UNIVERSITÉ DE RENNES 1

ESIR ECOLE SUPERIEURE D'INGENIEURS DE RENNES

27/01/2019

# TD6

> *« J'atteste que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée »*

Léo Guilpain – ESIR3 - IoT

# Table des matières

# Introduction

Le but de ce TP est de déployer notre galerie d'images sur un cluster de serveurs en utilisant Docker Swarm.

# Partie I - Retour sur les bases de Docker

## Question 01

**Docker :**

- Docker est basé sur les conteneurs.
- Un seul processus dans un conteneur. Le processus est isolé du reste du système.
- Docker est utilisé pour isoler des applications individuelles.

**VM :**

- La date et l'heure ne sont présent dans une VM.
- Composé de plusieurs processus
- Même le hardware est virtualisé dans une VM.
- Une VM a un OS et des applications
- Une VM isole un système entier.

Je me suis servi de ce site : https://devopsconference.de/blog/docker/docker-vs-virtual-machine-where-are-the-differences/

## Question 02

Contrairement à une VM, un Docker n'isole pas le système entier. Cela signifie que l'OS et les applications sont isolés par rapport au conteneur. Le processus s'exécutant sur le conteneur est alors isolé du reste du système.

## Question 03

Comme je l'ai dit dans les deux premières questions, un conteneur Docker ne peut exécuter qu'un seul processus. Cela signifie donc que si le WordPress de ma tante nécessite plus d'un seul processus alors je serai dans l'obligation d'avoir plusieurs conteneurs Docker.

## Question 04

Lorsque l'on installe Docker, généralement on crée le groupe docker. Ce groupe est « limité » à l'utilisateur root. Ensuite il faut ajouter notre utilisateur au groupe docker « sudo usermod -a -G docker $user ». En faisant un volume sur notre répertoire root, on peut modifier les fichiers de notre ordinateur depuis le conteneur sans même être superutilisateur. Je me suis servi de ce site : https://julienc.io/18/utiliser_le_client_docker_sans_etre_root

## Partie II - Docker Swarm

Tout d'abord Docker Swarm permet d'organiser un ensemble de conteneur sur plusieurs serveurs et non sur un seul comme c'était le cas avant. Cela permet de rendre notre installation plus résistante au crash. Si le worker1 crash alors le worker2 est toujours en mesure de prendre le relais.

Dans un premier temps, on initialise le swarm sur le manager. L'adresse IP renseignée ici est l'adresse du manager.

```
vagrant@manager:~$ sudo docker swarm init --advertise-addr 192.168.42.17
Swarm initialized: current node (sbtudnuhl2lghoeqyuvxqija1) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-38u1gbu4f76mdok6yeeys5y3mlkkngy0h9sxscrnm62chxo2ar-6gsxa1yywv0nc7kr2v7
nxiog4 192.168.42.17:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

*Figure 1 : Swarm init*

On récupère le token puis on se connecte en SSH aux workers. On rejoint ensuite le swarm avec la commande suivante :

```
vagrant@worker1:~$ sudo docker swarm join --token SWMTKN-1-38u1gbu4f76mdok6yeeys5y3mlkkngy0h9sxscrnm62chxo2ar
-6gsxa1yywv0nc7kr2v7nxiog4 192.168.42.17:2377
This node joined a swarm as a worker.
```

*Figure 2 : docker join*

En exécutant la commande : "docker info", on peut voir que le swarm est bien activé.

```
Swarm: active
 NodeID: sbtudnuhl2lghoeqyuvxqija1
 Is Manager: true
 ClusterID: r5ppynklzy3bjyqtyb2o64ipk
 Managers: 1
 Nodes: 3
```

*Figure 3 : Docker info*

Enfin, avec la commande "docker node ls" on peut voir que les workers sont bien présents dans le swarm. De plus, le manager est bien le leader de ce swarm.

```
vagrant@manager:~$ sudo docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
sbtudnuhl2lghoeqyuvxqija1 *   manager             Ready               Active              Leader              18.09.1
z9e5hmahsaujvfxvs12gbs2p2     worker1             Ready               Active                                  18.09.1
vwzsnaop73xwnaz4kr4gu0dma     worker2             Ready               Active                                  18.09.1
```

*Figure 4 : docker node ls*

On cherche à mettre les machines dans le même réseau. On crée donc un réseau que l'on nomme "test_net".

```
vagrant@manager:~$ sudo docker network create -d overlay --attachable test_net
8e6gih1vik1zotis89bzh35co
vagrant@manager:~$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
9197d651d533        bridge              bridge              local
2fc5a4ca30ed        docker_gwbridge     bridge              local
87e5f01dcd7a        host                host                local
hax8x842g5fa        ingress             overlay             swarm
cb5f51d0e202        none                null                local
8e6gih1vik1z        test_net            overlay             swarm
```

*Figure 5 : create network*

Dans la commande, on peut voir plusieurs éléments :

- overlay : visible par tout le monde
- -- attachable : workers peuvent utiliser ce réseau

Dans un premier temps, on met la base de données sur un conteneur.

On exécute les commandes suivantes afin de faire en sorte que la base de données soit disponible sur tous les workers :

```
vagrant@worker1:~$ chmod -R 777 /vagrant/myappgallery
vagrant@worker1:~$ sudo docker rm -f redis-datastore
redis-datastore
vagrant@worker1:~$ sudo docker run --net test_net --name redis-datastore -v /vagrant/myappgallery/data:/data -d redis --appendonly yes
2965d08ddcf8e7dc5e71f5a60ce692086651ff4ea2602adbf276af8180611203
vagrant@worker1:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
2965d08ddcf8        redis               "docker-entrypoint.s…"   20 seconds ago      Up 8 seconds        6379/tcp            redis-datastore
vagrant@worker1:~$ sudo docker run --net test_net -it --link redis-datastore:redis-cli --rm redis redis-cli -h redis-datastore
redis-datastore:6379> keys *
1) "Album:3"
2) "imageCounter"
3) "imageIdCounter"
4) "Image:2"
5) "albumCounter"
6) "Image:1"
7) "albumIdCounter"
8) "Album:4"
redis-datastore:6379>
```

*Figure 6 : base de données*

La première commande permet de donner tous les droits au dossier « data ». En suite on fait un volume pour récupérer les données de notre base de données.

En testant depuis worker2, on voit bien qu'il a accès à la base de données.

```
vagrant@worker2:~$ sudo docker run --net test_net -it --link redis-datastore:redis-cli --rm redis redis-cli -h redis-datastore
redis-datastore:6379> keys *
1) "Album:3"
2) "imageCounter"
3) "imageIdCounter"
4) "Image:2"
5) "albumCounter"
6) "Image:1"
7) "albumIdCounter"
8) "Album:4"
```

*Figure 7 : keys **

On veut désormais utiliser des services :

```
vagrant@manager:~$ sudo docker service create --name registry --publish published=5000,target=5000 registry:2
jmanvfxmkpnf8j4orea3mt6c8
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
vagrant@manager:~$ sudo docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
jmanvfxmkpnf        registry            replicated          1/1                 registry:2          *:5000->5000/tcp
```

*Figure 8 : creation d'un registre*

On crée dans un premier temps un registre qui va stocker toutes les images.

Depuis les worker, on fait “curl http://localhost:5000/v2/”. On obtient ceci :

```
vagrant@worker1:~$ sudo curl http://localhost:5000/v2/
{}vagrant@worker1:~$
```

*Figure 9 : curl*

Les workers ont bien accès au registre.

On souhaite désormais alimenter notre registre en image.

On remplit notre “docker-compose.yml” comme ci :

```yaml
version: '2'
services:
  web:
    image: "localhost:5000/myappgallery"
    build: .
    ports:
     - "3000:3000"

```

*Figure 10 : docker-compose.yml*

Sur le manager, on exécute la commande “docker-compose build” puis on “push” l’image sur le registre.

```
vagrant@manager:/vagrant/myappgallery$ sudo docker-compose push
Pushing web (localhost:5000/myappgallery:latest)...
```

*Figure 11 : docker-compose build*

Après avoir push les images, on se place dans les “workers” puis on pull les images depuis le registre.

```
vagrant@worker1:~$ sudo docker pull localhost:5000/myappgallery
Using default tag: latest
latest: Pulling from myappgallery
ab1fc7e4bf91: Pull complete
35fba333ff52: Pull complete
f0cb1fa13079: Pull complete
3d1dd648b5ad: Pull complete
49f7a0920ce1: Pull complete
1d199f738c5f: Pull complete
8968c17918fc: Pull complete
b7a5354dec59: Pull complete
144107ca3c0f: Pull complete
ad5cd3d1a0eb: Pull complete
adbcb52b5355: Pull complete
c2822f01d6e8: Pull complete
Digest: sha256:18eeaf676ef04c77c9f8105d7c067359c438886e1a0c06071b3165f08ae89e8a
Status: Downloaded newer image for localhost:5000/myappgallery:latest
vagrant@worker1:~$
```

*Figure 12 : docker pull*

On voit bien ici que cela s'est fait correctement. Cela signifie que les images sont bien correctement stockées sur le registre.

Dans un premier temps, on test le bon fonctionnement de notre registre en créant un service à partir de l'image que l'on vient de push "localhost:5000/myappgallery". Dans ce service, on créé 3 replicas.

```
vagrant@manager:/vagrant/myappgallery$ sudo docker images
REPOSITORY                   TAG                 IMAGE ID            CREATED             SIZE
localhost:5000/myappgallery   latest              3272833b7d46        2 hours ago         924MB
node                         8                   1f6c34f7921c        37 hours ago        893MB
registry                     <none>              116995fd6624        5 days ago          25.8MB
vagrant@manager:/vagrant/myappgallery$ sudo docker service create --name myappservice --replicas=3 localhost:5000/myappgallery
ts7hojssuxqw14hhyagpzbf35
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
vagrant@manager:/vagrant/myappgallery$
```

*Figure 13 : création d'un service*

On a donc créé un service nommé "myappservice" qui sera répliqué 3 fois. Comme on peut le voir sur la commande suivante le service est correctement exécuté 3 fois.

```
vagrant@manager:/vagrant/myappgallery$ sudo docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                                PORTS
ts7hojssuxqw        myappservice        replicated          3/3                 localhost:5000/myappgallery:latest
1l6hxdxsbo5j        registry            replicated          1/1                 registry:2                           *:5000->5000/tcp
vagrant@manager:/vagrant/myappgallery$
```

*Figure 14 : docker service ls*

Notre service a bien été créé. Pour vérifier le fonctionnement du service on exécute la commande suivante.

```
vagrant@manager:/vagrant/myappgallery$ sudo docker service logs myappservice
myappservice.3.25cjpsjfzw34@manager    |
myappservice.3.25cjpsjfzw34@manager    | > myappgallery@0.0.0 start /usr/src/myappgallery
myappservice.3.25cjpsjfzw34@manager    | > node ./bin/www
myappservice.3.25cjpsjfzw34@manager    |
myappservice.3.25cjpsjfzw34@manager    | events.js:183
myappservice.3.25cjpsjfzw34@manager    |       throw er; // Unhandled 'error' event
myappservice.3.25cjpsjfzw34@manager    |       ^
myappservice.3.25cjpsjfzw34@manager    |
myappservice.3.25cjpsjfzw34@manager    | Error: Redis connection to redis:6379 failed - getaddrinfo EAI_AGAIN redis:6379
myappservice.3.25cjpsjfzw34@manager    |     at GetAddrInfoReqWrap.onlookup [as oncomplete] (dns.js:67:26)
```

*Figure 15 : dockers service logs*

Comme on peut le voir, notre service est fonctionnel cependant comme le service de redis n'est pas implémenté il y a une erreur.

Toute cette partie est fonctionnelle mais ce n'est pas exactement comme cela qu'il faut faire. J'ai donc créé un "docker-stack.yml" pour pouvoir lancer les services depuis un docker stack deploy.

```yaml
version: "3"
services:
  web:
    image: localhost:5000/myappgallery
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
    ports:
      - 3000:3000
    networks:
      - test_net

  redis:
    image: redis
    ports:
      - "6379:6379"
    volumes:
      - "/vagrant/myappgallery/data:/data"
    deploy:
      placement:
        constraints: [node.role == manager]
    command: redis-server --appendonly yes
    networks:
      - test_net
networks:
  test_net:
    external: true
```

*Figure 16 : docker-stack.yml*

Comme on peut le voir, on exécute 2 services. LE premier est le service web. Ce dernier va utiliser l'image que l'on a précédemment « push » dans le registre. Ce service aura 3 répliques. Le deuxième service correspond à l'image redis. Un volume sera fait pour récupérer les données. On rajoute une contrainte lors du déploiement pour forcer le déploiement sur le manager. Le manager est notre source « sûre ». Ce dernier n'est pas sensé crashé. C'est pourquoi nous avons déployé la base de données sur celui-ci. En revanche, ce n'est pas la meilleure solution car s'il arrive un problème au manager, alors nous perdons toutes les données présentes dans la base.

Ces deux services seront déployés sur le réseau que l'on a préalablement créé « test_net »

On exécute ensuite la commande :

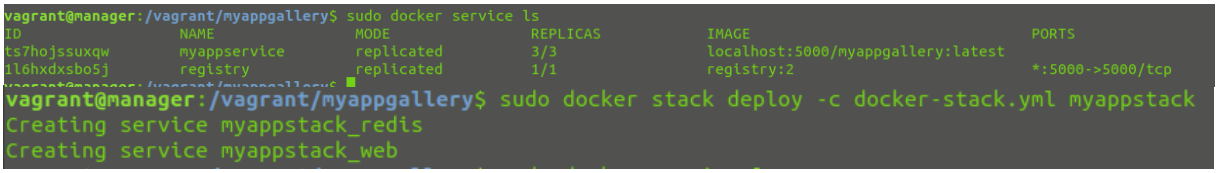


*Figure 17 : stack deploy*

Les deux services ont donc bien été créé correctement.

Pour vérifier le bon fonctionnement, Il suffit de renseigner l'adresse du manager ou alors des workers dans la barre de recherche

Vous pouvez voir ci-dessous l'exemple avec l'adresse IP du manager :
http://192.168.42.17:3000/

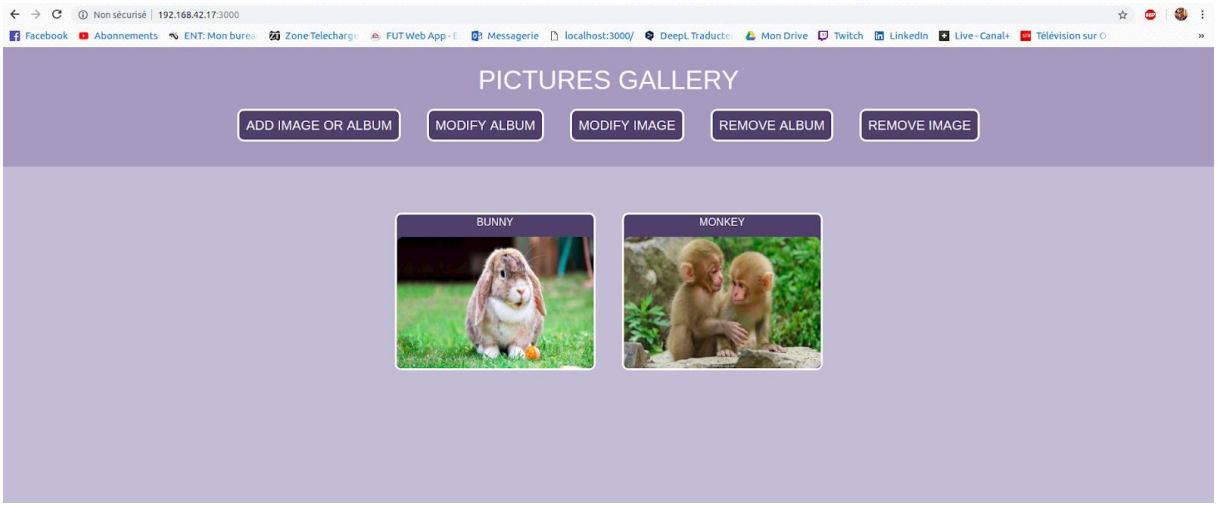


*Figure 18 : gallery*

Le site s'affiche normalement donc nos services fonctionnent correctement.

# Conclusion

Dans ce TP sur la création de service, la difficulté a été de mettre en place nos différents services. Après avoir paramétrer correctement notre fichier « docker-swarm.yml », notre gallérie fut de nouveau fonctionnelle. Malheureusement, je n'ai pas eu le temps de gérer mes photos avec Minio.