TP R4D09 Fondamentaux de la conteneurisation

4 Création d'images Docker

Création d'une image docker :

La commande : - `export DOCKER_BUILDKIT=1` active BuildKit, un moteur de construction de conteneurs Docker plus rapide et plus sécurisé que l'ancien moteur de construction.

La commande : - `export COMPOSE_DOCKER_CLI_BUILD=1` indique à Docker Compose d'utiliser la même interface CLI que Docker pour construire des images, plutôt que l'ancienne méthode utilisant l'API de Docker.

Accéder au conteneur : `docker exec -it <mycontainer> bash`

Docker crée des conteneurs applicatifs. Pour le cloud.

SEARCH se renseigner et cgroups/namespace/firejail/capabiliies Docker pull/Push/ docker run = create + start

- Image arborescence sans programme/service/processus
- Tous les arreter/les supprimé = `docker "stop ou rmi" $(docker ps -a)`
- Commit cree une image en lien l'arborescence et les processus.
- Ont limite le nombre de ligne dans un dockerfile, pour reduire les couche.
- EXPOSE permet a d'autre container e venir sur ce port.ex bdd. CMD ne crée pas de couche.
- ENTRYPOINT est le binaire et CMD les arguments. Mais les deux font le boulot. `docker build -t .` cree une image. `dockerbuild buildx` : `docker logs <mycontainer>` : `docker attach` : s'attache à un processus qui récupère la sortie du process.
- Fournies par la boîte à outils du générateur BuildKit. La commande docker buildx build offre : création d'instances de générateur étendues, la construction simultanée sur plusieurs nœuds, la configuration des sorties, la mise en cache de la construction en ligne et la spécification de la plate-forme cible. En outre, Buildx prend également en charge de nouvelles fonctionnalités qui ne sont pas encore disponibles pour la construction standard de docker, telles que la création de listes de manifestes, la mise en cache distribuée et l'exportation des résultats de construction vers des archives d'images OCI.
- VETH reseau virtuel docker. Recopie le paquet de la carte vers le docker donc très rapide. Le mieux est 1 bridges par application.

`--restart` redémarre le container à chaque boot.

4.1 Build d'une image Docker Debian

1: docker build -t ubuntu:juju .

2:

RUN : Démarre un conteneur = docker run correspond à faire un "docker create + docker start"

ENV : Permet de définir une valeur à une variable d'environement. Commande : `docker run -e VARIABLE=AutreValeur image_name`

FROM : A partir de quel arborescence de base exemple debian, ubuntu, python.

3: Réduire le nombre de couche permet augmenter la rapidité. Une couche est une opération dans un dockerfile, RUN,CMD, etc...

4: Traceroute :

```
FROM ubuntu:latest
|
|RUN apt update -y && apt upgrade -y && apt install -y \
|
|traceroute
|
|CMD traceroute iutbeziers.fr
```

5: -rm permdocker run -e VARIABLE=AutreValeur image_name et de le remove après execution du docker donc d'éviter d'avoir une longue liste de docker en arrière plan "docker ps".

6-7: `docker run --entrypoint traceroute ubuntu:juju www.google.fr`

8: docker commit ubuntu:juju

9-10: docker login registry.iutbeziers.fr docker build -t julien.alleaume /chemin/repertoire docker tag julien.alleaume registry.iutbeziers.fr/julien.alleaume/adet-ping docker push registry.iutbeziers.fr/julien.alleaume docker search registry.iutbeziers.fr/julien.alleaume

11: doccker pull registry.iutbeziers.fr/mathieu.puig

4.2 Installation d’un "insecure registry" sur votre poste de travail

{ "insecure-registries" : ["http://myregistrydomain.com:5000"] }

1: docker run -d -p 5000:5000 --restart=always --name registry -v "$(pwd)":/var/lib/registry registry:2

2: docker pull ubuntu:latest 3: docker tag ubuntu:latest localhsot:5000/ubuntu:latest 4: docker push localhost:5000/ubuntu:latest 5: docker image remove ubuntu:latest 6: docker image remove localhost:5000/ubuntu:latest 7: docker pull localhost:5000/ubuntu:latest

4.3 Création d’un Dockerfile afin de générer une image debian ssh

```
FROM registry.iutbeziers.fr/debianiut

RUN apt-get update && \
```

```
    apt-get install -y openssh-server && \
    mkdir /var/run/sshd && \
    echo 'root:password' | chpasswd

RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/'
/etc/ssh/sshd_config && \
    #echo 'Port 2222' >> /etc/ssh/sshd_config && \
    echo 'PasswordAuthentication yes' >> /etc/ssh/sshd_config

EXPOSE 2222

CMD ["/usr/sbin/sshd", "-D"]
```

docker build -t debian-ssh . docker run -d -p 2222:22 --name debian-ssh debian-ssh ssh root@localhost -p 2222

4.4 Dockérisation d'une application Python

4.4.1 Sans le container lancez l'appliquette suivante fonctionnant avec Python3

Déployement de python dans un conteneur :

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Le Python c'est bon mangez en"

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0', port=9999)
```

4.4.2

Installation de Flask et création du fichier python.

Création du fichier APP.py :

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Le Python c'est bon mangez en"

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0')
```

```
#Fichier Dockerfile pour faire tourner l'APP flask :
FROM python:3.10

WORKDIR /usr/app/src

RUN pip3 install flask
RUN export ENV FLASK_APP=app.py && export ENV FLASK_ENV=development

COPY app.py ./

CMD [ "python", "./app.py"]
```

`docker run -p 9999:5000 python:APP` ("-p 9999:5000" expose le port sur le réseau de la salle ou internet et redirige le port 5000 vers le 9999)

-p et EXPOSE" pas obliger pour rendre accessible le port, cela marche sans le -p mais accessible uniquement en local.

4.4.3

1 :

Création du fichier appv2.py :

```
from flask import Flask,jsonify, request
import os

app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Le Python c'est bon mangez en\n"

@app.route('/whoami')
def get_tasks():
    ipv4 = os.popen('ip addr show eth0').read().split("inet ")
[1].split("/")[0]
    return jsonify({'ip_hote': ipv4}), 200

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0',port=5000)
```

2 :

VOIR FICHIERS LOCAUX POUR CONF

Test de nip.io:

```
host 192.168.1.21.nip.io
192.168.1.21.nip.io has address 192.168.1.21

host whoami.192.168.1.21.nip.io
whoami.192.168.1.21.nip.io has address 192.168.1.21
```

Créations des certificats :

```
mkcert 192.168.1.21.nip.io "whoami.192.168.1.21.nip.io"
"*.192.168.1.21.nip.io"
```

Teste de l'application avec la commande `curl -H ost:whoami.192.168.1.95.nip.io http://whoami.192.168.1.95.nip.io`

5 Réseaux docker

5.1:

docker network ls

5.2:

docker network create -d bridge Question2

5.3:

Le NAT est fait en iftable et les network sont de base en NAT. Pour rattacher le container au reseau cree : `docker run -itd --network=Question2 MonContainer`

5.4:

Macvlan et IPvlan sont directement rattaché au réseau de la salle/datacenter. En se basant sur le subnet.
macvlan: docker network create -d macvlan --subnet=10.202.0.103/16 --gateway=10.202.255.254 -o parent=enx34298f74bdf7 Question4

5.5:

Vlan de mon interface réseau : `ip link add link eth0 name eth0.10 type vlan id 10`

ipvlan: `docker network create -d ipvlan --subnet=10.202.0.103/24 --gateway=10.202.255.254 -o parent=enx34298f74bdf7.20 ipvlan20`

Possibilité de ne pas mettre le '.20' derrière le nom de la carte réseau.

5.6:

Macvlan (vieux protocole qui sert lorsqu'une application à besoins d'une MAC) et IPvlan (Protocole de layer 2+, plus rapide et fiable que macVlan) sont directement rattaché au réseau de la salle/datacenter. En se basant sur le subnet. Le NAT est fait en iftable et les networks sont de base en NAT.

5.7:

Il utilise le DNS par défaut de la machine hôte, sinon il faut le configurer en faisant `docker run --dns image_name`.

6 Tips & Tricks

6.1 Connexion à distance au daemon Docker

docker context create ubuntuvm --docker "host=ssh://student@VOTRE_IP" Permet d'avoir different environnement, plus précisément, un contexte définit l'ensemble des images, conteneurs, volumes et autres ressources Docker disponibles pour une commande donnée

6.2 Débugguer un container

Busybox permet d'avoir des commande d'administration, débuggage, etc... Temmoraire car il suffit de simplement enlever le binaire pour enlever de potentiel vulnérable.

Pensé à `chmod +x busybox`, sinon impossible à executer dans le conainer.

`docker run -d --rm httpd` Démarre un container WEB. `docker cp ./busybox 5942f144bd32:/` Copie du binaire dans le container. `docker exec -it 5942f144bd32 /busybox ip a` Exécution de commande via busybox dans le container.

Installation de nsenter permet l'execution de programme dans différent namespace.

`nsenter -t 35159 -p -u -n -i`