Build automatisé d'images Docker avec Gitlab

Jean-Marc Pouchoulon

février 2024

1 Pré-requis

- Avoir un accès à un serveur GitLab (gitlab.com ou gitlab.com).
- Avoir Docker installé sur votre machine.
- Avoir créer un projet sur votre serveur Gitlab. Le Dockerfile suivant vous permet de construire une image Docker identique à registry.iutbeziers.fr/debianiut:

```
FROM debian:bookworm
LABEL maintainer="Jean-Marc Pouchoulon x

# désactivation des éléments interactifs
ENV DEBIAN_FRONTEND noninteractive
ENV DEBCONF_NONINTERACTIVE_SEEN true
ENV LC_ALL=fr_FR.UTF-8
ENV LANG=fr_FR.UTF-8

# Utilisation du dépot Debian de l'IUT (à décommenter si vous êtes à l'IUT)
#RUN echo "deb http://debian.iutbeziers.fr/debian bookworm main contrib non-free"  > /etc/apt/sources.list
#RUN echo "deb http://debian.iutbeziers.fr/debian bookworm main contrib non-free"  > /etc/apt/sources.list
#RUN echo "deb http://deb.debian.org/debian bookworm main contrib non-free bookworm-updates"  > /etc/apt/sources.list

RUN apt-get update && apt-get upgrade -y  && apt-get install -y \
 git  \
 wget \
 cmatrix \
 curl \
 bzip2 \
 telnet \
 debconf \
 locales \
 iproute2 \
 isc-dhcp-client \
 net-tools \
 sudo \
 vim \
 nano \
```

```
    openssh-client \
    procps \
    man \
    httpie \
    dnsutils \
    mtr \
    ldap-utils \
    iputils-ping \
    inetutils-traceroute \
    nmap

# Créer le fichier si vous êtesun adepte de vim et décommentez la ligne suivante
# COPY .vimrc /root/.vimrc

ENV LC_ALL=fr_FR.UTF-8
ENV LANG=fr_FR.UTF-8
ENV LANGUAGE=fr_FR:fr
ENV LC_TIME=fr_FR.UTF-8
ENV LC_COLLATE=fr_FR.UTF-8



RUN echo "Europe/Paris" > /etc/timezone && \
    dpkg-reconfigure -f noninteractive tzdata && \
    sed -i -e 's/# fr_FR.UTF-8 UTF-8/fr_FR.UTF-8 UTF-8/' /etc/locale.gen && \
    echo 'LANG="fr_FR.UTF-8"'>/etc/default/locale && \
    dpkg-reconfigure --frontend=noninteractive locales && \
    update-locale LANG=fr_FR.UTF-8


# Adding git user
RUN mkdir -p /home/git/.ssh

CMD exec /bin/bash -c "trap : TERM INT; sleep infinity & wait"
```

2 Installation d'un runner Gitlab sur votre machine

Les runners Gitlab sont des agents qui exécutent les jobs dans des pipelines. Ils peuvent être installés sur des machines virtuelles, des conteneurs Docker, des machines physiques, des instances cloud, etc. Ce sont eux qui communiquent avec le serveur Gitlab pour récupérer les jobs à exécuter et ils n'ont pas d'être visibles depuis le serveur Gitlab. Au niveau d'une machine (votre PC de salle par exemple), on a besoin d'installer le binaire gitlab-runner qui permettra de piloter les runners Gitlab sur votre machine et de communiquer avec le serveur Gitlab afin d'aller chercher les jobs.

1. Installez un runner Gitlab sur votre machine. (voir la documentation pour les détails d'installation)

```
curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash
cat <<EOF | sudo tee /etc/apt/preferences.d/pin-gitlab-runner.pref
Explanation: Prefer GitLab provided packages over the Debian native ones
Package: gitlab-runner
Pin: origin packages.gitlab.com
Pin-Priority: 1001
EOF
apt update && apt install gitlab-runner
```

TIPS:

on pourrait aussi installer gitlab-runner dans un container Docker sur votre machine.

```
docker run --restart always -d
        --name gitlab-runner-registry
        -v /var/run/docker.sock:/var/run/docker.sock
        -v volume-gitlab-runner:/etc/gitlab-runner gitlab/gitlab-runner:latest --env TZ=Europe/Paris
```

3 Accès au daemon Docker par une socket TLS

Il est nécessaire de configurer le daemon Docker afin qu'il accepte les connexions sécurisées via un socket TLS depuis le runner Gitlab quand il est sous forme de container et n'a pas accès à la socket Docker de votre machine hôte.

1. Installez mkcert pour pouvoir générer des certificats TLS pour Docker.

```
apt install libnss3-tools mkcert
mkcert --install
Created a new local CA
The local CA is now installed in the system trust store!
```

2. Générez un certificat pour votre machine.

```
mkdir -p /opt/certs
cd /opt/certs
mkcert nom_de_machine.iutbeziers.fr votreip 127.0.0.1
cp /root/.local/share/mkcert/rootCA.pem /opt/certs/ca.pem
cp /root/.local/share/mkcert/rootCA.pem /root/.docker/ca.pem
```

3. Modifiez le daemon Docker pour qu'il accepte les connexions sécurisées via un socket TLS.
   On va utiliser les certificats générés précédemment pour sécuriser la socket Docker en TLS sur le port 2376. l'intérêt de cette configuration est de pouvoir utiliser le runner Gitlab sous forme de container pour construire des images Docker. On pourra faire appel depuis le container runner Gitlab à l'API Docker de la machine pour construire les images.

```
systemctl  edit docker
### Editing /etc/systemd/system/docker.service.d/override.conf
### Anything between here and the comment below will become the new contents of the file

[Service]

ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2376 \
--tlsverify \
--tlscacert=/opt/certs/ca.crt \
--tlscert=/opt/certs/nom_de_machine.iutbeziers.fr+2.pem \b
--tlskey /opt/certs/nom_de_machine.iutbeziers.fr+2-key.pem

### Lines below this comment will be discarded
...
systemctl daemon-reload
systemctl restart docker
```

4. Générez des certificats côté client pour tester la connexion au daemon Docker en TLS.

La communication entre le runner Gitlab et le daemon Docker nécessite que le client Docker (dans le runner) dispose d'un certificat client pour être connecté au daemon Docker de votre machine par TLS.

```
# générez un certificat pour votre machine
mkdir -p /root/.docker
mkcert -client nom_de_machine.iutbeziers.fr 127.0.0.1 10.6.255.24
# afin d'éviter de spécifier le certificat client à chaque commande docker,
# on renomme le certificat client et la clé privée que l'on vient de générer en cert.pem et key.pem.
mv nom_de_machine.iutbeziers.fr+2.pem /root/.docker/cert.pem
mv nom_de_machine.iutbeziers.fr+2-key.pem /root/.docker/key.pem
# on copie aussi le certificat X509 de l'autorité de certification dans le répertoire .docker
cp /root/.local/share/mkcert/rootCA.pem /opt/certs/ca.pem
```

TIPS:

Utilisez la commande suivante afin de vérifier le contenu du certificat:
openssl x509 -in /.docker/cert.pem -text -noout

On vérifie ensuite que l'on peut se connecter au daemon Docker en TLS.

```
export DOCKER_HOST="votre_IP.iutbeziers.fr:2376"
export DOCKER_TLS_VERIFY=1 # redondant avec --tlsverify
docker --tlsverify info
    Client: Docker Engine - Community
     Version:    25.0.2
     Context:    default
     Debug Mode: false
     Plugins:
...
```

4 Création de runners Gitlab sur gitlab.com

Tout d'abord il vous faut créer un runner Gitlab sur votre serveur Gitlab au niveau de votre projet:

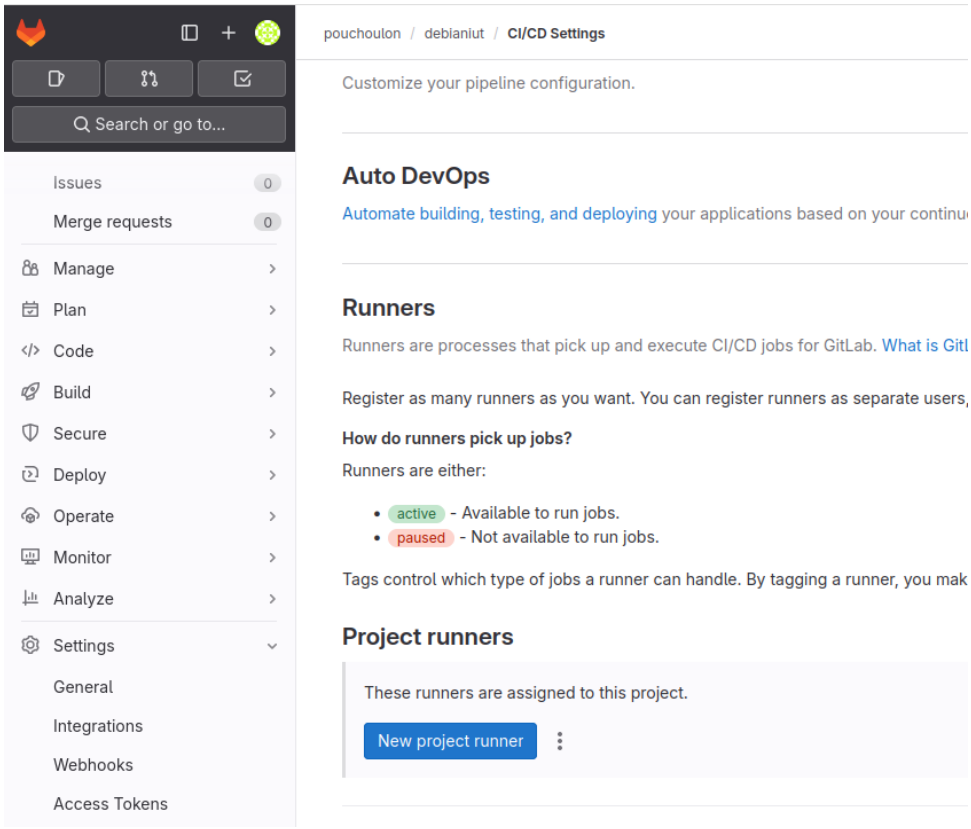

FIGURE 1 – Création d'un runner Gitlab sur gitlab.com

On va choisir le type de runner que l'on souhaite créer en fonction de votre machine.

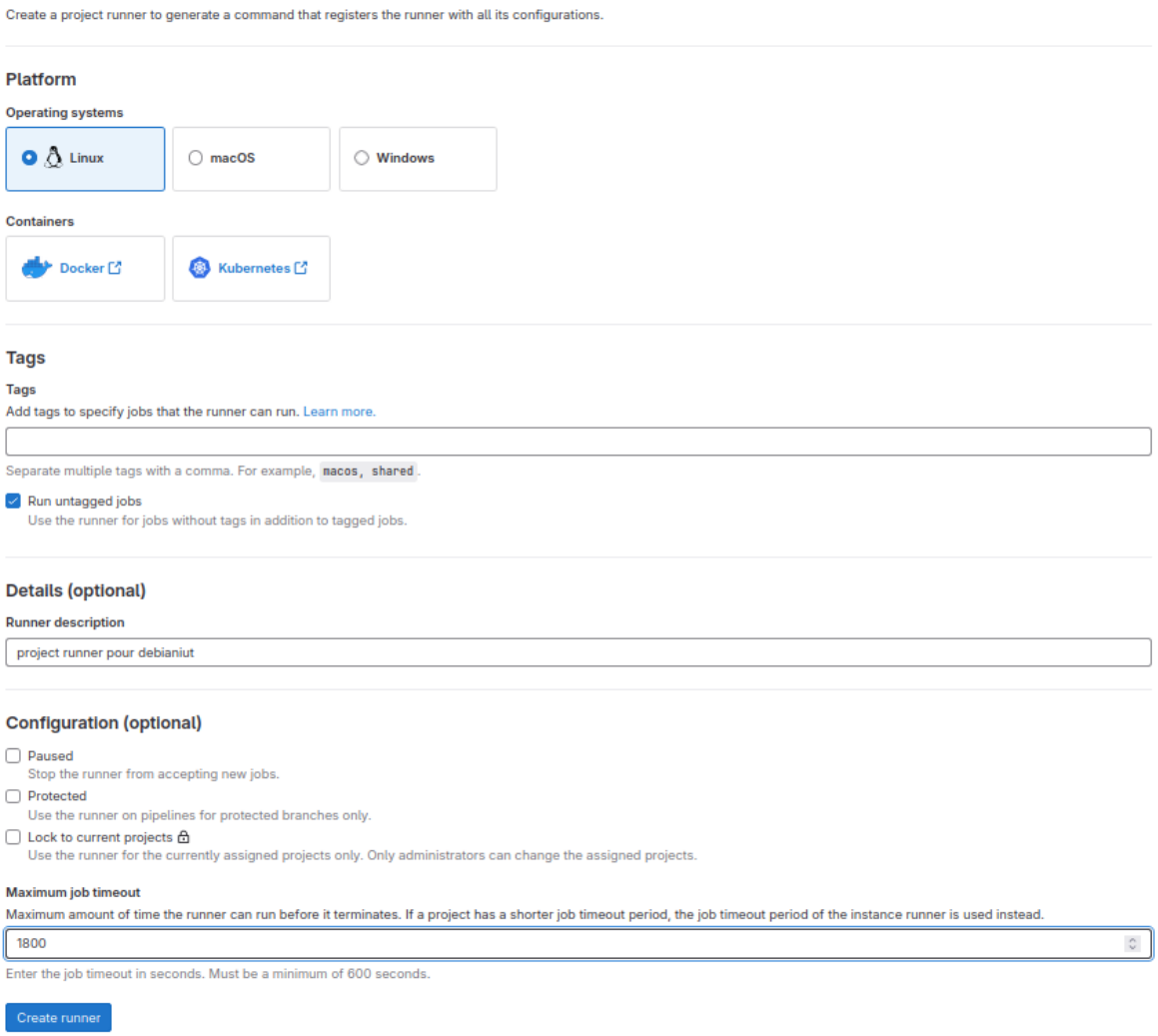

FIGURE 2 – choix pour le runner

Notez bien le numéro de token affiché par Gitlab qui vous sera demandé pour l'enregistrement du runner sur votre machine.

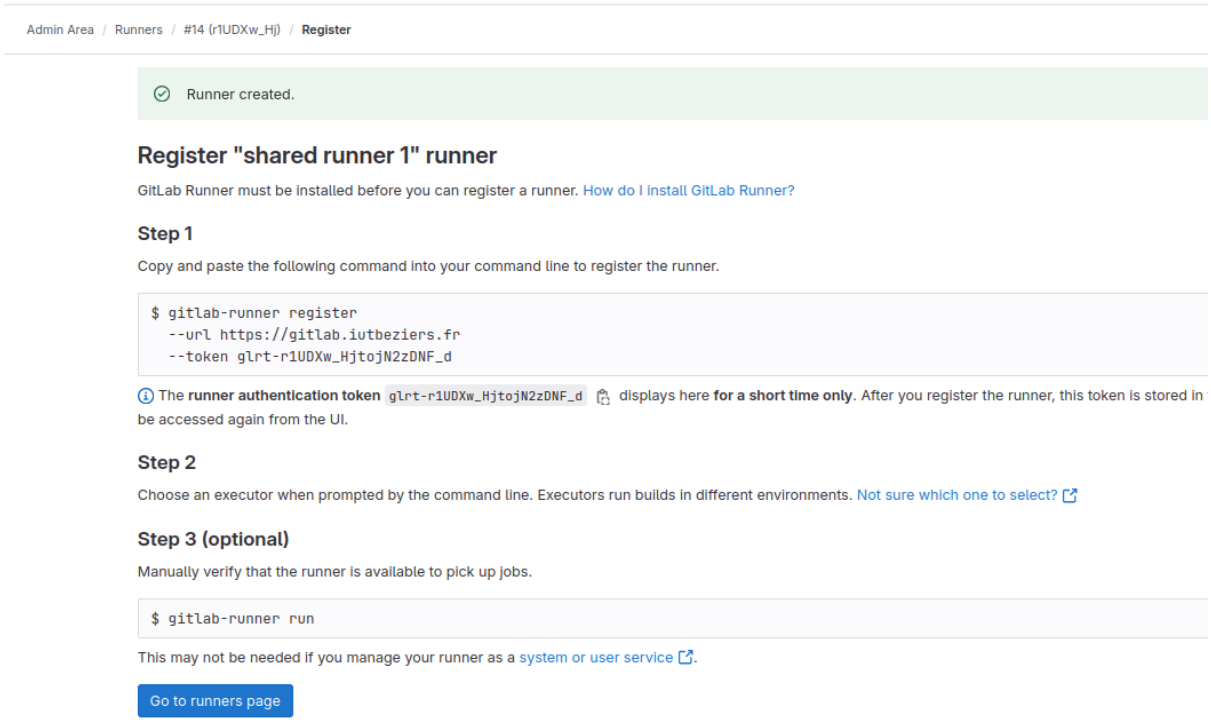

FIGURE 3 – Runner créé sur gitlab.com

4.1 Variables d'environnement pour le runner

On met en place des variables d'environnement pour le runner Gitlab. Vous reconnaîtrez les variables d'environnement contenant vos clefs. Afin d'utiliser le registry de l'IUT il vous faut renseigner vos identifiants LDAP pour pousser vos images sur le registry de l'IUT.

L'utilisation de ces variables d'environnement sera faite dans le fichier .gitlab-ci.yaml qui doit être présent à la racine de votre projet Gitlab et qui vous est donnée ensuite.

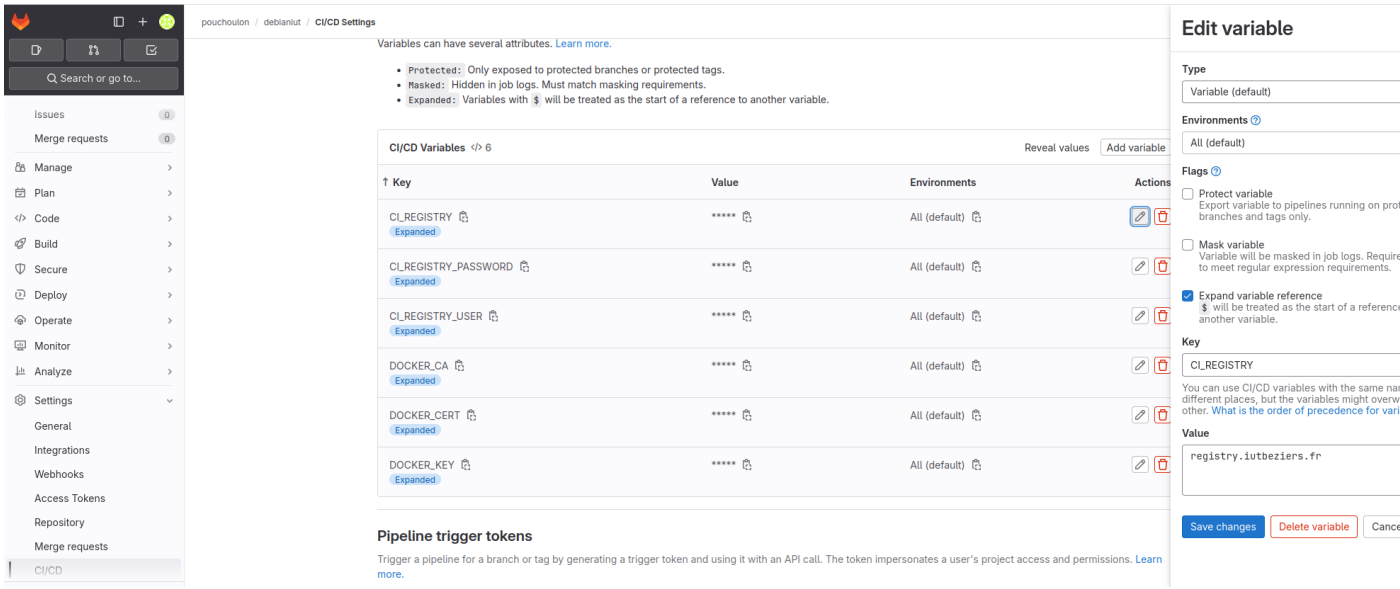

FIGURE 4 – Variables d'environnement pour le runner

4.2 Création d'un runner Gitlab shell sur votre machine

Vous pouvez choisir le type d'exécution de votre runner Gitlab. Il en existe plusieurs types:

- Shell
- Docker
- Docker-ssh
- Parallels
- VirtualBox
- Kubernetes

L' "executor Docker" est le plus utilisé car il permet de construire des images Docker pour les pousser ensuite sur un registry Docker. Il isole les jobs dans des conteneurs Docker.

1. Créez un runner Gitlab shell

```
gitlab-runner register -n  \
--url "https://gitlab.com/"  \
 --registration-token glrt-XXXXXXXX  \
 --executor shell  \
 --description "runner shell portainer1"
```

Vérifiez sur l'interface graphique que le runner a bien communiqué avec le serveur Gitlab. Faites une modification sur un Docker et verifiez

4.3 Création d'un runner Gitlab D.I.N.D (Docker in Docker)

1. Créez le runner Gitlab dind avec cette commande et le token récupéré précédemment.

```
sudo gitlab-runner register -n \
--url "https://gitlab.com/" \
--registration-token glrt-XXXXXXXX \
--executor docker \
--description "docker runner dind portainer1" \
--docker-image "docker:24.0.5" \
--docker-privileged
```

2. Utilisez les instructions suivantes pour votre fichier .gitlab-ci.yaml

```yaml
default:
image: docker:24.0.5
services:
  - docker:24.0.5-dind

variables:
DOCKER_TLS_VERIFY: "1"
DOCKER_CERT_PATH: ".docker"
DOCKER_HOST: "tcp://portainer1.iutbeziers.fr:2376"
DOCKER_DRIVER: overlay2
SECURE_LOG_LEVEL: 'debug'
CI_REGISTRY_IMAGE: registry.iutbeziers.fr/debianiut
CI_REGISTRY: registry.iutbeziers.fr
CI_DEFAULT_BRANCH: main


stages:
 - build_image
 - test
 - deploy
 - container_scanning



before_script:
  - mkdir -p $DOCKER_CERT_PATH
  - echo "$DOCKER_CA" > $DOCKER_CERT_PATH/ca.pem
  - echo "$DOCKER_CERT" > $DOCKER_CERT_PATH/cert.pem
  - echo "$DOCKER_KEY" > $DOCKER_CERT_PATH/key.pem
  - docker --tlsverify  info
  - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" "$CI_REGISTRY"



build_image:
stage: build_image
script:
  - docker pull debian:bullseye
  - docker pull debian:bookworm
  - docker buildx build -t registry.iutbeziers.fr/debianiut:bullseye -f ./Dockerfile.bullseye .
  - docker buildx build -t registry.iutbeziers.fr/debianiut:bookworm -f ./Dockerfile.bookworm .
  - docker tag registry.iutbeziers.fr/debianiut:bookworm registry.iutbeziers.fr/debianiut:latest
  - docker buildx build -t registry.iutbeziers.fr/debianiut:bookwormsystemd -f ./Dockerfile.bookworm.systemd .

#include:
#  - template: Security/Container-Scanning.gitlab-ci.yml

test:
before_script:
  echo "Test un jour, test toujours"
stage: test
script:
  - echo "Test un autre jour"

deploy:
stage: deploy
script:
  - docker push registry.iutbeziers.fr/debianiut:bullseye
```

```
    - docker push registry.iutbeziers.fr/debianiut:bookworm
    - docker push registry.iutbeziers.fr/debianiut:latest
    - docker push registry.iutbeziers.fr/debianiut:bookwormsystemd
    - docker logout $CI_REGISTRY
    - rm -rf $DOCKER_CERT_PATH

container_scanning:

variables:
  # No need to clone the repo, we exclusively work on artifacts. See
  # https://docs.gitlab.com/ee/ci/runners/configure_runners.html#git-strategy
  GIT_STRATEGY: none
  TRIVY_USERNAME: "$CI_REGISTRY_USER"
  TRIVY_PASSWORD: "$CI_REGISTRY_PASSWORD"
  TRIVY_AUTH_URL: "$CI_REGISTRY"
  TRIVY_NO_PROGRESS: "true"
  TRIVY_CACHE_DIR: ".trivycache/"
  TRIVY_IMAGE: "aquasec/trivy:latest"
  TRIVY_COMMAND: "docker run -u root aquasec/trivy:latest"
  FULL_IMAGE_NAME: "registry.iutbeziers.fr/debianiut:latest"
script:
  - echo $FULL_IMAGE_NAME
  - docker pull $TRIVY_IMAGE
  # cache cleanup is needed when scanning images with the same tags, it does not remove the database
  - time $TRIVY_COMMAND image --clear-cache
  # update vulnerabilities db
  - time $TRIVY_COMMAND image --download-db-only
  # Builds report and puts it in the default workdir $CI_PROJECT_DIR, so `artifacts:` can take it from there
  - time $TRIVY_COMMAND image --scanners vuln --exit-code 0 --format template --template "@/contrib/gitlab.tpl"
      --output "gl-container-scanning-report.json"  "$FULL_IMAGE_NAME"
  # Prints full report
  - time $TRIVY_COMMAND image --scanners vuln --exit-code 0 "$FULL_IMAGE_NAME"
cache:
  paths:
    - .trivycache/
```

3. Commentez et expliquer ce fichier gitlab-ci.yaml
4. Faites valider votre chaine de build par votre enseignant.
5. Faites une modification sur ce projet pour arrêter le deploiement de l'image sur le registry de l'IUT.