TRAVAUX DIRIGÉS
ORCHESTRATEUR - DOCKER SWARM

I. INTRODUCTION

Nous allons étudier une premier exemple d'orchestrateur, la solution embarquée par docker, Swarm. Un orchestrateur permet de gérer un ensemble de conteneurs, intégrant le nombre de répétition d'un même service pour s'adapter à la demande (scalabilité), maintenant ces services actifs, gérant aussi la mise à jour des conteneurs, gérant aussi un cluster de VM, répartissant la charge sur chacune des VM et réagissant automatiquement aux changements d'état du cluster.

II. ENVIRONNEMENT

Nous allons travailler sur le système Debian des salles du plateau technique. Vous sauvegardez donc tous vos fichiers sur un projet Github afin de pouvoir redéployer rapidement vos infrastructures si nous changeons de salles

Vous allez installer Vagrant en suivant la procédure d'installation décrite sur le site de vagrant (`https://developer.hashicorp.com/vagrant/install`). Vérifier aussi que Virtualbox est bien installé sur la machine, il servira d'hyperviseurs pour nos VM. Vous allez monter le cluster de VM suivant :

- une VM manager en debian 12.5, avec le service docker installé (vous écrirez un script permettant d'installer docker en configurant l'accés au repository)
- 2 VM worker en debian 12.5 avec le service docker

Ces 4 VM doivent avoir des adresses IP dans le même sous réseaux 192.168.0.10x. Ensuite vous allez vous connecter à la machine manager en ssh et exécuter la commande d'initialisation de Swarm

```
docker swarm init --advertise-addr=192.168.1.101 # l'adresse IP du manager
```

Cette commande ne fonctionne que si l'utilisateur vagrant fait partir du groupe docker. Elle vous affichera la commande permettant d'ajouter des noeuds à votre cluster, avec un token particulier. Exécutez cette commande sur chacun des noeuds worker de votre cluster.

```
docker swarm join --token #{token fourni} 192.168.1.101:2377 #adresse IP et port de
    gestion du cluster.
```

Pour visualiser les noeuds de votre cluster, il faut, depuis la VM manager, exécuter la commande

```
docker node ls
```

Votre cluster est à présent disponible.

III. DÉPLOYER UN SERVICE SUR UN CLUSTER SWARM

III.1. **déploiement manuel.** pour expérimenter le déploiement, nous allons exécuter 3 containers nginx à partir de la commande **docker run** à partir de la VM manager.

```
docker run -d -p80  --name=nginx1 nginx
docker run -d -p80  --name=nginx2 nginx
docker run -d -p80  --name=nginx3 nginx
```

si nous exécutons ensuite la commande **docker ps**, nous constatons que les 3 instances du service web sont lancées. Si nous supprimons le premier au travers de la commande **docker rm nginx1**, nous pouvons constater qu'à présent seul 2 instances tournent toujours :

III.2. **deploiement sous forme de service orchestré.** Pour déployer un service, sous la forme d'un conteneur, administré par l'orchestrateur swarm, il faut utiliser la commande **docker service create** sur le manager. Les options de cette commande sont :

- le nom du service (–name)
- le nombre de réplicas lancés (–replica)
- la publication de port mappés (–publish published=#port_utilisé target=#port_mappé)
- comment se comporte le service en cas d'erreur (–restart-condition, –restart-delay, ...
- les caractéristiques des noeuds affectés au service –limit-cpu, –limit-memory, ...

Pour plus de détail, vous avez l'ensemble des options possibles sur `https://docs.docker.com/reference/cli/docker/service/create/`

Nous allons à nouveau déployer un service web avec 3 instances.

```
docker service create -- replicas=3  --name nginx-service nginx
```

```
docker service ls
```

permet de voir les services lancés et le nombre d'instances qui tournent.

```
docker service ps nginx-service
```

permet de voir les instances et le node sur laquelle chacune est lancée.

si nous supprimons une instance, elle va être relancée immédiatement. Faite le test. vous verrez que l'instance choisie et supprimée avec docker rm ne sera plus présente mais qu'une nouvelle instance aura été lancée à la place.

III.3. **gestion des services.** lorsqu'un service est lancé, il peut être nécessaire d'adapter les conditions d'usage de ce service afin de gérer la montée en charge, ou mettre à jour notre service.

III.3.1. *Scalabilité.* Comme le service a été lancé en mode réplicas, nous pouvons adapter le nombre d'instance. Pour cela, il suffit de lancer la commande

```
docker service scale nginx-service=4
```

ce qui permettra de monter à 4 le nombre de replicas utilisés.

III.3.2. *inspection d'un service.* Afin de disposer de toutes les informations sur un service, de la même façon que pour les container, les images,... nous pouvons utiliser la commande inspect sous la forme

```
docker service inspect nginx-service
```

III.3.3. *mise à jour.* Si une nouvelle version du container est disponible, vous pouvez transmettre cette mise à jour de façon transparente et sans interruption de service. Pour cela il faut utiliser la commande

```
docker service update -- image #nom de l'image nginx-service
```

III.3.4. *suppression d'un service.* une fois que le service est inutile, nous ne pouvons pas le supprimer avec une commande **docker rm** car nous avons vu que les instances sont relancées automatiquement. Il faut utiliser la commande

```
docker service rm  nginx-service
```