Linux Professional Institute

DEVOPS TOOLS ENGINEER

Présentation de Docker Avancée

Formation Certifiante DevOps Tools Engineer

DevOps Tools Ingénieur | Linux Professional Institute

docker

Les réseaux sur Docker

Lors de l'installation de Docker, trois réseaux sont créés automatiquement. On peut voir ces réseaux avec la commande **docker network ls**. Un réseau de type bridge est créé

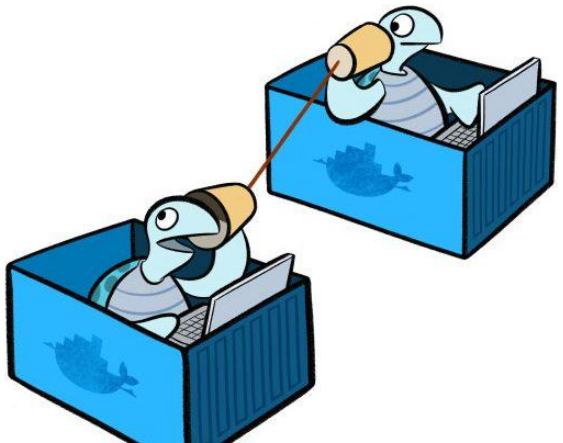
Le réseau Bridge est présent sur tous les hôtes Docker. Lors de la création d'un conteneur, si l'on ne spécifie pas un réseau particulier, les conteneurs sont connectés au Bridge **docker0**

La commande **ifconfig** ou **ip a** fournit les informations sur le réseau ponté (bridge).

La commande **docker network inspect bridge**, retourne les informations concernant ce réseau

```
docker run -itd --name=container1 alpine
docker run -itd --name=container2 alpine
docker network inspect bridge
```

Les conteneurs sont connectés au Bridge par défaut **docker0** et peuvent communiquer entre eux **par adresse IP**, les conteneurs se trouvent alors, **sur le même réseau**.



Réseau par défaut 172.17.0.0 /16

de 172.17.0.1 à 172.17.255.254

65 534

communication entre conteneur

```
docker run -tid --name conteneur1 alpine
docker exec -ti conteneur1 sh
>ip a
```

Machine docker 172.17.0.1/16

Réseaux définis par l'utilisateur

Un réseau Bridgé est le type de réseau le plus utilisé dans Docker.

Il est recommandé d'utiliser des réseaux Bridgés définis par l'utilisateur pour contrôler quels conteneurs peuvent communiquer entre eux.

Les réseaux Bridgés créés par les utilisateurs sont semblables au réseau bridge par défaut créé à l'installation de Docker Docker0.

```
docker network create --driver bridge blue
docker network create -d bridge --subnet 172.30.0.0/16 mynetwork

docker network inspect blue
docker network ls

docker run --network blue -itd --name test1 alpine
docker network inspect blue
```

Connexion d’un container au réseau blue

```
docker run -itd --name c1 alpine
```

```
docker network connect blue c1
```

link

docker run -tid --name conteneur1 alpine

docker run -tid --name conteneur2 –link conteneur1 alpine

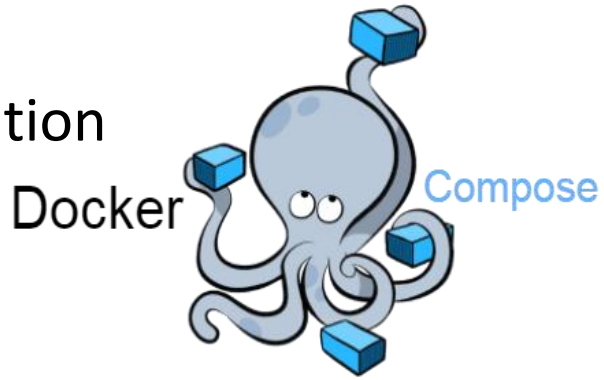
docker exec -ti conteneur2 sh
>cat /etc/hosts

résolution de noms (machine)

Docker Compose est un outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services inter-connectés.

je vais par exemple décrire un ensemble composé de n conteneurs :

un conteneur PostgreSQL
un conteneur pour le code de mon application
un conteneur pour ,,,,,,,,,,,,,,,,,,,
un conteneur pour ,,,,,,,,,,,,,,,,



Je pourrai alors démarrer mon ensemble de conteneurs en une seule commande docker-compose up.

Sans Docker Compose, j'aurais dû lancer 3 commandes docker run avec beaucoup d'arguments pour arriver au même résultat.

Par ailleurs, cela aurait nécessité que je rédige un README plutôt précis pour que les autres membres de mon équipe obtiennent le même résultat.

Avec Docker Compose, cette configuration est faite dans un fichier qui est versionné avec le reste du code de l'application.

Dans le fichier **docker-compose.yml**, chaque conteneur est décrit avec un ensemble de paramètres qui correspondent aux options disponibles lors d'un docker run : l'**image** à utiliser, les **volumes** à monter, **les ports** à ouvrir, etc.

Mais on peut également y décrire des éléments supplémentaires, comme la possibilité de « construire » (docker build) une image à la volée avant d'en lancer le conteneur.

Exemple d'une configuration Docker Compose

https://docs.docker.com/compose/wordpress/

docker-compose up
docker-compose down
docker-compose restart
docker-compose exec
docker-compose logs

docker-compose up :
//démarre les services décrits dans mon docker-compose.yml et ne me rend pas la main.

docker-compose down:
// stoppe les services.

docker-compose restart:
//redémarre l'ensemble des services.
docker-compose restart nginx :
//redémarre un des services (ici nginx).

docker-compose exec rails bash :
//me fournit une console bash au sein du conteneur rails.

docker-compose logs :
//me retourne l'ensemble des logs des services depuis le dernier démarrage et me rend la main.

docker-compose logs -f rails :fait la même chose pour le conteneur rails uniquement.

Docker machine

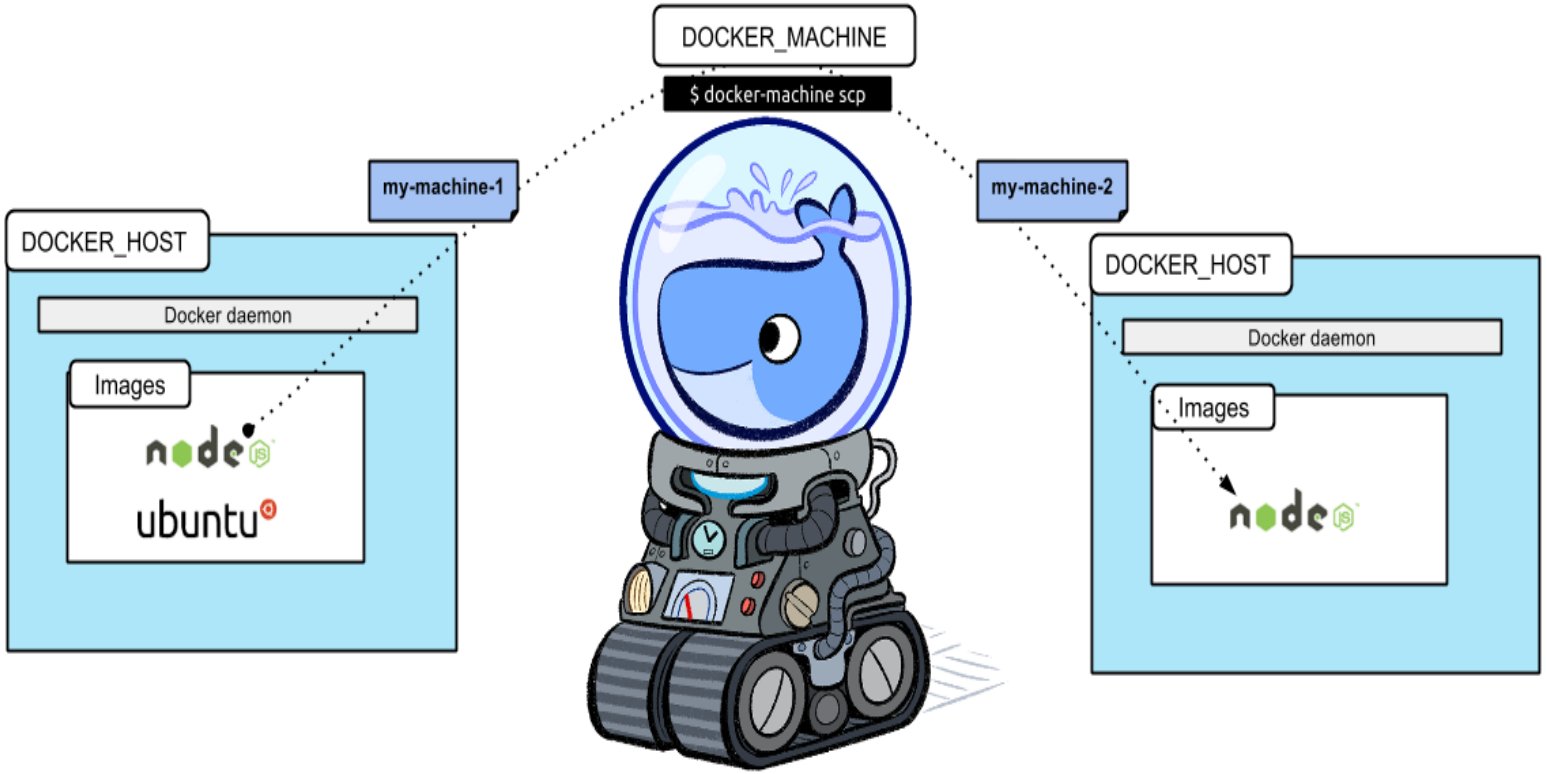
Docker-**machine** est un outil qui permet de provisionner des machines (physiques ou virtuelles) afin d'y installer le nécessaire pour faire fonctionner docker.

Docker Machine permet de provisionner sur **virtualbox**, mais également beaucoup de **service cloud**, comme **Azure**

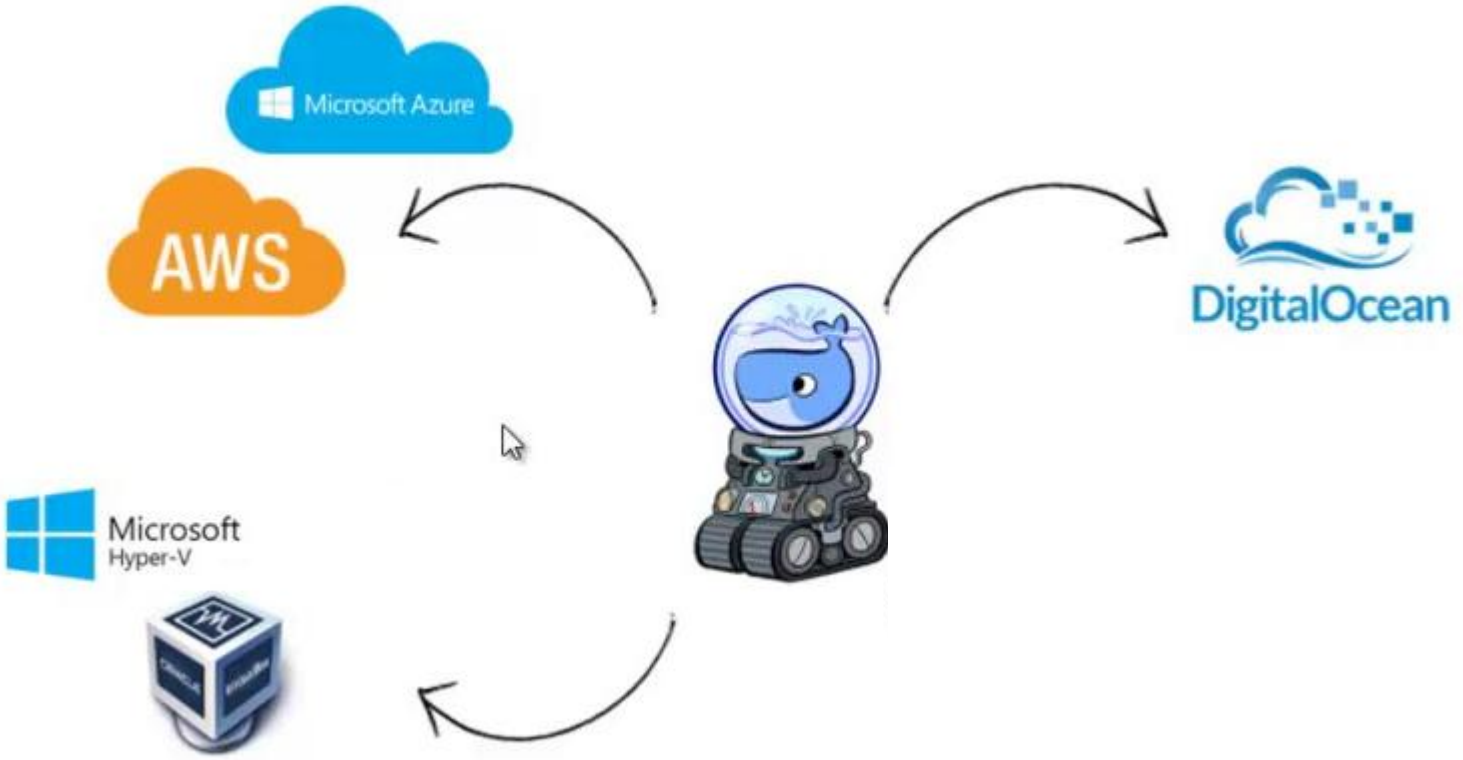
Il installera donc docker, **un jeu de clé pour une connexion ssh**, et ceux sur de nombreuses distribution GNU/Linux (debian, centos, archlinux ...).

→est un outil qui permet d'**installer Docker engine** sur des hôtes virtuels et distants

Docker machine



Drivers docker-machine



Installation du Docker machine

https://docs.docker.com/machine/install-machine/

```
base=https://github.com/docker/machine/releases/download/v0.16.0 &&
 curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
 sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

```
docker-machine version

docker-machine create --driver virtualbox  manager
docker-machine create --driver virtualbox  worker1
docker-machine create --driver virtualbox  worker2
docker-machine create --driver virtualbox  worker3

docker-machine ls
docker-machine start Machine1
docker-machine stop Machine1
docker-machine restart Machine1
docker-machine rm Machine1
docker-machine ssh Machine1
```

```
docker-machine ip manager
```

L'orchestration de conteneur

Dans le développement moderne, les applications ne sont plus **monolithiques**, au contraire elles sont composées de douzaines voire de centaines de composants mis en conteneurs, **peu structurés**, qui **doivent fonctionner ensemble** pour permettre à telle ou telle application de **fonctionner correctement**.

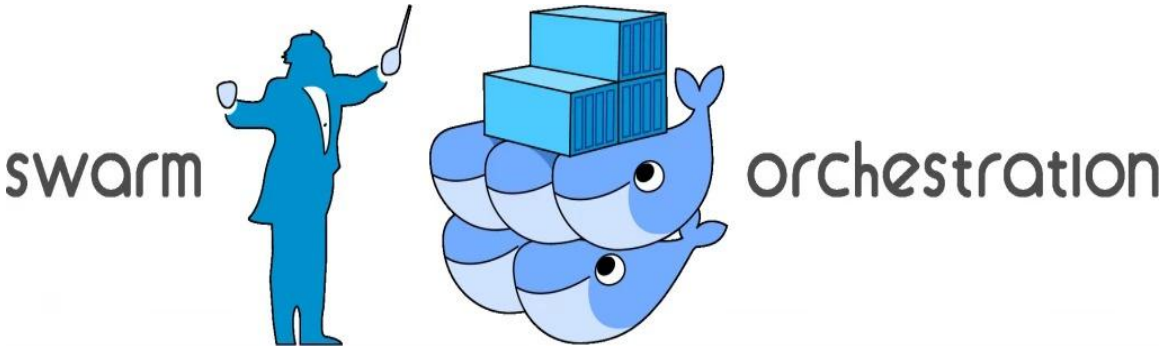
L'orchestration de conteneur désigne **le processus d'organisation du travail des composants individuels et des niveaux d'application**.

les moteurs d'orchestration de conteneurs permettent aux utilisateurs de **contrôler** le moment où les conteneurs commencent et s'arrêtent, les grouper en **clusters** et **coordonner tous les processus** qui composent une application.

Les outils d'orchestration de conteneur permettent aux utilisateurs de **guider le déploiement de conteneur** et d'a**utomatiser les mises à jour,** le **contrôle d'état et les procédures de basculement**.

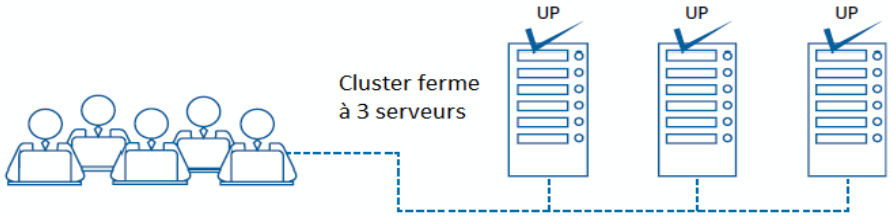
Docker Swarm est un outil qui permet **d'orchestrer le déploiement de conteneurs Docker** au sein d'un cluster. L'idée étant d'exécuter des commandes Docker comme vous le feriez sur un hôte Docker classique, mais de distribuer automatiquement ces commandes sur les différents nœuds du cluster.

L'avantage de Docker Swarm est qu'il est déjà intégré dans Docker par défaut.

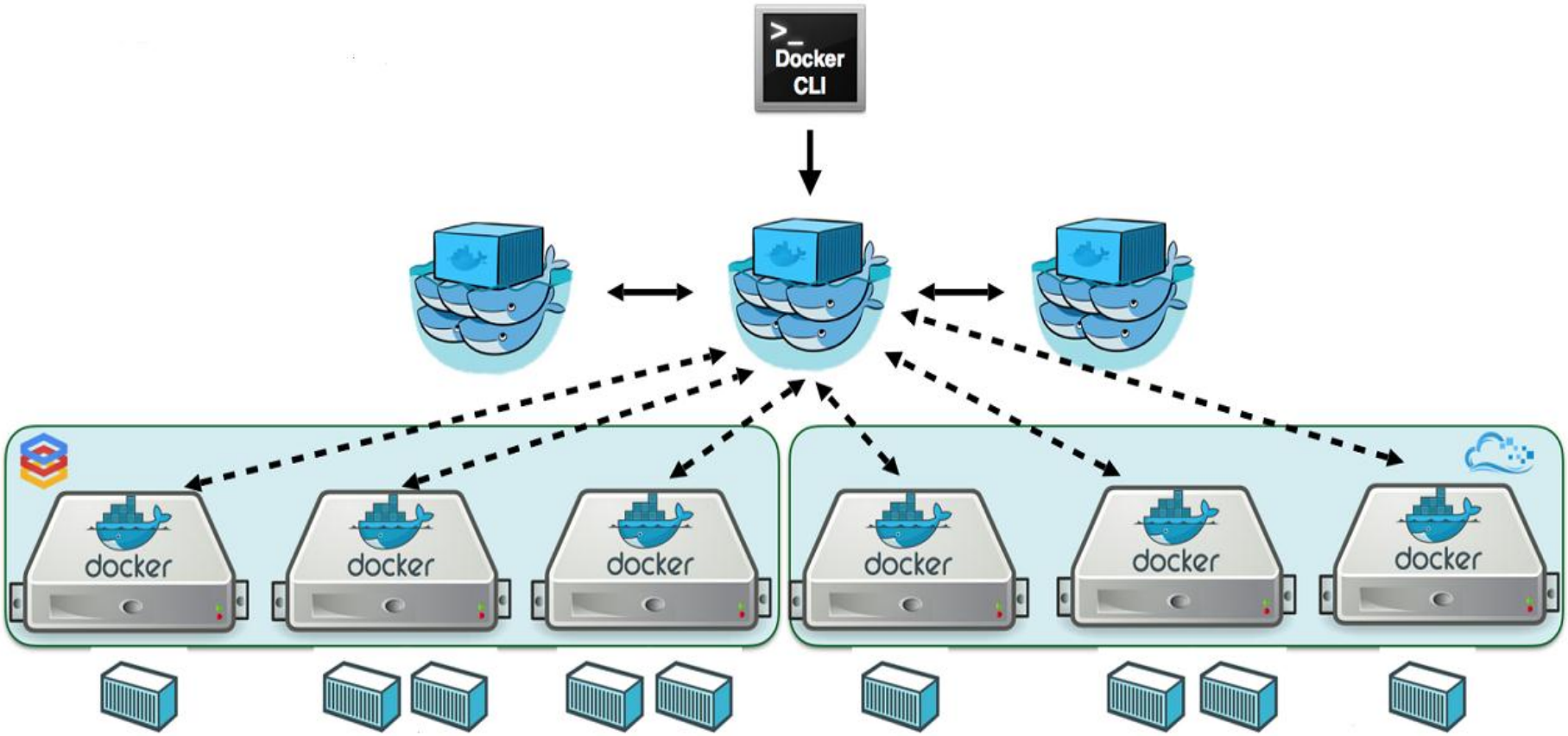


Docker Swarm ou **le monde Swarm** Permet la gestion de cluster pour Docker

Un **cluster informatique**, **cluster de serveurs** ou grappe de serveurs, est un groupe de serveurs indépendants fonctionnant comme un seul et même système



Docker Swarm



Fonctionnalités

Swarm est un service qui a **une vue sur toutes les instances** (serveurs) Docker, et qui est **le point d'entrée universel vers tous ces nœuds**.

Par exemple, c'est Swarm qui se charge d'instancier un conteneur sur le cluster.

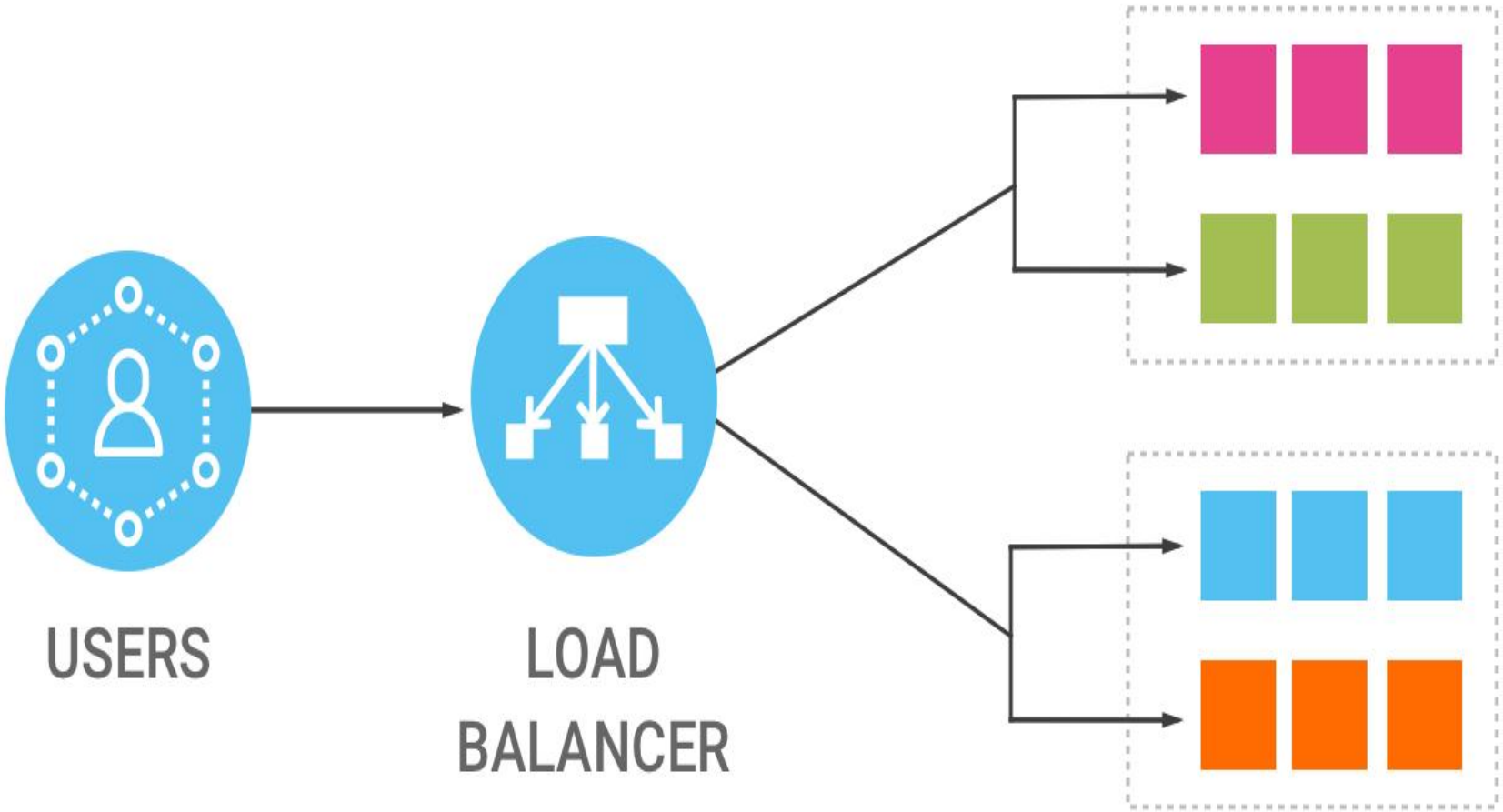
Swarm choisit l'emplacement de ses déploiements en fonction de certaines règles, filtres ou stratégies de déploiement. Par exemple, on peut spécifier les besoins d'un container à un certain niveau de CPU et de RAM.

Swarm effectuera son choix d'hôte Docker de façon à répondre à ces conditions.

- Les bilans de santé sur les conteneurs
- Le lancement d'un ensemble fixe de conteneurs pour une image Docker particulière
- La mise à l'échelle du nombre de conteneurs de haut en bas en fonction du chargement (scale)
- L'exécution de la mise à jour continue du logiciel à travers les conteneurs
- Swarm utilise un **load balancer** embarqué pour gérer la répartition de charge

Load balancer

la répartition de charge



Leader ou manager a pour rôle de gérer l'ensemble de cluster

- Les stacks
- Les services
- Les nœuds
- Les updates

Worker

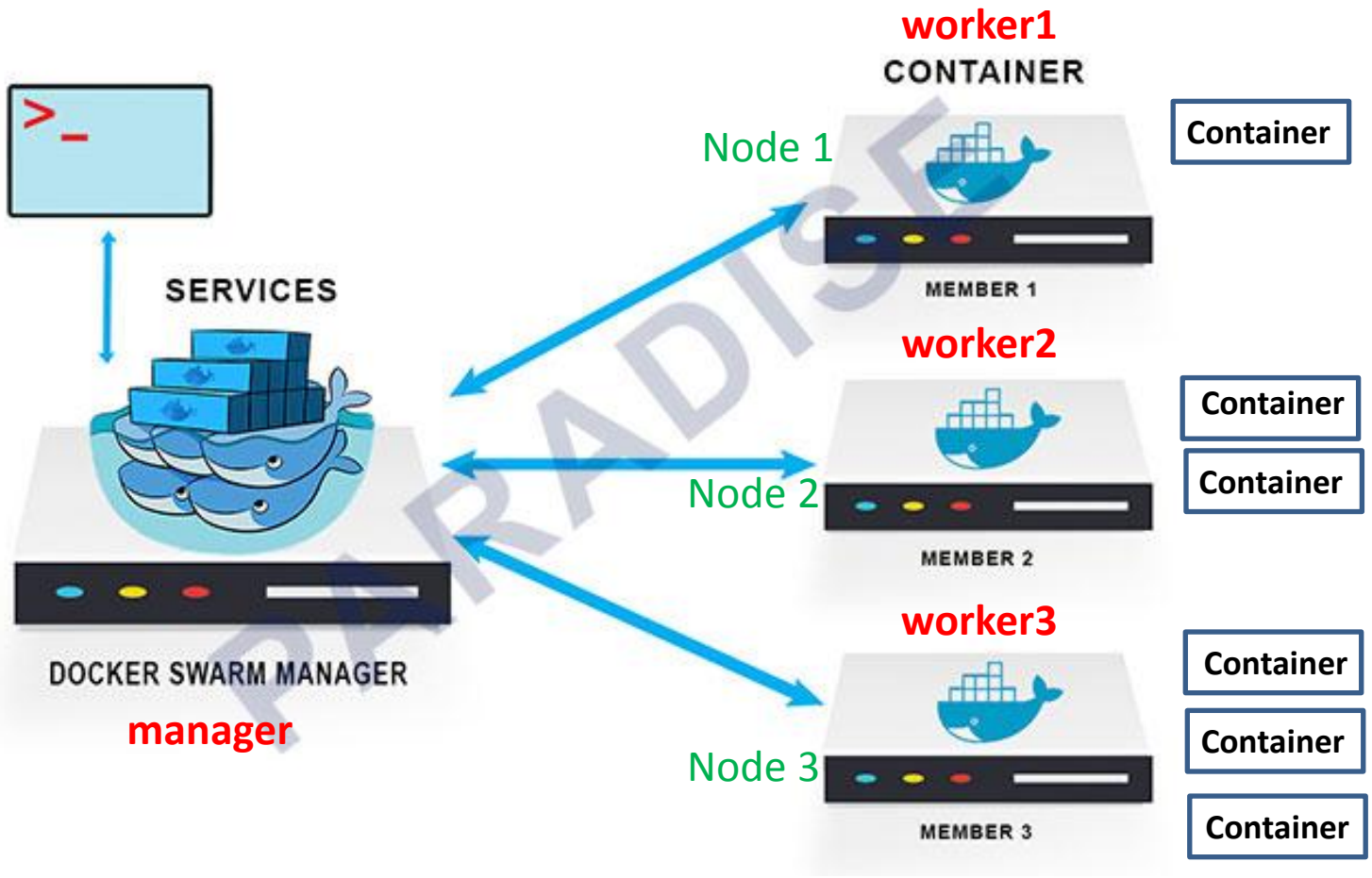
Un nœud **worker** est une instance du moteur Docker participant a Swarm

Le leader distribue les tâche s aux nœuds travailleurs: les workers

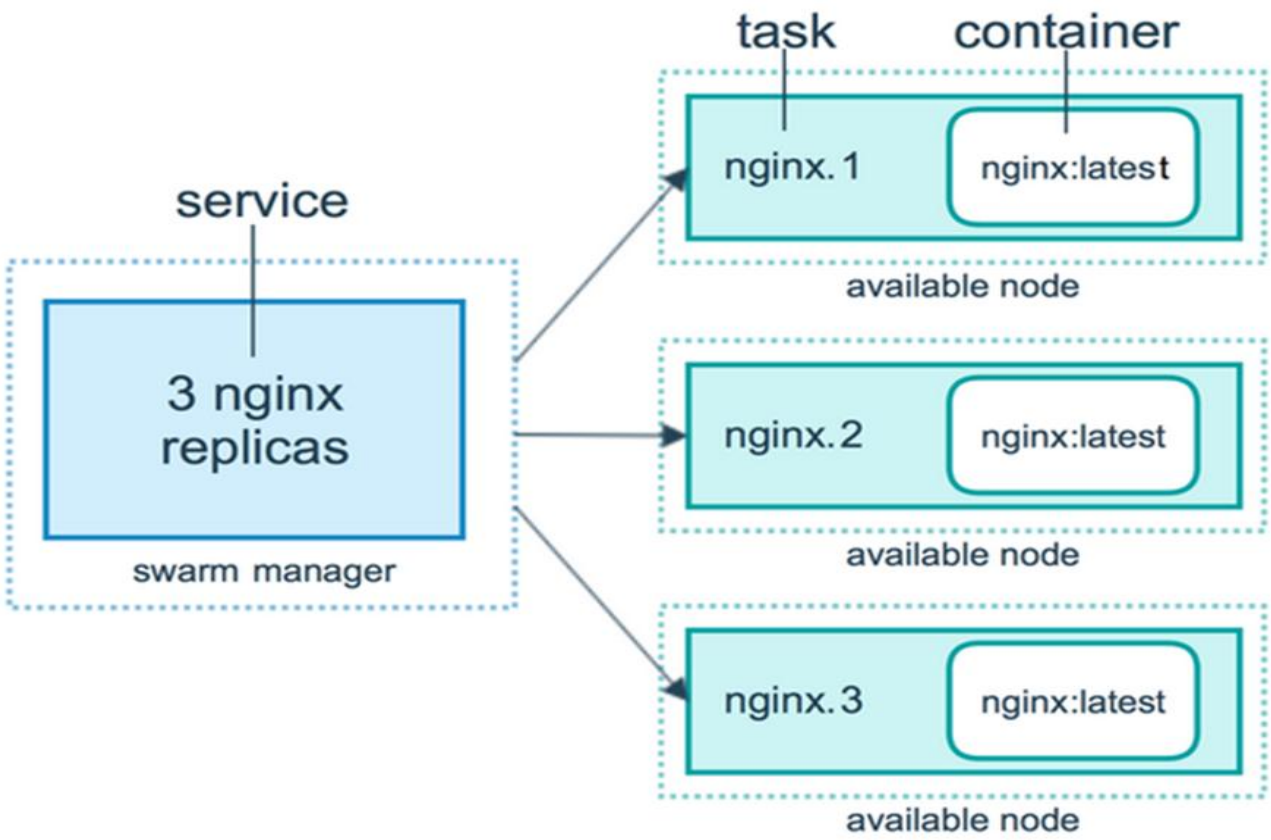
Un agent s'exécute sur chaque nœud de travail et rapporte les tâches qui lui sont affectées

Les Workers notifie au leader, l'état de ses tâches assignées afin que le manager puisse maintenir l'état de chaque worker

Lors qu'un Leader est Down, un nœud du cluster est élu pour jouer ce rôle


worker1
CONTAINER
Node 1
Container
SERVICES
MEMBER 1
worker2
Container
Container
Node 2
MEMBER 2
worker3
Container
Container
Container
Node 3
MEMBER 3
DOCKER SWARM MANAGER
manager

Un service est composé de un ou plusieurs conteneur(s) repliqués



La première chose à faire c'est d'initialiser Swarm. Vous allez faire un SSH dans la machine manager1 et initialiser Swarm là-dedans

.

$docker-machine ip manager

>docker-machine ssh manager

> docker swarm init --advertise-addr **MANAGER_IP**

exp: >| docker swarm init — advertise-addr 192.168.1.20

vous pouvez voir le statut de votre Swarm

- docker node ls

Pour ajouter un travailleur à cet swarm, exécutez la commande suivante :

> docker-machine ssh worker1

> docker swarm join --token SWMTKN-1-2kt9kl8ipzego29d1r9tun8ytdflpv3e7uixfx5i2 b39ul4mak-aynd86csf4c6wikswbbdmr4y0 192.168.99.101:2377

> docker-machine ssh manager
> docker node ls

//Pour connaître la commande de jointure d'un travailleur

> docker swarm join-token worker

Faites la même chose en lançant des sessions SSH pour travailleur2/3/4/5 et en collant la même commande puisque l'on veut que tous soient des nœuds de travail.

docker@manager:~$ docker node ls

docker@manager:~$ docker service create --replicas 5 -p 80:80 --name web nginx

docker@manager:~$ docker service ls

cela montre que les répliques ne sont pas encore prêtes 0,1,2

docker@manager:~$ docker service ps web

docker service ls
docker service ps web

docker ps

Mise à l'échelle et réduction d'échelle

```
$ docker service scale web=8    //web scaled to 8
```

```
docker service ls
docker service ps web
```

Inspecter les nœuds

```
docker node inspect self
docker node inspect worker1
```

```
docker node ls
docker node ps worker1
docker node inspect worker1
```

Supprimer le service

docker service ls
docker service inspect web

docker service rm web

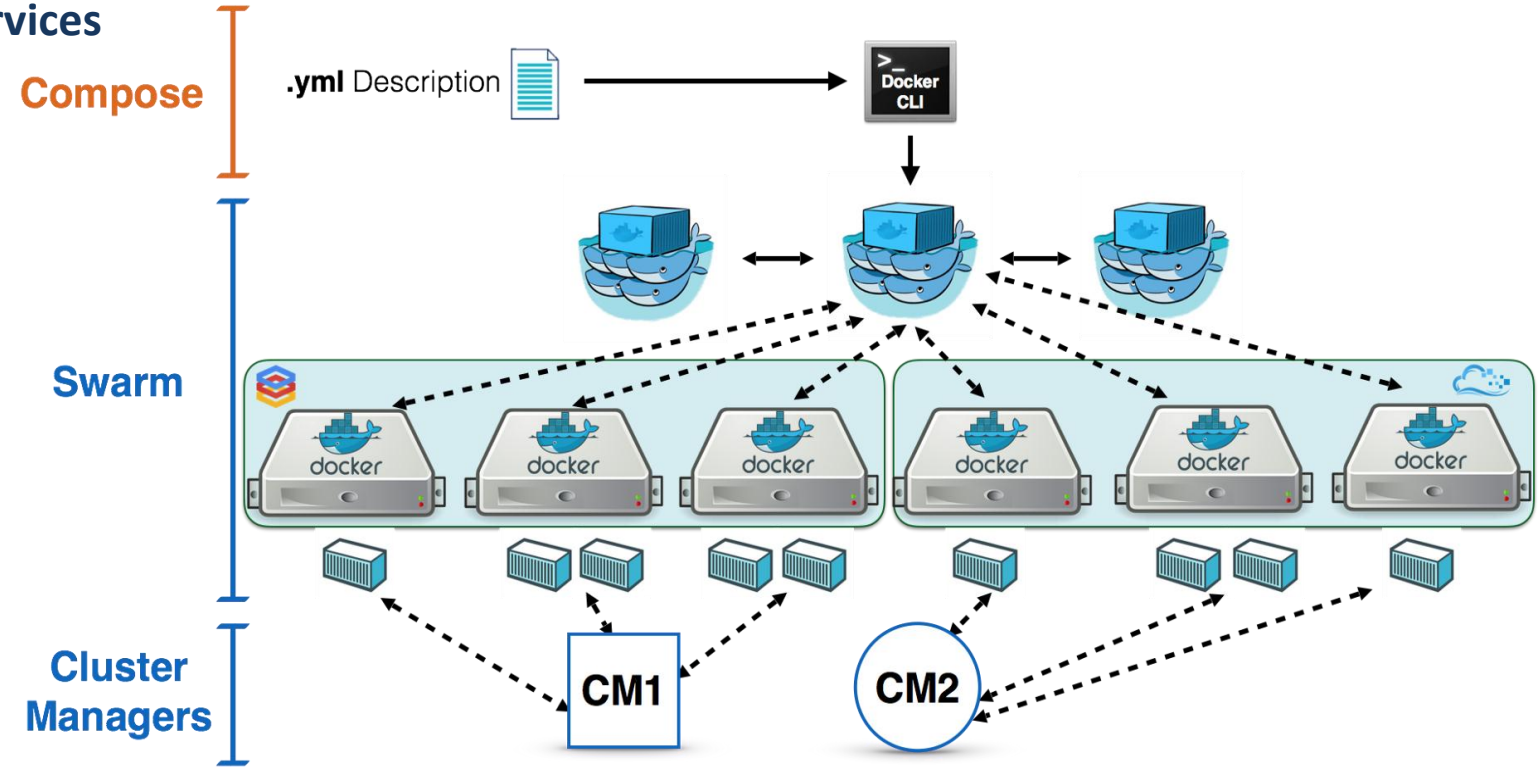
Application des mises à jour progressives

docker service update --image <imagename>:<version> web

Construction d'un stack avec Compose

Stack

Un stack est composé de plusieurs conteneur(s) Micro services
qui forment alors un Stack déployé avec un fichier compose **V3 permet de déployer des services**



```
version: '3'

services:
 web:
  image: nginx
  volumes:
    - "web-conf:/etc/nginx/conf.d/"
    - "web-files:/usr/share/nginx/html/"
  ports:
    - 8080:80
  logging:
   driver: gelf
   options:
    gelf-address: "udp://localhost:12201"
  deploy:
    mode: replicated
    replicas: 3
    placement:
     constraints: [node.role == worker]
 db:
  image: mysql
  ports:
    - 3306:3306
  environment:
    - MYSQL_ROOT_PASSWORD=root
  deploy:
    mode: replicated
    replicas: 3
    placement:
     constraints: [node.role == worker]
volumes:
 web-conf: {}
 web-files: {}
~
"docker-compose.yml" 33L, 566C
```

> docker stack deploy – compose-file docker-compose.yml web

>docker stack ls

>Docker stack service

> Manager

```
ludo@leader:~/web$ docker stack deploy --compose-file docker-compose.yml web
Creating network web_default
Creating service web_web
Creating service web_db
ludo@leader:~/web$
```