Table des matières

[Objectif 2](#_Toc70958610)

[Infrastructure 3](#_Toc70958611)

[1. Prérequis 3](#_Toc70958612)

[2. Installation de jenkins 3](#_Toc70958613)

[3. Déploiement du serveur applicatif 4](#_Toc70958614)

[1. Conception 4](#_Toc70958615)

[2. Utilisation 5](#_Toc70958616)

[4. Installation du registry 5](#_Toc70958617)

[1. Conception 5](#_Toc70958618)

[2. Utilisation 6](#_Toc70958619)

[5. Installation du Code Repository 6](#_Toc70958620)

[1. Pré-requis 6](#_Toc70958621)

[Installation de GitLab local 6](#_Toc70958622)

[2. Utilisation de GitLab 7](#_Toc70958623)

[Les configurations 8](#_Toc70958624)

[1. Jenkins et Gitlab 8](#_Toc70958625)

[L’intégration continue 12](#_Toc70958626)

[Etape 1 et 2 : Lancement du pipeline CI lors d’un push sur GitLab 12](#_Toc70958627)

[Etape 3 : Récupération du code sur GitLab 12](#_Toc70958628)

[Etape 4 : Monter un container Docker via le Dockerfile simple\_api 12](#_Toc70958629)

[Etape 5.1 : Démarrer le container 13](#_Toc70958630)

[Etape 5.2 : Lancer une commande curl pour s’assurer que l’application fonction 13](#_Toc70958631)

[Etape 6 : Utiliser CLAIR 14](#_Toc70958632)

[Etape 7 : Pousser l’image sur le Registry 15](#_Toc70958633)

[Etape 8 : Déploiement de l’application via Ansible 15](#_Toc70958634)

[Etape 9 : Lancer un test sécurité avec Arachni 16](#_Toc70958635)

# Objectif

L’objectif de ce POC est de simuler un environnement local de développement permettant de satisfaire au fonctionnement décrit par le schéma ci-dessous :

1 : Lancement d’un push par un développeur > lancement du pipeline CI

2 : Le serveur Code Repository informe Jenkins

3 : Jenkins récupère le code poussé sur le serveur Code Repository

4 : Jenkins monte un container via le Dockerfile simple\_api

5 : Jenkins test le fonctionnent du server via la commande curl -u toto:python -X GET http://:<host IP><API exposed port>/pozos/api/v1.0/get\_student\_ages

6 : Jenkins lance le scanner d’image « clair »

7 : Jenkins pousse l’image validé sur le serveur Registry

8 : Jenkins déploye l’image sur le server App\_Server va Ansible

9 : Jenkins lance un test E2E afin de s’assurer la validité sécurité de l’app via Arachi de Gautlt

**App\_Server**

Server de production

**Deployment Server**

Server d’intégration continue permettant de faire les tests

****

****

****

**Registry**

Stock les images utilisées dans l’environnement

**Code Repository**

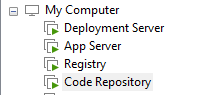
Permet de stocker et de mettre à disposition le code générant l’environnemen**t**

****

# Infrastructure

## Prérequis

Monter 4 VM dans VMware de la façon suivante :



Sur chacune des machines virtuelles, installer :

* Centos7.6.
* Editer les préférences réseaux via la commande *nmtui* afin d’automatiquement lancer la détection de réseau.
* Installer Docker via les instructions du site [*https://docs.docker.com/engine/install/centos/*](https://docs.docker.com/engine/install/centos/)
* Installer Git via la commande yum install git

De plus, puis-ce que nous allons souhaitez démarrer automatique nos applications au démarrage des serveurs, il faudra démarrer automatique docker engine. Le démarrage automatique des images docker se fera via les paramètres des docker-compose.yml.

* Démarrer docker engine automatiquement via la commande systemctl enable /usr/lib/systemd/system/docker.service

## Installation de jenkins

Accès à la documentation en ligne : https://www.jenkins.io/doc/book/installing/linux/

Après installation de la VM, l’installation de wget, suivi du process

sudo wget -O /etc/yum.repos.d/jenkins.repo \

https://pkg.jenkins.io/redhat-stable/jenkins.repo

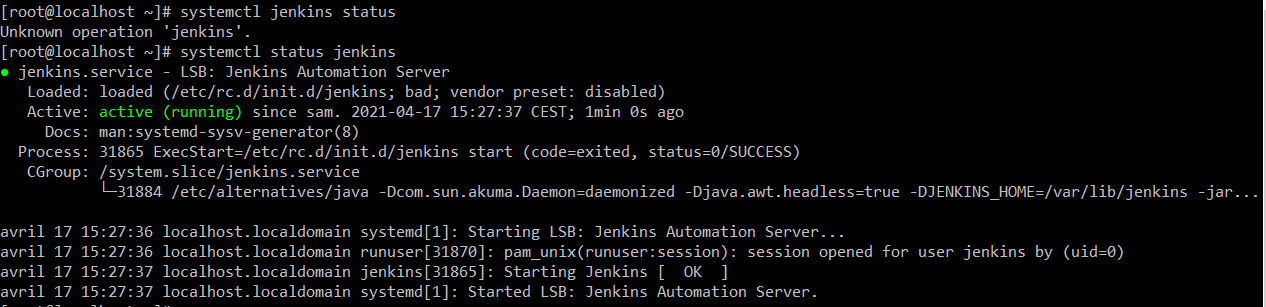
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

sudo yum upgrade

sudo yum install jenkins java-1.8.0-openjdk-devel

sudo systemctl daemon-reload

Le resultat que nous avons via la commande *systemctl status jenkins*



Puis ouvrir les port dans le firewall

YOURPORT=8080

PERM="--permanent"

SERV="$PERM --service=jenkins"

firewall-cmd $PERM --new-service=jenkins

firewall-cmd $SERV --set-short="Jenkins ports"

firewall-cmd $SERV --set-description="Jenkins port exceptions"

firewall-cmd $SERV --add-port=$YOURPORT/tcp

firewall-cmd $PERM --add-service=jenkins

firewall-cmd --zone=public --add-service=http --permanent

firewall-cmd –reload

## Déploiement du serveur applicatif

### Conception

Le dossier App\_Server est organisé de la façon suivante :

D’un point de vue fichier, nous avons

**Simple\_api**

* Monte une image python :2.7-stretch comme demander par POZOS
* On installe l’ensemble des modules nécessaires au fonctionnement de flask
* On monte un volume nommé « data » dans le container
* On copie dans la racine du container le fichier student\_age.py
* On copie dans le volume monté le fichier student\_age.json
* On spécifie la commande à lancer dans le container afin de pointer sur le fichier python
* On expose le container sur le port 5000

**website**

* On monte l’image l’image php :8.0-apache comme demander par POZOS
* On copie le fichier index.php dans le dossier /var/www/html du container
* On expose le container sur le port 8080

**Docker-compose**

Lance le container php-apache

* Nomme le container : website
* Monte le container via un build du Dockerfile
* Demande le lancement automatique du container au démarrage de Docker
* Expose l’image sur le port 8080
* Créer une dépendance afin de lancer ce container avant l’autre
* Stipule les credentials via 2 variables d’environnement

Lance le container flask

* Nomme le container : flask
* Monte le container via un build du Dockerfile
* Demande le lancement automatique du container au démarrage de Docker

### Utilisation

Téléchargez le dossier de travail via la commande

git clone <https://github.com/FlorianVigier/4DVOP_POZOS.git>

Puis, vous devrez vous rendre dans le dossier télécharger puis dans le dossier App\_Server

cd 4DVOP\_POZOS/App\_Server

Puis lancer la commande

docker-compose up

L’application est lancé sur votre serveur.

## Installation du registry

### Conception

Le dépôt du registre ne compose qu’un fichier docker-compose construit de la façon suivante :

**Docker-compose**

Lance le container regristry

* Nomme le container : registry en version 2.7
* Monte l’image registry supporté par Docker
* Demande le lancement automatique du container au démarrage de Docker
* Expose l’image sur le port 5000
* On monte un volume local storage dans /var/lib/registry afin de presister les images

Lance le container front\_end

* Nomme le container : registry\_interface
* Monte l’image joxit/docker-registry-ui:static
* Demande le lancement automatique du container au démarrage de Docker
* Expose l’image sur le port 80
* Créer une dépendance afin de lancer ce container avant l’autre
* Stipule les 2 variables d’environnement afin de pointer la page sur le registre

### Utilisation

Téléchargez le dossier de travail via la commande

git clone <https://github.com/FlorianVigier/4DVOP_POZOS.git>

Puis, vous devrez vous rendre dans le dossier télécharger puis dans le dossier Registry

cd 4DVOP\_POZOS/Registry

Puis lancer la commande

docker-compose up

## Installation du Code Repository

### Pré-requis

Ne pas oublier d’éditer les préférences de la VM Centos afin de monter à 4Go de ram.

Cela permettra d’éviter les problèmes de TimeOut sur certains services.

Installation de Gitlab en local

### Installation de GitLab local

Pour installer Gitlab en local, il suffira de suivre les instruction présentent sur le site [*https://about.gitlab.com/install/#centos-8*](https://about.gitlab.com/install/#centos-8)mais vous trouverez ci-dessous un extrait des commandes.

**Installation des dépendances**

sudo dnf install -y curl policycoreutils openssh-server perl

sudo systemctl enable sshd

sudo systemctl start sshd

sudo firewall-cmd --permanent --add-service=http

sudo firewall-cmd --permanent --add-service=https

sudo systemctl reload firewalld

sudo dnf install postfix

sudo systemctl enable postfix

sudo systemctl start postfix

**Installation du package GitLab**

curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.rpm.sh | sudo bash

sudo EXTERNAL\_URL="https://gitlab.example.com" dnf install -y gitlab-ee

L’installation se terminera par le message suivant :



### Utilisation de GitLab

La page de GitLab est accessible via l’IP de la VM. La premiere fois, vous aurez à saisir un nouveau mot de passe. Le login se fera via le compte *root*.

* User : Root
* Password : a définer.

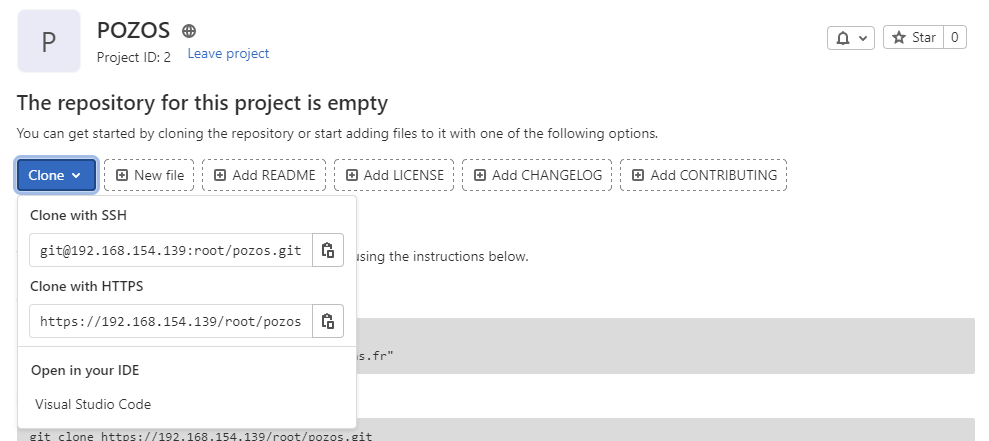
Pour créer le lien entre GitLab et le registry

Edit de /etc/gitlab/gitlab.rb

# Les configurations

## Jenkins et Gitlab

1/ Dans GitLab, veuillez créer un projet et récupérer l’adresse HTTPS.

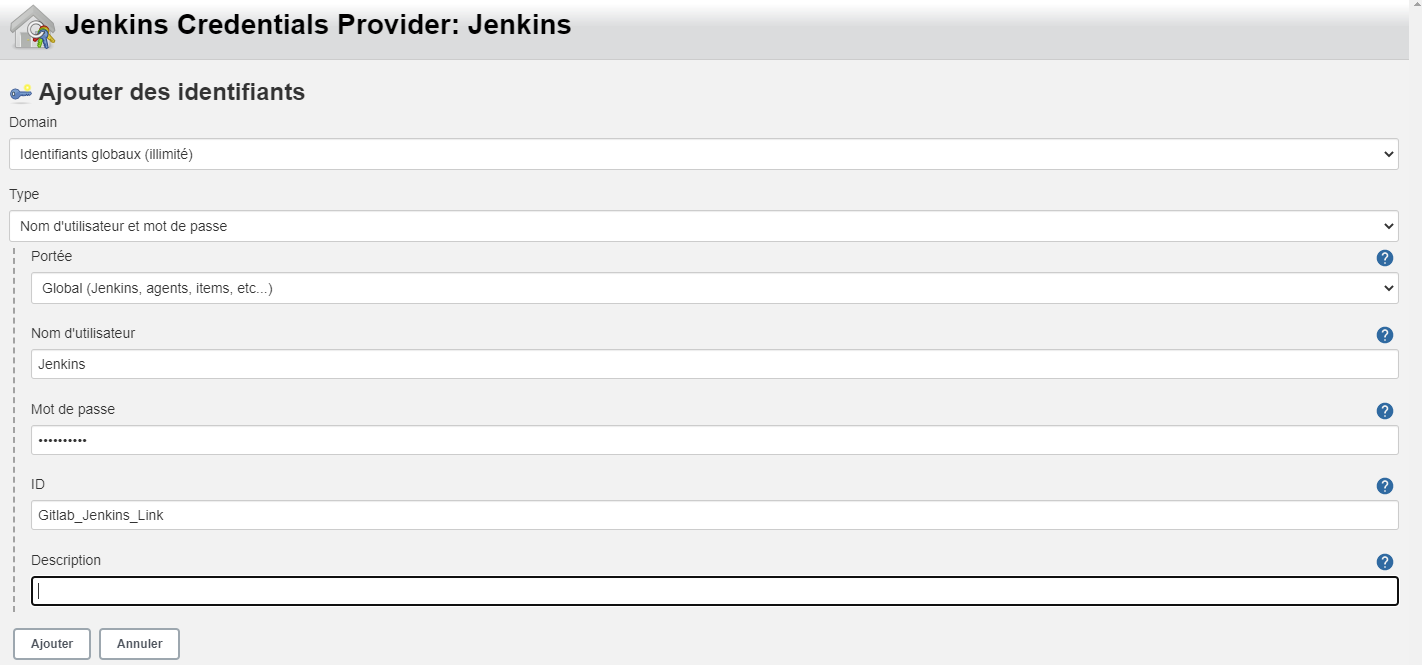


2/ Dans Jenkins, veuillez créer un compte en oubliant pas de :

* Valider Git
* Saisir l’adresse du projet GitLab



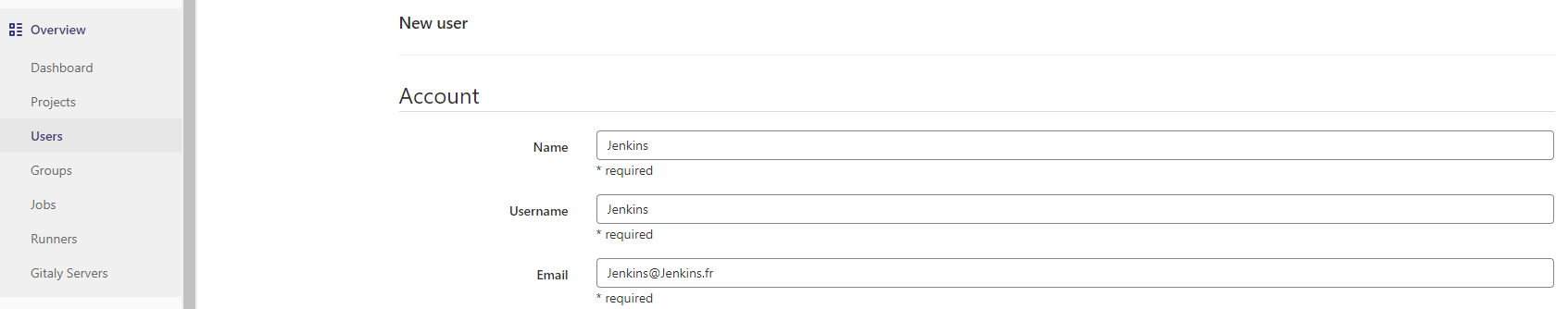
* Créer des crédentials si besoin



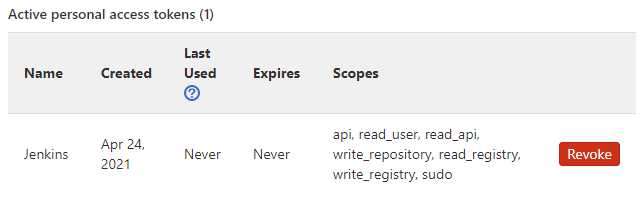
git config --system http.sslVerify false

git config --global http.sslVerify false

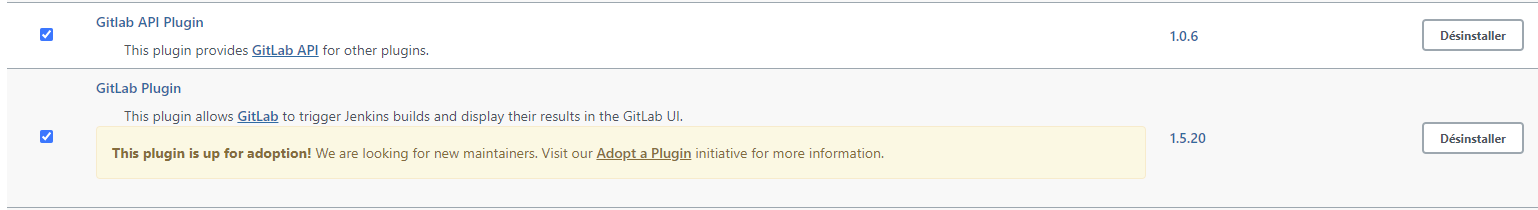
1/ Sur la page GitLab, création d’un compte pour l’association avec Jenkins



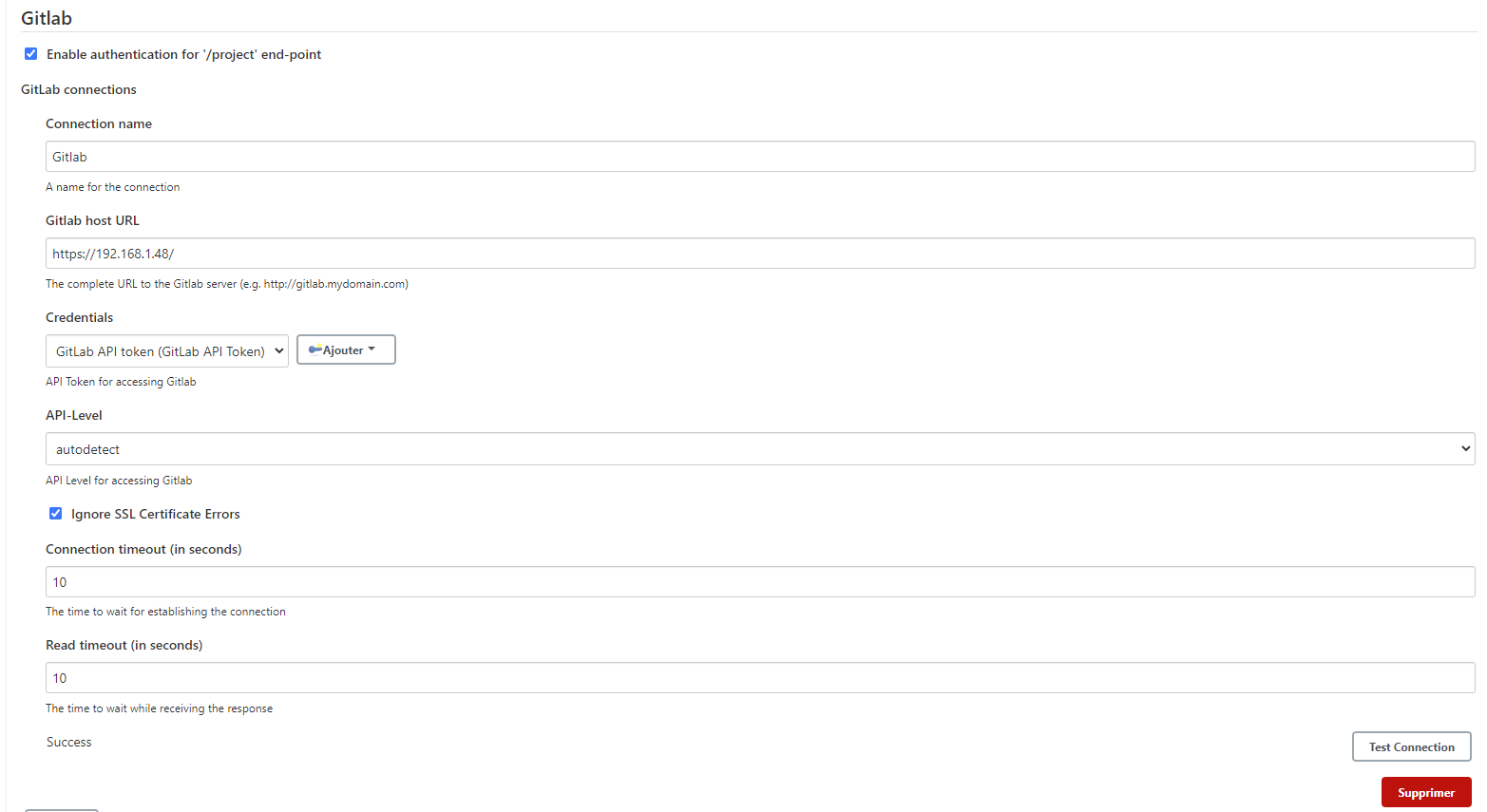
Création d’untoken à destination de jenkins.



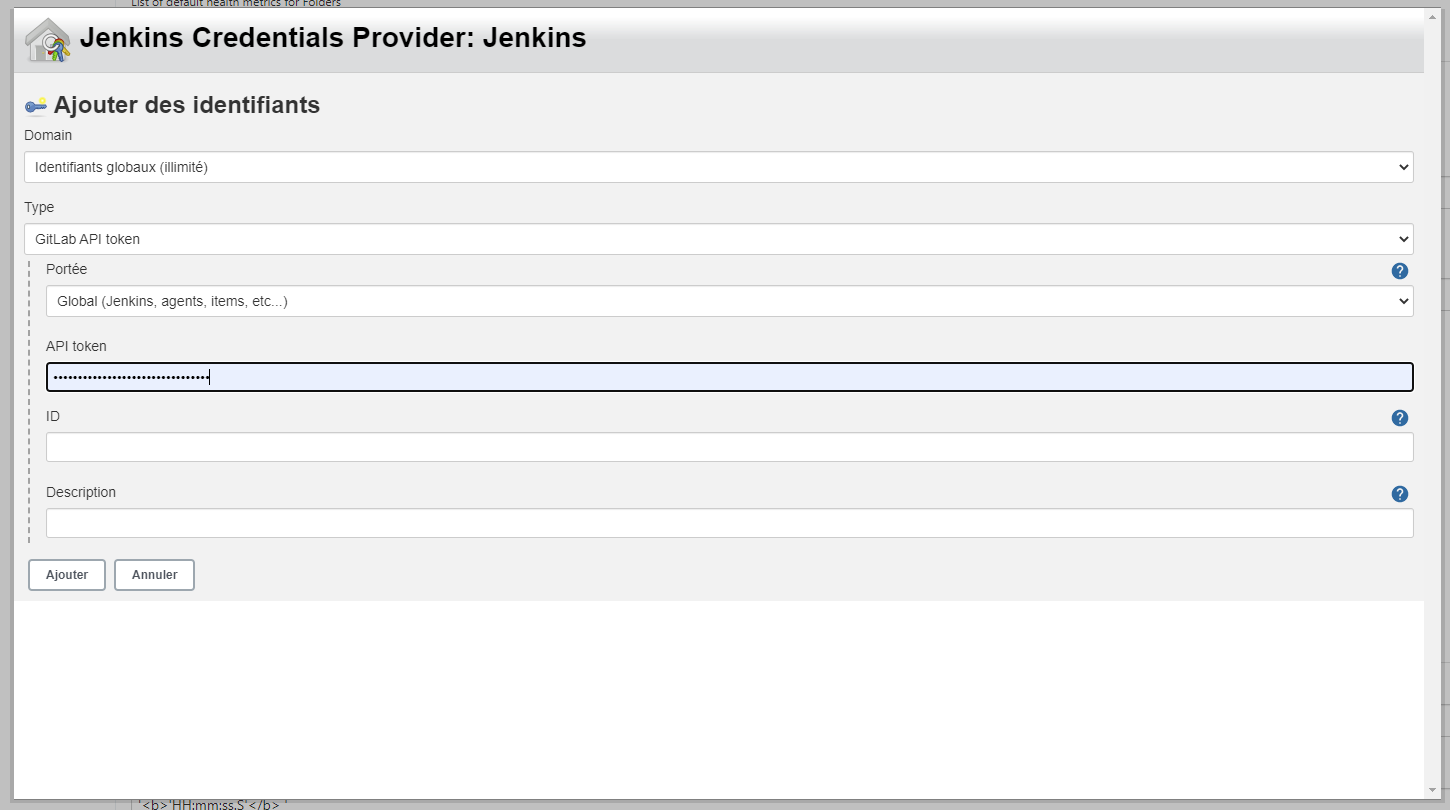
2/ Dans Jenkins, installer les plugins GitLab.

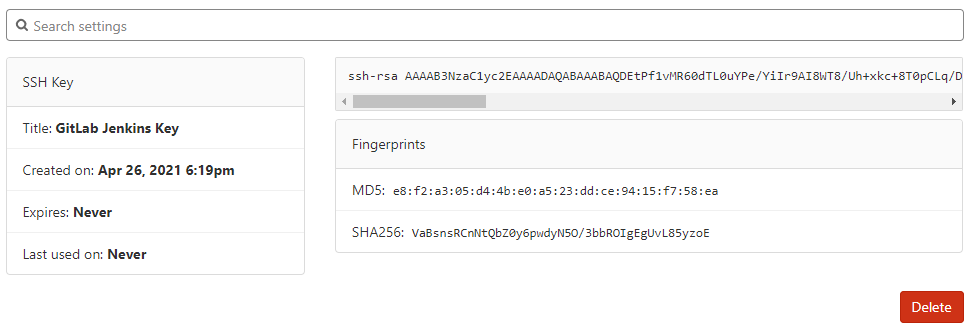


3/ Puis dans Jenkins, déclarer des credentials avec le GitLab API Token.



Dans la page de configuration, sous la rubrique GitLab, ajouter à jenkins un GitLab API token.

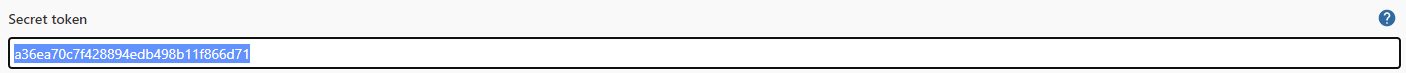




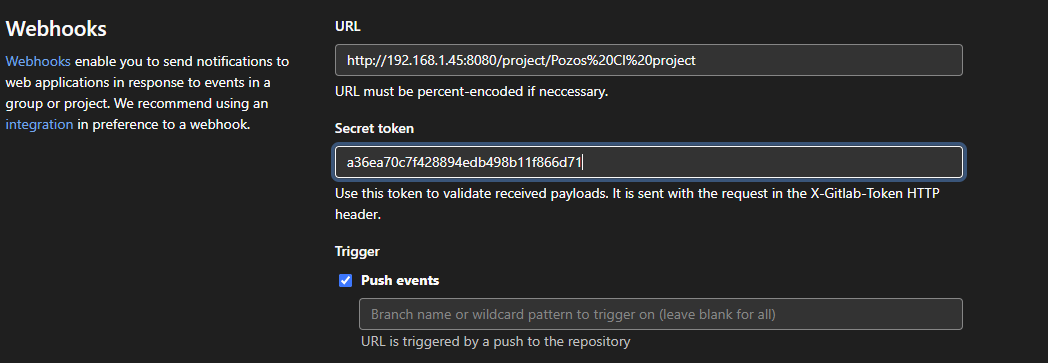
Creation du webhook pour lancer la CI sur push

Dans Jenkins :

Creation d’un token dans la rubrique « Ce qui déclenche le build »



Puis dans GitLab



# L’intégration continue

Première chose : Installer le plugin Docker dans Jenkins.

Vous trouverez ci-dessous l’ensemble des étapes demandés ainsi que les explications inhérentes à chaque configuration.

|  |  |
| --- | --- |
| Etape 1 et 2 : Lancement du pipeline CI lors d’un push sur GitLab | |
| Pour cela, dans le configuration du projet, il suffit de cocher la case « Build when a change is pushed to GitLab webhook » dans la rubrique « Ce qui déclence le build ».  Assurer vous que la case « push event » soit également cochée. | |
| La configuration |  |

|  |  |
| --- | --- |
| Etape 3 : Récupération du code sur GitLab | |
| Pour télécharger la source du repository GitLab, il suffit d’ajouter l’URL du projet dans la « Gestion de code source » | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 4 : Monter un container Docker via le Dockerfile simple\_api | |
| Puis-ce que le repo local est téléchargé intégralement lors de l’étape précédente dans la racine du container, nous pouvons monter notre image grâce à la commande build en navigant de façon classique dans les dossiers du « repo ». | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 5.1 : Démarrer le container | |
| Là encore, nous allons nous appuyer sur une commande docker classique : docker run. | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 5.2 : Lancer une commande curl pour s’assurer que l’application fonction | |
| Pour lancer une commande curl, il faut juste comprendre le principe suivant : notre application n’a pas d’accès à un réseau externe. Elle n’a donc pas d’adresse IP. Pour accéder à l’application, nous utiliserons donc <https://0.0.0.0:5000>. | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 6 : Utiliser CLAIR | |
| Nous montons la base de données hébergeant les vulnérabiltiées puis nous créons une attente le temps de s’assurer que le container et l’application soit opérationnelles.  Nous montons le scanner dans un container lié a sa base de données.  Puis nous exécutons un test sur le container qui vient d’être créé. | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 7 : Pousser l’image sur le Registry | |
| Pour pousser l’image sur le Registry, il faut  1/ Tager l’image avec l’url du Registry et un identifiant unique  2/ Pousser l’image à l’aide de la commande push | |
| La configuration |  |
| Le résultat console |  |

### Etape 8 : Déploiement de l’application via Ansible

1/ Installer localement Ansible via la commande yum install ansible.

2/ Déclaration de l’hôte à Ansible via le fichier hosts dans le dossier /etc/ansible/hosts.

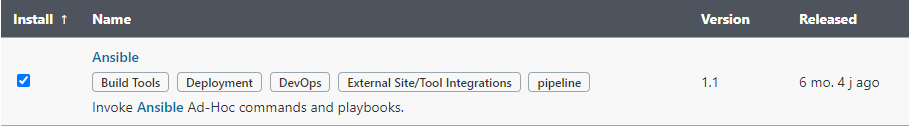
Dans le fichier, ajout de l’ip de la machine : 192.168.1.39

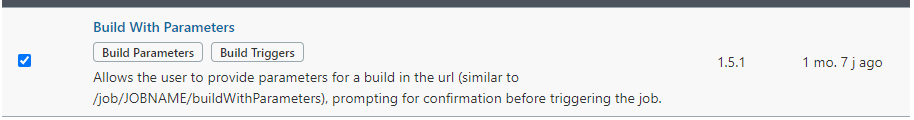
3/ Déclaration des clés SSH pour autoriser Ansible à administrer notre server.

ssh-keygen -t rsa -b 2048

ssh-copy-id [root@192.168.1.39](mailto:root@192.168.1.39)

4/ Installer les plugins Ansible et Build With Parameters

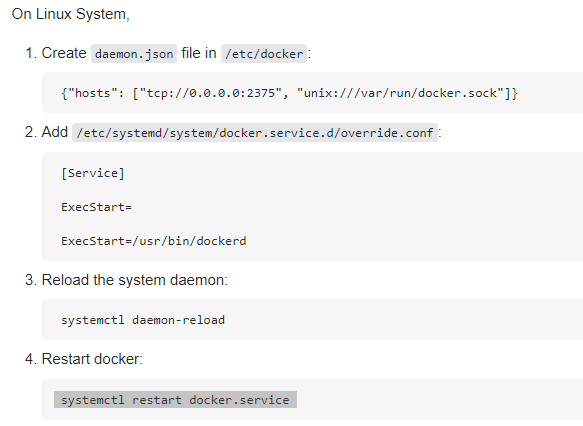






### Etape 9 : Lancer un test sécurité avec Arachni

**Tips : En cas d’erreur apach au lancement d’un build (montage de l’image) :**



Tips : Supprimer une image du registry

Lancer d’adresse http://<IP du registry>:<Port du registry>/v2/<nom\_de\_l’image>/tags/list

Puis lancer la commande : curl -v --silent -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET http://localhost:5000/v2/<name>/manifests/<tag> 2>&1 | grep Docker-Content-Digest | awk '{print ($3)}'

Puis lancer la commande curl -v --silent -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X DELETE http://127.0.0.1:5000/v2/<name>/manifests/sha256:6de813fb93debd551ea6781e90b02f1f93efab9d882a6cd06bbd96a07188b073

Yum install epel-release

Yum install jq

REGISTRY\_STORAGE\_DELETE\_ENABLED=true

registry='localhost:5000'

name='my-image'

curl -v -sSL -X DELETE "http://${registry}/v2/${name}/manifests/$(

curl -sSL -I \

-H "Accept: application/vnd.docker.distribution.manifest.v2+json" \

"http://${registry}/v2/${name}/manifests/$(

curl -sSL "http://${registry}/v2/${name}/tags/list" | jq -r '.tags[0]'

)" \

| awk '$1 == "Docker-Content-Digest:" { print $2 }' \

| tr -d $'\r' \

)"

# Monitoring

## System monitoring

Datadog est une solution de monitoring via une plateforme cloud.

Pour créer le monitoring, il faut lier un environnement (dans notre cas, docker) avec l’application cloud via une clé.

|  |  |
| --- | --- |
| Etape 1 : Création du compte | |
| Création d’un compte sur <https://app.datadoghq.eu/> | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 2 : Création du compte | |
| Déclaration de l’environnement, nous créons un API\_KEY nommé Pozos. | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 3 : Création du compte | |
| Nous installons l’agent datadog sur la VM à contrôler. | |
| La configuration | docker run -d --name dd-agent -v /var/run/docker.sock:/var/run/docker.sock:ro -v /proc/:/host/proc/:ro -v /sys/fs/cgroup/:/host/sys/fs/cgroup:ro -e DD\_API\_KEY=632e89622c4544ecc719c4ecdf51dd9f -e DD\_SITE="datadoghq.eu" gcr.io/datadoghq/agent:7 |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 4 : Exploitation | |
| Dans le menu *Dashboard* du compte cloud Datadog, nous retrouvons directement notre environement avec 3 containers Docker :   * Le container Flask * Le container Website * Le container agent Datadog (dd-agent) | |
| Le résultat |  |

## Software metrics

## Log monitoring

Splunk est également une solution de monitoring via une plateforme cloud.

|  |  |
| --- | --- |
| Etape 1 : Création du compte | |
| Création d’un compte sur <https://www.splunk.com/fr_fr> en choisissant de tester la version cloud.  Puis allez dans la rubrique DevOps puis Log Observer. | |
| La configuration |  |
| Le résultat console |  |