DOCKER SWARM

INTRODUCTION À DOCKER SWARM

- Docker Swarm est l'outil d'orchestration natif de Docker qui permet de gérer un ensemble de nœuds Docker comme un cluster. Il simplifie le déploiement, la mise à l'échelle, et la gestion des services conteneurisés sur plusieurs hôtes. Swarm utilise une approche déclarative pour la gestion des services, où l'état désiré du cluster est spécifié par l'utilisateur.

POURQUOI UTILISER DOCKER SWARM ?

- Haute disponibilité : Évite le point de défaillance unique
- Mise à l'échelle facile : Déployer plusieurs instances de l'application
- Auto-guérison : Remplace automatiquement les conteneurs défaillants
- Gestion simplifiée : Contrôle centralisé via le nœud manager

COMPOSANTS CLÉS DE DOCKER SWARM

- Nœud Manager : Coordonne le cluster
- Nœud Worker : Exécute les conteneurs
- Service : Définit l'application à déployer (plusieurs réplicas possibles)

CRÉATION D'UN CLUSTER SWARM À 2 NŒUDS

- Prérequis:
- Deux serveurs avec Docker Engine installé
- Communication réseau entre les serveurs (résolution DNS ou IP)
- Ports ouverts : 2377 (gestion du cluster), 7946 (découverte des nœuds), 4789 (trafic overlay)

- Étapes:
- 1. Initialiser le Swarm sur le nœud Manager :
- docker swarm init

- 2. Joindre un nœud Worker :
- docker swarm join --token <token> <manager_ip>:2377

- 3. Vérifier les nœuds dans le Swarm :
- docker node ls

- Exemple de sortie :
- ID | HOSTNAME | STATUS | AVAILABILITY | MANAGER STATUS
- abcd1234 | manager1 | Ready | Active | Leader

AJOUT DE NŒUDS AU CLUSTER

- Ajout d'un nouveau nœud Worker :

- 1. Générer le token pour les Workers :
- docker swarm join-token worker

- 2. Joindre le nœud Worker au Swarm :
- docker swarm join --token <worker-token> <manager-ip>:2377

- 3. Vérifier que le nœud est ajouté :
- docker node ls

PROMOTION ET DÉGRADATION DE NŒUDS

- Promotion d'un nœud Worker en Manager :

- Commande :
- docker node promote <worker-name>

- Dégradation d'un nœud Manager en Worker :
- Commande :
- docker node demote <manager-name>

- Ces actions permettent d'ajuster les rôles dans le cluster en fonction des besoins (équilibrage de charge, maintenance, etc.).

DRAINAGE ET SUPPRESSION DE NŒUDS

- Drainage d'un nœud :
- docker node update --availability drain <node-name>
- Le drainage empêche le nœud de recevoir de nouvelles tâches et redistribue celles en cours vers d'autres nœuds.
- Réactivation d'un nœud :
- docker node update --availability active <node-name>
- Suppression d'un nœud du cluster :
- docker node rm <node-name> (après avoir quitté le Swarm : docker swarm leave).

GESTION DES MANAGERS DANS UN CLUSTER SWARM

- Dans Docker Swarm, un Manager est responsable de la gestion de l'état du cluster.
- Un seul Manager est désigné comme Leader pour prendre les décisions finales.

- Docker Swarm utilise l'algorithme de consensus Raft pour assurer la cohérence et l'élection des Managers.
- Le quorum nécessaire est calculé comme (n/2) + 1, où n est le nombre de Managers.
- quorum = (nombre de Managers / 2) + 1

- Pour un cluster robuste, il est recommandé d'avoir un nombre impair de Managers et de les distribuer sur plusieurs zones géographiques.

GESTION DES MANAGERS DANS UN CLUSTER SWARM

- Tolérance aux Pannes
- Le cluster peut tolérer la perte de jusqu'à (n - 1) / 2 nœuds Managers.

- Exemple :
- Avec 3 Managers : Tolérance à 1 panne.
- Avec 5 Managers : Tolérance à 2 pannes.

CRÉATION ET GESTION DES SERVICES SWARM

- Un service est un ensemble de conteneurs répliqués ou distribués à travers le cluster.

- Création d'un service avec Nginx (3 réplicas) :
- docker service create --name my-web --replicas 3 -p 80:80 nginx

- Vérification des services :
- docker service ls
- docker service ps <service-name>

- Mise à jour d'un service (changer l'image) :
- docker service update --image nginx:latest my-web

- Rollback d'un service :
- docker service rollback my-web

TYPES DE SERVICES ET PLACEMENT

- Docker Swarm propose deux types de services :
- **Répliqué : Le nombre de réplicas est défini par l'utilisateur.**
- **Global : Une instance par nœud.**

- Création d'un service global :
- docker service create --name my-agent --mode global monitoring-agent

- Pour contrôler le placement des services, utilisez les labels :
- docker node update --label-add type=cpu-intensive <node-name>
- Utilisez des contraintes pour assigner des services à des nœuds spécifiques.
  - docker node update --label-add type=database worker1
  - docker service create --name db-service --constraint 'node.labels.type == database' mysql

DOCKER CONFIG POUR GÉRER LES FICHIERS DE CONFIGURATION

Docker Config permet de gérer et partager des fichiers de configuration sur tous les nœuds du cluster.

- Création d'une configuration :
- docker config create my_nginx_conf /path/to/nginx.conf

- Utilisation dans un service :
- docker service create --name my-nginx --config src=my_nginx_conf,target=/etc/nginx/nginx.conf nginx

Rotation des Configs

- Créer la Nouvelle Config
  - docker config create my_nginx_conf_v2 /path/to/new_nginx.conf
- Mettre à Jour le Service
  - docker service update --config-rm my_nginx_conf --config-add src=my_nginx_conf_v2,target=/etc/nginx/nginx.conf my-nginx

DOCKER STACK POUR LE DÉPLOIEMENT D'APPLICATIONS

- Docker Stack permet de déployer une application composée de plusieurs services via un fichier docker-compose.yml (version 3+).

- Déployer une stack :
- docker stack deploy -c docker-compose.yml my-stack

- Exemple de fichier docker-compose.yml :

```yaml
version: '3.8'
services:
  web:
    image: nginx
    ports:
      "80:80"
    deploy:
      replicas: 5
      placement:
        constraints:
          node.role == worker
```

DIFFÉRENCES ENTRE DOCKER COMPOSE ET DOCKER STACK

- Docker Stack utilise un fichier docker-compose.yml similaire à Docker Compose, il ajoute des options spécifiques à Swarm comme :

- **Réplicas** : nombre d'instances d'un service à déployer sur le cluster.
- **Placement** : contraintes pour définir sur quels nœuds un service doit s'exécuter.
- **Mises à jour progressives (Rolling updates)** : définir comment les mises à jour sont appliquées à un service (parallélisme, délai, rollback).
- **Ressources** : limitation des ressources CPU et mémoire pour chaque service.
- **Tolérance aux pannes (Fault tolerance)** : gestion de la résilience des services.

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

- replicas : Nombre d'instances à exécuter pour le service.

replicas: 5

- 5 réplicas pour assurer une haute disponibilité.

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

update_config : Configurer les mises à jour progressives (rolling updates).

- parallelism : Nombre de conteneurs à mettre à jour simultanément.
- delay : Temps d'attente entre chaque mise à jour.
- failure_action : Action à prendre en cas d'échec de mise à jour (rollback, pause, continue).
- monitor : Temps d'attente avant de considérer la mise à jour comme réussie ou échouée.
- max_failure_ratio : Ratio d'échec maximal avant de stopper la mise à jour.
  - Calcul du ratio : si vous avez un service avec **5 réplicas** et que vous définissez **max_failure_ratio à 0.2**, Docker Swarm permettra jusqu'à **1 échec (20 % de 5)** pendant la mise à jour. Si **plus de 1 échec se produit**, Docker **considérera la mise à jour comme échouée** et n'appliquera pas les modifications.
- Exemple :

```
deploy:
  replicas: 3
  update_config:
    parallelism: 2
    delay: 10s
    monitor: 10s
    failure_action: rollback
    max_failure_ratio: 0.3
```

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

restart_policy : Définir la politique de redémarrage.

- condition : Conditions de redémarrage (on-failure, always, none).
- delay : Délai avant de redémarrer un conteneur échoué.
- max_attempts : Nombre maximum de tentatives de redémarrage.
- window : Spécifie la durée de la **période d'observation**. Pendant cette période, **Docker enregistre les échecs et prend des décisions sur le redémarrage** des réplicas en **fonction du nombre d'échecs** qui se produisent dans cette fenêtre.
  - Définition de la fenêtre : Par exemple, si vous définissez window à 10s, Docker observera les échecs des réplicas sur une période de 10 secondes.
  - Limite des redémarrages : Si, pendant cette période, **un service échoue plusieurs fois (par exemple, 3 fois),** Docker peut décider de ne **pas redémarrer immédiatement** le service pour éviter un cycle de redémarrages incessant. Cela peut être utile pour **donner au service le temps de se stabiliser**.

```
restart_policy:
  condition: on-failure
  max_attempts: 3
  window: 30s
```

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

resources : Limitation des ressources CPU et mémoire.

- limits : Limites de CPU et de mémoire pour chaque conteneur.
- reservations : Réservation minimale de CPU et de mémoire.
- Exemple :

```
resources:
  reservations:
    cpus: '0.25'
    memory: 256M
  limits:
    cpus: "1"
    memory: 1024M
```

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

placement : Contraintes pour définir sur quels nœuds un service doit être exécuté.

- constraints : Utiliser des contraintes pour restreindre un service à des nœuds spécifiques (node.role == worker ou node.labels.type == db).
- Nb: Les contraintes sont en ET ou non en OU
- Exemple :

```
constraints:
  - node.role == worker
  - node.hostname != swarm-worker01
```

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

healthcheck : Configurer les vérifications d'état (health check) d'un service.

- test : Commande à exécuter pour vérifier l'état.
- interval : Temps entre chaque vérification.
- timeout : Temps maximum pour considérer la vérification comme échouée.
- retries : Nombre de tentatives avant de considérer un service comme non sain.

```yaml
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:3000/users"]
  interval: 60s
  timeout: 30s
  retries: 3
```

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

preferences avec spread :

- La directive preferences est utilisée pour influencer la manière dont Docker Swarm distribue les tâches (conteneurs) d'un service sur les différents nœuds.
- Avec l'option spread, Docker cherche à répartir les tâches de manière équilibrée en fonction d'un critère particulier, souvent un label associé aux nœuds.

```
deploy:
    constraints:
      - node.role == worker
    preferences:
      - spread: node.labels.zone
```

- spread: node.labels.zone indique que Docker Swarm doit essayer de répartir les conteneurs aussi équitablement que possible entre les nœuds qui ont des labels zone.
- Par exemple, si vous avez trois nœuds étiquetés **zone=us-east, zone=us-west et zone=eu-central**, Swarm tentera de distribuer les conteneurs de manière à ce que chaque zone reçoive un nombre équilibré de tâches.

OPTIONS SPÉCIFIQUES À SWARM DANS DEPLOY

endpoint_mode :

- L'option endpoint_mode spécifie le mode d'équilibrage de charge (load balancing) que Docker Swarm utilisera pour un service particulier.

Il existe deux modes d'équilibrage de charge :

- vip (Virtual IP) : Le mode par défaut. Tous les conteneurs d'un service sont accessibles via une seule adresse IP virtuelle. Docker Swarm utilise une adresse IP virtuelle partagée pour équilibrer la charge entre les réplicas d'un service.
- dnsrr (DNS Round-Robin) : Utilise un équilibrage de charge au niveau DNS en renvoyant une liste d'adresses IP des différents réplicas du service. Ce mode est utile pour des services qui gèrent leur propre équilibrage de charge ou dans des cas d'utilisation spécifiques comme certains systèmes de cache DNS.

```
deploy:
    endpoint_mode: vip

deploy:
    endpoint_mode: dnsrr
```

MISE À JOUR DES SERVICES

- Mise à jour de l'image :
- docker service update --image myimage:latest myapp

- Rollback en cas d'échec :
- docker service rollback myapp

INTRODUCTION À L'ORCHESTRATION

- L'orchestration est un processus qui permet de gérer et de coordonner les tâches et les actions d'un système complexe.
- Dans le contexte de l'informatique**, l'orchestration est utilisée pour gérer et coordonner les conteneurs**, **les machines virtuelles, les clusters et les services dans un environnement distribué.**

- La principale raison d'utiliser l'orchestration est **d'améliorer l'agilité, l'évolutivité et la fiabilité des systèmes**. En utilisant l'orchestration, vous pouvez facilement **gérer des centaines ou des milliers de conteneurs ou de machines virtuelles,** déployer des applications en quelques minutes, et équilibrer automatiquement la charge sur les différents éléments du système.

- Exemples d'utilisation de l'orchestration:
  - Déploiement d'une application en conteneurs sur un cluster de machines
  - Gérer la scalabilité des instances pour un service web
  - Automatisation de la configuration de machines
  - Automatisation de la découverte de service pour une application distribuée
  - Gestion de la disponibilité et la tolérance aux pannes d'un système
  - Il est important de noter que les systèmes d'orchestration ne se limitent pas aux conteneurs, mais peuvent également gérer des machines virtuelles et des services répartis sur différents systèmes.

CONCEPTS DE BASE DE L'ORCHESTRATION

- **Conteneurs** : Un conteneur est une **instance d'une image logicielle**, qui permet de lancer une application ou un service dans un environnement isolé. Les conteneurs partagent le noyau de l'hôte, ce qui les rend plus légers et plus rapides à démarrer que les machines virtuelles.

- **Images** : Une image est un fichier qui **contient tout ce dont une application a besoin pour s'exécuter**, y compris le code source, les bibliothèques, les configurations, etc. Les images sont utilisées pour créer des conteneurs.

- **Déploiements** : Un déploiement **décrit comment une application ou un service doit être déployé sur un cluster**, y compris le nombre d'instances souhaitées, les métadonnées, les stratégies de mise à jour, etc.

- **Services** : Un service est une **abstraction qui permet de diriger le trafic vers un groupe de conteneurs**. Les services peuvent être configurés pour assurer la disponibilité, l'équilibrage de charge, et la tolérance aux pannes.

- **Réseaux** : Les réseaux permettent **aux conteneurs de communiquer entre eux et avec l'extérieur**. Ils peuvent être configurés pour isoler les conteneurs, pour créer des sous-réseaux, pour connecter des conteneurs à des services externes, etc.

- **Volumes** : Les volumes sont des **espaces de stockage qui peuvent être utilisés par les conteneurs**. Ils permettent de stocker les données de manière persistante, même si les conteneurs sont redémarrés ou répliqués.

- Il est important de comprendre que ces concepts de base sont liés les uns aux autres et qu'ils sont utilisés de différentes manières selon le système d'orchestration choisi.

LES SYSTÈMES D'ORCHESTRATION

- **Docker Swarm** : **Docker Swarm est un système d'orchestration intégré à Docker** qui permet de gérer des conteneurs sur **plusieurs hôtes**. Il permet de créer des clusters de conteneurs, de déployer des applications sur des clusters, de gérer les réseaux et les volumes, et de gérer la scalabilité des applications.

- **Mesos** : **Mesos est un système d'orchestration distribué qui gère les ressources des machines d'un cluster de manière centralisée**. Il permet de gérer des conteneurs et des machines virtuelles, de planifier les tâches sur des machines libres, et de gérer la scalabilité et la tolérance aux pannes des applications.

- **Kubernetes** : Kubernetes est un système d'orchestration open-source pour les conteneurs qui permet de gérer des clusters de conteneurs à l'échelle. **Il fournit des fonctionnalités avancées pour la gestion de la scalabilité, de la disponibilité, de la tolérance aux pannes, de la sécurité et de la mise à jour des applications**. Il est également extensible pour intégrer des fonctionnalités supplémentaires, et peut être **utilisé avec des outils tels qu'ETCD ou Prometheus pour améliorer les fonctionnalités de base.**

- Il est important de noter que **chacun de ces systèmes d'orchestration a ses propres avantages et inconvénients** et qu'il est important de choisir celui qui **convient le mieux à vos besoins en fonction de vos exigences en matière de scalabilité, de disponibilité, de sécurité, de coûts, etc**.
- Kubernetes est devenu le système d'orchestration le plus populaire et est utilisé dans de nombreux environnements d'entreprise et Cloud, mais d'autres systèmes comme Docker Swarm ou Mesos peuvent être adaptés pour certaines utilisations ou certaines exigences.

ORCHESTRATION AVEC DOCKER

- **Utilisation de Docker pour orchestrer des conteneurs** : Docker fournit des commandes pour créer et gérer des conteneurs, telles que "docker run", "docker start", "docker stop" pour démarrer, arrêter et supprimer des conteneurs. Il est également possible de créer des conteneurs à partir d'images existantes en utilisant "docker pull" pour récupérer des images sur un dépôt d'images et "docker create" pour créer un conteneur à partir de cette image.

- **Création d'images** : Les images Docker sont créées en utilisant un fichier "Dockerfile" qui décrit les étapes pour créer l'image. Il peut inclure des instructions pour copier les fichiers, installer des paquets, définir des variables d'environnement, etc. Les images peuvent également être créées à partir d'autres images existantes en utilisant les instructions "FROM" dans un Dockerfile.

- **Gestion des déploiements** : Il est possible de gérer les déploiements avec Docker en utilisant les commandes "**docker service create**" pour créer un service, "**docker service update**" pour mettre à jour un service, et "**docker service rm**" pour supprimer un service. Il est également possible de gérer les déploiements en utilisant des outils de **gestion de configuration tels que Ansible ou Terraform qui peuvent automatiser les tâches de déploiement.**

- **Utilisation des réseaux et des volumes** : Les conteneurs peuvent être connectés à des réseaux et des volumes en utilisant les options de la commande "docker run" ou en définissant les options de réseau et de volume dans un fichier "docker-compose.yml" pour gérer les déploiements.

TYPES DE RÉSEAUX DANS SWARM

- Types de Réseaux
- bridge : Réseau par défaut pour les conteneurs sur un même hôte.
- host : Le conteneur partage la pile réseau de l'hôte.
- none : Aucun réseau n'est attaché au conteneur.
- overlay : Réseau distribué sur plusieurs nœuds Swarm.
- Mode Ingress (par défaut) : Le service est accessible via n'importe quel nœud du Swarm.

SERVICE DISCOVERY

- Docker Swarm intègre un service de résolution DNS qui permet aux services de se découvrir mutuellement par leur nom.

- Pour que la découverte fonctionne, les services doivent être sur le même réseau overlay :
- docker network create -d overlay my-app-net
- docker service create --name db --network my-app-net mysql
- docker service create --name web --network my-app-net nginx

RÉSEAUX OVERLAY DANS DOCKER SWARM

- Les réseaux overlay permettent aux conteneurs de différents nœuds de communiquer entre eux.

- Création d'un réseau overlay personnalisé :
- docker network create -d overlay my-overlay-net

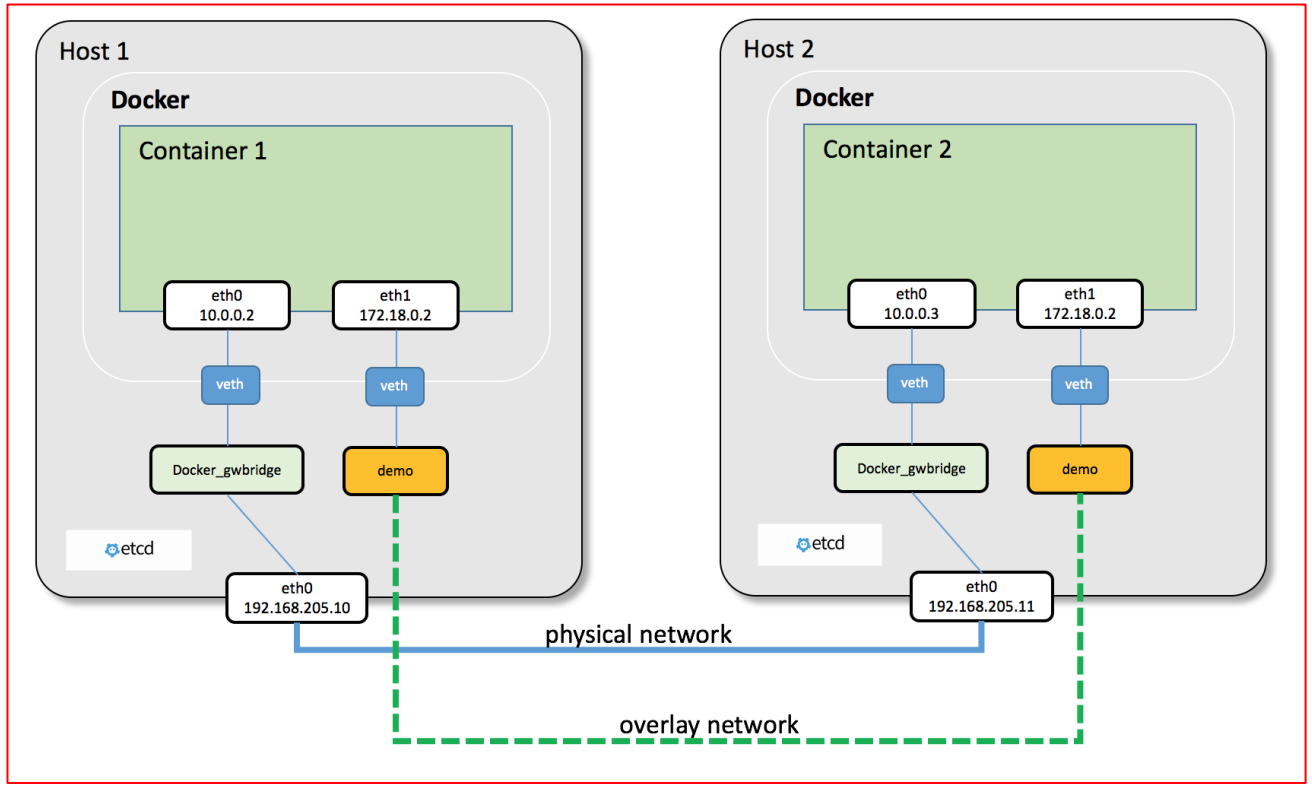
- Attacher un service à un réseau overlay :
- docker service create --name my-service --network my-overlay-net nginx

RÉSEAUX OVERLAY DANS DOCKER SWARM

Publier des Ports avec Différents Modes

- Mode Ingress (par défaut) : Le service est accessible via n'importe quel nœud du Swarm.
- docker service create --name my-service -p 80:80 nginx
- Mode Host : Le service est accessible uniquement via le nœud où il s'exécute.
- docker service create --name my-service --network host -p 80:80 nginx

OVERLAY (UTILISÉ DANS L'ORCHESTRATION) #SWARM



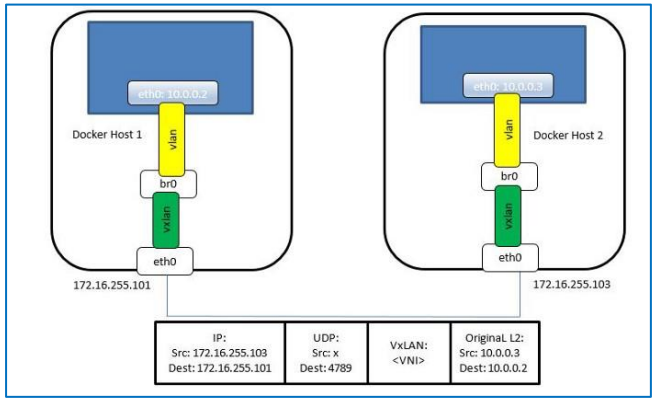

```
(Host) Docker01 : 172.16.255.101
Container1 : eth0: 10.0.0.2

(Host) Docker03 : 172.16.255.103
Container2 : eth0: 10.0.0.3
```

Dans un environnement docker avec 2 hôtes, chaque hôte a 1 conteneur à l'intérieur :

OVERLAY (UTILISÉ DANS L'ORCHESTRATION) #SWARM

- Lorsque vous créez un réseau overlay, Docker crée un espace de noms pour le réseau sur l'hôte.
- Il créera ensuite un périphérique de pont (par exemple br0) et une interface vxlan.
- Lorsque vous créez un conteneur attaché à ce réseau, il sera attaché au bridge.
- Lorsque vous envoyez ensuite du trafic entre les conteneurs sur différents hôtes, le périphérique réseau sur le conteneur l'envoie au périphérique vxlan et au pont br0, jusqu'à l'hôte.
- L'hôte utilise l'en-tête vxlan pour acheminer le paquet vers le nœud de destination.



- La méthode recommandée pour créer un réseau superposé de nos jours se fait en 2 étapes :

1. Créez un réseau swarm entre les nœuds que vous souhaitez mettre en réseau.

2. Créez un overlay sur le dessus et les nodes de swarm se découvriront automatiquement.

Avant de commencer à créer un réseau superposé à l'aide de Swarm, assurez-vous que les ports suivants sont ouverts et accessibles sur tous les nœuds hôtes Docker :

- Port TCP 2377
- Port TCP et UDP 7946
- Port UDP 4789

TP PRATIQUE: CRÉATION D'UN CLUSTER SWARM

- 1. Créez un cluster avec 3 Managers et 3 Workers.
- 2. Déployez une application multi-services avec Docker Stack.
- 3. Configurez des réseaux overlay, utilisez des configs et testez la découverte de services.
- 4. Effectuez une mise à jour d'image et gérez un rollback.

- Prendre le compose et déjà créer le .env avec
- les variables indiqués dans le compose
- transformer ce compose en stakc pour swarm :
- - Ajouter la partie deploy

- BDD: 1 replicas
- Front : 2 replicas
- Redis : 2 replicas

- paralellism : 1
- resources limits
- contraints
- - BDD sur le master
- - Le reste sur les worker

INTRODUCTION À KUBERNETES

Intelligent scheduling

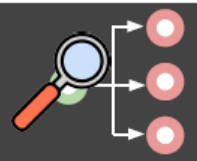
Vous fournissez à Kubernetes un cluster de nœuds qu'il peut utiliser pour exécuter des tâches conteneurisées. Vous indiquez à Kubernetes la quantité de CPU et de mémoire (RAM) que chaque conteneur a besoin. Kubernetes peut adapter les conteneurs sur vos nœuds afin d'utiliser au mieux utilisation de vos ressources.

Self healing

Kubernetes redémarre les conteneurs qui échouent, remplace les conteneurs, tue les conteneurs qui ne répondent pas à votre contrôle de santé défini par l'utilisateur, et ne les annonce pas aux clients jusqu'à ce qu'ils soient prêts à servir.

Horizontal scaling

Kubernetes vous permet de faire évoluer facilement vos applications manuellement avec une simple ligne de commande ou dynamiquement sur la base de métriques standard comme le CPU et la mémoire, ou sur la base de paramètres personnalisés.



Service discovery & Load Balancing

Kubernetes peut exposer un conteneur en utilisant son nom DNS ou en utilisant sa propre adresse IP. Si le trafic vers un conteneur est important, Kubernetes peut équilibrer la charge et distribuer le trafic réseau afin que le déploiement soit stable.

Automated Rollout & Rollback

Vous pouvez décrire l'état souhaité pour vos conteneurs déployés en utilisant Kubernetes, et il peut changer l'état réel vers l'état souhaité à une vitesse contrôlée. Par exemple, vous pouvez automatiser Kubernetes pour créer de nouveaux conteneurs pour votre déploiement, retirer les conteneurs existants et adopter toutes leurs ressources vers le nouveau conteneur.

Secret & Config management

Kubernetes vous permet de stocker et gérer des informations sensibles, comme les mots de passe, les jetons OAuth, et les clés ssh. Vous pouvez déployer et mettre à jour les secrets et la configuration d'application sans avoir à reconstruire vos images de conteneur, et sans exposer les secrets dans votre configuration de la pile.

INTRODUCTION À KUBERNETES

Intelligent scheduling

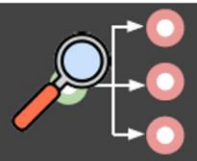
Vous fournissez à Kubernetes un cluster de nœuds qu'il peut utiliser pour exécuter des tâches conteneurisées. Vous indiquez à Kubernetes la quantité de CPU et de mémoire (RAM) que chaque conteneur a besoin. Kubernetes peut adapter les conteneurs sur vos nœuds afin d'utiliser au mieux utilisation de vos ressources.

Self healing

Kubernetes redémarre les conteneurs qui échouent, remplace les conteneurs, tue les conteneurs qui ne répondent pas à votre contrôle de santé défini par l'utilisateur, et ne les annonce pas aux clients jusqu'à ce qu'ils soient prêts à servir.

Horizontal scaling

Kubernetes vous permet de faire évoluer facilement vos applications manuellement avec une simple ligne de commande ou dynamiquement sur la base de métriques standard comme le CPU et la mémoire, ou sur la base de paramètres personnalisés.



Service discovery & Load Balancing

Kubernetes peut exposer un conteneur en utilisant son nom DNS ou en utilisant sa propre adresse IP. Si le trafic vers un conteneur est important, Kubernetes peut équilibrer la charge et distribuer le trafic réseau afin que le déploiement soit stable.

Automated Rollout & Rollback

Vous pouvez décrire l'état souhaité pour vos conteneurs déployés en utilisant Kubernetes, et il peut changer l'état réel vers l'état souhaité à une vitesse contrôlée. Par exemple, vous pouvez automatiser Kubernetes pour créer de nouveaux conteneurs pour votre déploiement, retirer les conteneurs existants et adopter toutes leurs ressources vers le nouveau conteneur.

Secret & Config management

Kubernetes vous permet de stocker et gérer des informations sensibles, comme les mots de passe, les jetons OAuth, et les clés ssh. Vous pouvez déployer et mettre à jour les secrets et la configuration d'application sans avoir à reconstruire vos images de conteneur, et sans exposer les secrets dans votre configuration de la pile.