/docker.sock gitlab/gitlab-runner:latest*

enregistrer le conteneur auprès de Gitlab :

docker run

--rm -t -i

-v C:\gitlab-runner\config:/etc/gitlab-runner gitlab/gitlab-runner

register

--non-interactive

--executor "docker"

--docker-image openjdk:XX *correspondant à l’environnement jdk de son projet (17, 19, )*

--url "https://gitlab.com/"

--registration-token "<TOKEN>" Token de projet/settings/CCID/Runners

--description "xxxxxxxxx-runner"

--run-untagged="true"

--locked="false"

*docker run --rm -t -i -v C:\gitlab-runner\config:/etc/gitlab-runner gitlab/gitlab-runner register --non-interactive --executor "docker" --docker-image openjdk:17 --url "https://gitlab.com/" --registration-token "<TOKEN>" --description "xxxxxxxxx-runner" --run-untagged="true"--locked="false"*

**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.

**ATTENTION, lors de la création du Tokken, selectionner un rôle Owner**

Le sauvegarder quelquepart

**Compléter le fichier pom.xml du projet avec les accès au projet GitLab**

<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>

**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*

**Pipeline d’installation des dépendances**

**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**"

**Info serveur Apache**

dans le dossier /etc/apache2/sites-available

il y a les fichiers (.conf) de configuration du serveur apache2

dans chaque fichier .conf (faire un cat), il y a la configuration,

dont le ***DocumentRoot qui indique le chemin où se situe le fichier index.html***

on peut supprimer un fichier .conf avec la commande **rm**

on peut également modifier le fichier .conf avec la commande **nano**

Une fois que c’est fait, il faut recharger le serveur apache : **systemctl restart apache2**

**Pipeline de déploiement continu de l’application**

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

**2) Compléter le fichier pom.xml du projet avec les accès au projet GitLab**

**3) Initialiser les variables d’environnement dans Gitlab**

*SSH\_PRIVATE\_KEY*

*SERVER\_IPADDRESS / xxx.xxx.xxx.xxx*

*H\_USER / root*

**4) Script du pipeline**

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

check-ssh:  
 stage: .pre  
 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:  
 - ssh $SSH\_USER@$SERVER\_IPADDRESS "hostname && echo 'Welcome!!!'"  
 - ssh $SSH\_USER@$SERVER\_IPADDRESS "**mkdir -p /opt/Square\_Games**"  
  
*# garde en mémoire entre les différents jobs les fichiers de ces répertoires*  
cache:  
 key: ${CI\_COMMIT\_REF\_SLUG}  
 paths:  
 - m2/repository  
 - ./target  
 - ./service // dossier où se situe le fichier game.service  
*# la suite nécessite une image maven*  
image: maven:3.9.0-eclipse-temurin-17  
build:  
 stage : build  
 script:  
*# install maven et compile le source avec le fichier local settings.xml*  
- **mvn install -s settings.xml**  
 - **mvn compile -s settings.xml**  
*# déploie le package dans le référentiel maven distant*  
deploy:  
 stage: deploy  
 script:  
 - **mvn deploy -s settings.xml**  
  
copyfichier:  
 stage: deploy

*# il faut recharger l’image alpine*  
 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:  
 *# copie le fichier compile dans le dossier /opt/Square\_games du serveur*  
- **scp ./target/API\_Square\_Games.jar** $SSH\_USER@$SERVER\_IPADDRESS:**/opt/**  
 - **scp ./service/game.service** $SSH\_USER@$SERVER\_IPADDRESS:**/etc/systemd/system/**  
 *# lance le service systemD a partir du fichier de configuration game.service*

- ssh $SSH\_USER@$SERVER\_IPADDRESS **killall java**  
- ssh $SSH\_USER@$SERVER\_IPADDRESS **systemctl enable game.service**  
 - ssh $SSH\_USER@$SERVER\_IPADDRESS **systemctl start game.service**

**5) créer dans l’éditeur le fichier de configuration de systemD**

mettre ce fichier dans un dossier propres aux services, ce qui facilitera sa récupération dans le script du pipeline (voir plus haut).

# creation du fichier de configuration system

**[Unit]**  
# description du service  
Description=service de demarrage de api\_squaregames  
# lance le service apres ce service  
After=network-online.target

**[Service]**  
Type=simple  
# execute le fichier compile Square\_Games.jar  
ExecStart=java -jar /opt/API\_Square\_Games.jar  
# relance le programe en cas d'interuption  
Restart=on-failure  
# Configures the time to wait before service is stopped forcefully.  
TimeoutStopSec=300

# le service est accessible pour tous les users  
**[Install]**  
WantedBy=multi-user.target

**6) vérifier le bon fonctionnement de l’ensemble sur le serveur**

systemctl status xxxxxx, xxxxxx étant le nom du fichier xxxxxx.service

curl [http://localhost:8080**/test**](http://localhost:8080/test)

8080/test étant une requête GET appelé « **test** » du microservice sur le port 8080

**7) Déploiement des micro-services**

L'ensemble des services doit être accessible depuis un seul point d'entrée

→ VirtualHost dans Apache HTTPD

pour le port 443/HTTPS, cela nécessite la génération et la maintenance d'un certificat SSL de type Let's Encrypt.

**pour le port 80/http :** (à éviter pour des Pb de sécurité)

Créer un fichier .conf dans *etc/*apache2/sites-available/ :

<VirtualHost \*:**80**>

ServerName **xxx.yyy.zzz.aaa**

ServerAdmin [root@](mailto:root@51.15.197.26)**xxx.yyy.zzz.aaa**

ProxyPass / [http://**xxx.yyy.zzz.aaa**:8080/](http://xxx.yyy.zzz.aaa:8080/) // si l’appli utilise le port 8080

ProxyPassReverse / **xxx.yyy.zzz.aaa**:8080/

ProxyRequests Off // pour des raisons de sécurité

</VirtualHost>

**pour le port 443/https :**

Créer un fichier .conf dans *etc/*apache2/sites-available/

*# VHost sur chevriercampus.me port 443 (https)*  
<VirtualHost \*:443>  
 ServerName chevriercampus.me  
 ServerAdmin root@chevriercampus.me  
 *# proxy*  
 ProxyPass / http://51.15.197.26:8080/  
 ProxyPassReverse / http://51.15.197.26:8080/  
 ProxyRequests Off  
 *# SSL / TLS*  
 SSLEngine on  
 SSLCertificateFile /etc/letsencrypt/live/chevriercampus.me/fullchain.pem  
 SSLCertificateKeyFile /etc/letsencrypt/live/chevriercampus.me/privkey.pem  
 *# fichier log*  
 ErrorLog /var/log/apache2/error.chevriercampus.log  
 CustomLog /var/log/apache2/chevriercampus.me.log combined  
</VirtualHost>

Activer le module SSL

**a2enmod ssl**

**systemctl reload apache2** → recharger la configuration

**apache2ctl -M | grep ssl** → vérification activation du module

Créer un certificat SSL

**apt update**

**apt install certbot** → install le paquet certbot

**certbot certonly --webroot -w /var/www/example -d example.com -d www.example.com**

→ Cette commande va générer un certificat valable pour les domaines (option -d) example.com et www.example.com, qui correspondent à un site web existant et fonctionnel placé dans le dossier racine (option -w) var/www/example.

→ À l'issue de la commande précédente sera crée une arborescence sous /etc/letsencrypt qui contient vos certificats et des fichiers de configuration nécessaires aux procédures de renouvellement et de révocation. Seront crées également une tâche cron (/etc/cron.d./certbot) et un timer systemd (lib/systemd/system/certbot.timer) permettant de renouveler automatiquement les certificats avant qu'ils n'arrivent à échéance (les certificats Let'sEncrypt ne sont valables que 3 mois).

→ L’emplacement du certificat et de la clé privée est indiqué à la fin de la procédure, typiquement :

/etc/letsencrypt/live/example.com/privkey.pem pour la clé privée

/etc/letsencrypt/live/example.com/fullchain.pem pour le certificat

**Commun http / https**

activer / désactiver un fichier.conf

se place dans dossier ***etc*apache2/sites-avalaible/**

**apache2ctl -S** pour afficher les Vhost configurés

**a2ensite** xxxxx.conf pour charger un fichier de configuration

**a2dissite** xxxxx.conf pour décharger un fichier de configuration

**systemctl** restart apache2

activer les deux modules proxy et proxy\_http :

**a2enmod proxy**

**a2enmod proxy\_http**

Redémarrer Apache HTTPD pour prendre en compte la nouvelle configuration :

s**er**

Vérifier le bon fonctionnement sur un terminal :

curl **xxx.yyy.zzz.aaa**:**80/test**

**Déploiement continu avec création d’images docker-image**

le fichier Dockerfile doit être à la racine du projet.

Utiliser GitLab.

**Installer un runner sous ubuntu**

curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash

sudo apt-get install gitlab-runner

**Ajoutez les gitlab-runner utilisateur au dockergroupe :**

sudo usermod -aG docker gitlab-runner

**Dans le cas présent, il faut créer 2 runners :**

- apidocker

- apishell

REGISTRATION\_TOKEN est le tokken dispo sous gitlab/settings/cicd/menu runner

**Enregistrer le runner apishell**

sudo gitlab-runner register -n \

--url "https://gitlab.com/" \

--registration-token REGISTRATION\_TOKEN \

--executor **shell** \

--description "Runner-api-shell"

**Enregistrer le runner apidocker**

sudo gitlab-runner register -n \

--url "https://gitlab.com/" \

--registration-token REGISTRATION\_TOKEN \

--executor docker \

--description "My Runner"

--docker-image alpine:latest

**Executer les runners**

sudo gitlab-runner run

**Initier le fichier créer un fichier .gitlab-ci.yml à la racine du projet :**

# la suite nécessite une image maven

build:

image: maven:3.9.0-eclipse-temurin-11

stage : build

# Runner à utiliser

tags:

- apidocker

script:

# install maven et compile le source

- mvn install

- mvn compile

# la suite nécessite une image maven

test:

image: maven:3.9.0-eclipse-temurin-11

stage : test

# Runner à utiliser

tags:

- apidocker

script:

# test

- mvn test

# Build and push the Docker image to the GitLab image

# registry using Docker-in-Docker.

dind-build:

stage: deploy

# Runner à utiliser

tags:

- apishell

script:

# GitLab has a built-in Docker image registry, whose

# parameters are set automatically. You can use some

# other Docker registry though by changing the login and

# image name.

- docker login -u "stephanechevrier" -p "AQWzsx12&"

- docker build -t stephanechevrier/realworld-api .

- docker push stephanechevrier/realworld-api

**FAQ Changement de poste**

**Sur le nouveau poste, créer une nouvelle clef ssh.**

Récupérer la clef SSH créée dans le dossier /.ssh :

CAT key.pub → clef publique → A rajouter dans *.ssh*/authorized\_keys du serveur

CAT key → clef privée → A rajouter dans GitLab dans User Settings

**Configurez les paramètres Maven avant d'ouvrir un projet**

Vous pouvez configurer IntelliJ IDEA pour afficher les paramètres Maven avant d'ouvrir un nouveau projet Maven. Comme il s'agit de paramètres au niveau de l'application, ils sont appliqués à tout nouveau projet que vous ouvrez.

File |New Project Setup | Preferences for New Projects.

Build, Execution, Deployment | Build Tools | Maven.

Maven settings dialog : select the Show settings dialog for new Maven projects.

**Initier le fichier settings.xml dans le répertoire Users/xxxxxx/.m2**

**FAQ serveur distant**

***etc/*apache2/sites-available** → emplacement des fichiers .conf de Apache

**var/log** → emplacement des fichiers de logicielles

**tail -f syslog** → visualisation dynamique du fichier syslog. Permet par exemple de voir que l’API plante et se relance en permanence.

**etc/systemd/system** → emplacement des fichiers .conf de systemd

**vérifier que la variable d’environnement path contient bien les chemins nécessaires**

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/jvm/jdk-17/bin

**vérifier que la variable d’environnement JAVA\_HOME contient le bon chemin vers java**

si ok java -version retourne un message correct