TIW - Cloud computing

Swarm - la suite

Fabien Rico (fabien.rico@univ-lyon1.fr)
Jean Patrick Gelas

Univ. Claude Bernard Lyon 1

séance 2

Pourquoi a-t-on besoin de cluster ?

Pourquoi a-t-on besoin de cluster ?

- Pour augmenter la puissance de calcul :
  - Cluster spark/Hadoop ;
  - Instance *worker* d'un site.
- Pour palier les problèmes
  - sur un logiciel ;
  - sur une machine.
- Pour la maintenance :
  - mise à jour des applications ;
  - mise à jour des systèmes ;
  - remplacement du matériel.

Comme retirer une machine du cluster ?

Lorsqu'il faut arréter une des machine du cluster, que faut-il faire ?

Comme retirer une machine du cluster ?

Lorsqu'il faut arréter une des machine du cluster, que faut-il faire ?
Si le noeud est un simple *worker*

1. Déplacer les services du noeud pour éviter les problèmes
   `docker node update --avaibility (drain|active) NOM`
2. Attendre la stabilisation
3. Rétirer le noeud du cluster

Si le noeud est un *manager*

1. S'assurer d'avoir toujours un autre manager (gestion de la notion de quorum ?)
   `docker node promote NOM_DUN_AUTRE_NOEUD`
2. Sortir le noeud des managers
   `docker node denote NOM`
3. Agir comme pour un worker

Mais que se passe-t-il en cas de panne ?

Gestion des images

Les dockers utilisés sont basés sur des images. Jusqu'à présent vous avez utilisé :

- des images publiques téléchargées automatiquement sur un serveur ;
- des images privées construites sur la machine qui lance le docker.

Mais dans un cluster, on ne maitrise plus le noeud qui hébergera le docker. Il faut donc utiliser *un serveur d'image dans lequel vous pouvez déposer une image construite* :

- serveur d'image distant (par exemple le dépot gitlab) ;
- serveur d'image local installé dans le cluster
  - un docker `registry` sur l'un des noeud ;
  - un *service* `registry` localisé sur un des noeud mais distribué sur tout les noeuds via le cluster.

  La seconde méthode permet de créer le service simplement car il sera considéré comme local et ne demandera pas de protocole sécurisé.

Problématique

On a vu en TP que chaque docker est dans un réseau interne uniquement accessible depuis l'hôte. Comment les faire communiquer ?

- Via les publications de ports (option `-p`), c'est compliqué et limité.
- Les machines hôtes sont dans un réseau qui peut être géré différemment (firewall).
- Les machines hôtes peuvent ne pas être dans le même réseau.

besoin

Pour que tous fonctionne bien, il faut que les dockers soient :

- dans un réseau commun ;
- dans un réseau privé qui ne contient que les docker ;
- avec un serveur de nom identique.

Réseau overlay

Définition (réseau overlay)

Un réseau *overlay* ou *superposé* est un réseau virtuel qui est construit sur un réseau existant. Les noeud connectés à ce réseau peuvent communiquer directement quel que soit les contraintes du réseau sous-jacent.

- Utilisation des vxlan.
- Les trames du réseau virtuel sont encapsulées dans les paquets du réseau réel.
- Les manager du cluster s'occupent de l'acheminement des paquets.

Fonctionnement

Il faut créer un réseau avec le *driver overlay* :

```
docker network create --driver overlay --attachable NOMRESEAU
```

Ensuite on peut ajouter le réseau aux services ou définir des services avec ce réseau :

```
docker service update --network-add NOMRESEAU NOMSERVICE
docker service create --network NOMRESEAU ... NOMIMAGEDUSERVICE
```

Remarque :
Pour que le réseau overlay soit fonctionnel sous docker, il faut que les machines puissent communiquer avec les ports 2377(TCP) pour swarm), 7946 (TCP,UDP) pour communication entre les noeuds et 4789(UDP) pour le trafique.

Le réseau *ingress*

C'est un réseau particulier qui :

- existe par défaut ;
- est un réseau overlay ;
- s'occupe des entrées ;
- contient le répartiteur interne à swarm.

En résumé c'est via ce réseau que les services qui ont plusieurs réplicats sont contactés.

Idée de base

Dupliquer les services permet

- d'augmenter la puissance de calul ;
- de fiabiliser le service ;
- de ne pas dépendre du matériel.

Mais ce n'est pas magique.

- Il y a un problème pour conserver les données locales.
- Il faut gérer les informations de sessions.
- Il faut assurer la cohérence des données.

Cas des données statiques

C'est le cas le plus simple, les données ne changent pas durant l'utilisation.

- Par exemple, les configurations des serveurs, le code, une base de donnée accéssible en lecture.
- Il suffit que chaque hôte obtiennent une copie de ces données :
  - ajouter les données dans les images docker ;
  - télécharger les données automatiquement au démarrage ;
  - générer les données automatiquement.

Données dynamiques

Si les données sont modifiées par le service :

- fichiers téléversés ;
- données finales de la base de données ;
- ...

Jusqu'à présent nous avons utilisé des volumes simples :

- les données importantes du docker sont stockées sur l'hôte ;
- en cas de reconstruction du docker, elles sont automatiquement réutilisées.

Mais si le docker change d'hôte ?

Gestions des données dynamique

Solution :

- Imposer la localisation du docker sur une machine particulière.

```
docker service create ... \
      --constraint DESCRIPTIONDELACONTRAINTE ...
```

- Utiliser des volumes *partagés entre les machines*.
- Envoyer les données dans un serveur externe.

Volume partagés

Pour construitre un volume partagé, il faut :

- Avoir un répertoire commun entre les machines hôtes (via les protocole vu en cours de stockage).
- Utiliser un volume dont le driver est compatible avec le protocole

```
docker service create ... \
    --mount 'type=volume,src=<VOLUME-NAME>,
    dst=<CONTAINER-PATH>,volume-driver=local,
    volume-opt=type=nfs,
    volume-opt=device=<SERVEURNFS>:<REPERTOIREPARTAGE>,
    "volume-opt=o=addr=<SERVEURNFS>,vers=4,
    soft,timeo=180,bg,tcp,rw"'
```

Mais ce n'est pas une solution parfaite :

- les drivers proposés utilisent des protocoles peu adaptés.
- cela ne permet pas l'édition concurente (donc les serveurs actif/actif).

Cas des sessions

Pour le bon fonctionnement des services, il faut tenir comptes des données en mémoire par exemple les sessions web. Comment fait-on ?

- On impose à un client de toujours contacter le même réplicat (load balancing selon ip source par exemple)
- On utilise un répartiteur intelligent qui *choisi* le réplicat pour toute la durée d'une session.
- On partage les informations via des services spécialisés (Redis).
- On limite au maximum ces besoins via des actions idempotentes et sans états.
- ...

Cas complexe

Mais si ce qu'on veut dupliquer est le serveur de données lui même ?

- Serveur de BDD répliqué en actif-actif
- Serveur de fichiers répliqué
- ...

On doit alors utiliser des services qui gèrent eux même les problématiques.