**Florian Vigier 317918 – Adrien Gauthier 288321 – Aude Grantais 281205**

**POZOS Companie**

**4DVOP - Rendu de projet**



Table des matières

[Préambule 2](#_Toc72318354)

[Objectif 3](#_Toc72318355)

[Infrastructure 4](#_Toc72318356)

[1. Prérequis 4](#_Toc72318357)

[2. Installation de jenkins 4](#_Toc72318358)

[3. Déploiement du serveur applicatif 5](#_Toc72318359)

[4. Installation du registry 7](#_Toc72318360)

[5. Installation du Code Repository 8](#_Toc72318361)

[Les configurations 10](#_Toc72318362)

[1. Jenkins et Gitlab 10](#_Toc72318363)

[2. Jenkins et Docker 13](#_Toc72318364)

[3. Jenkins et Ansible 13](#_Toc72318365)

[L’intégration continue 14](#_Toc72318366)

[Etape 1 et 2 : Lancement du pipeline CI lors d’un push sur GitLab 14](#_Toc72318367)

[Etape 3 : Récupération du code sur GitLab 14](#_Toc72318368)

[Etape 4 : Monter un container Docker via le Dockerfile simple\_api 14](#_Toc72318369)

[Etape 5.1 : Démarrer le container 15](#_Toc72318370)

[Etape 5.2 : Lancer une commande curl pour s’assurer que l’application fonction 15](#_Toc72318371)

[Etape 6 : Utiliser CLAIR 16](#_Toc72318372)

[Etape 7 : Pousser l’image sur le Registry 17](#_Toc72318373)

[Etape 8 : Déploiement de l’application via Ansible 17](#_Toc72318374)

[Etape 9.1 : Lancer un test sécurité avec Arachni 18](#_Toc72318375)

[Etape 9.2 : Test de l’application via Arachni scanner 19](#_Toc72318376)

[Monitoring 20](#_Toc72318377)

[1. Installation de la solution Datadog 20](#_Toc72318378)

[2. Monitoring logiciel 22](#_Toc72318379)

[3. Monitoring matériel 23](#_Toc72318380)

[4. Log monitoring 24](#_Toc72318381)

[Bonus Stage 25](#_Toc72318382)

[Etape 1 et 2 : Lancement du pipeline CI lors d’un push sur GitLab 25](#_Toc72318383)

[Etape 3 : Récupération du code sur GitLab 25](#_Toc72318384)

[Etape 4 : Monter un container Docker via le Dockerfile simple\_api 26](#_Toc72318385)

[Etape 5.1 : Démarrer le container 26](#_Toc72318386)

[Etape 5.2 : Lancer une commande curl pour s’assurer que l’application fonction 26](#_Toc72318387)

[Etape 6 : Utiliser CLAIR 27](#_Toc72318388)

[Etape 7 : Pousser l’image sur le Registry 28](#_Toc72318389)

[Etape 8 : Déploiement de l’application via Ansible 28](#_Toc72318390)

[Etape 9 : Test de l’application avec Arachni 29](#_Toc72318391)

# Préambule

Ce rendu de projet est composé d’une liasse de document. Cette liasse est organisée de la façon suivante :

**Dossiers**

**Descriptions**

**Fichiers**

# 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 déploie 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**

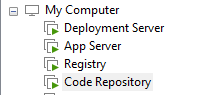
Permet de stocker et de mettre à disposition le code générant l’environnement

****

# 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

1. **Conception**

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

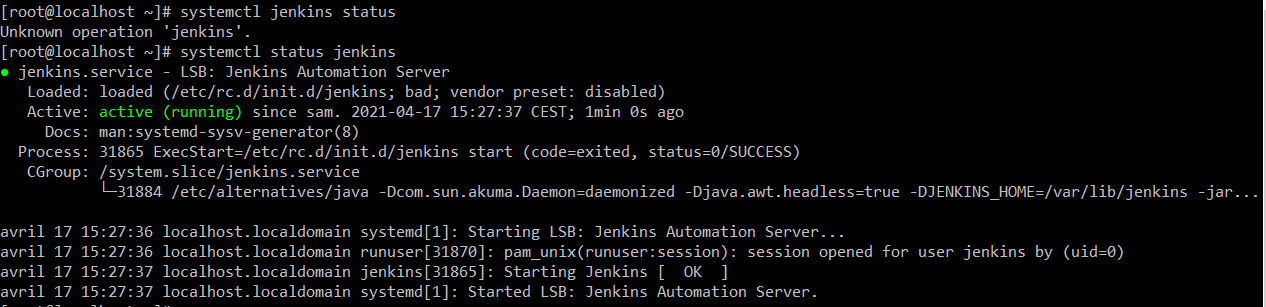
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

sudo yum upgrade

sudo yum install jenkins java-1.8.0-openjdk-devel

sudo systemctl daemon-reload

Le résultat 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

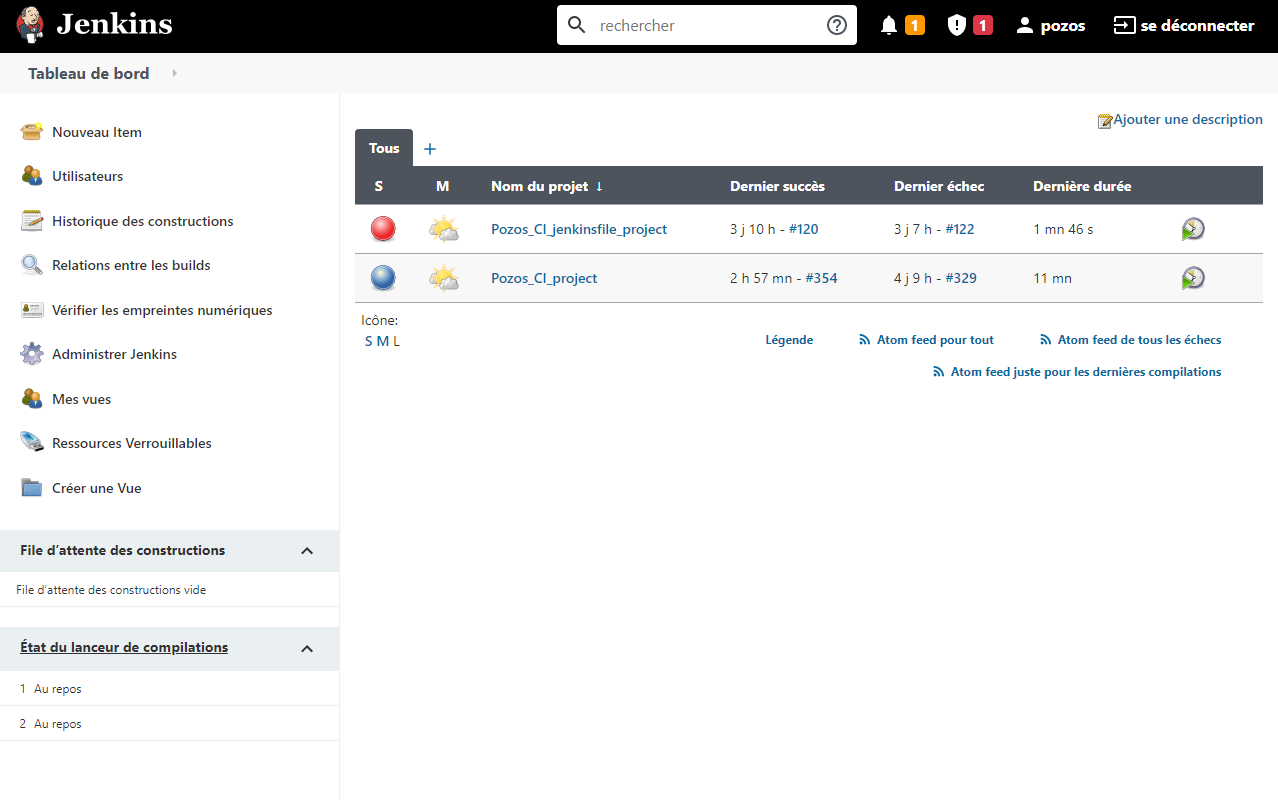
firewall-cmd $PERM --add-service=jenkins

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

firewall-cmd –reload

1. **Utilisation**

*Dans votre navigateur, accédez à l’interface de Jenkins via l’URL de la machine virtuel : dans notre cas :* [*http://192.168.1.45:8080*](http://192.168.1.45:8080)



## Déploiement du serveur applicatif

1. **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

1. **Utilisation**

Téléchargez le dossier de travail via la commande

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

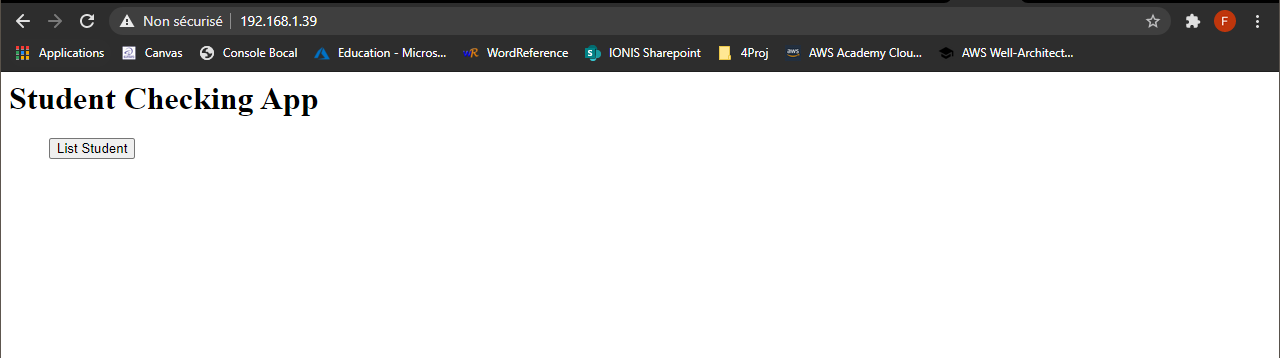
Puis, vous devrez vous rendre dans le dossier télécharger puis dans le dossier App\_Server

cd 4DVOP\_POZOS/App\_Server

Puis lancer la commande

docker-compose up

*Dans votre navigateur, accédez à l’interface de Jenkins via l’URL de la machine virtuel : dans notre cas :* [*http://192.168.1.39*](http://192.168.1.39)



## 

## Installation du registry

1. **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

1. **Utilisation**

Téléchargez le dossier de travail via la commande

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

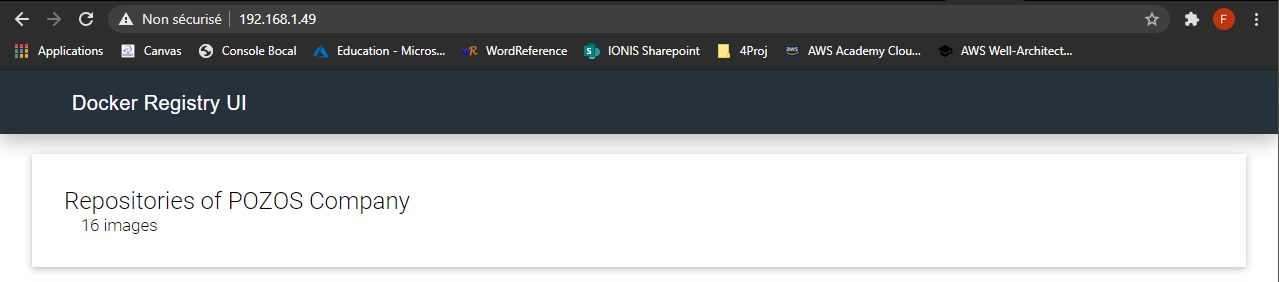
Puis, vous devrez vous rendre dans le dossier télécharger puis dans le dossier Registry

cd 4DVOP\_POZOS/Registry

Puis lancer la commande

docker-compose up

*Dans votre navigateur, accédez à l’interface de Jenkins via l’URL de la machine virtuel : dans notre cas :* [*http://192.168.1.49*](http://192.168.1.49)



## Installation du Code Repository

1. **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.

1. **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 :

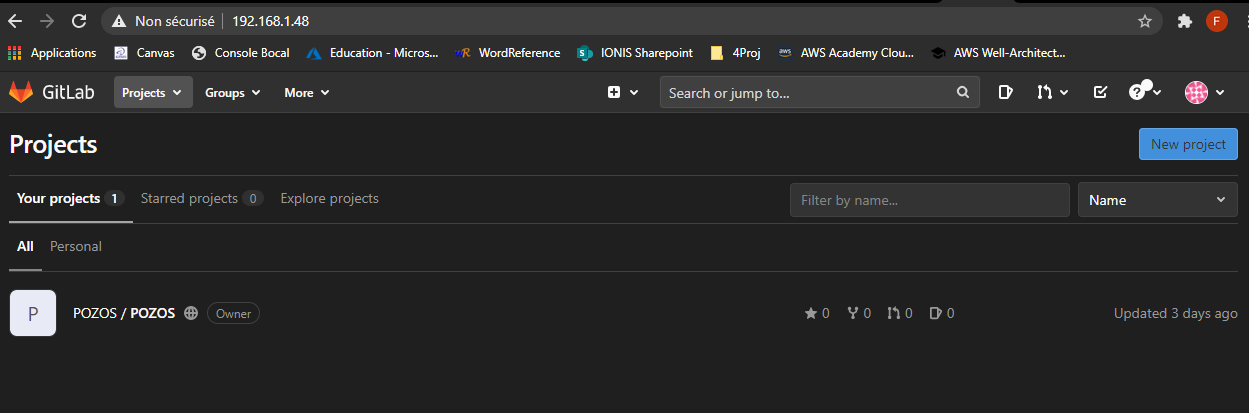


1. **Utilisation de GitLab**

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

* User : Root
* Password : a définir.

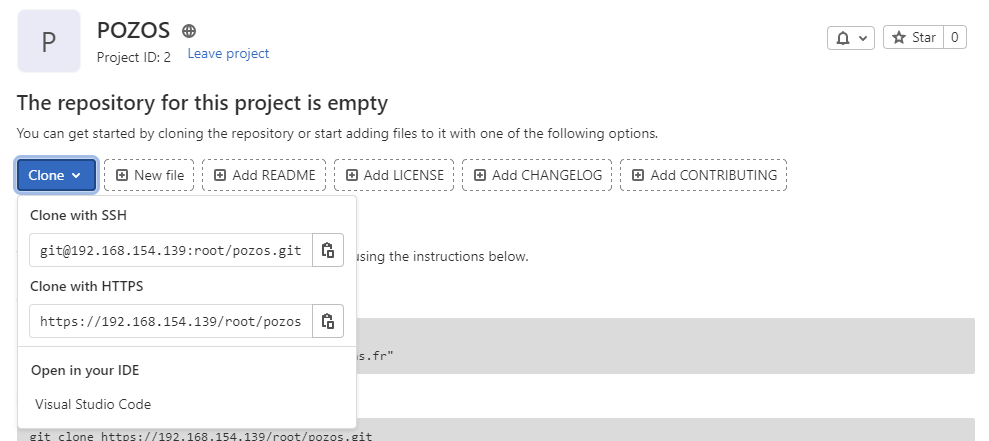
*Dans votre navigateur, accédez à l’interface de Jenkins via l’URL de la machine virtuel : dans notre cas :* [*http://192.168.1.48*](http://192.168.1.48)



# Les configurations

## Jenkins et Gitlab

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

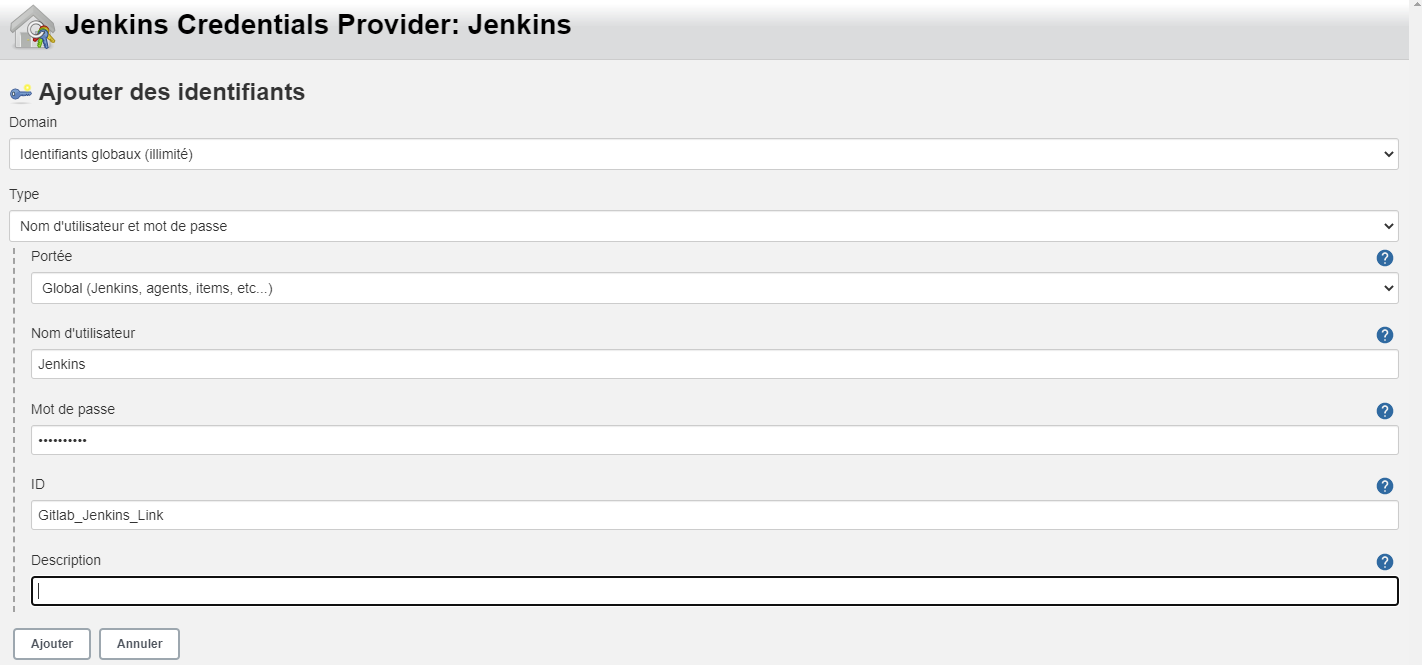


2/ Dans Jenkins, veuillez créer un compte en n’oubliant pas de :

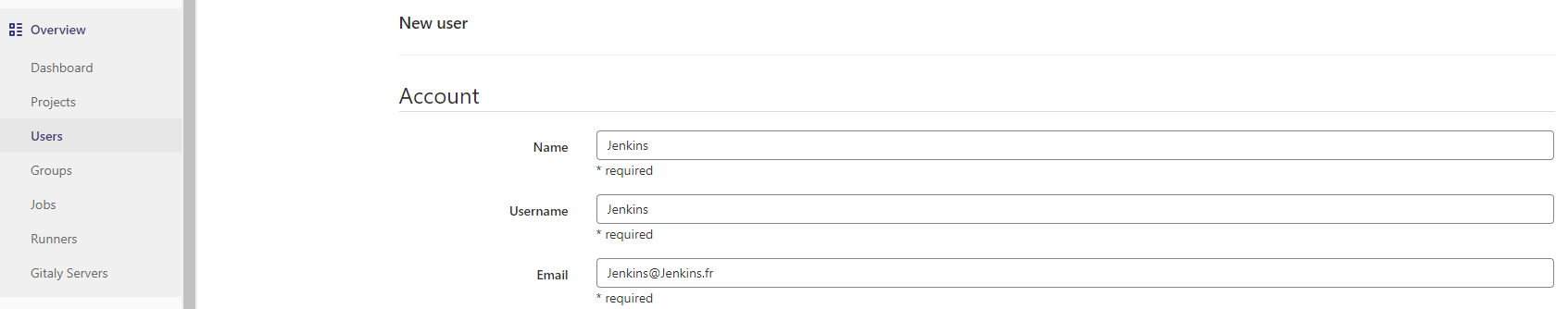
* Valider Git
* Saisir l’adresse du projet GitLab



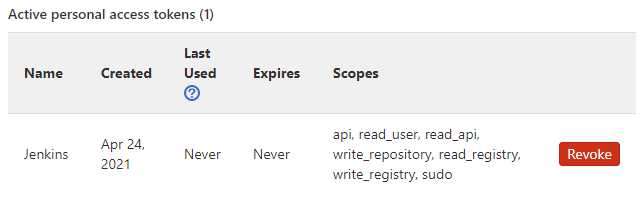
* Créer des crédentials Gitlab si besoin.



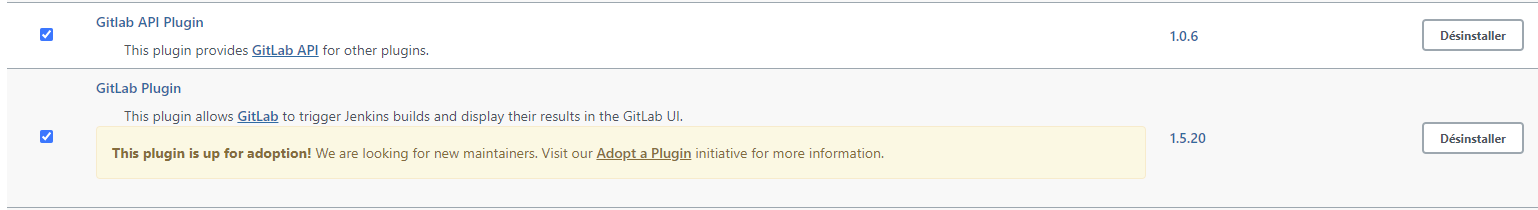
3/ Sur la page GitLab, création d’un compte dédié à l’association avec Jenkins afin de ne pas lier le compte root.



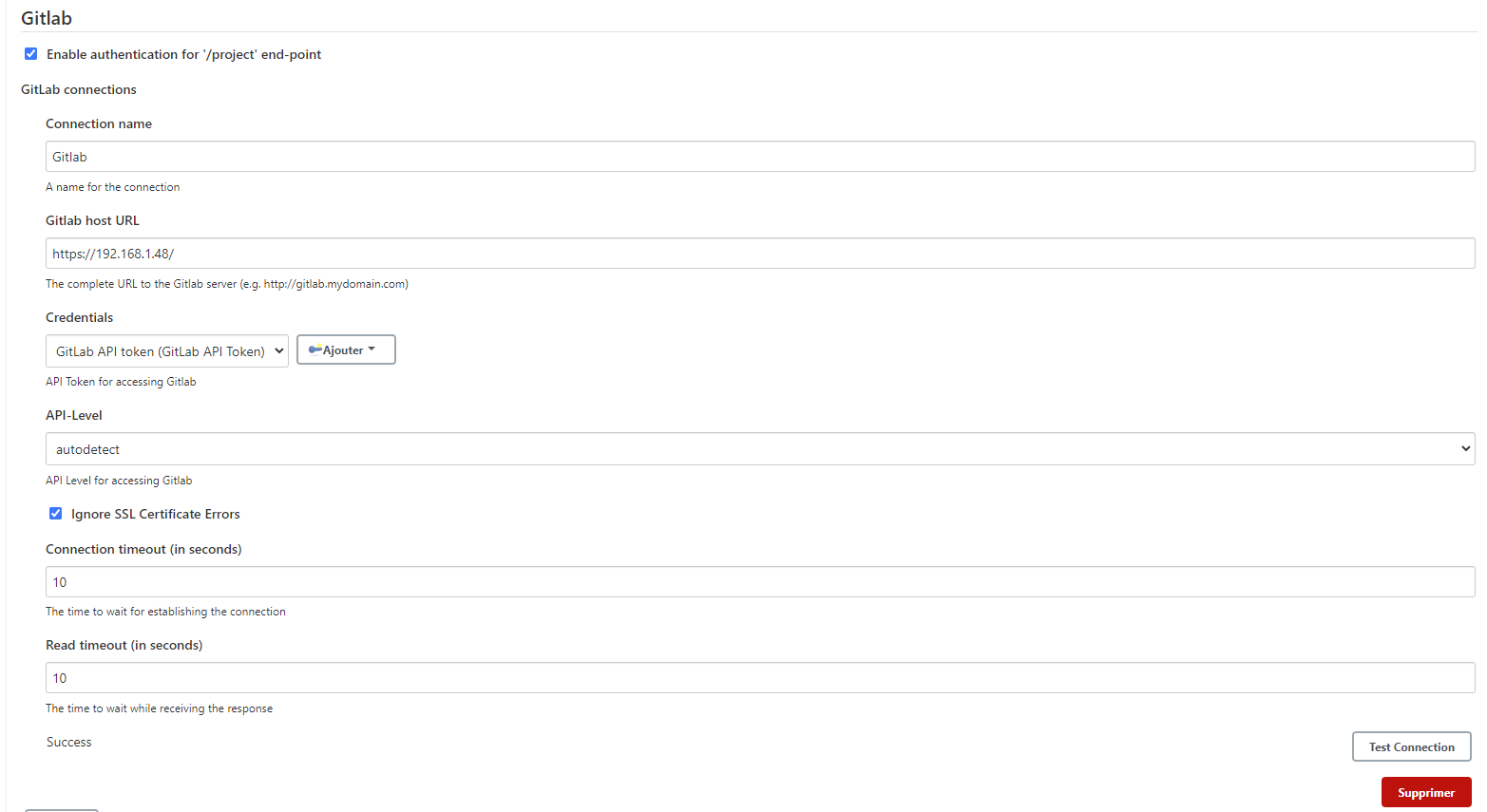
4/ Création d’un token à destination de jenkins.



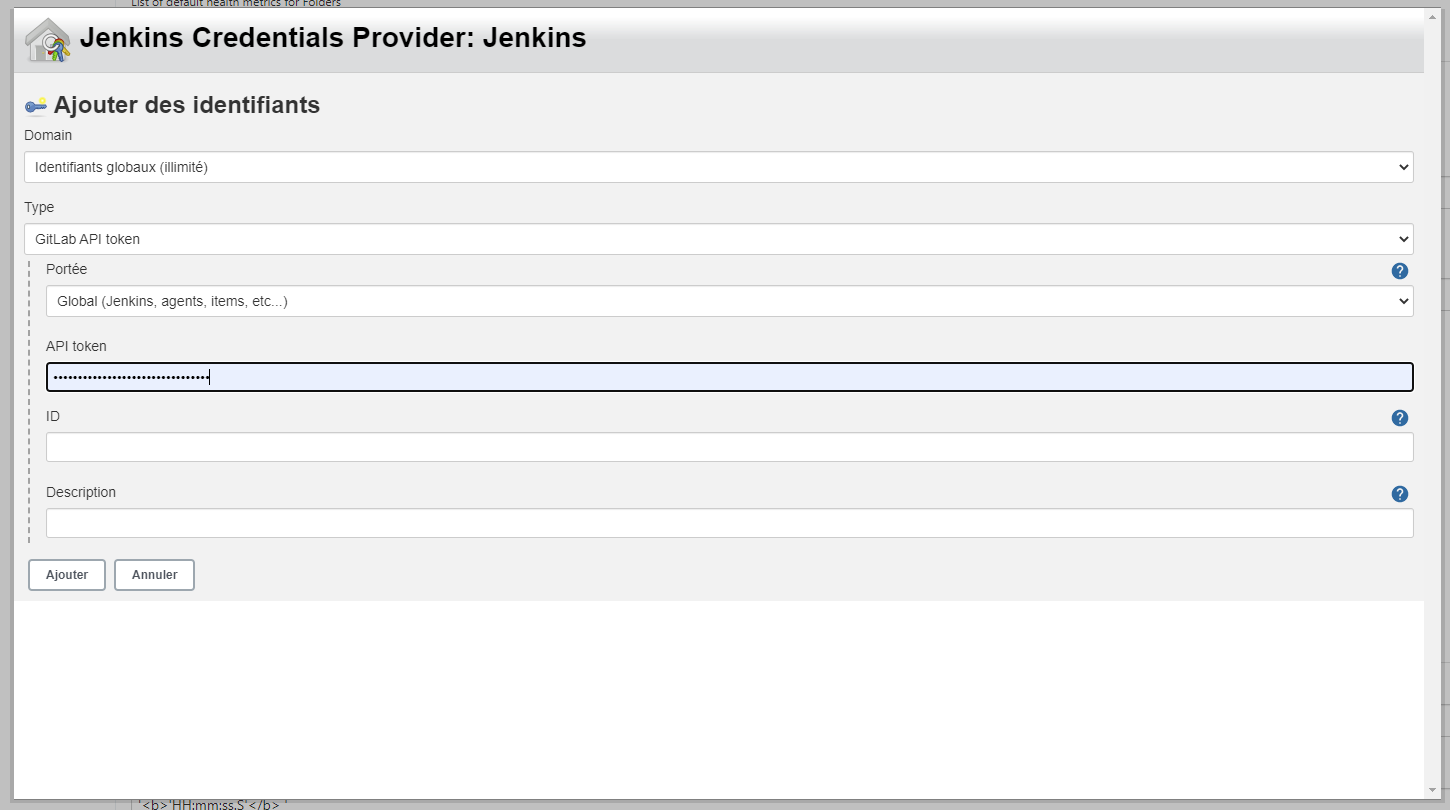
5/ Dans Jenkins, installer les plugins GitLab.



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

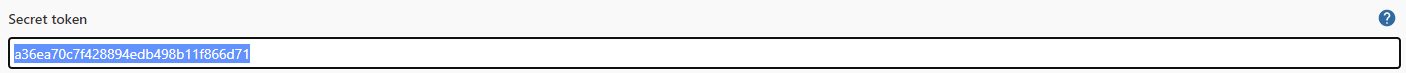


7/ Dans la page de configuration, sous la rubrique GitLab, ajouter à jenkins un GitLab API token.

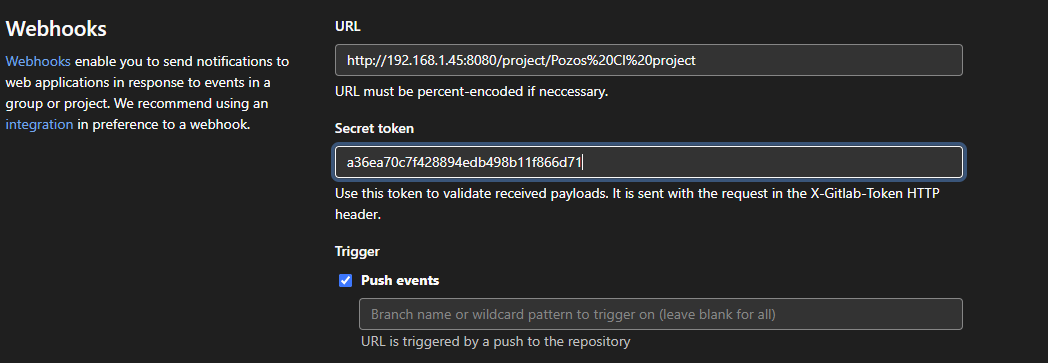


8/ Création du webhook pour lancer la CI sur push

Dans Jenkins, création d’un secret token dans la rubrique « Ce qui déclenche le build »



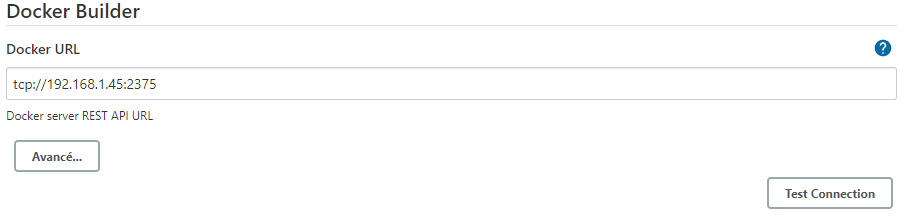
Puis dans GitLab, dans la rubrique webhook, ajouter l’URL du projet Jenkins ainsi que le secret token.



Voilà, Jenkins et Gitlab sont paramétrés pour fonctionner ensemble.

## Jenkins et Docker

Afin de permettre à Jenkins l’utilisation du plugin Docker, ne pas oublier de paramétrer le Docker Builder dans les paramètres Jenkins en saisissant l’URL de la machine supportant Docker avec le port 2375.



## Jenkins et Ansible

Afin de se servir l’Ansible pour le déploiement d’application, il sera nécessaire d’installer Ansible localement et utile d’installer le plugin afin de faire le lien avec l’API.

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 de production : 192.168.1.39

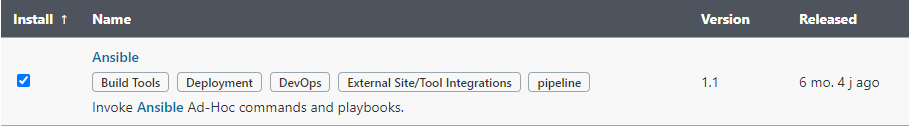


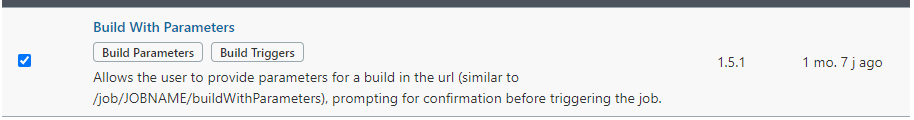
3/ Déclaration des clés SSH pour autoriser Ansible à administrer notre server.

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





Voilà, nous sommes donc prêt à déployer des applications.

# 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 vulnérabilités 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 à 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 |  |

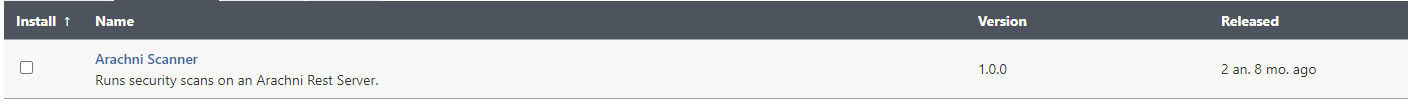
|  |  |
| --- | --- |
| Etape 8 : Déploiement de l’application via Ansible | |
| Utiliser le bloc invoke Ansible playbookpour lancer le playbook Ansible. | |
| La configuration |  |
| Le résultat console |  |

### Etape 9.1 : Lancer un test sécurité avec Arachni

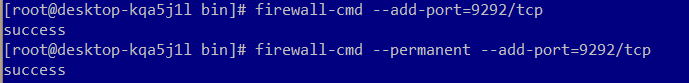
**1/ installation du server Arachni via le dépo officiel**

Wget [**https://github.com/Arachni/arachni/releases/download/v1.5.1/arachni-1.5.1-0.5.12-linux-i686.tar.gz**](https://github.com/Arachni/arachni/releases/download/v1.5.1/arachni-1.5.1-0.5.12-linux-i686.tar.gz)

**2/ Installer le plugin Arachni dans Jenkins**



**3/ Autoriser les communciations TCP sur le port 9292 de la VM.**



|  |  |
| --- | --- |
| Etape 9.2 : Test de l’application via Arachni scanner | |
| Utilisation du commande shell en remote host sur le server de production. | |
| La configuration |  |
| Le résultat console |  |

# Monitoring

## **Installation de la solution Datadog**

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 : Accès au site de datadog** |
| Création d’un compte sur <https://app.datadoghq.eu/> |

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

|  |  |
| --- | --- |
| **Etape 3 : Pour le monitoring des images dockers** | |
| Nous installons l’agent datadog sur les VM à contrôler et qui utilisent Docker. | |
| La configuration | docker run -d --name dd-agent --restart always -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 |

|  |  |
| --- | --- |
| **Etape 4 : Pour le monitoring de l’application Gitlab** | |
| Nous installons l’agent datadog sur la VM à contrôler et qui utilisent Gitlab. | |
| La configuration | 1/ Activer kes Metrics Prometheus dans la partie Admin / Settings / Metrics and Profiling.  2/ Installation de l’agent Datadog sur la machine Code\_Repository via la commande :  DD\_AGENT\_MAJOR\_VERSION=7 DD\_API\_KEY=632e89622c4544ecc719c4ecdf51dd9f DD\_SITE="datadoghq.eu" bash -c "$(curl -L <https://s3.amazonaws.com/dd-agent/scripts/install_script.sh>)"  3/ xxx |

|  |  |
| --- | --- |
| **Etape 5 : Pour le monitoring de l’application Jenkins** | |
| Nous installons l’agent datadog sur la VM à contrôler et qui utilisent Jenkins. | |
| La configuration | 1/ Changer le port de flask (5000 > 5005) car l’agent datadog utilise également le port 5000.  2/ Installation de l’agent Datadog sur la machine Deployment\_Server via la commande :  DD\_AGENT\_MAJOR\_VERSION=7 DD\_API\_KEY=632e89622c4544ecc719c4ecdf51dd9f DD\_SITE="datadoghq.eu" bash -c "$(curl -L <https://s3.amazonaws.com/dd-agent/scripts/install_script.sh>)"  3/ Dans la page d’administration, choisir l’option « Use Datadog Agent … » |

## **Monitoring logiciel**

|  |  |
| --- | --- |
| **Etape 1 : Exploitation** | |
| Dans le menu *Dashboard* du compte cloud Datadog, nous avons créé un dashboard regroupant les données de notre infrastructure. Elles comprend les containers Docker, Jenkins et Gitlab :   * Le container Flask * Le container Website * Les containers agent Datadog (xxx-ddAgent) * Les containers Registry * Jenkins * Gitlab   Nous retrouvons comme métriques   * Activité * Nombre d’instances démarrés / arrêtés * Historique d’évènement * Temps de fonctionnement | |
| Le résultat Docker et Jenkins |  |

## **Monitoring matériel**

|  |  |
| --- | --- |
| **Etape 1 : Exploitation** | |
| Dans le menu *Dashboard* du compte cloud Datadog, nous retrouvons notre dashboard avec les informations matériels choisis.  Ce sont ici, des données matériel (dans notre cas, le PC qui héberge les VMs). Il n’est cependant pas possible de monitorer des machines virtuelles VMware Workstation.  Nous retrouvons comme métriques   * Utilisation CPU * Activité réseau * Utilisation mémoire | |
| Le résultat |  |

## 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 |  |

# Bonus Stage

Il s’agit ici de reproduire notre travail lors du Pipeline Component, Intégration and build mais non plus via l’interface Jenkins, mais directement via un fichier Jenkinsfile.

Pour ce faire, nous créons un nouveau projet dans Jenkins mais à la place d’un projet freestyle, nous choisissons un pipeline.

|  |  |
| --- | --- |
| 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 via le Dockerfile précédent construit, grâce à la commande docker build.  Ps : L’arrêt et la suppression d’un container existant ne sert qu’en cas d’erreur lors d’un build précédent. | |
| 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 tester notre application, nous réutilisons la commande curl. | | |
| La configuration |  | |
| Le résultat console |  | |
| Etape 6 : Utiliser CLAIR | | |
| Nous montons la base de données hébergeant les vulnérabilités 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 à 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  3/ Supprimer l’image locale | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 8 : Déploiement de l’application via Ansible | |
| Utiliser commande ansiblePlaybook en lui donnant comme paramètre le chemin d’accés au playbook et la variable d’environnement du build. | |
| La configuration |  |
| Le résultat console |  |

|  |  |
| --- | --- |
| Etape 9 : Test de l’application avec Arachni | |
| Lancement de l’outil Arachni localement sur la machine de déploiement via l’agent ssh uilisant une clé SSH préalablement saisie dans les credentials Jenkins. | |
| La configuration |  |
| Le résultat console | **…** |