<!--

Conteneurs Docker

Modularisez et maîtrisez vos applications

<!--

Orchestration et clustering

<!--

Docker Machine

C'est l'outil de gestion d'hôtes Docker.

- Il est capable de créer des serveurs Docker "à la volée"
  - chez différents fournisseurs de cloud
  - en local avec VirtualBox pour tester son application en mode production et/ou distribué.
- L'intérêt de `docker-machine` est d'être parfaitement intégré dans l'outil de CLI Docker.
- Il sert également de base pour créer un Swarm Docker et distribuer les conteneurs entre plusieurs hôtes.

<!--

Docker Machine

- Concrètement, `docker-machine` permet de créer automatiquement des machines avec le Docker Engine et ssh configuré et de gérer les certificats TLS pour se connecter à l'API Docker des différents serveurs.
- Il permet également de changer le contexte de la ligne de commande Docker pour basculer sur l'un ou l'autre serveur avec les variables d'environnement adéquates.
- Il permet également de se connecter à une machine en ssh en une simple commande.

<!--

Docker Machine

Exemple :

```
docker-machine create  --driver digitalocean \
    --digitalocean-ssh-key-fingerprint 41:d9:ad:ba:e0:32:73:58:4f:09:28:15:f
    --digitalocean-access-token "a94008870c9745febbb2bb84b01d16b6bf837b4e0ce
    hote-digitalocean
```

Pour basculer `eval $(docker env hote-digitalocean);`

- `docker run -d nginx:latest` créé ensuite un conteneur sur le droplet digitalocean précédemment créé.
- `docker ps -a` affiche le conteneur en train de tourner à distance.

<!--

Orchestration

- Un des intérêts principaux de Docker et des conteneurs en général est de :
  - favoriser la modularité et les architectures microservice.
  - permettre la scalabilité (mise à l'échelle) des applications en multipliant les conteneurs.
- A partir d'une certaine échelle, il n'est plus question de gérer les serveurs et leurs conteneurs à la main.

<!--

L'orchestration

L'orchestration consiste à automatiser la création et la répartition des conteneurs à travers un cluster de serveurs. Cela peut permettre de :

- déployer de nouvelles versions d'une application progressivement.
- faire grandir la quantité d'instances de chaque application facilement.
- voire dans le cas de l'auto-scaling de faire grossir l'application automatiquement en fonction de la demande.

<!--

Qu'est-ce que Docker Swarm ?

- Swarm est l'outil de clustering et d'orchestration natif de Docker (développé par Docker Inc.).
- Il s'intègre très bien avec les autres commandes docker (on a même pas l'impression de faire du clustering).
- Il permet de gérer de très grosses productions Docker.
- Swarm utilise l'API standard du Docker Engine (sur le port 2376) et sa propre API de management Swarm (sur le port 2377).
- Il a perdu un peu en popularité face à Kubernetes mais c'est très relatif (voir comparaison plus loin).

<!--

Architecture de Docker Swarm

Nœuds Manager et Worker pour une architecture "master/worker" :

- Un ensemble de nœuds de contrôle pour gérer les conteneurs
- Un ensemble de nœuds worker pour faire tourner les conteneurs
- Les nœuds Manager et Worker sont en fait identiques (les nœuds managers sont aussi des workers

<!--

Consensus entre managers Swarm

- L'algorithme Raft : http://thesecretlivesofdata.com/raft/
- Pas d'*intelligent balancing* dans Swarm
  - l'algorithme de choix est "spread", c'est-à-dire qu'il répartit au maximum en remplissant tous les nœuds qui répondent aux contraintes données.

<!--

Docker Services et Stacks

- les services : la distribution d'un seul conteneur
- les stacks : la distribution des applications multiconteneurs basées sur docker-compose

<!--

```yaml
version: "3"
services:
  web:
    image: username/repo
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "4000:80"
    networks:
      - webnet
```

<!--

Service discovery

- Par défaut les applications ne sont pas informées du contexte dans lequel elles tournent
- La configuration doit être opérée de l'extérieur de l'application
  - par exemple avec des fichiers de configuration
  - ou des variables d'environnement
- La mise en place d'un système de découverte de services permet de rendre les applications plus autonomes dans leur (auto)configuration.
- Elles vont pouvoir récupérer des informations sur leur contexte (dev ou prod, us ou fr ?)
- Ce type d'automatisation de l'intérieur permet de limiter la complexité du déploiement.

<!--

Service Discovery

- Concrètement un système de découverte de service est un serveur qui est au courant automatiquement :
  - de chaque conteneur lancé
  - du contexte dans lequel il a été lancé.
- Ensuite il suffit aux applications de pouvoir interroger ce serveur pour s'autoconfigurer.
- Utiliser un outil dédié permet d'éviter de s'enfermer.

<!--

Service Discovery - Solutions

- Le DNS du réseau overlay de Docker Swarm avec des stacks permet une forme extrêmement simple et implicite de service discovery. En résumé, votre application microservice docker compose est automatiquement distribuée.
- Avec le protocole de réseau overlay Weave Net il y a aussi un service de DNS accessible à chaque conteneur
- Deux autre solutions populaires mais plus manuelles à mettre en œuvre :
  - Consul (Hashicorp): Assez simple d'installation et fourni avec une sympathique interface web.
  - etcd : A prouvé ses performances aux plus grandes échelle mais un peu plus

<!--

Répartition de charge (load balancing)

- Un load balancer : une sorte d'"aiguillage" de trafic réseau, typiquement HTTP(S) ou TCP.
- Un aiguillage intelligent qui se renseigne sur plusieurs critères avant de choisir la direction.
- Cas d'usage :
  - Éviter la surcharge : les requêtes sont réparties sur différents backends pour éviter de les saturer.

<!--

Répartition de charge (load balancing) (suite)

- Haute disponibilité : on veut que notre service soit toujours disponible, même en cas de panne (partielle) ou de maintenance.
- Donc on va dupliquer chaque partie de notre service et mettre les différentes instances derrière un load balancer.
- Le load balancer va vérifier pour chaque backend s'il est disponible (healthcheck) avant de rediriger le trafic.
    - Répartition géographique : en fonction de la provenance des requêtes on va rediriger vers un datacenter adapté (+ ou - proche)

<!--

Le loadbalancing de Swarm est automatique

- Loadbalancer intégré : Ingress
- Permet de router automatiquement le trafic d'un service vers les nœuds qui l'hébergent et sont disponibles.
- Pour héberger une production il suffit de rajouter un loadbalancer externe qui pointe vers un certain nombre de nœuds du cluster et le trafic sera routé automatiquement à partir de l'un des nœuds.

<!--

Solutions de loadbalancing externe

- HAProxy : Le plus répandu en loadbalancing
- Træfik : Simple à configurer
- NGINX : Serveur web générique mais a depuis quelques années des fonctions puissantes de loadbalancing et TCP forwarding.

<!--

Gérer les données sensibles dans Swarm avec les secrets Docker

- `echo "This is a secret" | docker secret create my_secret_data`
- `docker service create --name monservice --secret my_secret_data redis:alpine` => monte le contenu secret dans `/var/run/my_secret_data`

<!--

Présentation de Kubernetes

- Une autre solution très à la mode depuis 4 ans. Un buzz word du DevOps en France :)
- Une solution robuste, structurante et open source d'orchestration Docker.
- Au cœur du consortium Cloud Native Computing Foundation très influent dans le monde de l'informatique.
- Hébergeable de façon identique dans le cloud, on-premise ou en mixte.
- Kubernetes a un flat network (un overlay de plus bas niveau que Swarm) : https://neuvector.com/network-security/kubernetes-networking/

<!--

Comparaison Swarm et Kubernetes

- Swarm plus intégré avec la CLI et le workflow docker.
- Swarm est plus fluide, moins structurant mais moins automatique que Kubernetes.
- Swarm groupe les containers entre eux par stack mais c'est un groupement assez léger à l'aide d'un serveur DNS.
- Kubernetes au contraire crée des pods avec une meilleure cohésion qui sont toujours déployés ensembles
  - Kubernetes à une meilleure fault tolerance que Swarm
  - un service Swarm est un seul conteneur répliqué, un service Kubernetes est un groupes de conteneurs (pod) répliqué, plus proche des Docker Stacks.

<!--

Comparaison Swarm et Kubernetes

- Kubernetes a plus d'outils intégrés. Il s'agit plus d'un écosystème qui couvre un large panel de cas d'usage.
- Swarm a un mauvais monitoring et le stockage distribué n'est pas intégré de façon standard.
- Swarm est beaucoup plus simple à mettre en oeuvre et plus rapide à migrer qu'une stack Kubernetes.
- Swarm serait donc mieux pour les clusters moyen et Kubernetes pour les très gros

<!--