

27/01/2019

Léo Guilpain – ESIR3 - IoT

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 »*

Table des matières

[Introduction 2](#_Toc536196869)

[Partie I - Retour sur les bases de Docker 2](#_Toc536196870)

[Question 01 2](#_Toc536196871)

[Question 02 2](#_Toc536196872)

[Question 03 2](#_Toc536196873)

[Question 04 2](#_Toc536196874)

[Partie II - Docker Swarm 3](#_Toc536196875)

[Conclusion 9](#_Toc536196876)

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.

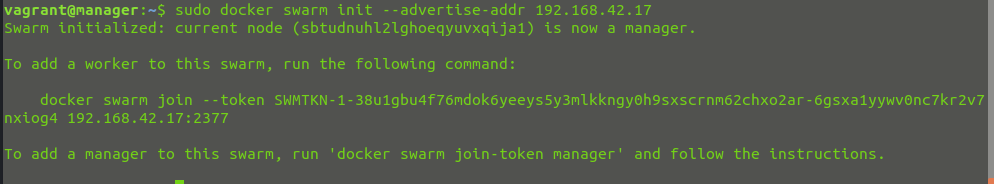


Figure 1 : Swarm init

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

https://lh6.googleusercontent.com/RXubbggDnml4yFtIWu3XMBF2Om3q75ImRLCwrVrv1C03_LDzIzNu58MrVD0Q3M4Fr1dEvQVotN7qlb44YX0Alw2QAjQszd91Ewnw0aalObH1ssBRz97IdL426QGe5R073uaaiAU

Figure 2 : docker join

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

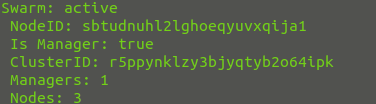


Figure 3 : Docker info

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.



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”.

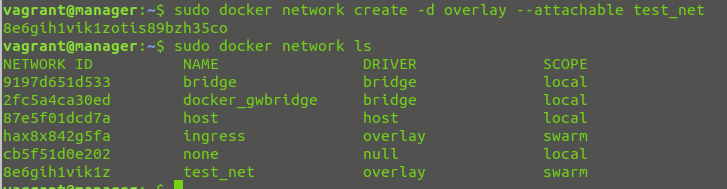


Figure 5 : create network

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 :

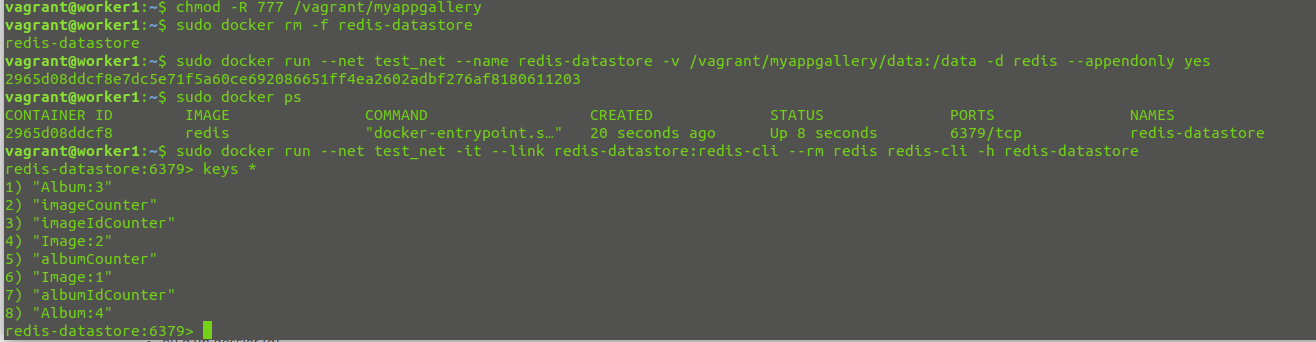


Figure 6 : base de données

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.



Figure 7 : keys \*

On veut désormais utiliser des services :

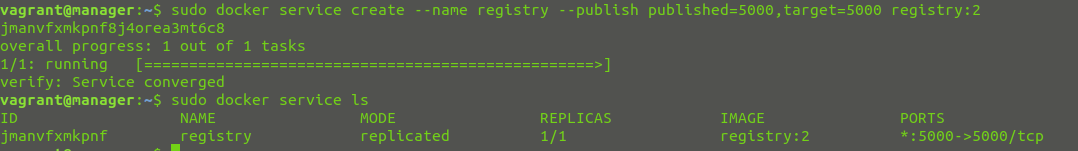


Figure 8 : creation d'un registre

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 :

https://lh4.googleusercontent.com/2oKMyvjZlOOOhYw6dwN67L5LpZr9osZdMZe6iYc1tsHk1iF-kz2BHGlJHHjJ8pQv5RBRzd_Pj9vJ7JsDskop41jRPvZehNvqEdQRd2cO4c-FJSCBWUBgGkXPKXZSfH19P2USvxg

Figure 9 : curl

Les workers ont bien accès au registre.

On souhaite désormais alimenter notre registre en image.

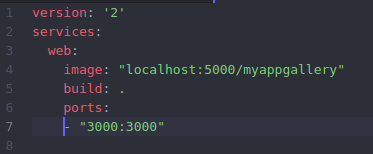
On remplit notre “docker-compose.yml” comme ci :   
  


Figure 10 : docker-compose.yml

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

https://lh3.googleusercontent.com/nAnfzeXSEp2TEm-xWYJJScQtO2vsBjM1WQDCsoCazd1rjTGWBIePdlp4v1wnrRsc6e0HNade19O4yx-piMCkPIcIh5lxFtV3FrAENfsRMWsM8Tm25N0UTWlrIl7cNNm690XV01I

Figure 11 : docker-compose build

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

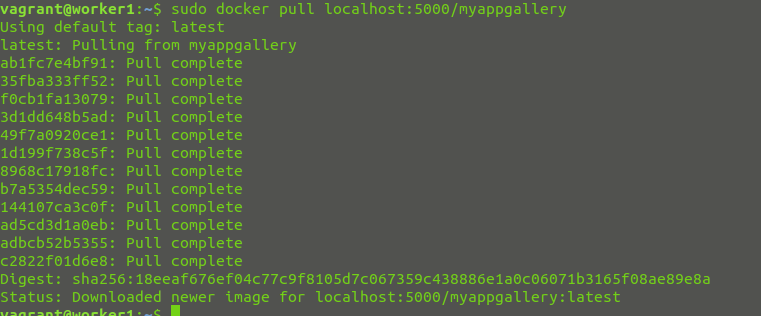


Figure 12 : docker pull

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.

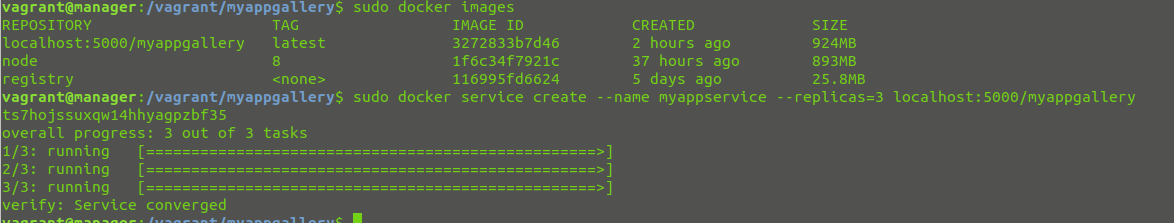


Figure 13 : création d'un service

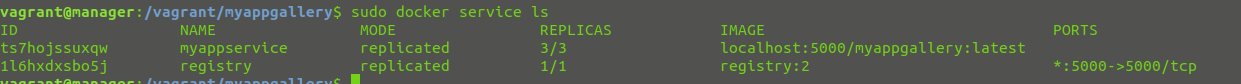
On a donc créé un service nommé “myappservice” qui sera répliqué 3 fois. Comme on peut le voir sur la commande suivante le service est correctement exécuté 3 fois.   
  


Figure 14 : docker service ls

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

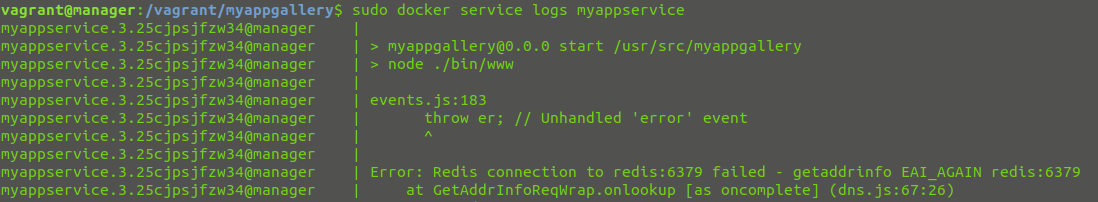


Figure 15 : dockers service logs

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

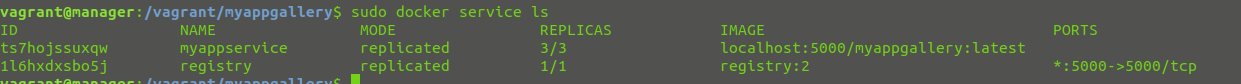
Toute cette partie est fonctionnelle mais ce n’est pas exactement comme cela qu’il faut faire. J’ai donc créé un “docker-stack.yml” pour pouvoir lancer les services depuis un docker stack deploy.



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 :

https://lh4.googleusercontent.com/zs9HMCyMW-avYx_0CvywmFuudIyg2fqMjq6dNDPCe0R4tSCVnvrFdlu7XPPyqqX36e2coFFxspXIk1LmTdWMckaOfzH0aR_y1cB80Ze3cL6DJ_B1C8yJfVRQ19hL21R_SOgfXZQ

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/>

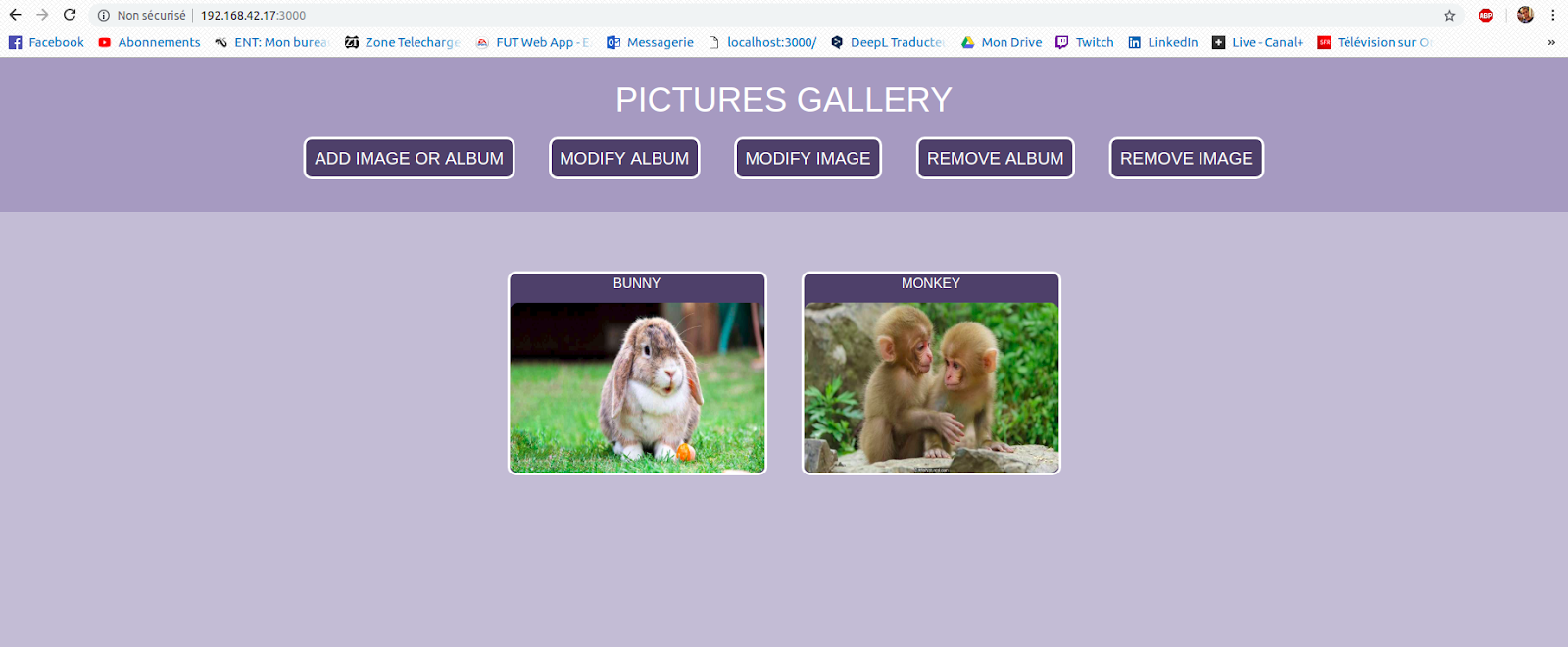


Figure 18 : gallery

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.