Conteneurisation et Orchestration

Partie 6 : Introduction aux clusters

Des Conteneurs à la Production

Jusqu'à présent, nous avons exécuté nos conteneurs sur **une seule machine**. C'est parfait pour le développement, mais pour la production, c'est risqué.

Que se passe-t-il si notre machine tombe en panne ?

- L'application devient indisponible.

Que se passe-t-il si le trafic augmente soudainement ?

- L'application devient lente ou plante.

La solution : ne pas dépendre d'une seule machine. Il nous faut un **cluster**.

C'est quoi, un Cluster ?

Un **cluster** est un groupe de serveurs (appelés **nœuds**) qui travaillent ensemble et sont vus de l'extérieur comme un seul système puissant.

- **Mise en commun des ressources** : La puissance de calcul (CPU), la mémoire (RAM) et le stockage de tous les nœuds sont combinés.
- **Gestion centralisée** : Un "chef d'orchestre" (l'orchestrateur) gère le cluster, répartit le travail et surveille la santé des nœuds.

Pourquoi un Cluster est Essentiel en Production

1. **Haute Disponibilité (High Availability, HA)**
    - Si un nœud tombe en panne, l'orchestrateur déplace automatiquement les conteneurs sur un autre nœud sain.
    - **Résultat : Pas d'interruption de service.**
2. **Scalabilité (Scalability)**
    - **Verticale** : Augmenter les ressources d'un nœud (CPU/RAM). Limité.
    - **Horizontale** : Ajouter de nouveaux nœuds au cluster pour augmenter la capacité totale.
    - **Résultat : L'application peut gérer les pics de charge.**

Pourquoi un Cluster est Essentiel en Production

3. **Équilibrage de Charge (Load Balancing)**
    - Le trafic entrant est réparti intelligemment entre les différents conteneurs de l'application.
    - **Résultat : Aucune instance de l'application n'est surchargée.**

Docker Swarm

Docker Swarm est l'outil d'orchestration **natif** de Docker. Il étend les capacités de Docker à un **cluster de machines**.

- **Simplicité** : Si vous connaissez Docker, vous connaissez déjà 80% de Swarm.
- **Scalabilité** : Permet de répartir la charge de vos conteneurs sur plusieurs serveurs.
- **Haute Disponibilité** : Gère la répartition des services et peut redémarrer automatiquement les conteneurs défaillants.
- **Équilibrage de Charge (Load Balancing)** : Répartit le trafic entrant entre les conteneurs d'un même service.

Architecture de Docker Swarm

Un Swarm est un cluster de nœuds Docker.

- **Nœuds Managers** :
  - Dirigent le cluster.
  - Prennent les décisions d'orchestration (où lancer les conteneurs).
  - Maintiennent l'état du cluster.
  - Il est recommandé d'avoir un nombre impair de managers (ex: 3 ou 5) pour la tolérance aux pannes.
- **Nœuds Workers** :
  - Exécutent les conteneurs (les "tâches").
  - Reçoivent leurs instructions des managers.

Initialisation d'un Cluster Swarm

- **Initialiser le premier manager** :

  ```
  docker swarm init --advertise-addr <IP_MANAGER>
  ```

  Cette commande génère un **token** pour joindre d'autres nœuds.

- **Joindre des workers au cluster**
  - Exécutez la commande `docker swarm join` (fournie par la commande `init`) sur chaque machine que vous souhaitez ajouter comme worker.
- **Joindre d'autres managers** (optionnel) :
  - `docker swarm join-token manager` (pour obtenir le token manager)
  - Exécutez la commande `join` sur les futurs managers.

Initialisation d'un Cluster Swarm

Ici un exemple sortie de la commande `docker swarm init` :

```
docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (dxn1zf6l61qsb1josjja83ngz) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \
    192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Déployer avec Docker Swarm

Pour déployer sur Swarm, on utilise un fichier `stack.yaml` (ou `compose.yml`). La syntaxe est très similaire, mais avec des ajouts spécifiques à Swarm.

La principale différence est la clé `deploy`, qui permet de définir :

- Le **nombre de réplicas** (instances) d'un service.
- Les **politiques de mise à jour** (comment déployer une nouvelle version).
- Les **contraintes de placement** (sur quel type de nœud lancer le service).
- Les **limites de ressources** (CPU, RAM).

Exemple de Fichier `stack.yaml`

```yaml
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    deploy:
      replicas: 3 # Lance 3 conteneurs Nginx
      update_config:
        parallelism: 1 # Met à jour les conteneurs un par un
        delay: 10s # Attend 10s entre chaque mise à jour
      restart_policy:
        condition: on-failure # Redémarre en cas d'erreur
```

Déploiement et Gestion d'une Stack

- **Déployer une stack** :
  `docker stack deploy -c stack.yaml nom_de_la_stack`
- **Lister les stacks** :
  `docker stack ls`
- **Lister les services d'une stack** :
  `docker stack services nom_de_la_stack`
- **Mettre à jour un service** :
  Modifiez l'image ou la configuration dans le `stack.yaml`, puis relancez la commande `deploy`. Swarm s'occupe de la mise à jour progressive.
- **Supprimer une stack** :
  `docker stack rm nom_de_la_stack`

Conclusion sur Docker Swarm

- **Docker Swarm** est une solution d'orchestration **simple et puissante**, intégrée à l'écosystème Docker.
- Il est idéal pour les équipes qui débutent avec l'orchestration ou pour des applications de taille moyenne.
- Il offre les fonctionnalités essentielles : **scalabilité, haute disponibilité et équilibrage de charge**, avec une courbe d'apprentissage très douce.