TD 6

Introduction

Au cours de ce TP nous allons déployer notre galerie d'images sur un cluster de serveurs utilisant Docker Swarm.

Partie I – Retour sur les bases de Docker

1)

Une première étape consiste à mettre en évidence les différences entre conteneurs et machines virtuelles. S'ils ont tous les deux pour but de déployer plusieurs services isolés sur une même plate-forme, il convient que leurs architectures ne sont pas tout à fait similaires.

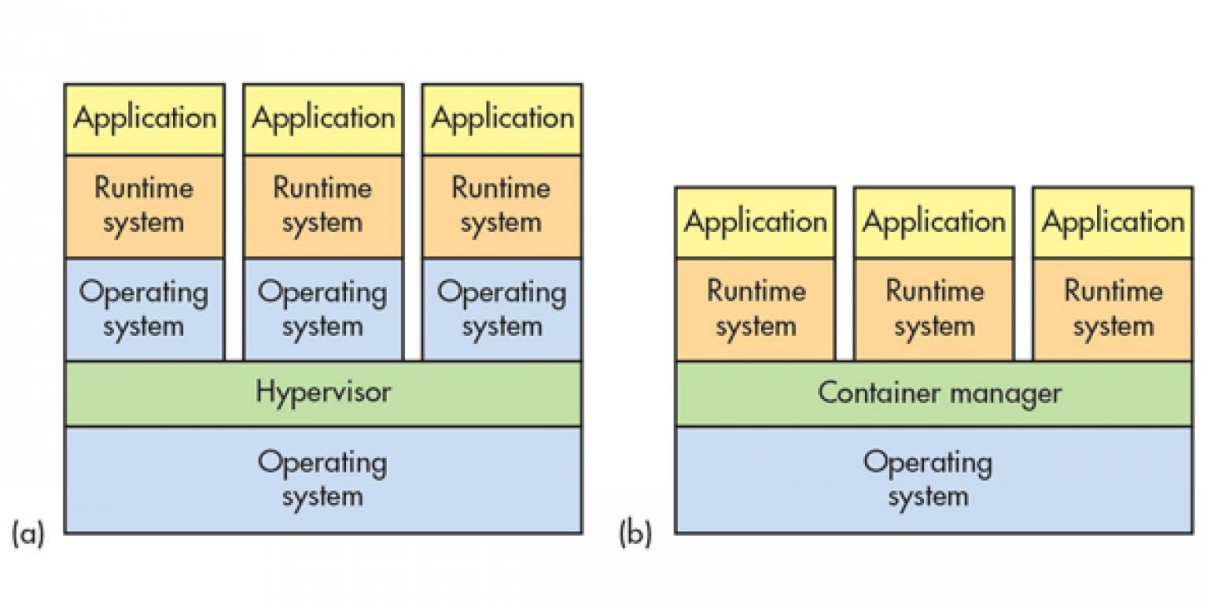


Figure 1 : Architecture d'une machine virtuelle (à gauche) et d'un conteneur (à droite)

Dans la Figure 1 à droite, on remarque que le système de conteneur a besoin d'un système d'exploitation sous-jacent lui permettant de fournir les services de base à toute application conteneurisée et utilisant la prise en charge de la mémoire virtuelle pour l'isolation. Cette approche permet entres autres de réduire le gaspillage des ressources dans la mesure où chaque conteneur ne renferme que l'application et les fichiers binaires ou bibliothèques associées. On utilise ainsi le même OS hôte pour plusieurs conteneurs.

D'autre part à gauche, un hyperviseur exécute des machines virtuelles dotées de leur propre système d'exploitation. Le conteneur de chaque application est donc libéré à la charge d'un OS, ce qui permet de la rendre plus facile à migrer, télécharger… La virtualisation a pour objectif de répondre à la demande suivante : Comment puis-je, sur une seule plateforme, exécuter une application historique ainsi qu'une nouvelle application ?

2)
Docker a pour but d'isoler des applications avec tous les binaires et dépendances nécessaires à leur exécution.

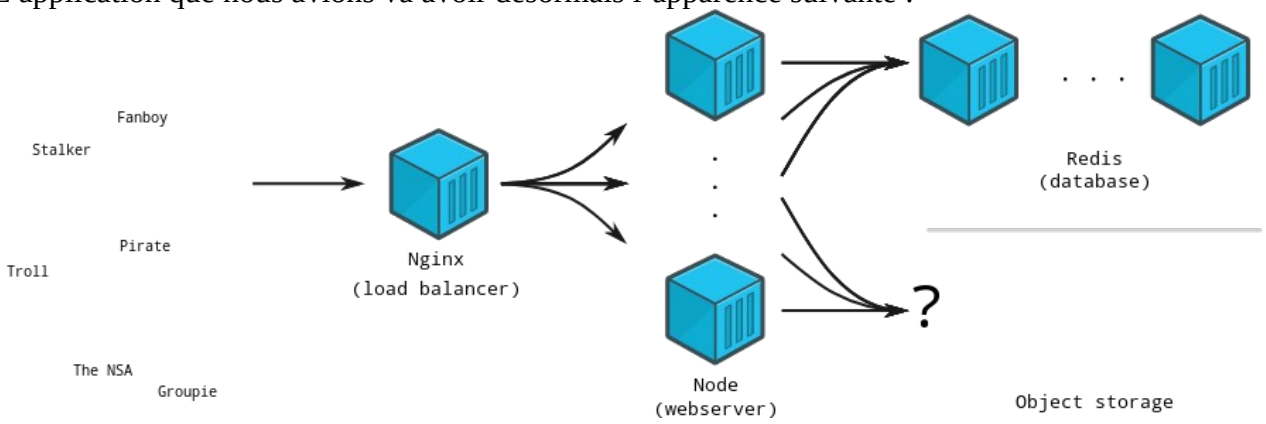
3)
Pour empaqueter un Wordpress il ne vaut mieux pas tout mettre dans un conteneur. En effet, un conteneur contient un unique processus. Il faut alors mettre en place plusieurs conteneurs. On va par exemple séparer le serveur de la base de données.

4)
Ce sont des incapables car ils n'auraient pas dû me laisser dans le groupe docker. En effet, en partageant le volume qui contient la gestion des utilisateurs, je peux modifier cette configuration et m'ajouter en tant qu'administrateur.

Partie II – Docker Swarm

1)
On reprend ici les travaux effectués lors du TD5 pour initialiser le swarm à l'aide d'Ansible. J'ai donc repris mon répertoire « 3VM » dans lequel j'avais fait cette partie.

L'application que nous avions va avoir désormais l'apparence suivante :



Nous allons notamment nous familiariser avec la notion de services avec Docker Swarm.

On commence par créer un réseau overlay. On se connecte au manager en ssh.

```
vagrant@manager:~$ sudo docker network create --attachable --driver overlay test-net
wgdcj45qqtw17q726rtfbrubf
```

On crée un service ayant pour nom « registry »

```
vagrant@manager:~$ sudo docker service create --name registry --publish published=5000,target=5000 registry:2
q9ecw9jxau638hkepe5s1egtu
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
q9ecw9jxau63        registry            replicated          1/1                 registry:2          *:5000->5000/tcp
vagrant@manager:~$
```

Dans notre application « myhbsapp » on va venir modifier le fichier « docker-compose.yml ». On ajoute la ligne : « image : 127.0.0.1:5000/myhbsapp ». 5000 correspond au port indiqué dans le published pour la création du service (voir au-dessus).

docker-compose... | manager_playbo... | worker_playboo...

```
version: '2'

services:
  web:
    image: 127.0.0.1:5000/myhbsapp
    build: .
    ports:
     - "3000:3000"
  redis:
    image: "redis:alpine"
    volumes:
      - "./myvolredis:/data"
```

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose up -d
sudo: docker-compose: command not found
vagrant@manager:/vagrant/myhbsapp$
```

Il est nécessaire d'installer docker-compose pour pouvoir réaliser cette opération. On ajoute ce code au fichier « install_docker.yml » :

```
# sudo apt-get install docker-compose=1.8.*
- name: Install docker-compose
  apt:
    name: docker-compose=1.8.*
    state: present
    update_cache: yes
  tags:
    - docker
```

On détruit les VM puis on les redémarre. Il faut donc répéter les opérations réalisées précédemment pour créer le réseau overlay ainsi que le service registry.

On lance donc notre app à l'aide de la commande suivante :

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the curren
t node.

To deploy your application across the swarm, use the bundle feature of the Docker experimental build.
```

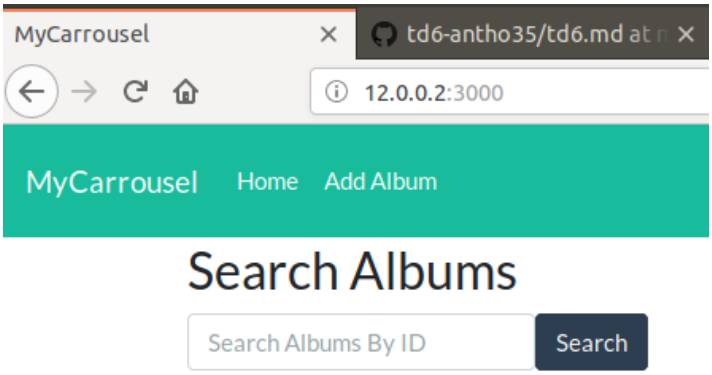
```
Pulling redis (redis:alpine)...
alpine: Pulling from library/redis
cd784148e348: Already exists
48d4c7155ddc: Pull complete
6d908603dbe8: Pull complete
0b981e82e1e2: Pull complete
7074f4a1fd03: Pull complete
447ac2b250dc: Pull complete
Digest: sha256:8e3ba72cd2bb6e508c430f303830e5dc5e4c210f06fd84ac91b2412352d555f7
Status: Downloaded newer image for redis:alpine
Creating myhbsapp_web_1
Creating myhbsapp_redis_1
```

Dans le code ci-dessus, l'image redis est récupérée vu qu'on l'avait déjà en local puis deux conteneurs sont créés : myhbsapp_web_1 et myhbsapp_redis_1.

On regarde si le docker-compose a fonctionné puis on peut tester si l'app est fonctionnelle.

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------------
myhbsapp_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
myhbsapp_web_1     npm start                        Up      0.0.0.0:3000->3000/tcp
```

```
vagrant@manager:/vagrant/myhbsapp$ curl http://localhost:3000
<!DOCTYPE html>
<html>
  <head>
    <title>MyCarrousel</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css">
    <link rel="stylesheet" href="https://bootswatch.com/4/flatly/bootstrap.min.css">
  </head>
  <body>
    <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
      <a class="navbar-brand" href="#">MyCarrousel</a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarsExampleDefault" aria-controls="
avbarsExampleDefault" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
```



En entrant l'addresse Ip du manager (12.0.0.2) ainsi que le bon port, on peut afficher notre application dans le navigateur.

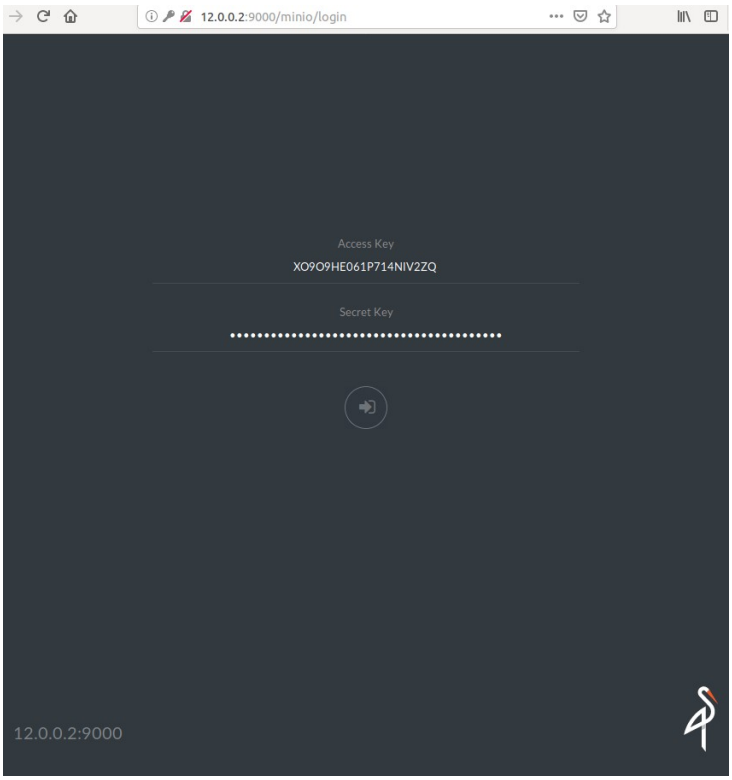
On peut à présent installer minio sur notre manager en lançant la commande :

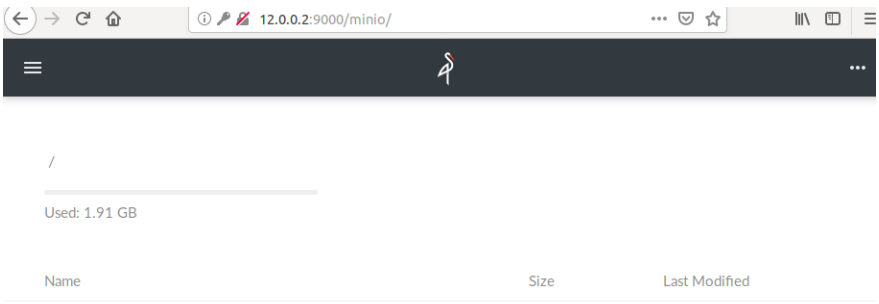
```
vagrant@manager:/vagrant/myhbsapp$ sudo docker run -p 9000:9000 -v /mnt/data:/data -v /mnt/config:/root/.minio minio/minio
server /data
Unable to find image 'minio/minio:latest' locally
latest: Pulling from minio/minio
407ea412d82c: Pull complete
6d27ac77351a: Pull complete
56e4d4a3fde5: Pull complete
Digest: sha256:80985bd9bc1735b48e5869fe157393712985dec28f9da6bb5c19a82f28a0998f
Status: Downloaded newer image for minio/minio:latest

Endpoint:  http://172.17.0.2:9000  http://127.0.0.1:9000

Browser Access:
   http://172.17.0.2:9000  http://127.0.0.1:9000

Object API (Amazon S3 compatible):
   Go:         https://docs.minio.io/docs/golang-client-quickstart-guide
   Java:       https://docs.minio.io/docs/java-client-quickstart-guide
   Python:     https://docs.minio.io/docs/python-client-quickstart-guide
   JavaScript: https://docs.minio.io/docs/javascript-client-quickstart-guide
   .NET:       https://docs.minio.io/docs/dotnet-client-quickstart-guide
```

En rajoutant « -it » à la commande précédente on peut récupérer notre AccessKey ainsi que notre SecretKey.

```
AccessKey: XO9O9HE061P714NIV2ZQ
SecretKey: xeeElLt8jk3NWOB5jTTHblUsGJUCNP4gThEFUpAi
```

Dans notre navigateur on entre l'adresse : 12.0.0.2:9000 pour aller sur notre VM manager en écoutant le port 9000.





On peut modifier notre fichier docker-compose.yml de façon à mettre minio dans un conteneur :

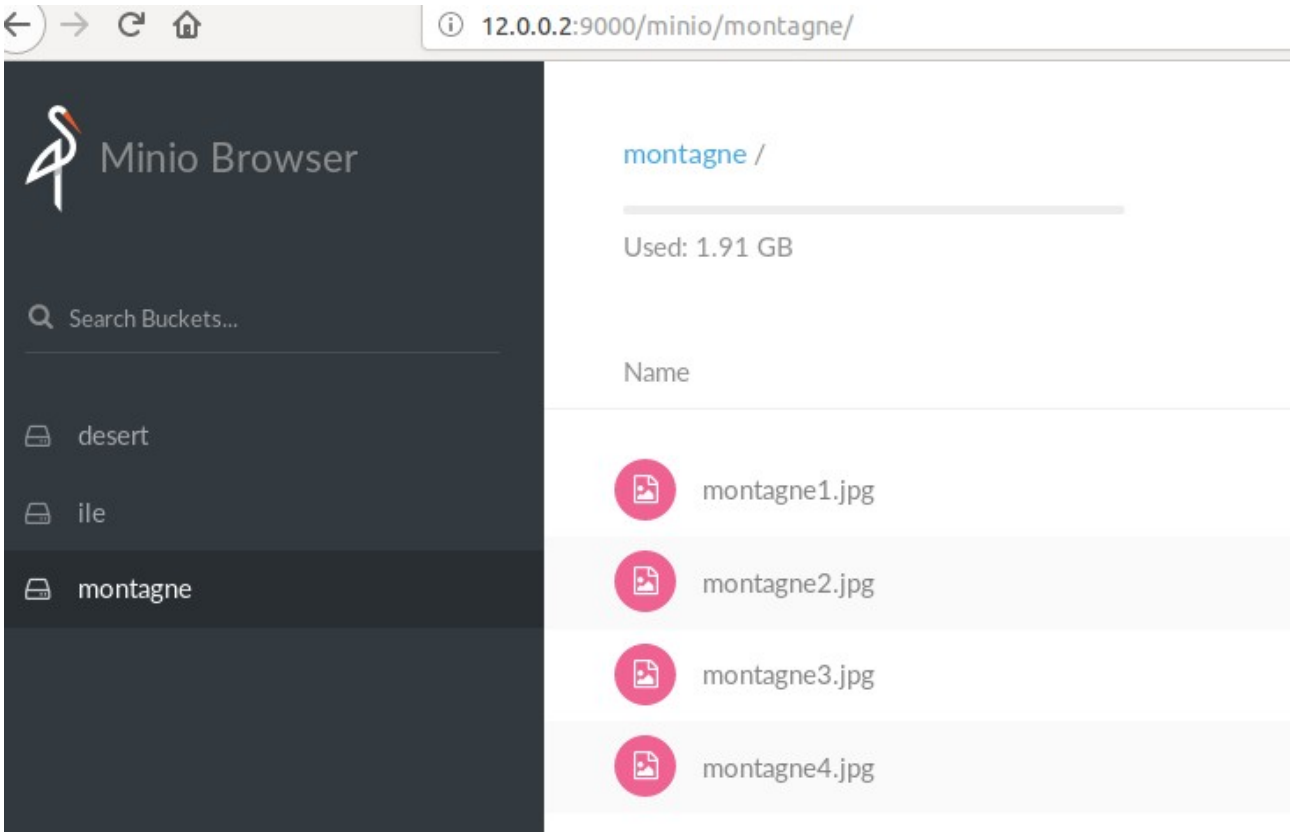
```yaml
minio:
  image: minio/minio
  command: minio server /data
  volumes:
    - "./data:/data"
  ports:
    - "9000:9000"
  environment:
    MINIO_ACCESS_KEY: myminio
    MINIO_SECRET_KEY: myminio123
```

On lance un « docker-compose up -d ».

On a maintenant 3 conteneurs. Le manager, est le chef d'orchestre : il choisira où il place ses répliques des images crées.

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------------
myhbsapp_minio_1   /usr/bin/docker-entrypoint ...   Up      0.0.0.0:9000->9000/tcp
myhbsapp_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
myhbsapp_web_1     npm start                        Up      0.0.0.0:3000->3000/tcp
```

J'ai commencé par ajouter 3 buckets sur minio : desert, ile et montagne qui correspondent aux trois albums de mon application.



On va maintenant procéder au déploiement sur le manager.

On éteint notre app :

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose down --volumes
Stopping myhbsapp_minio_1 ... done
Stopping myhbsapp_redis_1 ... done
Stopping myhbsapp_web_1 ... done
Removing myhbsapp_minio_1 ... done
Removing myhbsapp_redis_1 ... done
Removing myhbsapp_web_1 ... done
Removing network myhbsapp_default
```

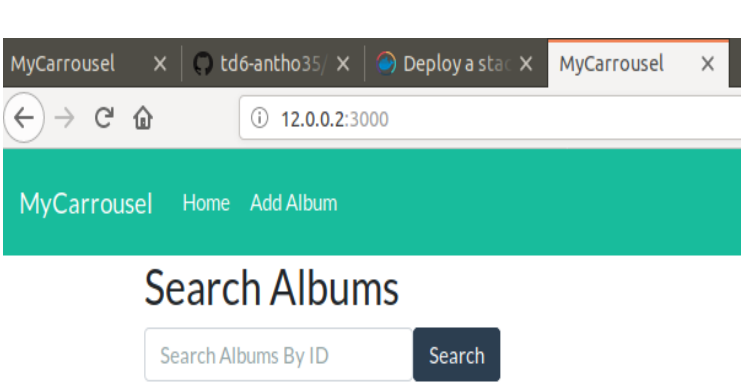
On envoie les images dans le registry créé précédemment :

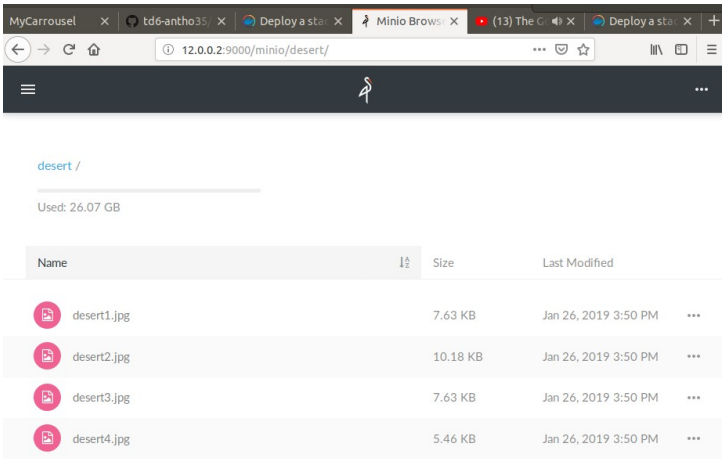
```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose push
Pushing web (127.0.0.1:5000/myhbsapp:latest)...
The push refers to repository [127.0.0.1:5000/myhbsapp]
d845c0ae3dc1: Pushed
581daf238b2c: Pushed
4a3c73e5e7fb: Pushed
ffb143619acd: Pushed
1f3213370fe1: Pushed
734b6a525613: Pushed
7bff100f35cb: Pushed
latest: digest: sha256:ef6a6bbdf1f722a3a5533ea09c5fdf940d75b820c54af4ffbad5eecaf3c29939 size: 1789
```

Puis on déploie sur manager :

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack deploy --compose-file docker-compose.yml myhbsapp
Ignoring unsupported options: build

Creating network myhbsapp_default
Creating service myhbsapp_minio
Creating service myhbsapp_web
Creating service myhbsapp_redis
```

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack services myhbsapp
ID                  NAME                MODE                REPLICAS            IMAGE                                PORTS
9osbidq3lvo0        myhbsapp_web        replicated          1/1                 127.0.0.1:5000/myhbsapp:latest       *:3000->3000/tcp
s37604kjna3u        myhbsapp_redis      replicated          1/1                 redis:alpine
```





L'application (12.0.0.2:3000) et minio (12.0.0.2:9000) sont toujours fonctionnels après le déploiement.

Lorsqu'une modification est réalisée sur minio, par exemple suppression d'une image, celle-ci est également effectuée dans le répertoire /data de notre application « myhbsapp »

Conclusion

C'était un TP conséquent qui m'a permit de découvrir la notion de services notamment mais aussi l'outil de stockage qu'est Minio.