# Dockerfiles - DockerCompose

lundi 30 janvier 2023  09:19

Formateur : Loup FORMENT

Suite cours Docker –Introduction :

Lien du git utilisé :

https://framagit.org/bkoj/freezer

Docker files pour l'agent Jenkins :

agent_ssh > agent.Dockerfile > ...

```
1   FROM archlinux
2
3   #Installation des paquets nécessaires
4
5   RUN pacman -Sy \
6       && pacman -S --noconfirm jdk11-openjdk python openssh \
7   #Création user
8       && useradd jenkins -m \
9       && mkdir /home/jenkins/.ssh
10
11  #Rajouter clé ssh publique jenkins
12  COPY jenkins_rsa.pub  /home/jenkins/.ssh/authorized_keys
13
14  RUN ssh-keygen -A \
15  && mkdir -p /run/sshd
16
17  #Exposer port 22
18  EXPOSE 22
19
20  #Démarrer openssh
21  CMD /usr/sbin/sshd -D
22
23
```

Pour exécuter le docker file :

```
[darksasuke@chocolat agent_ssh ]$ docker build -f agent.Dockerfile -t je
ns_agent .
```

Création d'un network : **docker network create "nom_reseau" --subnet xxxx.xxxx.xxxx.xxxx/xx --gateway xxxx.xxxx.xxxx.xxxx/xx**

```
vel@vel-Precision-3650-Tower:~/Atelier/docker$ docker network create freeze --subnet 1.2.3.0/24 --gateway 1.2.
331eb0ce266eb92de0f5e5ed11cf343f5530f46782ba17e922e6e30d64968998
vel@vel-Precision-3650-Tower:~/Atelier/docker$ docker network inspect freeze
[
    {
        "Name": "freeze",
        "Id": "331eb0ce266eb92de0f5e5ed11cf343f5530f46782ba17e922e6e30d64968998",
        "Created": "2023-01-30T09:27:37.172431108+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "1.2.3.0/24",
                    "Gateway": "1.2.3.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
```

```
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

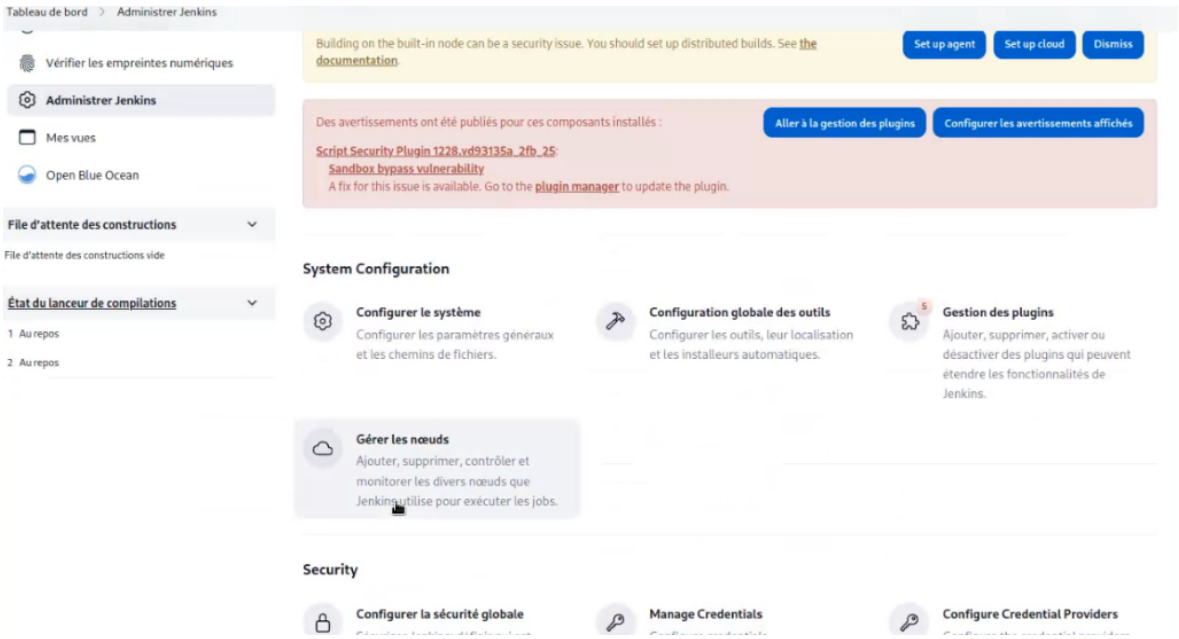

Création container dans le network :

```
vel@vel-Precision-3650-Tower:~/Atelier/docker$ docker run -d --name agent
eeze --ip 1.2.3.10 jenkins_agent
ed16221bb859a09d969dab811897730fcc9e0967432224b614015cdf77b3209d
```

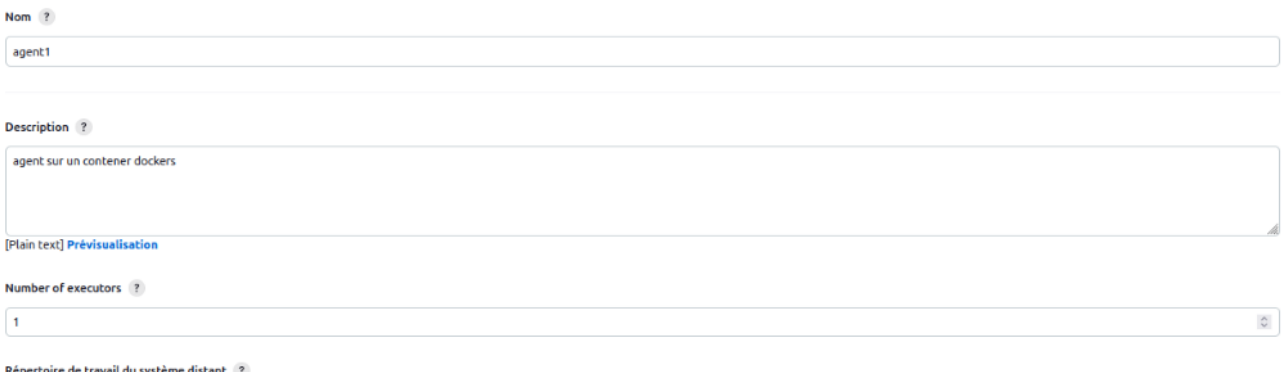
```
vel@vel-Precision-3650-Tower:~/Atelier/docker$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS
          PORTS     NAMES
ed16221bb859   jenkins_agent   "/bin/sh -c '/usr/sb…"   About a minute ago   Up About
a minute   22/tcp    agent
```

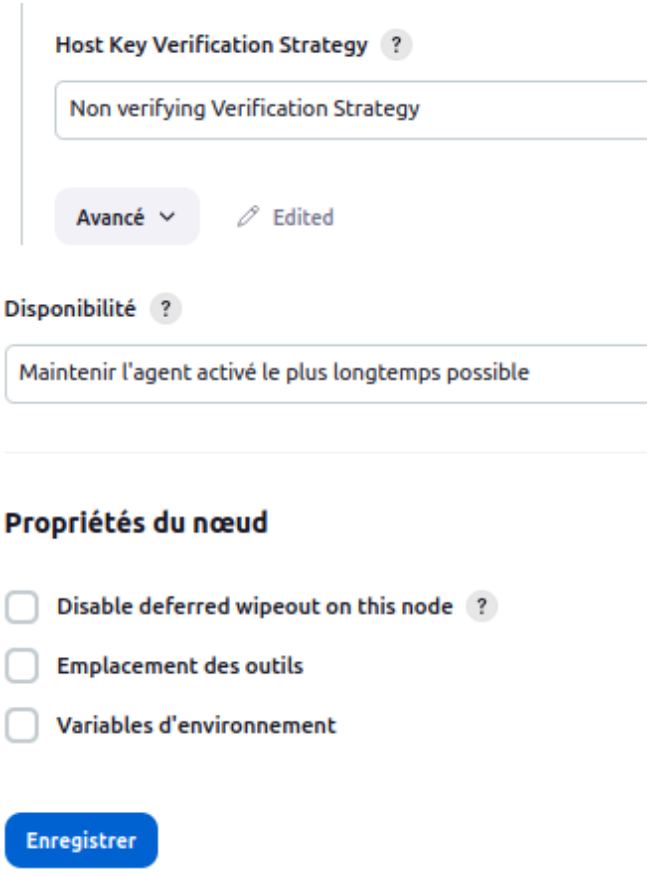
```
vel@vel-Precision-3650-Tower:~/Atelier/docker$ ssh jenkins@1.2.3.10
The authenticity of host '1.2.3.10 (1.2.3.10)' can't be established.
ED25519 key fingerprint is SHA256:BSWhD5gl5u38rgBRVzVodMDklVAX7lpEUFnIGQYukpw.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:14: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Création nœud jenkins :

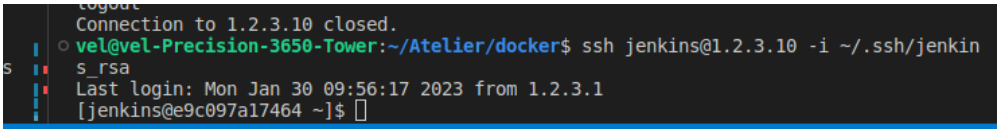



Configuration nodes Jenkins :



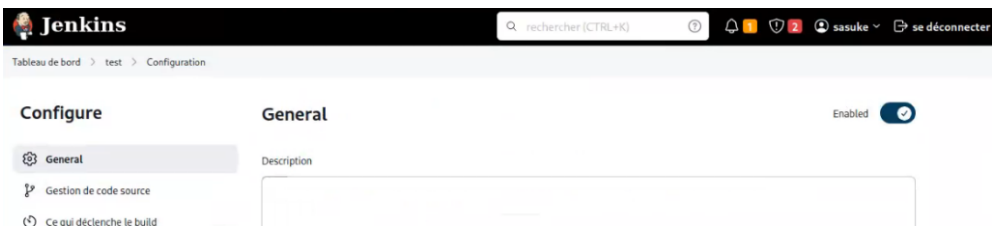



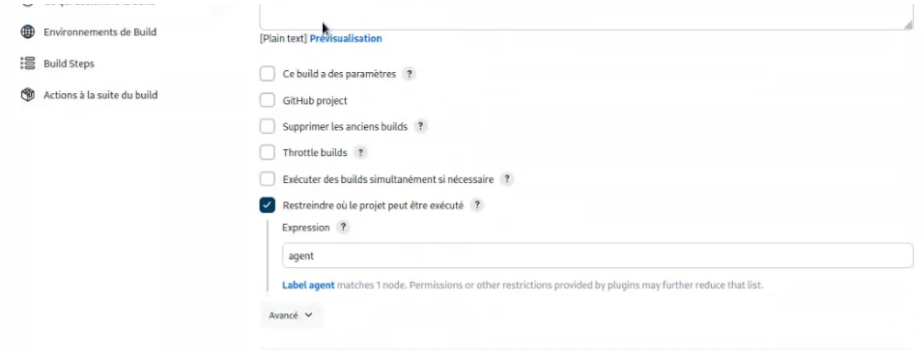

Si jamais une erreur de connexion apparaît : debug à réaliser : tentative de connexion SSH

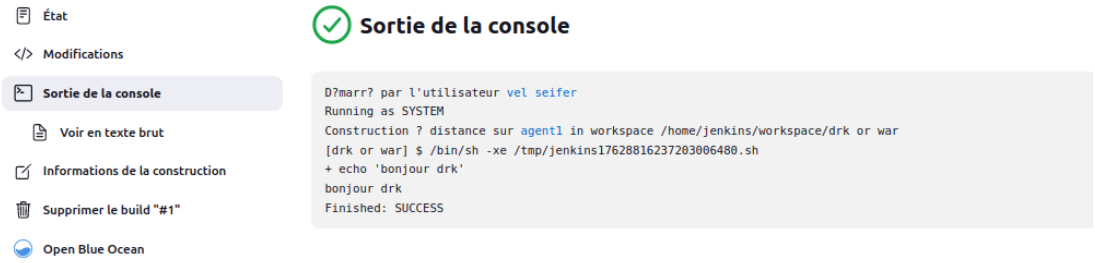



Si la connexion ne fonctionne pas -> refaire les clès est une solution qui peut fonctionner, mais demande de re-build le docker file et de run un nouveau container

Un nodes peut être tester avec un projet freestyle :





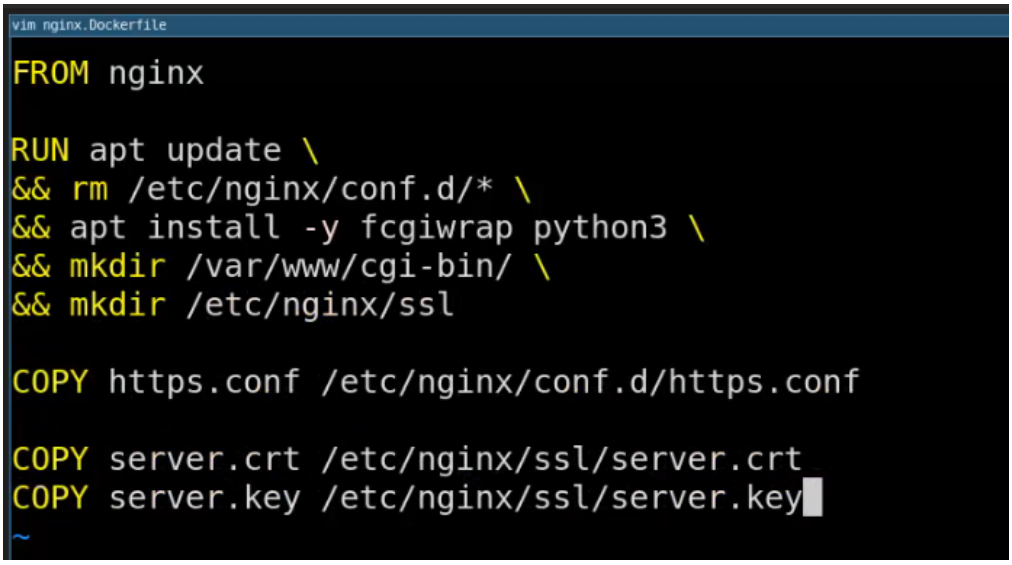



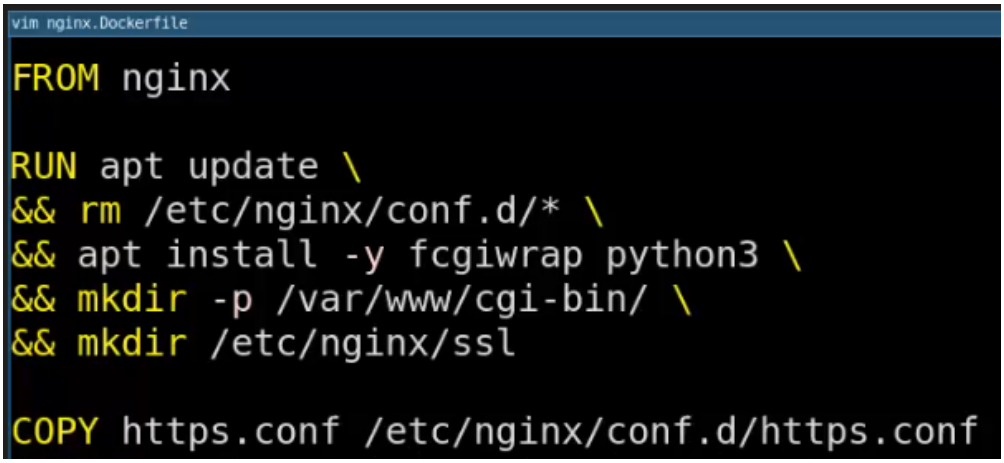

Dockerfiles Nginx

Correction :

Fichier dockerFiles :

HTTPS : Contenu du script :



```
COPY passphrase /etc/nginx/ssl/passphrase

COPY server.crt /etc/nginx/ssl/server.crt
COPY server.key /etc/nginx/ssl/server.key
```

Contenu du dossier :

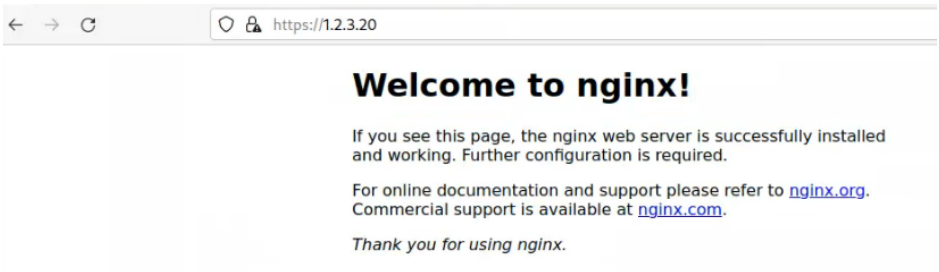
```
[darksasuke@chocolat nginx ]$ ls
https.conf  nginx.Dockerfile  server.crt  server.key
```

Extrait configuration fichier de conf, ne pas oublier la redirection dans la partie du serveur 80

```
server {
    listen               443 ssl;
    server_name          www.loup.fr;
    ssl_certificate     /etc/nginx/ssl/server.crt ;
    ssl_certificate_key /etc/nginx/ssl/server.key ;
    ssl_protocols        TLSv1.1 TLSv1.2 TLSv1.3;
    ssl_password_file /etc/nginx/ssl/passphrase;
    ssl_ciphers          HIGH:!aNULL:!MD5;
    root /usr/share/nginx/html;

     #location / {
#           proxy_pass http://backend;
 #    }


   # per user directory
  location ~ ^/~(.+?)(/.*)?$ {
          auth_basic           "Private Area";
        auth_basic_user_file /etc/nginx/pass/$1;
        alias     /usr/share/publics/$1$2;
        index     index.html index.htm;
        autoindex on;
    }
```




Ne pas oublier de modifier le fichier host pour ce TP

```
sudo vim /etc/hosts
# Static table lookup for hostnames.
# See hosts(5) for details.

1.2.3.4 chocolat vanille
1.2.3.20  loup.fr www.loup.fr data.loup.fr drive.loup.fr www.bob.com data.bob.com
10.125.25.52 sacha.fr data.sacha.fr www.sacha.fr tls.server.fr
10.125.25.68 oualid.fr data.oualid.fr
10.125.25.67 arandr.fr www.arandr.fr
10.125.25.50 www.jad.fr jad.fr
```

FCGI :

Pour faire fonctionner fcgi, il est nécessaire d'avoir un script pour lancer les modules :

vim wrapper.sh

```
#!/bin/bash

sed -i -E "s/^user .*$/user root;/g" /etc/nginx/nginx.conf

/usr/sbin/fcgiwrap -s unix:/run/fcgiwrap.socket &

/usr/sbin/nginx -g "daemon off;" &


wait -n
exit $?
```

Configuration FCGI dans le fichier de configuration du site web :

```
# CGI
location ~ /(.+\.cgi)$ {
     include       fcgiwrap_params;
     fastcgi_param DOCUMENT_ROOT /var/www/cgi-bin/;
     fastcgi_param SCRIPT_NAME   $1;
     fastcgi_pass  unix:/run/fcgiwrap.socket;
}

error_log /var/log/nginx/https.error.log;
access_log /var/log/nginx/https.access.log;
```

Fichier DockerFiles avec configuration CGI

vim nginx.Dockerfile

```
FROM nginx

RUN apt update \
&& rm /etc/nginx/conf.d/* \
&& apt install -y fcgiwrap python3 \
&& mkdir -p /var/www/cgi-bin/ \
&& mkdir /etc/nginx/ssl

COPY https.conf /etc/nginx/conf.d/https.conf
COPY passphrase /etc/nginx/ssl/passphrase
COPY fcgiwrap_params /etc/nginx/fcgiwrap_params
COPY main.cgi /var/www/cgi-bin/main.cgi


COPY server.crt /etc/nginx/ssl/server.crt
COPY server.key /etc/nginx/ssl/server.key


COPY wrapper.sh /wrapper.sh

CMD wrapper.sh
```

Fichier nécessaire :

~/docker/nginx

```
[darksasuke@chocolat nginx ]$ ls
fcgiwrap_params  main.cgi          passphrase  server.key
https.conf       nginx.Dockerfile  server.crt  wrapper.sh
```

ProFTPD :

Image utilisé :

https://hub.docker.com/r/instantlinux/proftpd

Démarrage d'un docker de test : argument **PSV_ADDRESS=xxxx.xxxx.xxxx.xxxx**

```
vel@vel-Precision-3650-Tower:~$ docker run --network=freeze --ip=1.2.3.40 -d --rm -e PASV_ADDRESS=1.2.3.14 insta
```

Le distribution utilisé derrière cette image est **Alpine**

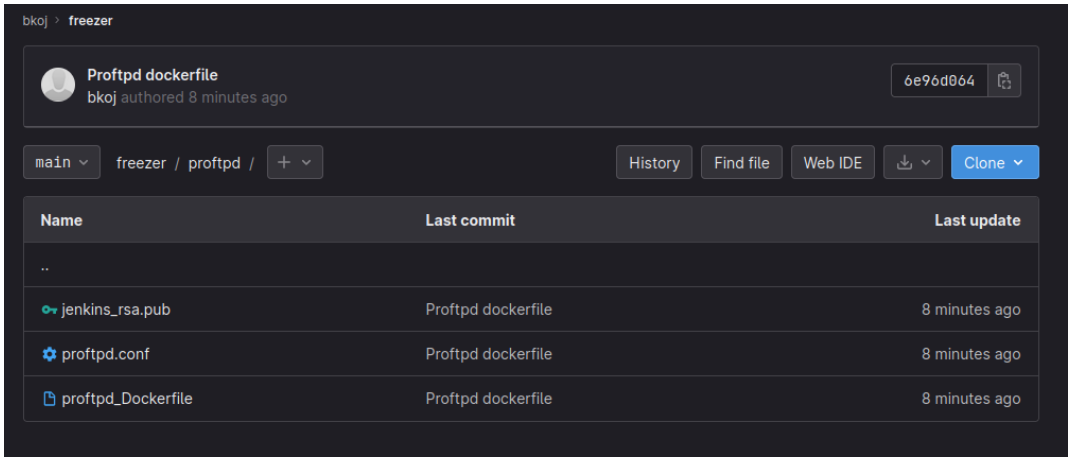
https://gitlab.com/instantlinux/docker-tools/-/blob/main/images/proftpd/Dockerfile

Pour se connecter sur l'image : **docker exec –it "nom_container" sh**

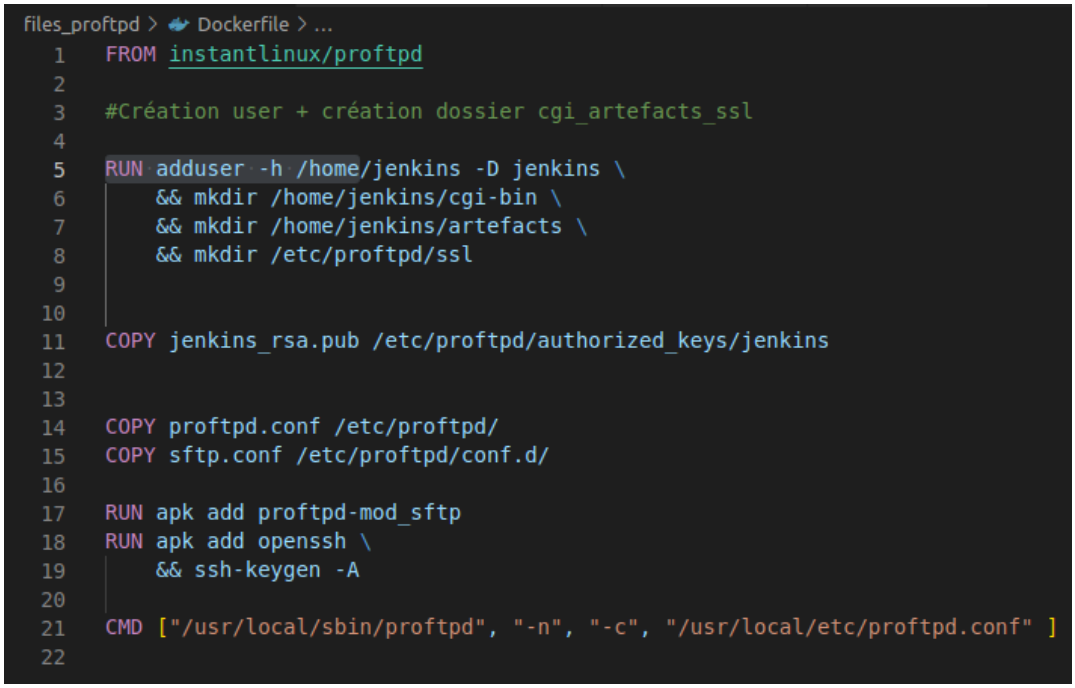
```
vel@vel-Precision-3650-Tower:~/Atelier/docker/correction_Freezer$ docker exec -it co
ndescending_cartwright sh
/ # ls
bin    etc    lib    mnt    proc   run    srv    tmp    var
dev    home   media  opt    root   sbin   sys    usr
/ #
```

Correction :

Pour ce docker files, nous avions besoin des éléments suivant :




Code dockerfiles :



**POSTGRES**

```
[darksasuke@chocolat postgre ]$ docker exec -it some-postgres bash
root@3538034528d4:/#
```

```
root@3538034528d4:/# psql -U postgres
psql (15.1 (Debian 15.1-1.pgdg110+1))
Type "help" for help.

postgres=#
```

```
postgres=# CREATE DATABASE freezer;
CREATE DATABASE
postgres=# \l
                                                 List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    | ICU Locale | Locale Provider |   Access privi
leges
-----------+----------+----------+------------+------------+------------+-----------------+---------------
--------
 freezer   | postgres | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            |
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            | =c/postgres
      +
           |          |          |            |            |            |                 | postgres=CTc/p
ostgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            | =c/postgres
      +
           |          |          |            |            |            |                 | postgres=CTc/p
ostgres
(4 rows)

postgres=# \c freezer
You are now connected to database "freezer" as user "postgres".
freezer=# create table users (id serial PRIMARY KEY,name VARCHAR(80) NOT NULL,  ) ;
```

Création des tables :

```
freezer=# create table musics (id SERIAL PRIMARY KEY, title VARCHAR(200), path VARCHAR(200));
CREATE TABLE
freezer=# \dt
         List of relations
 Schema |  Name  | Type  |  Owner
--------+--------+-------+----------
 public | musics | table | postgres
```

```
freezer=# create table users (id serial PRIMARY KEY,name VARCHAR(80) NOT NULL, pass VARCHAR(200), ip varch
ar(15));
CREATE TABLE
freezer=# \dt
         List of relations
 Schema |  Name  | Type  |  Owner
--------+--------+-------+----------
 public | musics | table | postgres
 public | users  | table | postgres
(2 rows)
```

```
freezer=# CREATE table rel (id_user INT,id_music INT , FOREIGN KEY (id_user) references users (id), FOREIG
N KEY (id_music) references musics (id), PRIMARY KEY (id_user,id_music) );
CREATE TABLE
freezer=# \dt
         List of relations
 Schema |  Name  | Type  |  Owner
--------+--------+-------+----------
 public | musics | table | postgres
 public | rel    | table | postgres
 public | users  | table | postgres
(3 rows)
```

Après création BDD -> réaliser DUMP de la BDD pour la récupérer

```
root@3538034528d4:/# pg_dump -U postgres freezer > freeze.sql
```

```
root@3538034528d4:/# pg_dump -U postgres freezer > freeze.sql
```

Après le dump : docker cp pour copier le dump de notre docker à notre pc

```
[darksasuke@chocolat postgre ]$ docker cp some-postgres:/freeze.sql .
[darksasuke@chocolat postgre ]$ ls
freeze.sql
```

Fichier DockerFiles :

```
FROM postgres

RUN psql -U postgres -c 'DROP DATABASE postgres; CREATE DATABASE freezer;'

COPY freeze.sql /

RUN psql freezer < freeze.sql
```

```
FROM postgres


RUN psql -U postgres -c 'DROP DATABASE postgres; CREATE DATABASE freezer;'

COPY freeze.sql /

RUN psql freezer < freeze.sql

CMD script.sh
```

### Initialization scripts

If you would like to do additional initialization in an image derived from this one, add one or more `*.sql`, `*.sql.gz`, or `*.sh` scripts under `/docker-entrypoint-initdb.d` (creating the directory if necessary). After the entrypoint calls `initdb` to create the default `postgres` user and database, it will run any `*.sql` files, run any executable `*.sh` scripts, and source any non-executable `*.sh` scripts found in that directory to do further initialization before starting the service.

**Warning**: scripts in `/docker-entrypoint-initdb.d` are only run if you start the container with a data directory that is empty; any pre-existing database will be left untouched on container startup. One common problem is that if one of your `/docker-entrypoint-initdb.d` scripts fails (which will cause the entrypoint script to exit) and your orchestrator restarts the container with the already initialized data directory, it will not continue on with your scripts.

For example, to add an additional user and database, add the following to `/docker-entrypoint-initdb.d/init-user-db.sh`:

```
#!/bin/bash
set -e

psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" <<-EOSQL
        CREATE USER docker;
        CREATE DATABASE docker;
        GRANT ALL PRIVILEGES ON DATABASE docker TO docker;
EOSQL
```

These initialization files will be executed in sorted name order as defined by the current locale, which defaults to `en_US.utf8`. Any `*.sql` files will be executed by `POSTGRES_USER`, which defaults to the `postgres` superuser. It is recommended that any `psql` commands that are run inside of a `*.sh` script be executed as `POSTGRES_USER` by using the `--username "$POSTGRES_USER"` flag. This user will be able to connect without a password due to the presence of `trust` authentication for Unix socket connections made inside the container.

Additionally, as of docker-library/postgres#253, these initialization scripts are run as the `postgres` user (or as the "semi-arbitrary user" specified with the `--user` flag to `docker run`; see the section titled "Arbitrary `--user` Notes" for more details). Also, as of docker-library/postgres#440, the temporary daemon started for these initialization scripts listens only on the Unix socket, so any `psql` usage should drop the hostname portion (see docker-library/postgres#474 (comment) for example).

Le Dockerfiles doit donc ressembler à ça :

```
FROM postgres
ENV POSTGRES_USER freezer
ENV POSTGRES_DB freezer


COPY freeze.sql /docker-entrypoint-initdb.d/
```

Contenu dossier postgresql

```
[darksasuke@chocolat postgre ]$ ls
freeze.sql  postgre.Dockerfile
```

Docker Compose :

What is Docker Compose used for?

Docker Compose is a tool that was developed **to help define and share multi-container applications**. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down.

https://docs.docker.com › get-started › 08_using_compose

Use Docker Compose - Docker Documentation

Rechercher : What is Docker Compose used for?

What is difference between Dockerfile and Docker compose?

The key difference between the Dockerfile and docker-compose is that **the Dockerfile describes how to build Docker images, while docker-compose is used to run Docker containers**. 18 mai 2022

https://docs.docker.com/compose/install/linux/

https://docs.docker.com/compose/gettingstarted/

Le fichier **docker-compose.yml** au format YAML "décrit" tous les *docker run* possibles avec images, noms, volumes, liens, ports, variables d'environnements, réseaux... Il n'est dès lors plus nécessaire de mémoriser et introduire tous ces paramètres en ligne de commande : ils sont tous inventoriés en liste dans ce fichier unique. Si vous maîtrisez vos images, conteneurs et paramètres d'exécution, vous pouvez facilement rédiger ce fichier.

En reprenant les conteneurs abordés dans le tutoriel précédent, le fichier docker compose équivalent serait alors :

```
version: '3'

services:
  web:
    image: php7.4-apache:projet
    container_name: projet-php-apache
    ports:
      # local:container
      - "1337:80"
    environment:
      - APACHE_DOCUMENT_ROOT=/var/www/html
      - COMPOSER_ALLOW_SUPERUSER=1
    volumes:
      # Volume stockant les fichiers web du projet
      - ./:/var/www/html
```

Installer docker compose : **apt install docker-compose**

```
vel@vel-Precision-3650-Tower:~/Atelier/docker/correction_Freezer/freezer$ sudo apt install docker-
```

Exemple de Docker-Compose :

```yaml
version: '3'

services:
  my_nginx:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - my_vol:/var/www/html

    networks:
      - freezer
volumes:
  my_vol:

network:
  freezer:
    #subnet: "1.2.3.0/24"
    #gateway: "1.2.3.1"

~
~
~
```

Commande d'exécution du docker compose :

```
[darksasuke@chocolat test ]$ docker-compose up
[+] Running 7/7
 ⠿ my_nginx Pulled                                                    1.9s
   ⠿ 3740c948ffd4 Already exists                                      0.0s
   ⠿ d2c0556a17c5 Already exists                                      0.0s
   ⠿ c8b9881f2c6a Already exists                                      0.0s
   ⠿ 693c3ffa8f43 Already exists                                      0.0s
   ⠿ 8316c5e80e6d Already exists                                      0.0s
   ⠿ b2fe3577faa4 Already exists                                      0.0s
[+] Running 3/0
 ⠿ Network test_default       Created                                 0.0s
 ⠿ Volume "test_my_vol"       Created                                 0.0s
 ⠿ Container test-my_nginx-1  Created                                 0.1s
Attaching to test-my_nginx-1
test-my_nginx-1  | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform conf
iguration
test-my_nginx-1  | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
test-my_nginx-1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
test-my_nginx-1  | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/defaul
```

```yaml
version: '3'

services:
  my_nginx:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - my_vol:/var/www/html
  my_pro:
    image: instantlinux/proftpd
      #build:
      #- context: proftpd
      #- file: proftpd.dockerfile

    volumes:
        - my_vol:/home/jenkins/cgi
    networks:
```

```
      freezer2:
        - ipv4_address: 1.2.4.10



volumes:
  my_vol:

networks:
   freezer2:
        driver: bridge
        ipam:
            config:
              - subnet: "1.2.4.0/24"
                gateway: 1.2.4.1
~
```

Docker-compose up –d : deployer en arrière-plan
Docker-compose down : remove toute les instances déployer avec docker-compose

```
[darksasuke@chocolat test ]$ docker-compose up -d
[+] Running 5/5
 ⠿ my_pro Pulled
   ⠿ c158987b0551 Already exists
   ⠿ 68eaa751a378 Already exists
   ⠿ 6f70029f2a20 Already exists
   ⠿ fd9677e478b3 Already exists
[+] Running 2/2
 ⠿ Container test-my_pro-1    Started
 ⠿ Container test-my_nginx-1  Started
[darksasuke@chocolat test ]$ sudo docker ps
[sudo] Mot de passe de darksasuke :
CONTAINER ID   IMAGE     COMMAND                  
            NAMES
173d92d2b719   nginx     "/docker-entrypoint.…"   
0->80/tcp   test-my_nginx-1
[darksasuke@chocolat test ]$ docker-compose down
[+] Running 3/3
 ⠿ Container test-my_nginx-1  Removed
 ⠿ Container test-my_pro-1    Removed
 ⠿ Network test_default       Removed
```

```
version: '3'
services:
  app:
    build:
      context: .
      dockerfile: docker/Dockerfile
    image: basic
    ports:
    - 82:80
```

Correction :

Extrait de fichier docker-compose.yaml

```
vim docker-compose.yaml
services:
  agent:
    image: agent
    build:
      context: agent_ssh
      dockerfile: agent.Dockerfile
    networks:
      freeze:
        ipv4_address: 1.2.3.20


  jen:
    image: my_jenkins
    build:
      context: jenkins
      dockerfile: jenkins.dockerfile
    networks:
      freeze:
        ipv4_address: 1.2.3.30
    environment:
      - JENKINS_ADMIN_ID=admin
      - JENKINS_ADMIN_PASSWORD=admin
      - SSH_PRIVATE_KEY=${SSH_PRIVATE_KEY}
```

```
my_proftpd:
  image: my_proftpd
  build:
    context: proftpd
    dockerfile: proftpd.Dockerfile
  networks:
    freeze:
      ipv4_address: 1.2.3.40
  volumes:
    - nginx_pro:/home/jenkins/cgi-bin/
  environment:
    - PASV_ADDRESS=1.2.3.40

my_nginx:
  image: my_nginx
  build:
    context: nginx
    dockerfile: nginx.Dockerfile
  networks:
    freeze:
      ipv4_address: 1.2.3.10
  volumes:
    - nginx_pro:/var/www/cgi-bin/my_nginx
```

Pour ce services, on va déclarer chaque containeur avec plusieurs arguments nécessaire à son bon fonctionnement (build/network par exemple)

```
#network utilisé
networks:
  freezer_bridge:
    driver: bridge
    ipam:
     config:
       - subnet: 6.6.6.0/24
         gateway: 6.6.6.1

volumes:
```

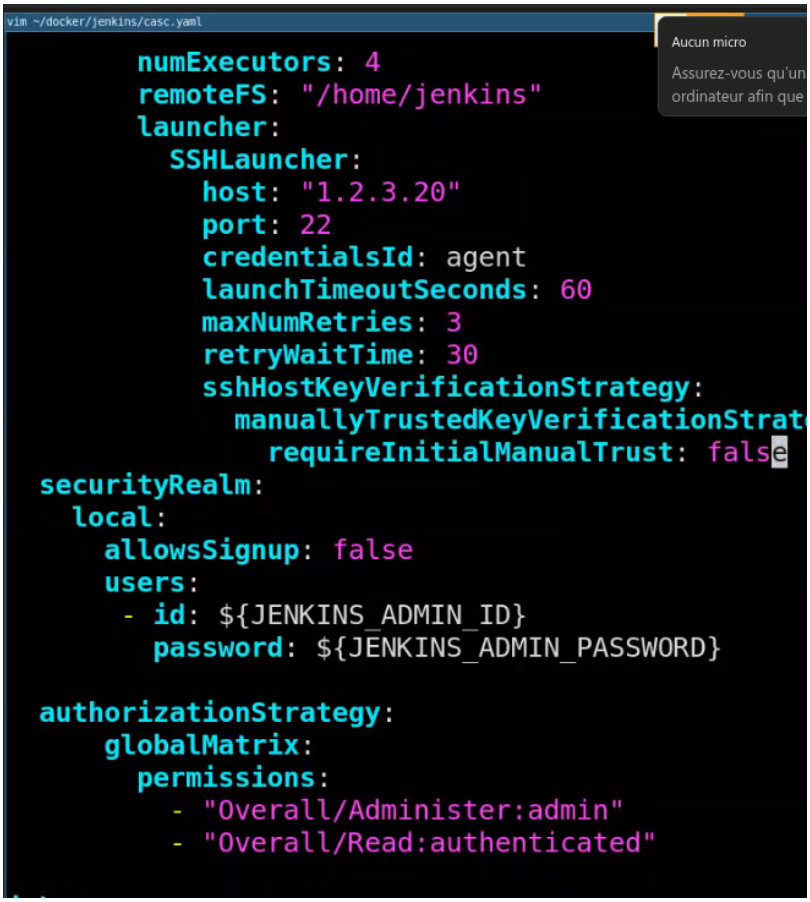
```
data:
```

On peut également déclarer des devices docker qui seront utilisé dans ce projet

Pour jenkins :

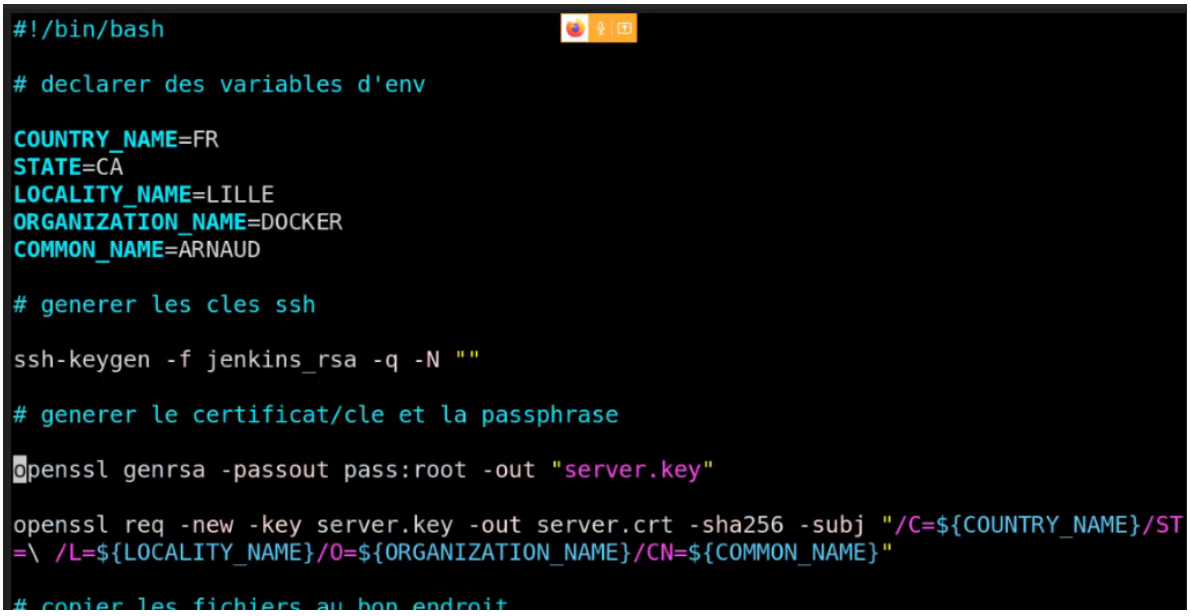
Fichier casc.yaml nécessaire avec le docker-compose pour configuration automatiquement Jenkins.

Extrait du fichier :




Création init.sh : ce script va servir à générer clé ssh/certificat, copier fichier aux bons endroits et à exécuter automatiquement le **docker-compose up --build**

Contenu **init.sh**



```
# copier les fichiers au bon endroit
```

```
#!/bin/bash

# declarer des variables d'env

COUNTRY_NAME=FR
STATE=CA
LOCALITY_NAME=LILLE
ORGANIZATION_NAME=DOCKER
COMMON_NAME=ARNAUD
PASSPHRASE=root


# generer les cles ssh

ssh-keygen -f jenkins_rsa -q -N ""

# generer le certificat/cle et la passphrase

openssl genrsa -passout pass:$PASSPHRASE -out "server.key"

openssl req -new -key server.key -out server.crt -sha256 -subj "/C=${COUNTRY_NAME}/ST=\ /L=${LOCALITY_NAME}/O=${ORGANIZATION_NAME}/CN=${COMMO
_NAME}"

# copier les fichiers au bon endroit

cp jenkins_rsa agent_ssh/
mv jenkins_rsa jenkins/
mv server.crt nginx/
mv server.key nginx/
echo "$PASSPHRASE" > nginx/passphrase

cp jenkins_rsa.pub agent_ssh/

ssh-keygen -e -f jenkins_rsa.pub | sed -E  "/Comment/d" > proftpd/jenkins_rsa.pub

rm jenkins_rsa.pub

docker compose up --build
```

Aucun micro

Assurez-vous qu'un micro est connecté à votre ordinateur afin que les autres utilisateurs...

```
#!/bin/bash

# declarer des variables d'env

COUNTRY_NAME=FR
STATE=CA
LOCALITY_NAME=LILLE
ORGANIZATION_NAME=DOCKER
COMMON_NAME=ARNAUD
PASSPHRASE=root


# generer les cles ssh

ssh-keygen -f jenkins_rsa -q -N ""

# generer le certificat/cle et la passphrase

openssl genrsa -passout pass:$PASSPHRASE -out "server.key"

openssl req  -x509 -new -key server.key -out server.crt -sha256 -subj "/C=${COUNTRY_NAME}/ST=\ /L=${LOCALITY_NAME}/O=${OR
{COMMON_NAME}"

# copier les fichiers au bon endroit

cp jenkins_rsa agent_ssh/
mv jenkins_rsa jenkins/
mv server.crt nginx/
mv server.key nginx/
echo "$PASSPHRASE" > nginx/passphrase

cp jenkins_rsa.pub agent_ssh/

ssh-keygen -e -f jenkins_rsa.pub | sed -E  "/Comment/d" > proftpd/jenkins_rsa.pub

rm jenkins_rsa.pub

docker compose up --build
```

```
-----BEGIN CERTIFICATE REQUEST-----
MIICUTCCATkCAQAwDDEKMAgGA1UECAwBIDCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBAIurgyIk10y0OI35GlEKoqWvBMxZox/VngP9qbMHg48tF+ksEjpB
LP4e7Z5WrRNBgCvATl64xYJKIZGFofe5iVeN0F/+XGVtu3el1PfcCrDRruZhQerr
xLlkhIJBhmwC/ygyGSU/h1TzSwAru3RBJsvIFYBBTROUTs0XsGp+YwIpHvXxqgTh
woHaLzW73jDPHlEmyr0pjqqOTM/Iavij8johWbw2AaRXGmUOIK9B+OWUgTrIKLrc
B6ll6y5t8Ust+JabFLsaOMei+c3MGnKE4zeiwKSlsvBQQQQw3Hq12kLsJ5w9tn7T
Kn/zVzBeYSJ4dTWCJnzqZueXu1RXGL3q08sCAwEAAaAAMA0GCSqGSIb3DQEBCwUA
A4IBAQAcCaqWxBmOQt7buFit/jyhtqYvE1s5JE+YY39uYn4A8iIfIfwlMpiawPGm
kc4zvl+M4TLxoEuFLnsL//ughbgjyDde5LCfOxOZfv9vWmJvhAMOT54xuoT3K6OP
J4YHSse+RvmHMeM9h0MC9AhqkeUu4WvPPh0t4tJftq1QdoJj6sSAmIuhA5MYe7pv
Nve1ezyIEzQHbi6fmyGVE4KLFuxK321aSMhLpQnaNFJ5edWEpvj7UzA2kh8/SMxy
```

```
hLqVfd6IhKVLn4YAx9rz956orQalNKL2U2Sk+h9y3rsBtAaq/A2iuyoEtKX1hEfG
keXenvQZJrh6A1lKcJuvUJgxWx2+
-----END CERTIFICATE REQUEST-----
[darksasuke@chocolat test ]$ openssl req -x509 -new -key server.key -ou
ANIZATION_NAME}/CN=${COMMON_NAME}"
req: No value provided for subject name attribute "C", skipped
req: No value provided for subject name attribute "L", skipped
req: No value provided for subject name attribute "O", skipped
req: No value provided for subject name attribute "CN", skipped
[darksasuke@chocolat test ]$ cat server.crt
-----BEGIN CERTIFICATE-----
MIIC+TCCAeGgAwIBAgIUFkcskZylwZJb5NHbY8vUYSRzjAEwDQYJKoZIhvcNAQEL
BQAwDDEKMAgGA1UECAwBIDAeFw0yMzAyMDIxNTQwNTBaFw0yMzAzMDQxNTQwNTBa
MAwxCjAIBgNVBAgMASAwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCL
q4MiJNdMtDiN+RpRCqKlrwTMWaMf1Z4D/amzB4OPLRfpLBI6QSz+Hu2eVq0TQYAr
wE5euMWCSiGRhaH3uYlXjdBf/lxlbbt3pdT33Aqw0a7mYUHq68S5ZISCQYZsAv8o
MhklP4dU80sAK7t0QSbLyBWAQU0TlE7NF7BqfmMCKR718aoE4cKB2i81u94wzx5R
Jsq9KY6qjkzPyGr4o/I6IVm8NgGkVxplDiCvQfjllIE6yCi63AepZesubfFLLfiW
mxS7GjjHovnNzBpyhOM3osCkpbLwUEEEMNx6tdpC7CecPbZ+0yp/81cwXmEieHU1
giZ86mbnl7tUVxi96tPLAgMBAAGjUzBRMB0GA1UdDgQWBBQzaBdFuXW0jitv0+I/
ej8xpvBE2zAfBgNVHSMEGDAWgBQzaBdFuXW0jitv0+I/ej8xpvBE2zAPBgNVHRMB
Af8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQBC0GFMuXrBx48YaHJOvGdIoXjN
AqS/qvSlOJmP5KzGkEMEUz+CImuhvF5RrFJ17EQdbxLJYWxbpKykt3C/Ok477svP
Zep4Ug6m3wPoY+rqTcBQMVSJ8kgGfF9t0CWAM/407gGUdD5HMVQMdcO12obVFNet
H7rxqO8QRAW5Nt8BXQ17QoX5jR2/zLLlQfUNK1h5C0kMLTClSBJOU/RfiX2YXIBz
7nylJMAmFpACUAq33nz6OkHHpl7Z8CKW6SE/l/X5kj8+TN889j5ElLyOBxemcZGC
orLoZBlhwrf+OZ7ZPCT3U+/LrJomoYI3cn9+ONY3iFfG576R5FhHwbZV3NV9
-----END CERTIFICATE-----
```

Pour lancer le docker-compose :

Bash **init.sh**

```
vel@vel-Precision-3650-Tower:~/Atelier/docker/correction_Freezer/freezer$ bash init.sh
 Creating network "freezer_freeze" with driver "bridge"
 Creating volume "freezer_nginx_pro" with default driver
 Building agent
 Sending build context to Docker daemon  7.168kB
 Step 1/6 : FROM archlinux
  ---> b457d2cc4d4c
 Step 2/6 : RUN pacman -Sy --noconfirm && pacman -S --noconfirm git python-pip jdk11-openjdk openssh python &&
 useradd jenkins -m && mkdir /home/jenkins/.ssh && pip install pylint
  ---> Running in eab8d40f3267
 :: Synchronizing package databases...
  core downloading...
  extra downloading...
  community downloading...
 resolving dependencies...
```

```
jen_1          | Processing provided DSL script
jen_1          | Creating pipeline Freezer
jen_1          | 2023-02-02 16:00:01.374+0000 [id=45]   INFO    j.j.plugin.JenkinsJobManagement#createOrUpdate
Config: createOrUpdateConfig for Freezer
jen_1          | 2023-02-02 16:00:01.741+0000 [id=62]   INFO    jenkins.InitReactorRunner$1#onAttained: System
 config adapted
jen_1          | 2023-02-02 16:00:01.741+0000 [id=62]   INFO    jenkins.InitReactorRunner$1#onAttained: Loaded
 all jobs
jen_1          | 2023-02-02 16:00:01.746+0000 [id=50]   INFO    jenkins.InitReactorRunner$1#onAttained: Config
uration for all jobs updated
jen_1          | 2023-02-02 16:00:01.772+0000 [id=118]  INFO    hudson.util.Retrier#start: Attempt #1 to do th
e action check updates server
jen_1          | 2023-02-02 16:00:01.901+0000 [id=44]   INFO    jenkins.InitReactorRunner$1#onAttained: Comple
ted initialization
jen_1          | 2023-02-02 16:00:01.987+0000 [id=32]   INFO    hudson.lifecycle.Lifecycle#onReady: Jenkins is
 fully up and running
jen_1          | [02/02/23 16:00:02] SSH Launch of agent on 1.2.3.20 completed in 2,080 ms
jen_1          | 2023-02-02 16:00:10.935+0000 [id=118]  INFO    h.m.DownloadService$Downloadable#load: Obtaine
d the updated data file for hudson.tasks.Maven.MavenInstaller
jen_1          | 2023-02-02 16:00:10.936+0000 [id=118]  INFO    hudson.util.Retrier#start: Performed the actio
n check updates server successfully at the attempt #1
```

Pour ce projet, les problèmes rencontré ont surtout été sur les formats des clé SSH et leur génération, c'est pour cette raison qu'un dossier init.sh a été réalisé