

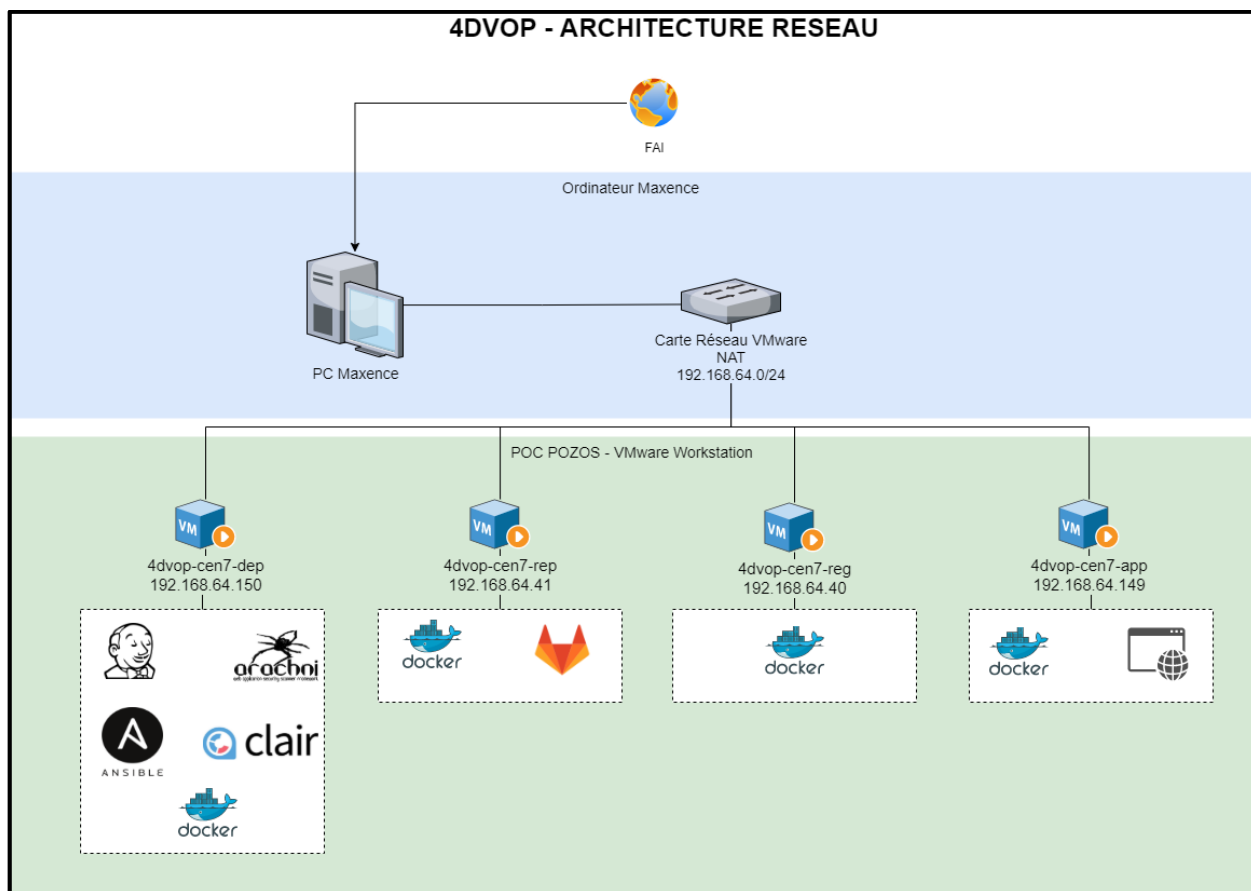
4DVOP - PROJET

Durée du projet : 1 mois

SOMMAIRE

2020-2021 - SUPINFO.....	1
SOMMAIRE	2
I. Architecture réseau	3
II. Contexte.....	3
III. Mise en œuvre du pipeline	4
A. Connexion aux différents environnements.....	4
B. Installation de Jenkins	4
C. Installation et configuration de GitLab on-premise	5
D. Construction de l'image docker via le Dockerfile du simple api	8
E. Démarrage de l'image et test de l'application	9
F. Scan de l'image simple api avec les outils de Clair.....	9
G. Push de l'image sur notre registre privé Docker	10
H. Déploiement de notre application	10
I. Test de sécurité de notre application avec Arachni.....	12
IV. Résultat du pipeline.....	14
A. Clone Gitlab.....	14
B. Build de l'image	15
C. Test de l'application.....	16
D. Scan avec Clair	16
E. Push de l'image sur le registre privé	17
F. Déploiement de l'application avec Ansible	17
G. Test avec Arachni.....	17

I. Architecture réseau



II. Contexte

Après le rendu de notre application conteneurisé pour le cours de 4DOKR, la DSI a été satisfaite de notre travail. Le département d'innovation souhaite améliorer son infrastructure existante en intégrant du CI/CD à travers une pipeline Jenkins qui permettrait de déployer, faire des tests et configurer les différents environnements.

Pour répondre à ce besoin nous avons 4 machines virtuelles :

- Un serveur de déploiement où Jenkins, Ansible, Clair et Arachni sont installés. Ces différents rôles permettront la création d'un pipeline, le déploiement des environnements et des tests de sécurité de nos applications (qui seront conteneurisés).
- Un serveur d'application qui hébergera notre application à travers un container accessible via une url sur le réseau.
- Un serveur de registre avec un container docker qui stockera les images de notre application. On pourra alors lui envoyer nos images et les récupérer.
- Un serveur d'hébergement avec gitlab d'installé de sorte à pouvoir centraliser nos ressources/ fichiers liés aux environnements. On y trouvera des fichiers de configurations (docker, ansible) et du code.

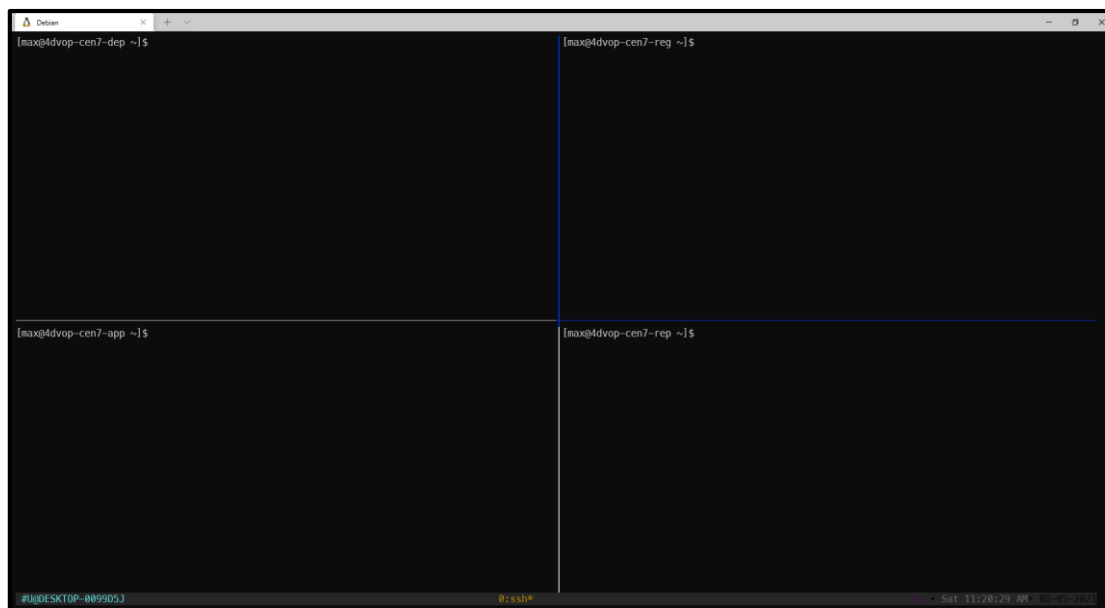
Nous travaillons sur des environnements Centos 7 comme demandé dans le cahier des charges.

III. Mise en œuvre du pipeline

A. Connexion aux différents environnements

Pour notre environnement nous utiliserons VMware Workstation où nous utiliserons une interface virtuelle de type NAT avec le réseau 192.168.64.xxx/24. Sa passerelle est 192.168.64.2. Chaque VM contiendra une carte réseau de type NAT sur ce réseau. Nous avons alloué des adresses IP Statiques pour éviter le changement via le DHCP de VMware.

Nous avons utilisé TMUX pour gérer les différentes connexions ssh en même temps.



Nos machines virtuelles ont la nomenclature suivante : **COURS-DISTRIBUTION-ROLE** par exemple **4dvop-cen7-dep**.

B. Installation de Jenkins

Nous avons utilisé Jenkins via le docker-compose qui nous avons été communiqué à savoir :

```
version: '2'
services:
  jenkins:
    # image: 'bitnami/jenkins:2'
    image: dirane/jenkins-docker-ansible-clair
    ports:
      - '80:8080'
      - '443:8443'
      - '50000:50000'
    privileged: true
    volumes:
      - 'jenkins_data:/bitnami'
      - '/var/run/docker.sock:/var/run/docker.sock'
volumes:
  jenkins_data:
    driver: local
```

Le conteneur contient Jenkins, Clair et d'autres utilitaires/plugins que nous utiliserons dans la suite.

C. Installation et configuration de GitLab on-premise

Pour installer gitlab il y avait deux possibilités, la première était de l'installer sur le serveur **4dvop-cen7-rep** et la seconde était d'utiliser un container gitlab sur le serveur **4dvop-cen7-rep**.

Nous détaillerons les deux installations/configurations mais pour des raisons de performances nous avons retenu la première solution.

Installation sur le serveur :

```
sudo yum install -y curl policycoreutils-python 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 yum install postfix

sudo systemctl enable postfix

sudo systemctl start postfix

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

sudo EXTERNAL_URL="https://https://192.168.64.41/" yum install -y gitlab-ee
```

Ensuite il suffit d'accéder à la page puis de se connecter avec le compte root.

L'ensemble de la documentation est disponible à cette adresse : [Download and install GitLab | GitLab](#)

Installation via un container docker :

```
web:
  image: 'gitlab/gitlab-ee:latest'
  restart: always
  hostname: 'gitlab.example.com'
  environment:
    GITLAB_OMNIBUS_CONFIG: |
      external_url 'http://192.168.64.41:8929'
      gitlab_rails['gitlab_shell_ssh_port'] = 2224
  ports:
    - '8929:8929'
    - '2224:22'
  volumes:
    - './config:/etc/gitlab'
    - './logs:/var/log/gitlab'
    - './data:/var/opt/gitlab'
```

```
sudo firewall-cmd --zone=public --add-port=443/tcp --permanent

sudo firewall-cmd --reload
```

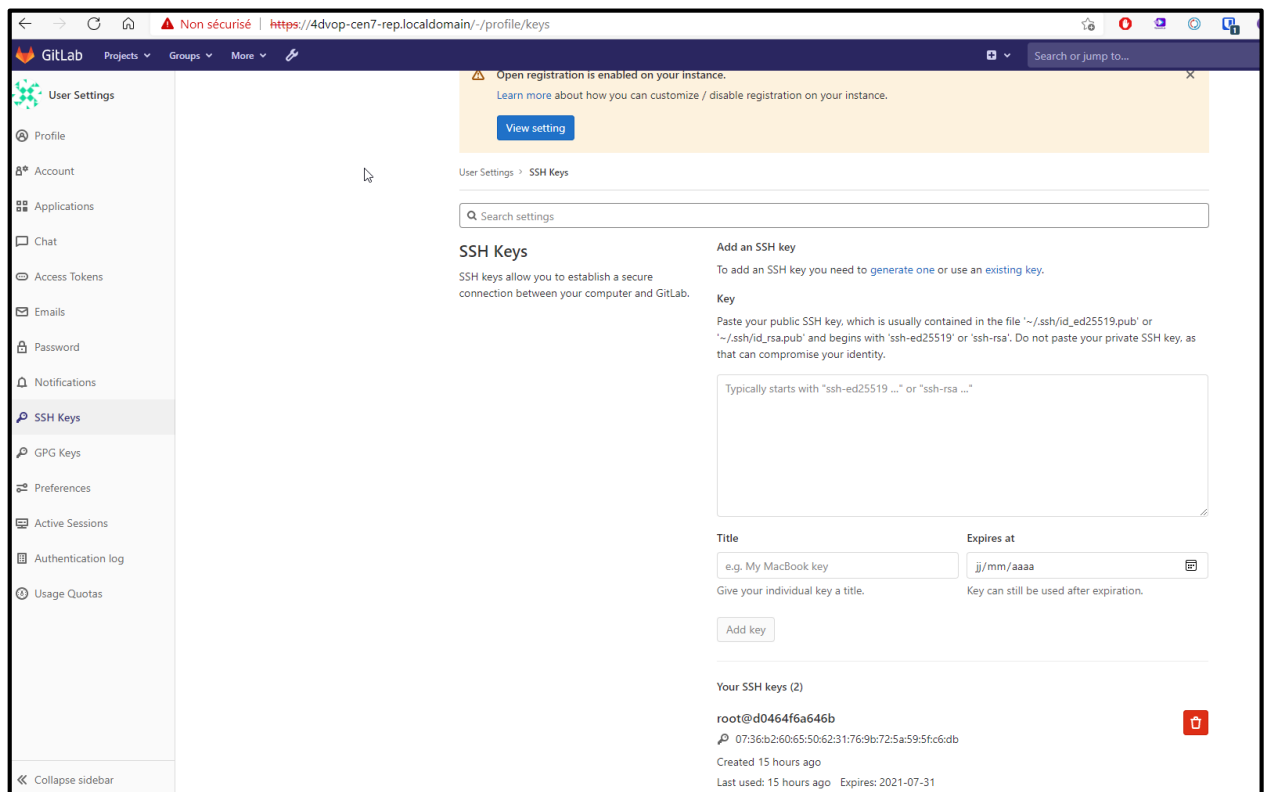
```
[max@4dvop-cen7-dep ~]$ sudo firewall-cmd --zone=public --add-port=443/tcp --permanent
[sudo] password for max:
success
```

L'objectif étant d'ouvrir les ports que ce soit via l'installation d'un container ou directement sur le serveur pour autoriser les connexions entrantes sur le port 443.

Nous avons ensuite dû générer une clé ssh dans le container Jenkins pour que le container puisse accéder au repository gitlab via ssh.

```
ssh-keygen
```

On copie ensuite la clé publique qu'on ajoute dans repository notre gitlab.



On ajoute ensuite les credentials sur Jenkins.

Portée	Global (Jenkins, agents, items, etc...)
ID	ssh-dep
Description	ssh-4dvop-cen7-dep
Username	root
Private Key	<div><input checked="" type="radio"/> Enter directly</div> <div><div>Key</div><div><div>Concealed for Confidentiality</div><div>Replace</div></div></div>
Passphrase	

Nous devons ensuite créer notre pipeline Jenkins et y ajouter le repository Gitlab.

Gestion de code source

☐ Aucune

☒ Git

Repositories

Repository URL

git@192.168.64.41:root/4dvop-project.git

Credentials

root (ssh-4dvop-cen7-dep) [Ajouter](#)

Avancé...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/master

Add Branch

Dans le cahier des charges il est demandé à ce que le repository informe Jenkins si un push est effectué. Si c'est le cas alors le pipeline se déclenche. Pour se faire on utilisera un webhook qui se configure de la manière suivante sur GitLab.

Open registration is enabled on your instance. [Learn more](#) about how you can customize / disable registration on your instance. [View setting](#)

Administrator > 4dvop-project > Webhook Settings > Webhook

Search settings

Webhook

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

http://192.168.64.150/project/4DVOP

URL must be percent-encoded if necessary.

Secret token

Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☒ **Push events**

Branch name or wildcard pattern to trigger on (leave blank for all)

URL is triggered by a push to the repository

☐ **Tag push events**

URL is triggered when a new tag is pushed to the repository

☐ **Comments**

URL is triggered when someone adds a comment

☐ **Confidential comments**

URL is triggered when someone adds a comment on a confidential issue

☐ **Issues events**

URL is triggered when an issue is created, updated, or merged

☐ **Confidential issues events**

URL is triggered when a confidential issue is created, updated, or merged

☐ **Merge request events**

URL is triggered when a merge request is created, updated, or merged

Dans les paramètres Jenkins on viendra indiquer l'url de notre Gitlab.

Dans le pipeline Jenkins que nous venons de créer il faudra indiquer ce qui déclenche le build de notre repo Gitlab.

The screenshot shows the Jenkins configuration page for project 4DVOP, specifically the 'Ce qui déclenche le build' (What triggers the build) tab. The page has a top navigation bar with tabs: General, Gestion de code source, Ce qui déclenche le build (selected), Environnements de Build, Build, and Actions à la suite du build. The main content area is titled 'Ce qui déclenche le build' and contains several sections of checkboxes and input fields. The first section, 'Déclencher les builds à distance', includes options like 'Déclencher les builds à distance (Par exemple, à partir de scripts)', 'Construire après le build sur d'autres projets', and 'Construire périodiquement'. The second section, 'Build when a change is pushed to GitLab', is checked and includes a 'GitLab webhook URL' field with the value 'http://192.168.64.150/project/4DVOP'. Below this is the 'Enabled GitLab triggers' section with checkboxes for 'Push Events', 'Push Events in case of branch delete', and 'Opened Merge Request Events'. The 'Build only if new commits were pushed to Merge Request' checkbox is also present. The 'Accepted Merge Request Events' and 'Closed Merge Request Events' checkboxes are also visible. The 'Rebuild open Merge Requests' section has a dropdown menu set to 'Never'. The 'Approved Merge Requests (EE-only)' and 'Comments' checkboxes are checked. The 'Comment (regex) for triggering a build' field contains the text 'Jenkins please retry a build'. The 'GitHub Pull Request Builder' section has checkboxes for 'GitHub hook trigger for GITScm polling' and 'Scrutation de l'outil de gestion de version'. The 'Avancé...' button is located at the bottom right of the configuration area. The bottom of the page shows the 'Environnements de Build' tab.

D. Construction de l'image docker via le Dockerfile du simple_api

Nous utilisons le plugin Docker présent sur Jenkins pour construire l'image mais il aurait été aussi possible de le faire à travers un script shell.

The screenshot shows the Jenkins configuration page for the 'Execute Docker command' plugin. The page has a top navigation bar with tabs: General, Docker command (selected), and Actions à la suite du build. The main content area is titled 'Execute Docker command' and contains several input fields and checkboxes. The 'Docker command' field is set to 'Create/build image'. The 'Build context folder' field contains the value '\$WORKSPACE/docker/simple_api'. The 'Tag of the resulting docker image' field contains the value '4dvop/simple_api'. The 'Filename of dockerfile' field contains the value 'Dockerfile'. The 'Attempt to pull a newer version of the image' checkbox is unchecked. The 'Don't use the cache when building the image' checkbox is checked. The 'List of build arguments, separated by comma, semi-colon or pipe (e.g. http_proxy=http://1.2.3.4:4321;foo=bar)' field is empty. The 'Remove intermediate containers after a successful build' checkbox is unchecked. The bottom of the page shows the 'Actions à la suite du build' tab.

Ici, on récupère le Dockerfile présent dans le repository Gitlab que nous avons téléchargé localement dans notre container Jenkins. Ensuite nous indiquons le fait qu'il ne faut pas utiliser

le cache. Cela permet de reconstruire l'image depuis le début, de cette manière si des modifications sont appliquées au Dockerfile alors ils seront appliqués au build.

E. Démarrage de l'image et test de l'application

Dans un deuxième temps nous allons exécuter du code shell qui sera utilisé sur le shell de notre container Jenkins.

Pour commencer cette partie, nous démarrons l'image précédemment créée puis nous effectuons un test pour savoir si le test est opérationnel.

```
Exécuter un script shell
Commande
docker run -p "5000:5000" --network=jenkins_default -d --name simple_api 4dvop/simple_api
sleep 10
STATUSCODE=$(curl --silent --output /dev/stderr --write-out %{http_code} -u toto:python -X GET http://192.168.64.150:5000/posos/api/v1.0/get_student_ages)
if test $STATUSCODE -ne 200; then
    docker rm -f simple_api
    exit 1
fi
```

L'url du container est accessible sur le port 5000 et sur le même réseau que notre container Jenkins. On fait un curl pour récupérer le code de la page, si la page est accessible (code 200) alors continuons le pipeline, sinon nous arrêtons le pipeline avec un exit 1 en ayant au préalable supprimé le container récemment créé à partir de l'image.

On pense évidemment à supprimer nos containers si jamais le pipeline se relance plus tard.

The screenshot shows two steps in a Jenkins pipeline configuration. The first step, 'Execute Docker command', has a dropdown menu set to 'Stop container(s)' and a text input field containing 'db, clair, simple_api'. The second step, also 'Execute Docker command', has a dropdown menu set to 'Remove container(s)' and a text input field containing 'db, simple_api, clair'. Both steps include a description: 'Comma separated list of containers to be stopped.' and 'Comma separated list of containers to be removed.' respectively. An 'Avancé...' button is visible at the bottom right.

F. Scan de l'image simple_api avec les outils de Clair

On lance le scan clair en utilisant un container qui contient la base de donnée Clair ainsi que le container permettant de scanner notre image. Les containers sont liés via le même réseau. Puis on lance le scan de notre site web, une fois terminé on continue le pipeline avec l'exit 0.

```
docker run -p 5432:5432 --network=jenkins_default -d --name db arminc/clair-db:2017-09-18
sleep 05
docker run -p 6060:6060 --network=jenkins_default --link db:postgres -d --name clair arminc/clair-local-scan:latest
sleep 05
clair-scanner --clair="http://clair:6060" --ip=jenkins_jenkins_1 192.168.64.40:4000/4dvop_simple_api || exit 0
```

G. Push de l'image sur notre registre privé Docker

Pour push l'image, il faut avoir créé le container qui nous servira de registre Docker.
On se connecte au serveur **4dvop-cen7-reg**.

```
[max@4dvop-cen7-reg ~]$ docker run -d -p 4000:5000 --restart=always --name registry registry:2 _
```

J'ai choisi de le rendre accessible sur le réseau sur le port 4000.

Ensuite il est obligatoire de préciser à tous les serveurs qu'il existe un registre privé non sécurisé sur notre réseau (non sécurisé car pas de certificat ssl d'installé sur notre container qui sert de registre) dans **/etc/docker/daemon.json**.

```
{  
  "insecure-registries" : ["192.168.64.40:4000"]  
}
```

Ensuite on push l'image depuis notre Jenkins avec le tag latest et comme nom d'image celui de notre image.

Execute Docker command

Docker command

Push image

Name of the image to push (repository/image)

4dvop_simple_api

Tag

latest

Registry

192.168.64.40:4000

Docker registry URL

https://192.168.64.40:4000

Registry credentials

- aucun -

Ajouter

H. Déploiement de notre application

Nous déployons notre application via l'image que nous venons de push sur notre registre privé vers notre serveur d'application à savoir **4dvop-cen7-app**. Pour ce faire nous utiliserons Ansible, car nous avons déjà travaillé avec et qu'il est facile à implémenter avec beaucoup de documentation.

Dans un premier temps il est nécessaire de déposer notre clé publique ssh Jenkins sur le serveur d'application avec un **ssh-copy-id** pour qu'Ansible puisse s'exécuter sans demander le mot de passe.

```
[max@4dvop-cen7-dep ~]$ ssh-copy-id max@192.168.64.40
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/max/.ssh/id_rsa.pub"
The authenticity of host '192.168.64.40 (192.168.64.40)' can't be established.
ECDSA key fingerprint is SHA256:0JJ3FUjzDKGjC30n87J7EXQ526kZq+PK9PdPut96lqI.
ECDSA key fingerprint is MD5:ba:f7:c5:65:f5:9a:12:93:e4:4c:5a:8f:2a:0d:69:ee.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
max@192.168.64.40's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'max@192.168.64.40'"
and check to make sure that only the key(s) you wanted were added.
```

Ensuite il faut créer le fichier de déploiement Ansible sur notre Gitlab.

```
simple_api_deploy.yaml 327 octets

1 - hosts: 4dvop-cen7-app
2   become: true
3   tasks:
4     - name: "Copie du repertoire docker"
5       copy:
6         src: /opt/bitnami/jenkins/jenkins_home/workspace/4DVOP/docker
7         dest: /home/max
8
9     - name: Déploiement du docker-compose
10      shell:
11        cmd: "docker-compose -f docker-compose.yml up -d "
12      chdir: /home/max/docker
```

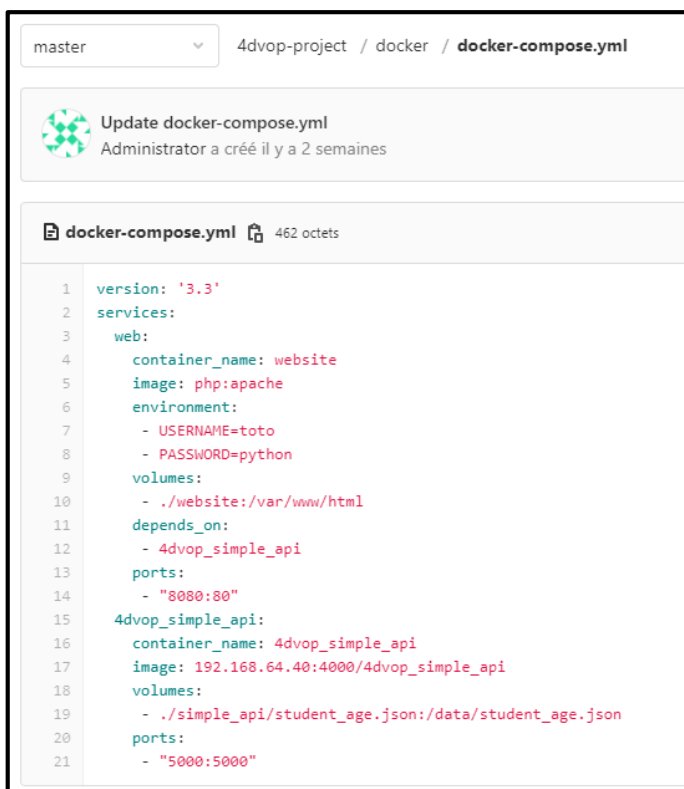
Dans ce déploiement Ansible on indique de copier notre dossier docker contenue dans le Gitlab que nous avons copié localement dans le dossier de notre pipeline et de l'envoyer dans le /home/max qui est sur le serveur d'application. Ensuite on exécute le docker-compose sur le serveur distant.

Nous avons spécifié l'host 4dvop-cen7-app dans le fichier server.ini.

```
server.ini 43 octets

1 4dvop-cen7-app ansible_host=192.168.64.149
```

Notre docker-compose va ensuite pull l'image qui est dans notre registre privé et la démarré.



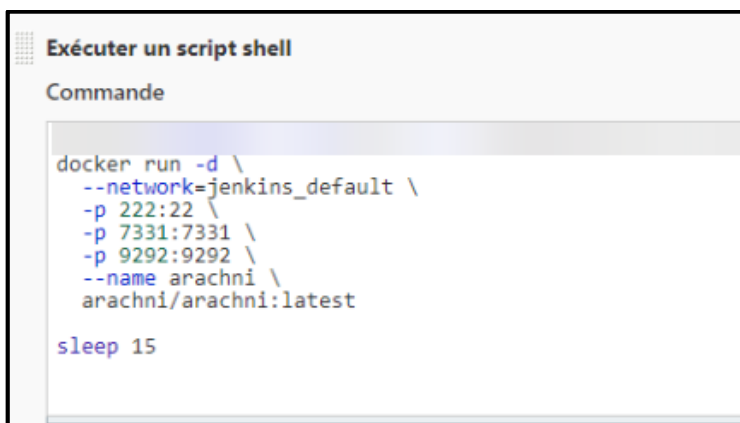
Pour lancer ce déploiement dans notre pipeline voici la configuration Jenkins.



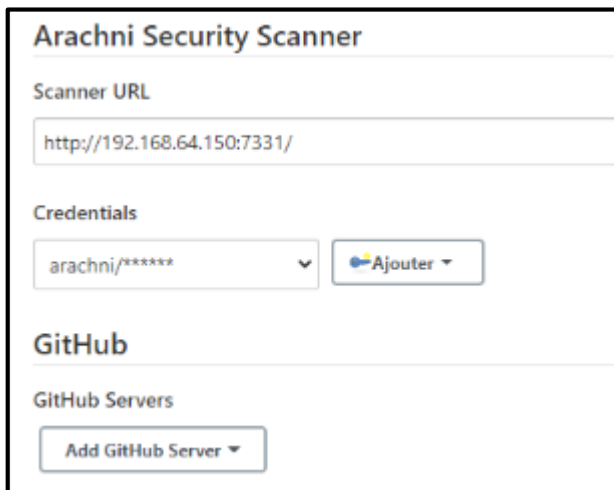
I. Test de sécurité de notre application avec Arachni

Pour utiliser Arachni il y a plusieurs possibilités à disposition. La première est d'utiliser Arachni avec un container ou de l'installer directement sur notre serveur de déploiement et de faire un curl de notre application pour nous retourner le résultat. La seconde est d'utiliser Arachni avec un container ou de l'installer directement sur notre serveur de déploiement et d'utiliser le plugin Arachni qui est disponible sur Jenkins puis d'exécuter un test et de nous retourner le résultat au format json par exemple. Dans ce POC nous avons choisi la seconde possibilité.

Nous allons donc dans un premier temps lancer le container Arachni à travers notre pipeline.

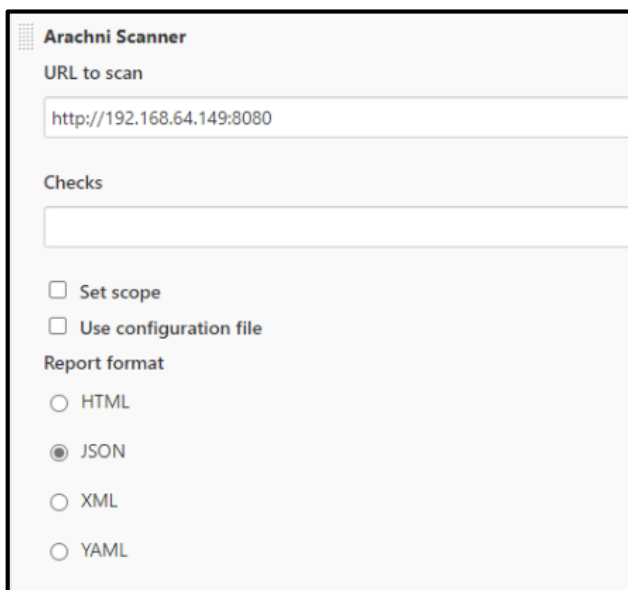


Ensuite il faut installer et configurer le plugin Arachni pour indiquer l'url et le port que le container Arachni utilise. Il faut penser à ajouter les credentials de Arachni dans notre Jenkins.



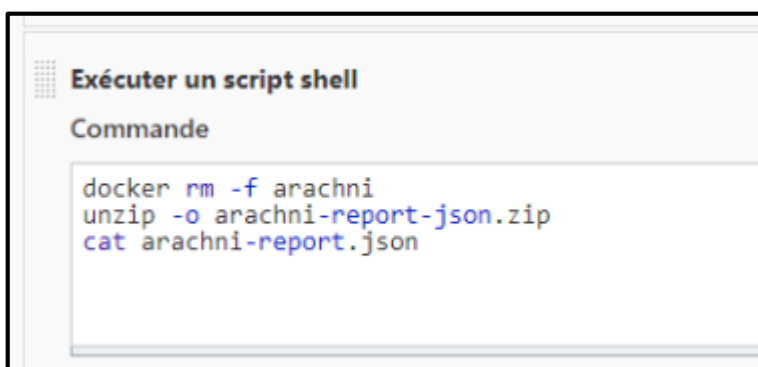
The image shows the 'Arachni Security Scanner' configuration page in Jenkins. It has three main sections: 'Scanner URL' with a text input field containing 'http://192.168.64.150:7331/'; 'Credentials' with a dropdown menu showing 'arachni/*****' and an 'Ajouter' button; and 'GitHub' with a section for 'GitHub Servers' and an 'Add GitHub Server' button.

Enfin il faut lancer le scan Arachni sur notre application disponible à l'url <http://192.168.64.149:8080> puis de stocker le résultat au format JSON (d'autres choix sont possibles) depuis notre pipeline.



The image shows the 'Arachni Scanner' configuration page in Jenkins. It has three main sections: 'URL to scan' with a text input field containing 'http://192.168.64.149:8080'; 'Checks' with an empty text input field; and 'Report format' with radio buttons for 'HTML', 'JSON' (selected), 'XML', and 'YAML'. There are also checkboxes for 'Set scope' and 'Use configuration file'.

Enfin, on souhaite connaître le résultat de notre scan/test, on va donc unzip le résultat et l'afficher dans notre résultat de pipeline (on pense aussi à arrêter et supprimer le container Arachni).



The image shows the 'Exécuter un script shell' configuration page in Jenkins. It has a section for 'Commande' with a text input field containing the following commands:

```
docker rm -f arachni
unzip -o arachni-report-json.zip
cat arachni-report.json
```

IV. Résultat du pipeline

A. Clone Gitlab



Sortie de la console

```
Started by user user
Running as SYSTEM
Building in workspace /opt/bitnami/jenkins/jenkins_home/workspace/4DVOP
The recommended git tool is: NONE
using credential ssh-dep
> git rev-parse --resolve-git-dir /opt/bitnami/jenkins/jenkins_home/workspace/4DVOP/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url git@192.168.64.41:root/4dvop-project.git # timeout=10
Fetching upstream changes from git@192.168.64.41:root/4dvop-project.git
> git --version # timeout=10
> git --version # 'git version 2.23.0'
using GIT_SSH to set credentials ssh-4dvop-cen7-dep
[INFO] Currently running in a labeled security context
> git /usr/bin/chcon --type=ssh_home_t /opt/bitnami/jenkins/jenkins_home/workspace/4DVOP/tmp/jenkins-gitclient-ssh4258549299678184476.key
> git fetch --tags --force --progress -- git@192.168.64.41:root/4dvop-project.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 3e25420809aa0adcf4b7541797f595a8d949d1e1 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 3e25420809aa0adcf4b7541797f595a8d949d1e1 # timeout=10
Commit message: "Update Dockerfile"
> git rev-list --no-walk 3e25420809aa0adcf4b7541797f595a8d949d1e1 # timeout=10
```

B. Build de l'image

```
[Docker] INFO: Step 1/10 : FROM python:3.6.2-stretch
[Docker] INFO:

[Docker] INFO: ---> 2acc6ba9e75f

[Docker] INFO: Step 2/10 : LABEL maintainer="maxence.peligny@supinfo.com"
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> ef0d565e3eb3

[Docker] INFO: Step 3/10 : COPY student_age.py /
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> 7e7eca755948

[Docker] INFO: Step 4/10 : RUN apt-get update -y && apt-get install python-dev python3-dev libssl-dev python-dev libldap2-dev libssl-dev -y
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> 84891855dc05

[Docker] INFO: Step 5/10 : RUN pip install flask flask_httpauth flask_simpleldap python-dotenv
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> ff691da15657

[Docker] INFO: Step 6/10 : RUN mkdir /data
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> b243f692d421

[Docker] INFO: Step 7/10 : COPY student_age.json /data
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> ed975ace750c

[Docker] INFO: Step 8/10 : VOLUME /data
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> 2d10f0b18861

[Docker] INFO: Step 9/10 : EXPOSE 5000
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> 8603137ba0c6

[Docker] INFO: Step 10/10 : CMD ["python", "./student_age.py"]
[Docker] INFO:

[Docker] INFO: ---> Using cache

[Docker] INFO: ---> 3f2975f1074f

[Docker] INFO: Successfully built 3f2975f1074f

[Docker] INFO: Successfully tagged 192.168.64.40:4000/4dvop_simple_api:latest

[Docker] INFO: Build image id:3f2975f1074f
```


E. Push de l'image sur le registre privé

```
[Docker] INFO: Pushing image 192.168.64.40:4000/4dvop_simple_api:latest
[Docker] INFO: Done pushing image 192.168.64.40:4000/4dvop_simple_api:latest
```

F. Déploiement de l'application avec Ansible

```
[4DVOP] $ /bin/sh -xe /tmp/jenkins1728777085303893409.sh
+ ansible-playbook --user max -b -i ansible/server.ini ansible/simple_api_deploy.yaml

PLAY [4dvop-cen7-app] *****

TASK [Gathering Facts] *****
ok: [4dvop-cen7-app]

TASK [Copie du repertoire docker] *****
ok: [4dvop-cen7-app]

TASK [D?ploiement du docker-compose] *****
changed: [4dvop-cen7-app]

PLAY RECAP *****
4dvop-cen7-app      : ok=3    changed=1    unreachable=0    failed=0
```

G. Test avec Arachni

```
051ea9663ab644d2b7bc8155391b9fa97eb9ec32a875ff77a0fb79c7ca462c29
+ sleep 15
Start Arachni Security Scan
Arachni server URL: http://192.168.64.150:7331/
Site under scan: http://192.168.64.149:8080
Scan started with id: 2f3c148280c69acf7733236221615713
Status: scanning - Pages found: 1 - Pages audited: 1
Status: done - Pages found: 1 - Pages audited: 1
Scan finished for id: 2f3c148280c69acf7733236221615713
[4DVOP] $ /bin/sh -xe /tmp/jenkins5558960254366251549.sh
+ docker rm -f arachni
arachni
+ unzip -o arachni-report-json.zip
Archive:  arachni-report-json.zip
  inflating: arachni-report.json
+ cat arachni-report.json
{
  "version" : "1.5.1",
  "seed" : "8dd95d0d849a7041cf0f62a607fb319d",
  "options" : {
    "audit" : {
      "parameter_values" : true,
      "exclude_vector_patterns" : [],
      "include_vector_patterns" : [],
      "link_templates" : []
    },
    "browser_cluster" : {
      "local_storage" : {}
    }
  }
}
```

```
  "plugins" : {}
}Finished: SUCCESS
```