Architecture logicielle, API, services web et interopérabilité

J2 - Découvrir l'orchestration de services web

Partie 0 — C’est quoi l’orchestration

Introduction

Le monde de la conteneurisation a apporté de nombreuses choses dans l’administration système, et le déploiement des services web. Mais une des choses principales que nous apportent les conteneurs (et particulièrement Docker) : c’est la possibilité de découper des architecture logicielle complexes en plusieurs briques, simples à manipuler. Pour répondre à ça Docker propose un outil adapté pour l’orchestration automatique d’instance : **Docker Swarm**.

En pratique, Docker Swarm est assez peu utilisé en production. Deux énormes concurrents existent : Kubernetes et Nomad, qui sont plus complets et plus puissants.

Cependant, Docker Swarm est une alternative simple sans complexité extravagante. Ce qui est parfait pour découvrir les concepts de l’orchestration, et faire tourner des petits lab pour tester vos projets reposant sur des containers.

Docker Swarm

Comme dit précédemment : Docker Swarm est un outil d’orchestration. Avec cet outil, on peut gérer automatiquement nos conteneurs avec des règles favorisant la Haute-disponibilité, et l’évolutivité (Scalability) de vos services. On peut donc imaginer 2 scénarios qui sont entièrement compatibles :

- Votre site a un pic de charge et nécessite plusieurs conteneurs : Docker Swarm gère la réplication et l’équilibrage des charges
- Une machine hébergeant vos Dockers est en panne : Docker Swarm réplique vos conteneurs sur d’autres machines.

Nous allons donc voir comment configurer ça, et faire un état des lieux des fonctionnalités proposées.

Les pré-requis

Voici les éléments dont nous aurons besoin pour ce TP, à récupérer dans les cours précédents :

- Vagrant sur son PC pour démarrer des VM (plusieurs noeuds du cluster Swarm qui hébergeront les conteneurs)
- Docker installé sur votre machine

Consignes de rendu

Faites un rapport simple des différentes commandes utilisées, étape par étape dans un document (word, markdown sur Github, etc.) avec des captures pour montrer que ça marche.
Envoyez à `gael@skalab.fr` avant la fin du TP à 16h.

Partie 1 - Préparer notre application pour Docker

Exercice 1 ⭐⭐

Créez un Dockerfile qui fait tourner Django et qui expose le port TCP/8000 en partant d'une image Python.
Confirmer que tout est ok : http://127.0.0.1:8000 doit afficher votre app watchlist.

Tips :

- préparer le code de l'"application" Django sur votre ordi
- installer Django dans le container en utilisant pip avec un fichier requirements.txt
- injecter l'application dans l'image (COPY)
- Choisissez la bonne commande à lancer (CMD) au démarrage du container

Exercice 2 ⭐⭐

Modifiez votre container pour être conforme aux recommandations de Django pour la production. Vous ne devez plus utiliser le serveur de test Django (python manage.py runserver), cela n'est pas fait pour être utilisé à grande échelle en production. Utilisez WSGI avec Gunicorn (https://docs.djangoproject.com/fr/5.2/howto/deployment/)
Adaptez votre image en ce sens.

Partie 2 - Mettre en place l'infra d'orchestration

Exercice 2 ⭐

Avec Vagrant, créer 3 VM Debian, nommée node1, node2, node3.
Vous devez pouvoir vous y connecter en SSH.
Les VMs doivent être sur le même réseau sans filtrage de ports, auquel vous devez accéder.

Exercice 3 ⭐

Installer Docker sur les 3 VM.
La commande `docker swarm` doit être disponible.

Exercice 4 ⭐

Maintenant que tout est prêt : nous allons fabriquer un cluster Swarm réparti sur ces 3 VM.

La première étape est de définir un Manager, celui-ci sera la tête du cluster, ainsi que le point d'accès vers les différentes machines. Dans notre cas, on va faire très simple, le manager sera `node1`.

Pour lancer le Swarm sur le manager, il suffit d'utiliser la commande `docker swarm init`.

Attention, si votre système possède un nombre de carte réseau supérieur à 1 *(Assez facile sur un serveur)*, il faut donner l'IP d'écoute. Par exemple, si l'IP de l'interface du réseau local (dans lequel les VMs communiquent) est *192.168.0.8*. Donc la commande que je vais lancer est `docker swarm init --advertise-addr 192.168.0.8`

Exercice 5 ⭐⭐

La commande précédente indique comment les autres nœuds doivent rejoindre le cluster.

Ajouter les autre noeuds au cluster, et contrôlez que tout le monde est là avec la commande `docker node ls` depuis le manageur.

Partie 3 - Déployer un service

Exercice 6 ⭐

Si vous êtes adepte de la commande `docker run` et que vous refusez `docker-compose`, sachez une chose : je ne vous aime pas. Comme vous m'êtes sympathique, voici une info qui ne servira pas : l'équivalent de `docker run` en Swarm est `docker service`.

Mais nous n'allons pas aborder ce sujet ici.

On va plutôt utiliser l'équivalent de `docker-compose`, qui est `docker stack`. Donc avant-tout, voici le fichier .yml

```
version: "3"
services:
  viz:
    image: dockersamples/visualizer
    volumes:
       - "/var/run/docker.sock:/var/run/docker.sock"
    ports:
       - "8080:8080"
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.role == manager
```

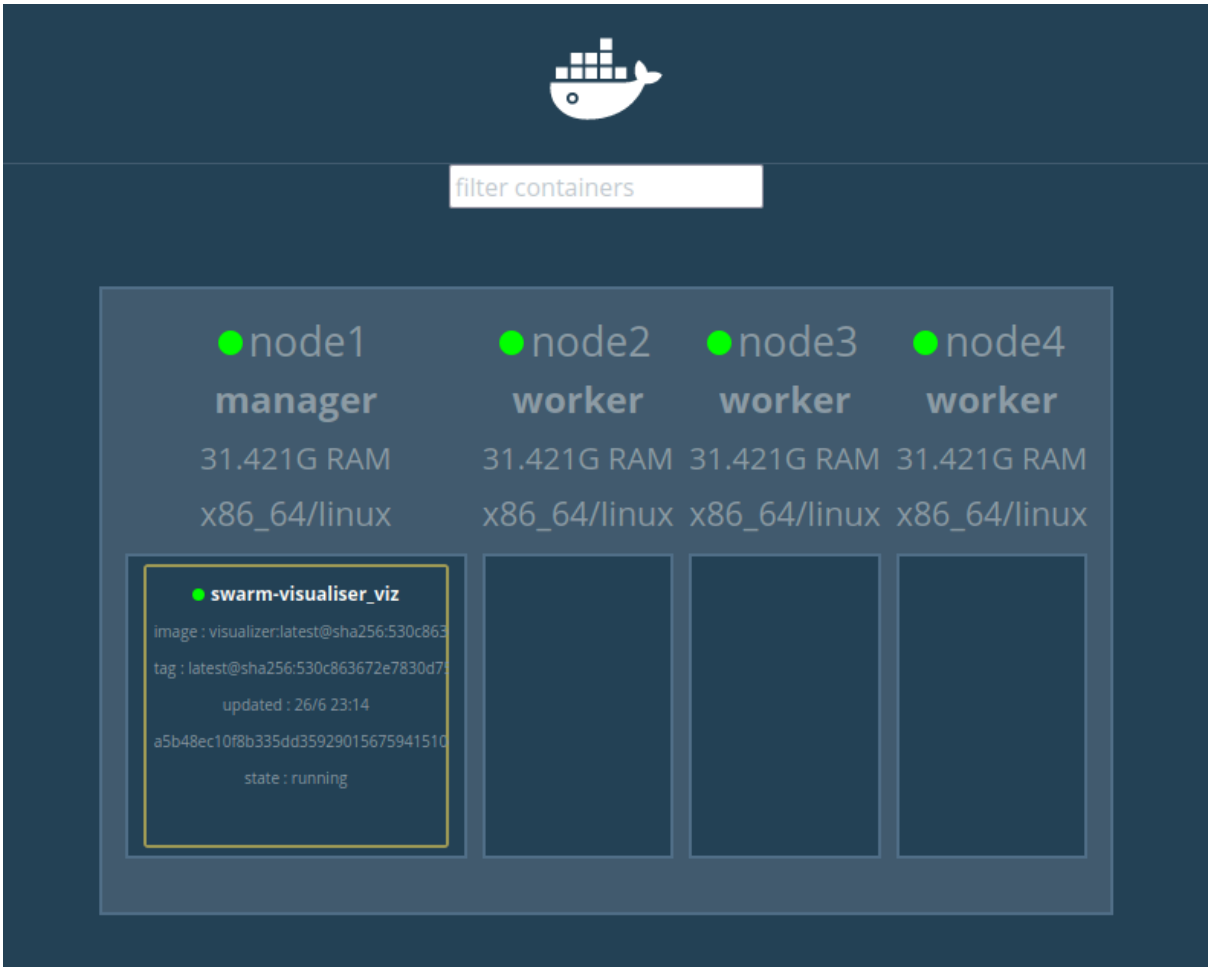
Avant de le démarrer, vous remarquerez sûrement la partie `deploy` qui permet de donner des indications à Swarm. On peut donc rajouter des contraintes pour déployer ça sur le/les managers, demander à l'hôte de limiter l'utilisation des ressources, ou gérer des répliques pour l'équilibrage des charges.

Ce premier conteneur servira à avoir un dashboard simple pour voir où se positionnent les Dashboard, et éviter de passer uniquement en CLI pour cette fonction.

On va donc déployer ce compose avec la commande suivante:

```
docker stack deploy --compose-file docker-compose.yml
swarm-visualiser
```

Une fois la commande terminée, il suffit d'ouvrir le serveur web du manager au port 8080.



Pas mal non ? C'est français.

Exercice 7 ⭐

Reprenez votre conteneur de l'app watchlist. Créez un service watchlist docker-compose simple qui expose votre application sur le port 80 du cluster.

Ajouter l'instruction suivante :

```
deploy:
    replicas: 4
```

A votre avis, ça va faire quoi ?

Lancez le service dans le cluster et visualisez le résultat.

Exercice 8 ⭐

Trouver une commande pour modifier le nombre de replica (on appelle ca “scale up” ou “scale down”) depuis le manager, sans toucher au docker-compose.yaml.

Passez à 2 replicas, observez le résultat dans l’interface de visualisation.

Passez à 20 replicas, observez le résultat dans l’interface de visualisation.

Exercice 9 ⭐⭐

Nous avons un léger problème…
Chaque replicas dispose de sa propre base de données. C’est un fichier local SQLite dans le container, qui n’est donc pas partagé avec les autres containers du cluster.

Modifier votre service watchlist :

- Votre container watchlist ne doit plus utiliser une base de données SQLite, mais utiliser une base de donnée PostgreSQL
- Ajouter un service PostgreSQL à votre docker-compose.yml
- Les fichiers de BDD de votre service PostgreSQL doivent être stocké dans un volume docker dédié
- Mettre au moins 2 replicas de votre service PostgreSQL

Tips : Pensez à gérer les migrations Django (l’écriture de schéma SQL dans base de donnée) au lancement de l’application..

Partie 4 - Gérer la haute-disponibilité

Exercice 10 ⭐

J’ai axé ce TP dans les fonctions de Swarm, et comment les utiliser. Et si je n’ai abordé la haute-disponibilité en priorité, c’est parce que chaque conteneur créé dans Swarm est déjà géré en HA ! Voyons voir.

Notez l’ID d’un conteneur watchlist et trouvez une commande pour arrêter ce conteneur.
Observez ce qu’il se passe dans l’interface (spoiler, il doit redémarrer tout seul).

Exercice 11 ⭐

La haute-disponibilité concerne aussi la perte d’un noeud lui-même (serveur qui prend feu, etc.).

Éteignez un noeud (la VM complète).
Observez ce qu’il se passe dans l’interface. Avons-nous perdu les conteneurs qui tournaient dessus ?

Conclusion

Docker-Swarm est un point d'entrée vers les clusters d’applications qui sont d’une complexité incroyable sans un outil adapté. Swarm permet facilement de répondre à des besoins particuliers, sans grande compétence technique. Dans un environnement de production, il est conseillé de s’orienter vers Kubernetes ou Nomad qui sont des alternatives bien plus complètes et puissantes.

C’est fini, bravo !