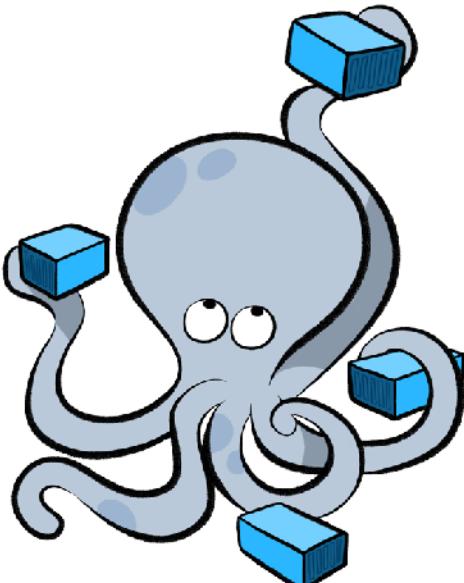


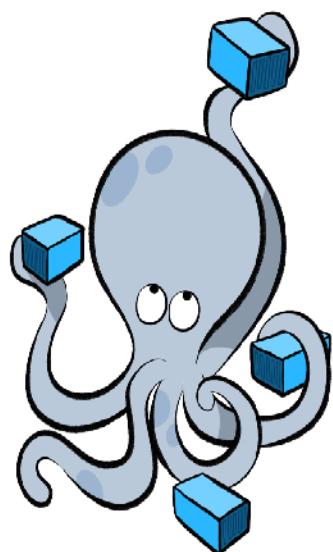
Docker Compose



100

Docker compose

- Docker Compose est un outil open source développé par Docker Inc et qui permet de définir et d'exécuter des applications **multi-conteneurs**.
- Il facilite la création, la configuration, le déploiement et la gestion d'applications conteneurisées.
- Docker Compose utilise des fichiers **YAML** pour décrire les services, les réseaux et les volumes nécessaires à l'application, ainsi que pour spécifier les relations entre les différents conteneurs.
- Ces fichiers peuvent être versionné et partagé.
- En utilisant Docker Compose, il est possible de déployer des applications conteneurisées de manière simple et reproductible sur différents environnements



101

Docker compose

- Docker Compose est utilisé pour exécuter plusieurs conteneurs en tant que service unique
- Plusieurs environnements isolés sur un seul hôte
- Par exemple, une application qui nécessite des conteneurs NGNIX et MySQL, vous pouvez créer un fichier qui démarre les deux conteneurs en tant que service (docker compose) au lieu de démarrer chacun séparément (docker run).
- Tous les services sont à définir au format YAML
- Il préserve les données de volume lors de la création des conteneurs
- Il Recrée uniquement les conteneurs qui ont changé

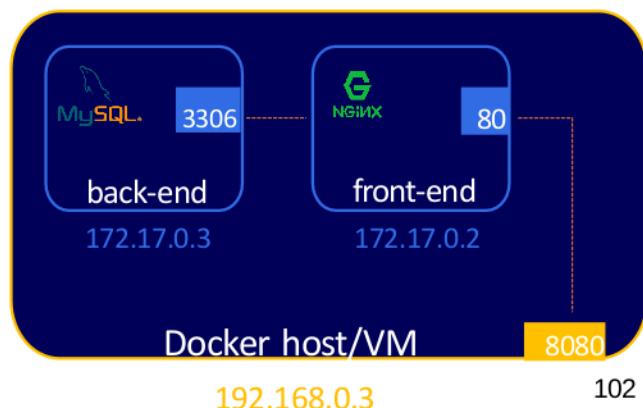
compose file: `docker-compose.yml`

Bring up the app:

`docker-compose up -d`

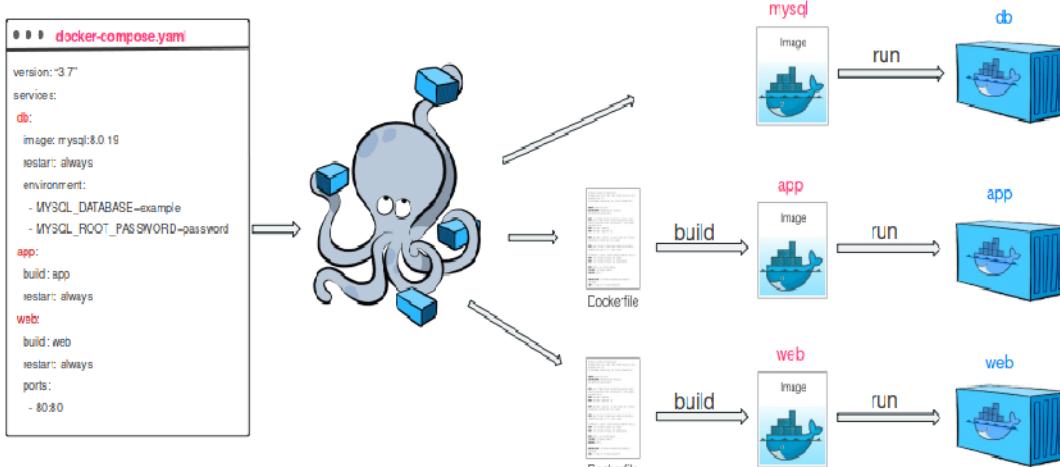
Bring down the app:

`docker-compose down`

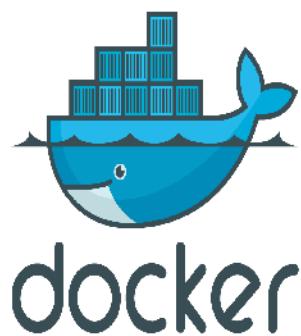
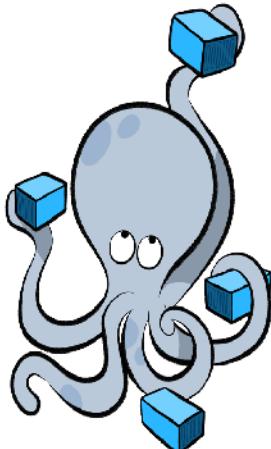


Docker compose

- Le fonctionnement de Docker Compose repose sur la définition de services, qui peuvent être des conteneurs Docker ou d'autres services, tels que des bases de données ou des serveurs web.
- Les services sont définis dans un fichier YAML qui spécifie leur configuration, leur image de conteneur, les ports à exposer, les volumes à utiliser et les dépendances entre les différents services.
- Docker Compose utilise ce fichier pour créer les conteneurs nécessaires et les lier entre eux.



Docker compose



- Docker est utilisé pour la gestion de conteneurs individuels,
- Docker Compose facilite la gestion des applications multi-conteneurs en utilisant un fichier de configuration YAML.

104

Docker Compose

Syntaxe de fichier Compose :

```
version: '3'  # si aucune version n'est spécifiée, alors v1 est utilisée.  
services:  # containers. identique à docker run  
  service_name: # nom du conteneur. c'est aussi le nom DNS à dans le réseau  
  image: # nom de l'image  
  Command: # remplace le CMD par défaut spécifié par l'image  
  ports: # identique à -p dans docker run  
  volumes: # identique à -v dans docker run  
  volumes: # identique à docker volume create  
  networks: # identique à docker network create
```

105

Docker Compose : versions

Version 1

- Les fichiers Compose qui ne déclarent pas de version sont considérés comme « version 1 »
- Ne prend pas en charge les volumes nommés, les réseaux définis par l'utilisateur ou les arguments de construction
- Chaque conteneur est placé sur le réseau de pont par défaut et est accessible depuis les autres conteneurs via son adresse IP.
- Pas de résolution DNS utilisant les noms de conteneurs

Version 2

- Résolution DNS via les noms de conteneurs
- Tous les services doivent être déclarés sous la clé 'service'
- Les volumes nommés peuvent être déclarés sous la clé volume et les réseaux peuvent être déclarés sous la clé network.
- Nouveau réseau de ponts pour connecter tous les conteneurs

Version 3

- Prise en charge de docker swarm

106

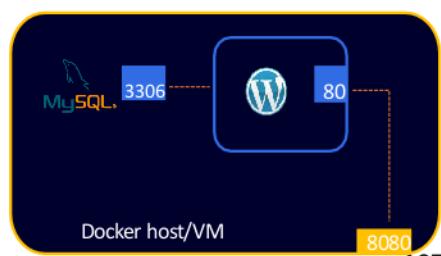
Docker Compose

```
version: '3.3'  
services:  
  wordpress:  
    image: wordpress  
    depends_on:  
      - mysql  
    ports:  
      - 8080:80  
    environment:  
      WORDPRESS_DB_HOST: mysql  
      WORDPRESS_DB_NAME: wordpress  
      WORDPRESS_DB_USER: wordpress  
      WORDPRESS_DB_PASSWORD: wordpress  
    volumes:  
      - ./wordpress-data:/var/www/html  
    networks:  
      - my_net  
  mysql:  
    image: mariadb  
    environment:  
      MYSQL_ROOT_PASSWORD: wordpress  
      MYSQL_DATABASE: wordpress  
      MYSQL_USER: wordpress  
      MYSQL_PASSWORD: wordpress  
    volumes:  
      - mysql-data:/var/lib/mysql  
    networks:  
      - my_net  
  volumes:  
    mysql-data:  
  networks:  
    my_net:
```

```
wordpress:  
  image: wordpress  
  depends_on:  
    - mysql  
  ports:  
    - 8080:80  
  environment:  
    WORDPRESS_DB_HOST: mysql  
    WORDPRESS_DB_NAME: wordpress  
    WORDPRESS_DB_USER: wordpress  
    WORDPRESS_DB_PASSWORD:  
      wordpress  
  volumes:  
    - ./wordpress-data:/var/www/html  
  networks:  
    - my_net
```

```
volumes:  
  mysql-data:  
networks:  
  my_net:
```

```
mysql:  
  image: mariadb  
  environment:  
    MYSQL_ROOT_PASSWORD: wordpress  
    MYSQL_DATABASE: wordpress  
    MYSQL_USER: wordpress  
    MYSQL_PASSWORD: wordpress  
  volumes:  
    - mysql-data:/var/lib/mysql  
  networks:  
    - my_net
```



107

Docker Compose

```
version: '3.3'  
services:  
  wordpress:  
    image: wordpress  
    depends_on:  
      - mysql  
    ports:  
      - 8080:80  
    environment:  
      WORDPRESS_DB_HOST: mysql  
      WORDPRESS_DB_NAME: wordpress  
      WORDPRESS_DB_USER: wordpress  
      WORDPRESS_DB_PASSWORD: wordpress  
    volumes:  
      - ./wordpress-data:/var/www/html  
    networks:  
      - my_net  
  mysql:  
    image: mariadb  
    environment:  
      MYSQL_ROOT_PASSWORD: wordpress  
      MYSQL_DATABASE: wordpress  
      MYSQL_USER: wordpress  
      MYSQL_PASSWORD: wordpress  
    volumes:  
      - mysql-data:/var/lib/mysql  
    networks:  
      - my_net  
volumes:  
  mysql-data:  
  networks:  
  my_net:
```

Services file: **docker-compose.yml**

Bring up: **docker-compose up -d**

Bring down: **docker-compose down**

Process state: **docker-compose ps**

```
root@docker-master:/home/osboxes/docker# docker-compose up -d  
Creating network "docker_my_net" with the default driver  
Creating volume "docker_mysql-data" with default driver  
Pulling mysql (mariadb:)...  
latest: Pulling from library/mariadb  
23884877105a: Pull complete  
bc38caa0f5b9: Pull complete  
2910811b6c42: Pull complete  
36505266dcc6: Pull complete  
e69dcc78e96e: Pull complete  
222f44c5392d: Pull complete  
efc64ea97b9c: Pull complete  
9912a149de6b: Extracting [=====>] 115B/115B
```

```
root@docker-master:/home/osboxes/docker# docker ps  
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS  
9784a2cc6e02        wordpress          "docker-entrypoint.s..."   5 minutes ago     Up 5 minutes       0.0.0.0  
2657b6db3f94        mariadb            "docker-entrypoint.s..."   5 minutes ago     Up 5 minutes       3306/tcp  
root@docker-master:/home/osboxes/docker#
```

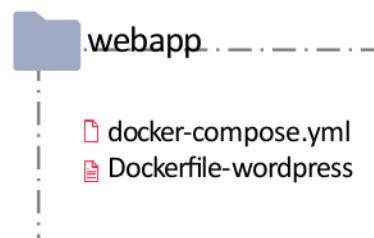
108

Docker Compose

```
version: '3.3'  
services:  
  wordpress:  
    image: wordpress  
    depends_on:  
      - mysql  
    ports:  
      - 8080:80  
    environment:  
      WORDPRESS_DB_HOST: mysql  
      WORDPRESS_DB_NAME: wordpress  
      WORDPRESS_DB_USER: wordpress  
      WORDPRESS_DB_PASSWORD: wordpress  
    volumes:  
      - ./wordpress-data:/var/www/html  
    networks:  
      - my_net  
  mysql:  
    image: mariadb  
    environment:  
      MYSQL_ROOT_PASSWORD: wordpress  
      MYSQL_DATABASE: wordpress  
      MYSQL_USER: wordpress  
      MYSQL_PASSWORD: wordpress  
    volumes:  
      - mysql-data:/var/lib/mysql  
    networks:  
      - my_net  
volumes:  
  mysql-data:  
  networks:  
  my_net:
```

Si l'image doit être créée avant le déploiement, incluez le fichier dockerfile dans le fichier compose

```
version: "3"  
services:  
  wordpress:  
    build:  
      context: .  
      dockerfile: Dockerfile-wordpress  
      image: wordpress  
      container_name: wordpress
```



To build only images: **docker-compose build**

build + deploy: **docker-compose up -d**

109

Docker Compose

```
version: '3.3'
services:
  wordpress:
    image: wordpress
    depends_on:
      - mysql
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: mysql
      WORDPRESS_DB_NAME: wordpress
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
    volumes:
      - ./wordpress-data:/var/www/html
    networks:
      - my_net
  mysql:
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - my_net
    volumes:
      mysql-data:
    networks:
      my_net:
```

Déploiement via des commandes :

```
docker network create --driver bridge my_net
docker volume create mysql-data

docker run --name wordpress -p 8080:80 -v ./wordpress-data:/var/www/html \
--net my_net -e WORDPRESS_DB_HOST=mysql \
-e WORDPRESS_DB_NAME=wordpress \
-e WORDPRESS_DB_USER=wordpress \
-e WORDPRESS_DB_PASSWORD=wordpress \
--name mysql -p 3306 -v mysql-data:/var/lib/mysql --net my_net
-e MYSQL_ROOT_PASSWORD=wordpress \
-e MYSQL_DATABASE=wordpress \
-e MYSQL_USER=wordpress \
-e MYSQL_PASSWORD=wordpress \
--name mariadb
```

110

Docker Compose

Docker Compose Commands:

- **Create a compose**
 - docker-compose up -d
- **List containers created by compose**
 - docker-compose ps / docker container ls
- **Stop a compose**
 - docker-compose stop
- **Start a compose**
 - docker-compose start
- **Restart a compose**
 - docker-compose restart
- **Delete a compose**
 - docker-compose down
- **Scale a service**
 - docker-compose scale service=number

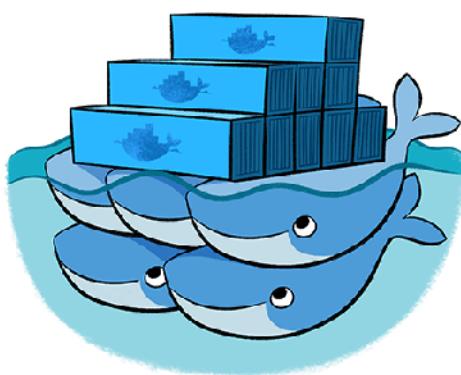
Docker Compose : Installation

- Deux façons d'installer docker compose :
 - Installer le **compose plugin**
 - sudo apt-get update
 - sudo apt-get install docker-compose-plugin
 - Installer le **compose standalone**
 - curl -SL https://github.com/docker/compose/releases/download/v2.24.5/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose
 - sudo chmod +x /usr/local/bin/docker-compose
- **Note** that Compose standalone uses the **-compose** syntax instead of the current standard syntax **compose**.
 - For example type **docker-compose up** when using Compose **standalone**, instead of **docker compose up** when using **compose plugin**.

Reference : <https://docs.docker.com/compose/reference/>

112

Docker Swarm



113

Orchestration des conteneurs

- Les conteneurs s'arrêtent lorsque le processus qu'ils contiennent se termine ou s'arrête à cause d'une erreur.
- De plus, un seul conteneur par service peut ne pas être suffisant pour gérer le trafic croissant vers l'application.
- Dans tous les scénarios ci-dessus, nous avons besoin d'un outil capable de relancer les conteneurs arrêtés ou d'en créer de nouveaux pour gérer le trafic croissant et garantir l'équilibrage de charge et la haute disponibilité de l'application à tout moment.
- C'est là que l'orchestration des conteneurs entre en jeu !!!

114

Orchestration des conteneurs

- Qu'est-ce que l'orchestration de conteneurs ?
 - L'orchestration de conteneurs consiste à gérer les cycles de vie des conteneurs, en particulier dans les environnements vastes et dynamiques.
 - Approvisionnement et déploiement de conteneurs
 - Ajouter ou supprimer des conteneurs pour répartir uniformément la charge des applications sur l'infrastructure hôte
 - déplacement de conteneurs d'un hôte à un autre s'il y a un manque de ressources dans un hôte, ou si un hôte s'arrête
 - Équilibrage de charge entre les conteneurs
 - Permet aux développeurs de déployer des applications à grande échelle de manière plus rapide et plus fiable

115

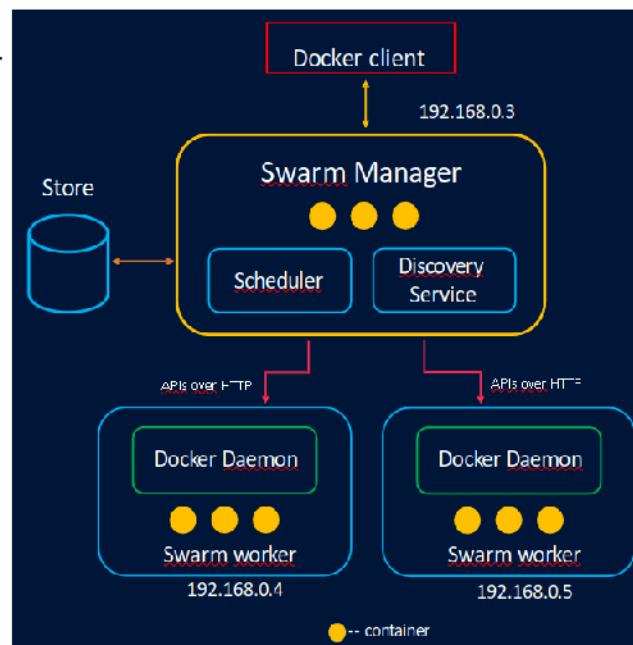
Docker Swarm

- Swarm est la solution d'orchestration de conteneurs intégrée à Docker. Son objectif principal est de gérer des conteneurs dans un cluster, c'est-à-dire un ensemble de machines connectées qui fonctionnent ensemble.
- Lorsqu'une nouvelle machine rejoint le cluster, elle devient un nœud de swarm.
- En utilisant Docker Swarm, nous pouvons obtenir une haute disponibilité, un équilibrage de charge et un accès décentralisé à nos applications.
- Que peut-on faire en utilisant Docker Swarm ?
 - Déployer de nouveaux conteneurs pour remplacer ceux défaillants
 - Augmenter le nombre de conteneurs pour l'équilibrage de charge
 - Restauration des applications vers les anciennes versions
 - Arrêter un nœud pour la maintenance sans aucun temps d'arrêt de l'application
 - Prend en charge l'orchestration, la haute disponibilité, la mise à l'échelle, l'équilibrage de charge, les mises à jour progressives, les restaurations, etc.
 - utilise le protocole TLS (Transport Layer Security) pour la communication et l'authentification avec les nœuds.

116

Architecture de Docker Swarm

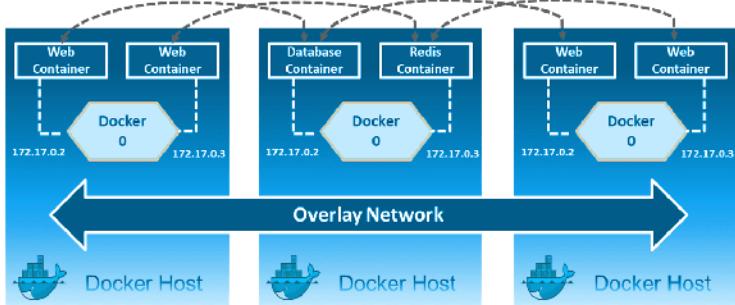
- Deux type de nœuds:
 - **Manager** : gère et orchestre le cluster
 - **Worker** : exécute les tâches assignées par le manager
- **Scheduler**: planifie les conteneurs sur les nœuds en fonction des règles et des filtres
- **Discovery service** : aide Swarm Manager à découvrir de nouveaux nœuds et à récupérer les nœuds disponibles
- **Store** : stocke l'état du cluster ; comme les informations sur les services de cluster et de swarm. C'est une base de données clé-valeur



117

Docker Swarm : Overlay network

- Les réseaux de type **Bridge** s'appliquent aux conteneurs exécutés sur le même hôte.
- Pour la communication entre les conteneurs exécutés sur différents hôtes, comme dans Swarm, nous devons utiliser un réseau **Overlay**.
- Si vous créez des services **swarm** et vous ne spécifiez pas de réseau, ils sont connectés au réseau **ingress** par défaut qui est de type overlay.
- Le réseau overlay s'étend sur l'ensemble du cluster Swarm, permettant la communication entre les conteneurs sur plusieurs nœuds



docker network ls

```
root@docker-master:/home/osboxes/docker# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
4a505a08943f    bridge    bridge      local
91bc90acf6b6    docker_gwbridge  bridge      local
305bfc47d41a    host      host       local
zyaa6fek5d7j    ingress   overlay    swarm
56ddd6ef1d30    my_net    bridge      local
5347bd97c94f    none     null       local
root@docker-master:/home/osboxes/docker#
```

118

Docker Swarm : Configuration

- Configuration de Docker Swarm :
 - Configurez Swarm Manager.
 - Ajoutez un worker node à Swarm manger.
- Configurez Swarm Manager :
 - Install Docker CE.
 - docker info | grep Swarm
 - docker swarm init --advertise-addr [Node Private IP]
 - docker info | grep Swarm
 - docker node ls

119

Docker Swarm : Configuration

docker swarm init --advertise-addr MANAGER_IP

```
root@docker-master # docker swarm init --advertise-addr 192.168.0.100
```

Swarm initialized: current node (pet61mspmfyvk6l0klisd9e9) is now a manager.

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

master node

```
docker swarm join --token SWMTKN-1-
```

```
3kabuqlekspyqdk02vj0strami9mcysj8lw4klp5cwmh3wm4ej-9rlom6m90chce1tgi0ke58t4l
```

```
192.168.0.100:2377
```

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

```
root@docker-slave01 # docker swarm join --token SWMTKN-1-
```

```
3kabuqlekspyqdk02vj0strami9mcysj8lw4klp5cwmh3wm4ej-9rlom6m90chce1tgi0ke58t4l
```

```
192.168.0.100:2377
```

This node joined a swarm as a worker.

worker node

```
root@docker-slave01:/
```

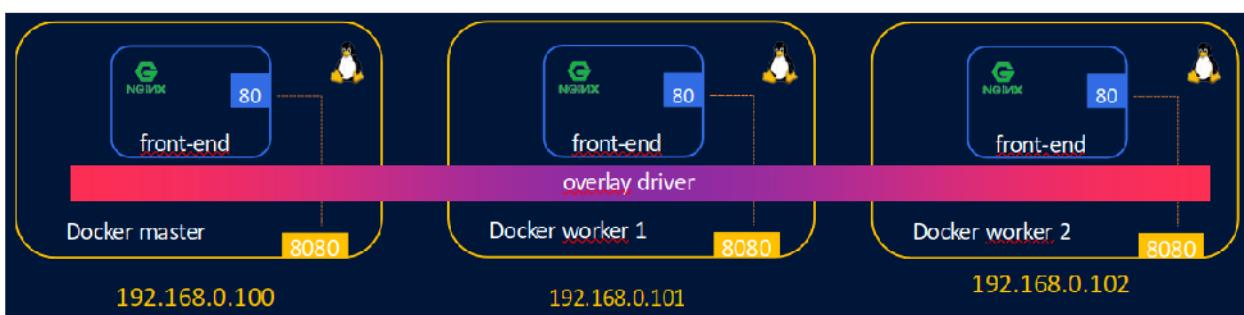
```
root@docker-slave01:/home/osboxes#
```

120

Docker Swarm : Configuration

docker node ls

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
m75v5bd6jd1vsy9usawecsu7r *	docker-master	Ready	Active	Leader
wywos5cppaol45967s0mtqj1n	docker-slave01	Ready	Active	
uimguox32lco6odxhpdq3nxnf2	docker-slave02	Ready	Active	



```
root@docker-master # docker network ls | grep overlay
```

```
qh17ylskxm31      ingress      overlay      swarm
```

121

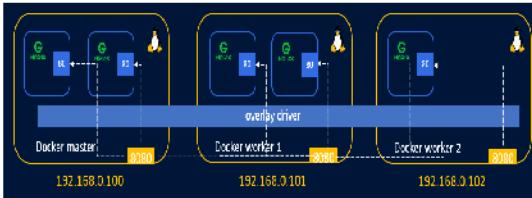
Docker Swarm : Service

- Pour déployer une image d'une application lorsque Docker Engine est en mode swarm, vous créez un service.
- Un **service** est l'image d'un microservice comme un serveur HTTP, une base de données ou tout autre type de programme exécutable que vous souhaitez exécuter dans un environnement distribué.
- Un service a besoin d'une image de conteneur à utiliser, du port sur lequel swarm rend le service disponible en dehors de swarm, d'un réseau overlay permettant au service de se connecter aux autres services et du nombre de répliques de l'image à exécuter dans swarm.

Créer un service :

```
docker service create --replicas 5 -p 80:80 --name web nginx
```

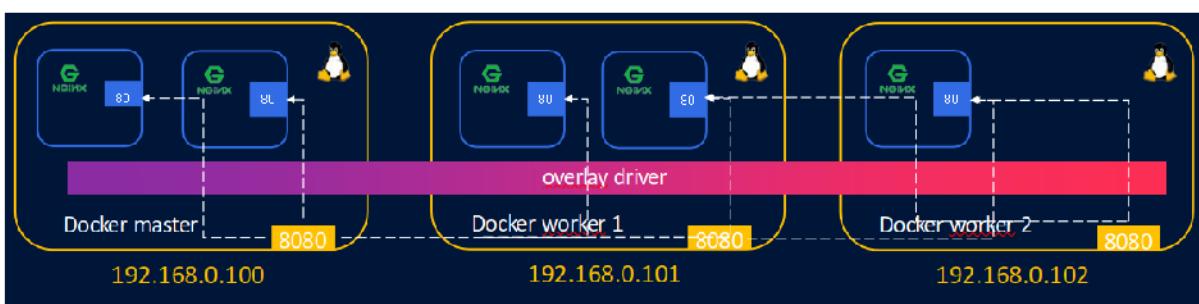
```
root@docker-master:/home/osboxes# docker service create --replicas 5 -p 80:80 --name web nginx
6obmcjveg1zk5eb9nd3ypj90e
overall progress: 5 out of 5 tasks
1/5: running [=====>]
2/5: running [=====>]
3/5: running [=====>]
4/5: running [=====>]
5/5: running [=====>]
verify: Service converged
```



Docker Swarm : Service

docker service ls

```
root@docker-master:/home/osboxes# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
6obmcjveg1zk    web      replicated  5/5      nginx:latest
root@docker-master:/home/osboxes#
```



Docker Swarm : Service

```
docker service create --replicas 5 -p 80:80 --name web nginx
```

docker service ps web

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
qpu0955z15rb	web.1	nginx:latest	docker-slave02	Running	Running 2 minutes ago
ytrie6t5kb50	web.2	nginx:latest	docker-master	Running	Running 2 minutes ago
rxubw2mh8hxv	web.3	nginx:latest	docker-slave01	Running	Running 2 minutes ago
rv9apxkz07xa	web.4	nginx:latest	docker-slave02	Running	Running 2 minutes ago
s1dgil2sp35d	web.5	nginx:latest	docker-slave01	Running	Running 2 minutes ago

192.168.0.1
00

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](#). Commercial support is available at [nginx.com](#).

Thank you for using nginx.

192.168.0.1
01

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](#). Commercial support is available at [nginx.com](#).

Thank you for using nginx.

192.168.0.1
04

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](#). Commercial support is available at [nginx.com](#).

Thank you for using nginx.

Master

Slave -01

Slave -02

124

Docker Swarm : Service

docker service ps web

Name of the service

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
qpu0955z15rb	web.1	nginx:latest	docker-slave02	Running	Running 2 minutes ago
ytrie6t5kb50	web.2	nginx:latest	docker-master	Running	Running 2 minutes ago
rxubw2mh8hxv	web.3	nginx:latest	docker-slave01	Running	Running 2 minutes ago
rv9apxkz07xa	web.4	nginx:latest	docker-slave02	Running	Running 2 minutes ago
s1dgil2sp35d	web.5	nginx:latest	docker-slave01	Running	Running 2 minutes ago

docker node ps docker-slave01

Name of the node

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
rxubw2mh8hxv	web.3	nginx:latest	docker-slave01	Running	Running 3 minutes ago
s1dgil2sp35d	web.5	nginx:latest	docker-slave01	Running	Running 3 minutes ago

125

Docker Swarm : Service

Supposons qu'un nœud est en panne à cause d'un problème

Dans ce cas, les conteneurs concernés dans ce nœud sont relancés dans les nœuds disponibles du cluster

docker service ls

```
root@docker-master:/home/osboxes# docker service ls
ID          NAME      MODE      REPLICAS      IMAGE
6obmcjveglzk    web      replicated   3/5          nginx:latest
```

```
root@docker-master:/home/osboxes# docker service ls
ID          NAME      MODE      REPLICAS      IMAGE
6obmcjveglzk    web      replicated   5/5          nginx:latest
```

docker service ps web

```
root@docker-master:/home/osboxes# docker service ps web
ID          NAME      IMAGE      NODE      DESIRED STATE      CURRENT STATE
4d65ub7ctbyg  web.1    nginx:latest  docker-master  Running        Running 18 seconds ago
qpu0955z15rb  \_ web.1  nginx:latest  docker-slave02  Shutdown       Running 37 seconds ago
ytrie6t5kb50   web.2    nginx:latest  docker-master  Running        Running 8 minutes ago
rxubw2mh8hxv   web.3    nginx:latest  docker-slave01  Running        Running 8 minutes ago
kaytsldhrdho   web.4    nginx:latest  docker-master  Running        Running 17 seconds ago
rv9apxkz07xa  \_ web.4  nginx:latest  docker-slave02  Shutdown       Running 37 seconds ago
sldgil2sp35d   web.5    nginx:latest  docker-slave01  Running        Running 8 minutes ago
```

126

Docker Swarm : Service

Scale a service (Mettre à l'échelle un service)

docker service scale <service_name>=value

docker service scale web=10

Update a service

docker service update --image <image_name>:<version> <service_name>

docker service update --image nginx:1.18 web

Rolling update (Mise à jour continue)

docker service update --image <new_image> --update-parallelism 2 --update-delay 10s <service_name>

docker service update --image nginx:1.18 --update-parallelism 2 --update-delay 10s web

127

Docker Swarm : Service

Supposons qu'un nœud doit être arrêté pour maintenance

Dans ce cas, nous vidons ce nœud de ses charges de travail sur d'autres nœuds

Drain a node: `docker node update --availability drain <node-name>`

Undrain a node: `docker node update --availability active <node-name>`

- Supposons que le nœud est opérationnel après la maintenance

Par défaut, swarm ne redéploie pas les conteneurs que le nœud exécutait précédemment.

Nous devons forcer le cluster à partager la charge entre les nœuds disponibles

Force workload balance: `docker service update --force <service_name>`

Rollback an update: `docker service update --rollback <service_name>`

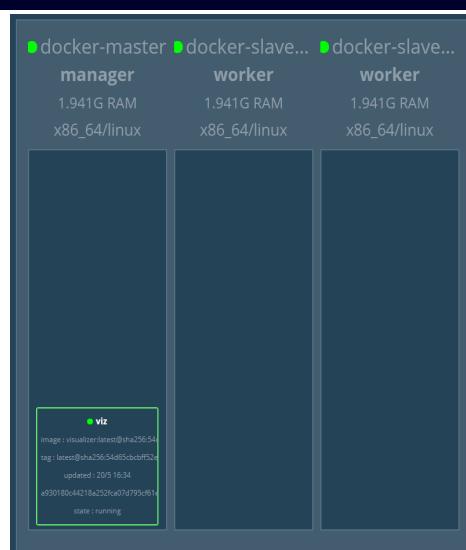
Remove a service: `docker service rm <service_name>`

128

Docker Swarm : Visualizer

Déployer le visualiseur

```
docker run -it -d -p 8080:8080  
-v/var/run/docker.sock:/var/run/docker.sock  
dockersamples/visualizer
```



Dashboard disponible à

`http://<node-ip>:8080`

- node-ip: n'importe quelle adresse IP du nœud participant au cluster
- bien que le pod soit lié au maître, il peut être atteint depuis tous les nœuds grâce au ingress routing mesh

<https://github.com/dockersamples/docker-swarm-visualizer>

129

Docker Swarm : Visualizer

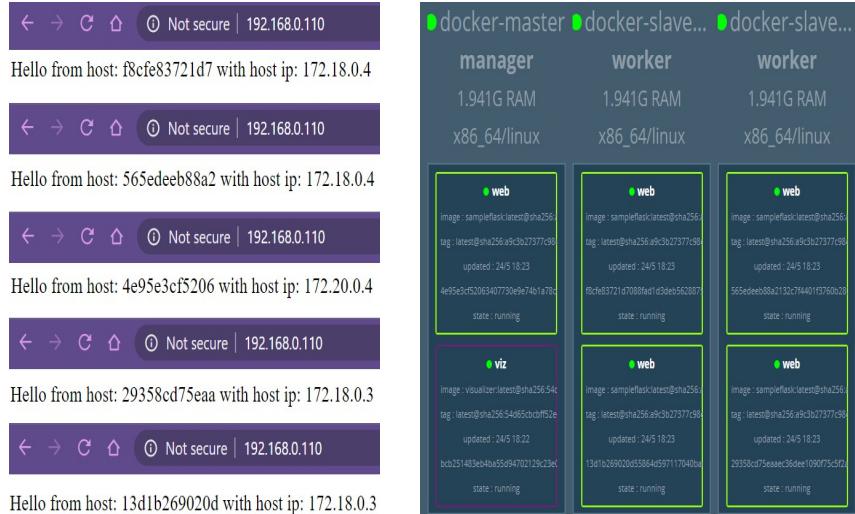
Demo: Visualizer

docker service create --replicas 5 -p 80:5000 --name web kunchalavikram/sampleflask:v2

visiter

<http://<node-ip>:80>

et actualisez la page pour voir l'effet d'équilibrage de charge



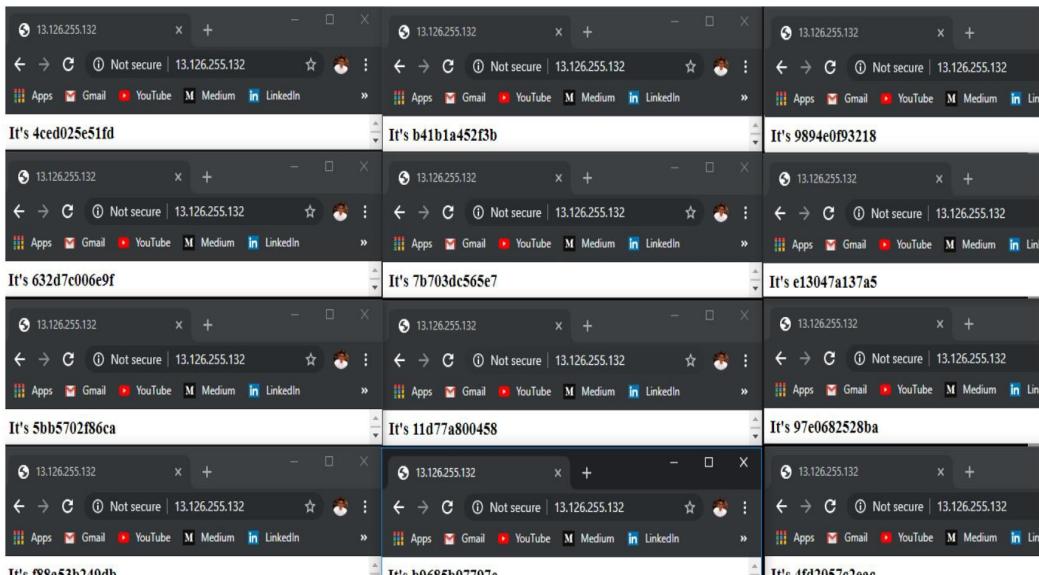
130

Docker Swarm : Visualizer

Demo: Visualizer & Load balancing

Trouvez une démo de visualizer et d'équilibrage de charge de swarm sur le lien ci-dessous

<https://levelup.gitconnected.com/load-balance-and-scale-node-js-containers-with-nginx-and-docker-swarm-9fc97c3cff81>



131

Docker Swarm: Networking

Creating overlay network

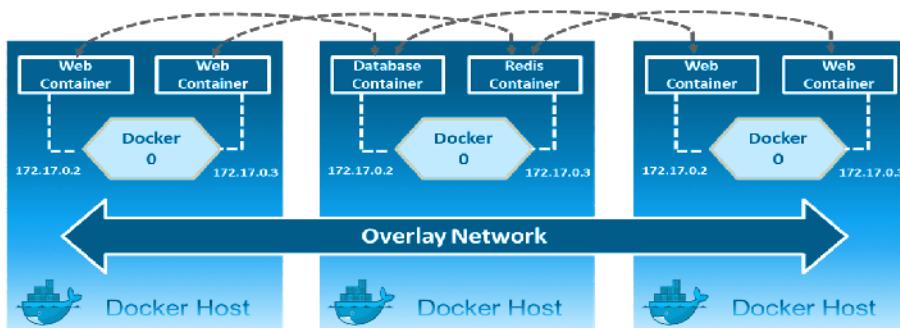
```
docker network create --driver overlay my_overlay
```

```
root@docker-master:/home/osboxes/docker# docker network ls | grep overlay
zyaa6fek5d7j      ingress          overlay        swarm
mv4mvtgnteeb     my_overlay       overlay        swarm
root@docker-master:/home/osboxes/docker#
```

```
docker service create --replicas 5 -p 80:80 --network my_overlay --name web nginx
```

Inspecting the service:

```
docker service inspect web
```



132

Docker Swarm: Networking

Ingress Network

Est-il possible d'exécuter 2 conteneurs avec le même port publié en mode non swarm?

```
docker run -d -p 80:5000 --name web1 flask
```

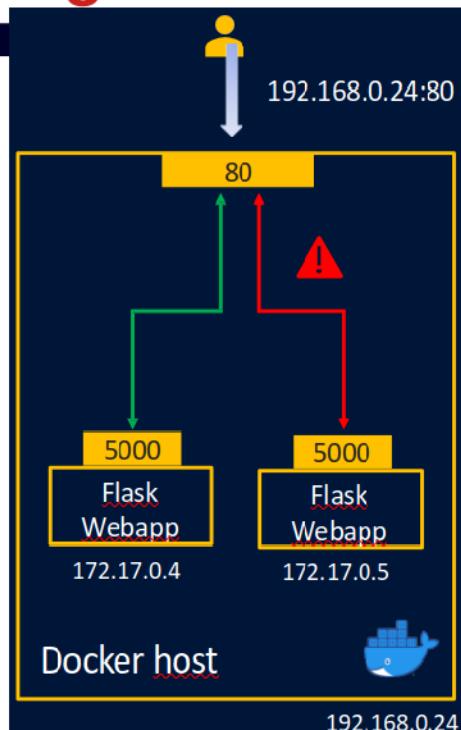
```
docker run -d -p 80:5000 --name web2 flask
```

Non!!! Cela entraîne un conflit de port. Le démon Docker n'autorise pas le même port pour plusieurs conteneurs en mode non swarm

Error : Bind for 0.0.0.0:80 failed: port is already allocated.

Alors, comment Docker permet-il d'exécuter plusieurs instances de la même application (avec le même mappage de port hôte) dans un nœud à l'intérieur d'un cluster Swarm ?

Ingress Routing Mesh



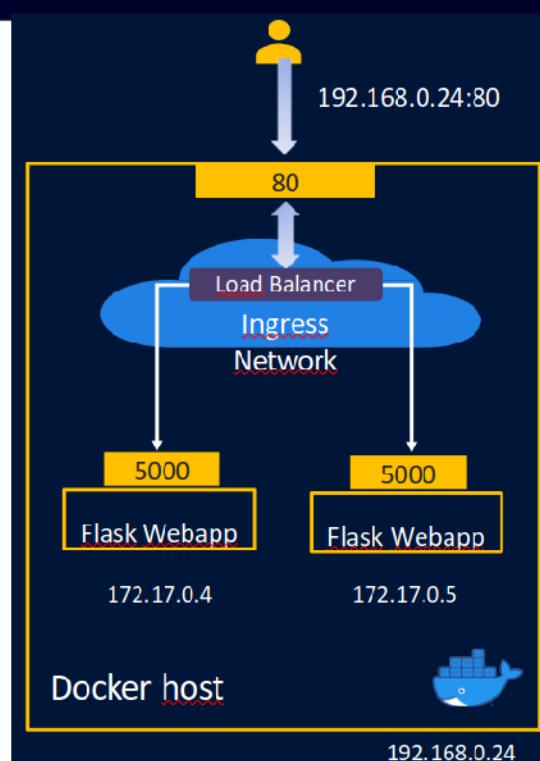
```
docker service create --replicas 5 -p 80:5000 --name web flask
```

133

Docker Swarm: Networking

Ingress Network

- Le réseau Ingress dispose d'un équilibrer de charge intégré qui redirige le trafic du port publié sur l'hôte vers les ports mappés du conteneur .
- Cela permet à plusieurs conteneurs à l'intérieur d'un hôte Docker d'utiliser le même port hôte sans collisions de ports.
- Aucune configuration externe n'est nécessaire, l'équilibrer de charge fonctionne immédiatement avec le réseau Ingress



```
docker service create --replicas 5 -p 80:5000 --name web flask
```

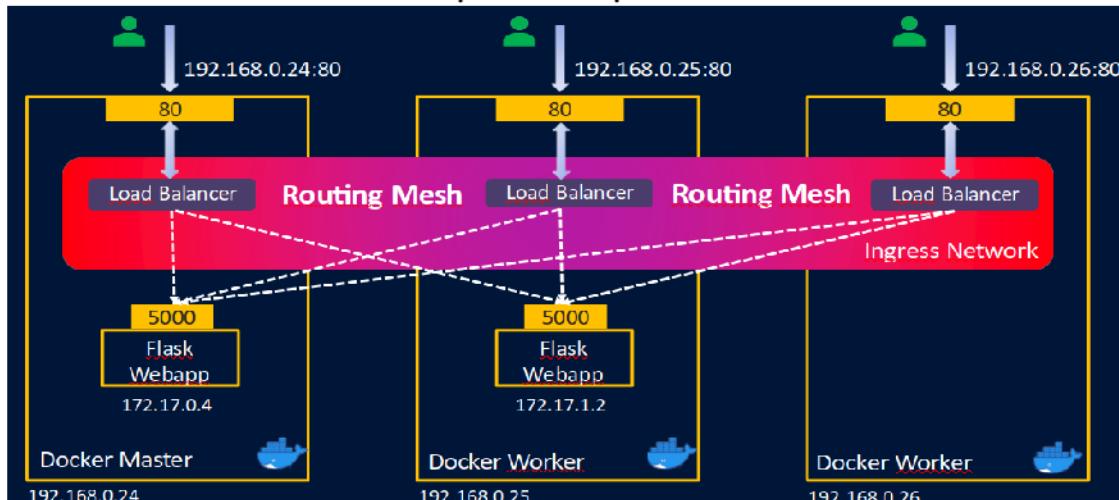
134

Docker Swarm: Networking

Ingress Routing Mesh

Le maillage réseau Ingress de swarm permet à chaque nœud du cluster d'accepter les connexions à n'importe quel port de service publié dans swarm en acheminant toutes les demandes entrantes vers les nœuds disponibles hébergeant un service avec le port publié.

```
docker service create --replicas 2 -p 80:5000 --name web flask
```

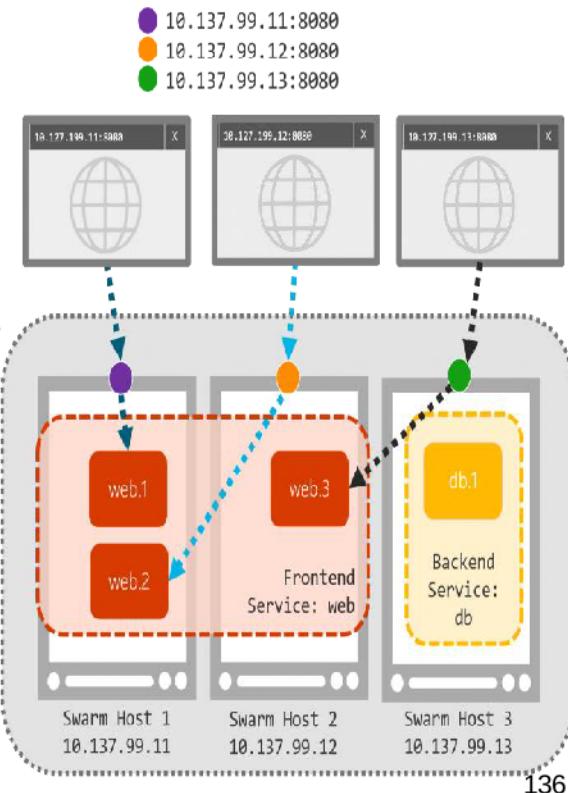


135

Docker Swarm: Networking

Ingress Routing Mesh

- Le maillage de routage permet à chaque nœud de swarm d'accepter des connexions sur les ports publiés pour tout service exécuté dans swarm, même si aucune tâche n'est en cours d'exécution sur le nœud.
- Le maillage de routage achemine toutes les demandes entrantes vers les ports publiés sur les nœuds disponibles vers un conteneur actif.
- Dans ce scénario, nous avons 3 nœuds et 3 répliques de conteneurs Web. Les conteneurs Web sont planifiés uniquement sur l'hôte 1 et l'hôte 2. Mais nous pouvons toujours accéder aux services Web à partir de l'hôte 3 grâce au maillage de routage en swarm.

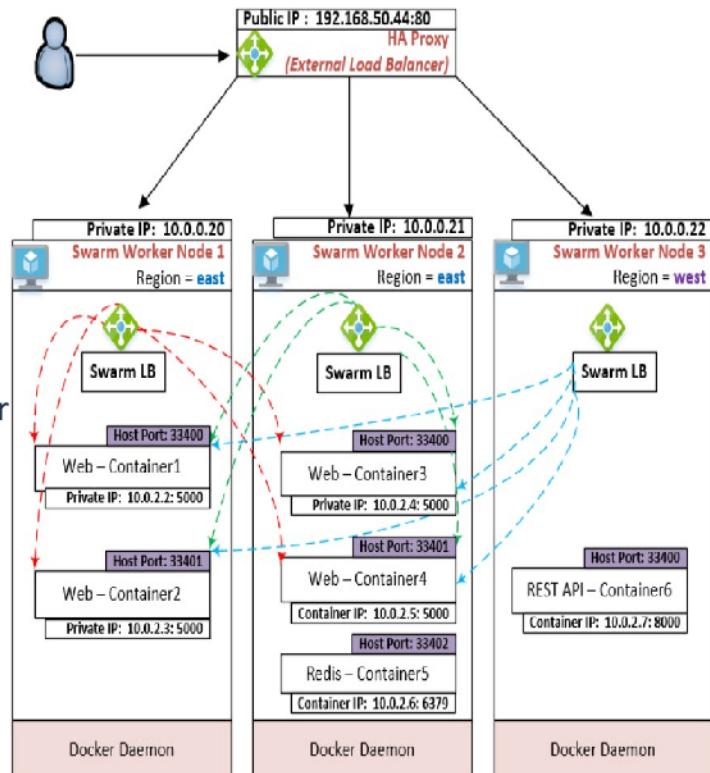


136

Docker Swarm: Networking

Load Balancer (ingress routing mesh)

Le réseau Ingress est un réseau overlay spécial qui facilite l'équilibrage de charge entre les nœuds d'un service. Lorsqu'un nœud Swarm reçoit une requête sur un port publié, il transmet cette requête à un module appelé IPVS. IPVS garde une trace de toutes les adresses IP participant à ce service, en sélectionne une et achemine la demande vers celle-ci, via le réseau Ingress.

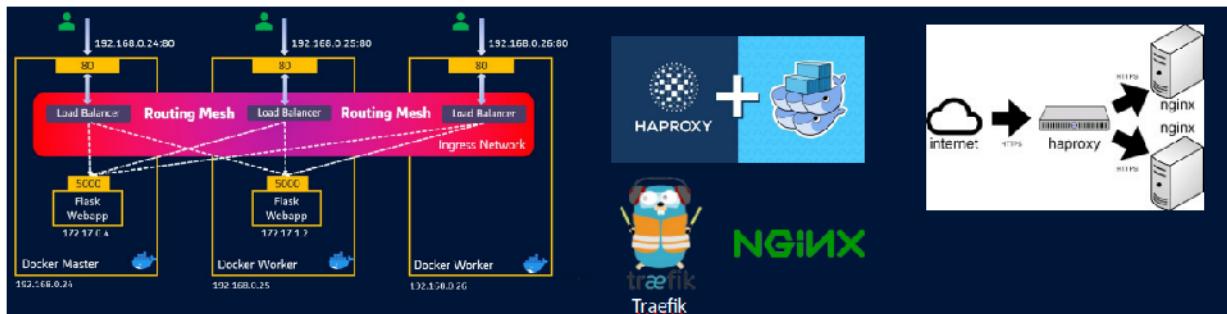


137

Docker Swarm: Networking

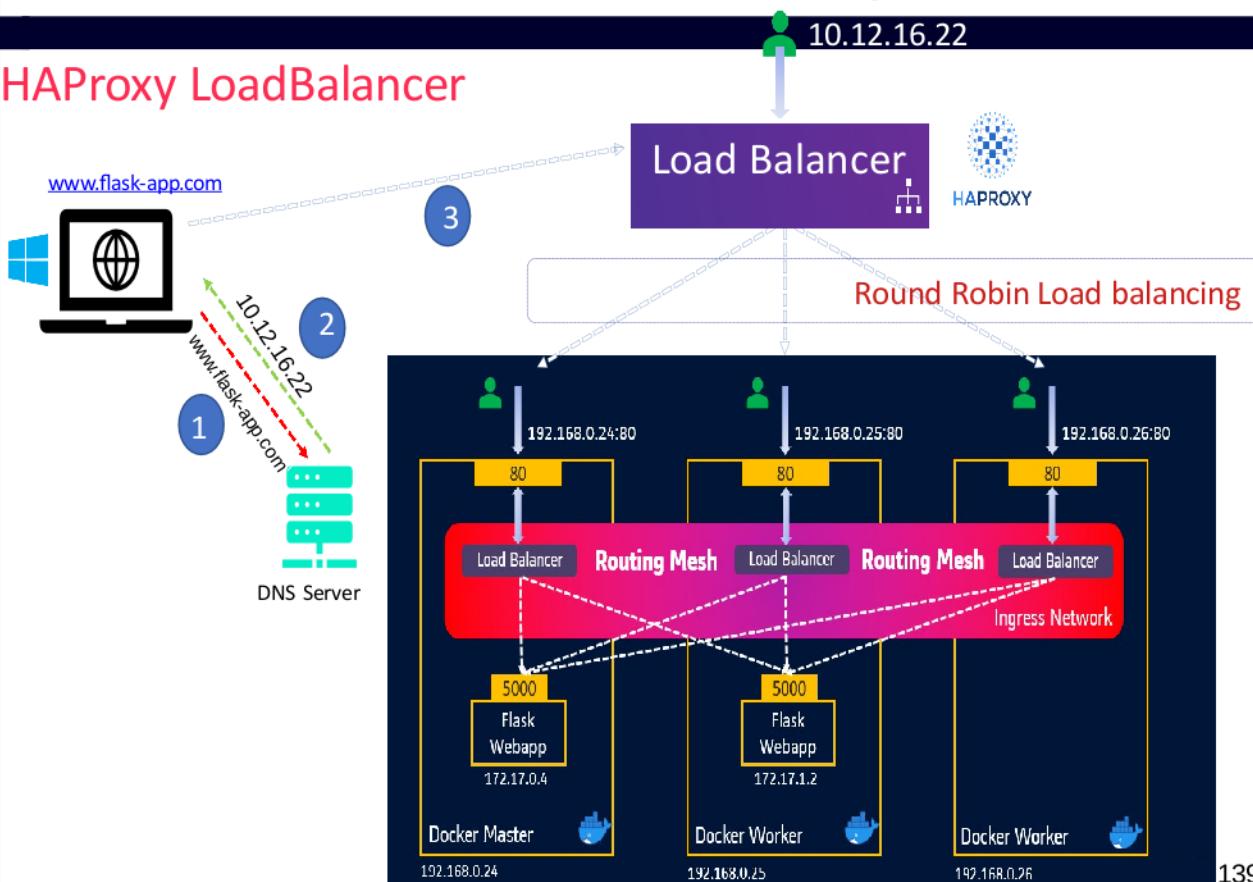
LoadBalancer

- L'un des principaux avantages de Docker Swarm est d'augmenter la disponibilité des applications grâce à la redondance.
- L'application est accessible depuis tous les nœuds disponibles du cluster en utilisant <node-ip>:port
- Mais comment l'utilisateur final peut-il accéder à cette application via un point de terminaison commun, c'est-à-dire un nom DNS ?
 - C'est via un LoadBalancer externe/Reverse Proxy
- L'équilibrEUR de charge HAProxy achemine le trafic externe vers le cluster et équilibre sa charge sur les nœuds disponibles
- D'autres équilibrEURS de charge populaires incluent Nginx et Traefik



Docker Swarm: Networking

HAProxy LoadBalancer

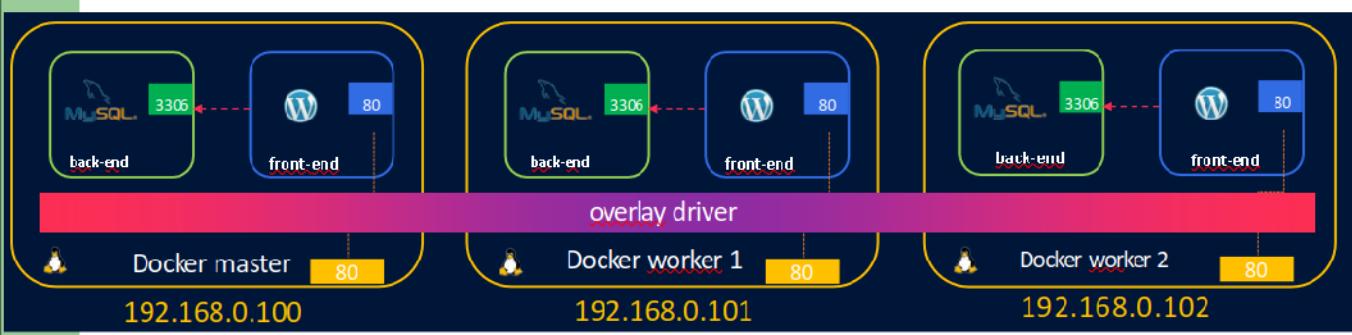


Docker Stack

142

Docker Stack

- Docker Stack est utilisée pour déployer une pile d'applications complète sur swarm.
- Docker-compose est mieux adapté aux scénarios de développement et constitue une solution de déploiement à hôte unique
- Utilise la même syntaxe que le fichier docker-compose, mais avec une nouvelle section '**deploy**' ajoutée au fichier
- Docker stack ignore les instructions '**build**'. Vous ne pouvez pas créer de nouvelles images à l'aide des commandes de pile. Il a besoin d'images prédéfinies pour exister



143

Docker Stack

```
version: '3.3'
services:
  wordpress:
    image: wordpress
    depends_on:
      - mysql
    ports:
      - 8080:80
    deploy:
      replicas: 2
      placement:
        constraints:
          - node.role == manager
environment:
  WORDPRESS_DB_HOST: mysql
  WORDPRESS_DB_NAME: wordpress
  WORDPRESS_DB_USER: wordpress
  WORDPRESS_DB_PASSWORD: wordpress
volumes:
  - wordpress-data:/var/www/html
networks:
  - my_net
mysql:
  image: mariadb
  deploy:
    replicas: 1
    placement:
      constraints:
        - node.role == worker
environment:
  MYSQL_ROOT_PASSWORD: wordpress
  MYSQL_DATABASE: wordpress
  MYSQL_USER: wordpress
  MYSQL_PASSWORD: wordpress
volumes:
  - mysql-data:/var/lib/mysql
networks:
  - my_net
networks:
  my_net:
  volumes:
    mysql-data:
    wordpress-data:
```

The diagram illustrates the comparison between Docker Stack and Docker Compose. It shows two side-by-side configurations with annotations.

Docker stack (Left):

```
wordpress:
  image: wordpress
  depends_on:
    - mysql
  ports:
    - 8080:80
  deploy:
    replicas: 2
    placement:
      constraints:
        - node.role == manager
  environment:
    WORDPRESS_DB_HOST: mysql
    WORDPRESS_DB_NAME: wordpress
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD: wordpress
  wordpress
  volumes:
    - wordpress-data:/var/www/html
  networks:
    - my_net
```

A red box highlights the `deploy: replicas: 2` section. A callout box points to it with the text: "Constraints permettent de déployer les conteneurs dans un nœud spécifique..".

Docker compose (Right):

```
wordpress:
  image: wordpress
  depends_on:
    - mysql
  ports:
    - 8080:80
  environment:
    WORDPRESS_DB_HOST: mysql
    WORDPRESS_DB_NAME: wordpress
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD: wordpress
  volumes:
    - ./wordpress-data:/var/www/html
  networks:
    - my_net
```

Docker stack vs docker compose

144

Docker Stack

Example-1: Creating Replicas

The diagram shows a Docker Stack configuration snippet with annotations. A yellow box highlights the `replicas: 3` section under the `deploy:` key for the `web_nginx` service. A bracket groups the `web_nginx` and `busybox_utility` services. An arrow points from this bracket to a terminal command.

```
1 version: "3.8"
2 services:
3   web_nginx: → Service
4     image: nginx - 1
5     ports:
6       - "8080:80"
7     deploy:
8       mode: replicated
9       replicas: 3
10  busybox_utility: → Service
11    image: radial/busyboxplus:curl
12    command: /bin/sh -c "while true; do sleep 10; done"
```

Terminal command:

```
docker service create --name web_nginx \
  -p 8080:80 \
  --replicas 3 \
  nginx
```

145

Docker Stack

Example-2: Using constraints and labels in docker compose file.

```
1 version: "3.8"
2 services:
3   web_nginx:
4     image: nginx
5     ports:
6       - "8080:80"
7     deploy:
8       mode: replicated
9       replicas: 4
10      placement:
11        constraints:
12          - "node.role!=manager"
13        preferences:
14          - spread: node.labels.datacenter
15   busybox_utility:
16     image: radial/busyboxplus:curl
17     command: /bin/sh -c "while true; do sleep 10; done"
```

```
docker node update \
  --label-add datacenter=dc_1 \
  <node name>
```

146

Docker Stack

Example-3: Resource limitations.

```
1 version: "3.8"
2 services:
3   web_nginx:
4     image: nginx
5     ports:
6       - "8080:80"
7     deploy:
8       mode: replicated
9       replicas: 4
10      placement:
11        constraints:
12          - "node.role!=manager"
13        preferences:
14          - spread: node.labels.datacenter
15      resources:
16        limits:
17          cpus: '0.50'
18          memory: 50M
19        reservations:
20          cpus: '0.25'
21          memory: 20M
22   busybox_utility:
23     image: radial/busyboxplus:curl
24     command: /bin/sh -c "while true; do sleep 10; done"
```

```
docker service update --name web_nginx \
  --limit-cpu=0.5 \
  --reserve-cpu=0.25 \
  --limit-memory=50m \
  --reserve-memory=20m \
  nginx
```

147

Docker Stack

Example-4: Using volume option.

```
1 version: "3.8"
2 services:
3   web_nginx:
4     image: nginx
5     ports:
6       - "8000:80"
7     deploy:
8       mode: replicated
9       replicas: 4
10      placement:
11        constraints:
12          - "node.role!=manager"
13        preferences:
14          - spread: node.labels.datacenter
15      resources:
16        limits:
17          cpus: '0.50'
18          memory: 50M
19        reservations:
20          cpus: '0.25'
21          memory: 20M
22
23      volumes:
24        - type: volume
25          source: myvolume
26          target: /data
27
28    busybox utility:
29      image: radial/busyboxplus:curl
30      command: /bin/sh -c "while true; do sleep 10; done"
31
32 volumes:
33   myvolume:
```

Deploy the Stack:

```
[root@Devops4Beginners ~]# docker stack deploy -c example4.yml app4
```

Inspect the service to see volume details:

```
[root@Devops4Beginners ~]# docker service inspect app4_web_nginx
```

```
"Mounts": [
  {
    "Type": "volume",
    "Source": "app4_myvolume",
    "Target": "/data",
    "VolumeOptions": {
      "Labels": {
        "com.docker.stack.namespace": "app4"
      }
    }
}
```

148

Docker Stack

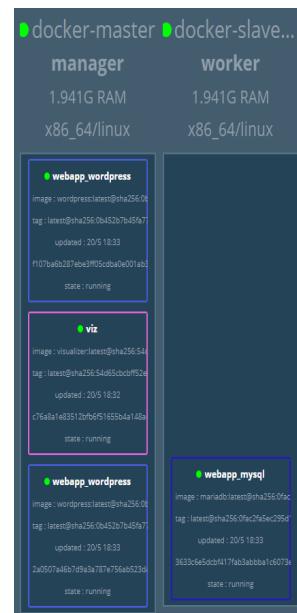
Deploy a stack `docker stack deploy -c <compose.yml> <stack-name>`

List stacks: `docker stack ls`

List processes: `docker stack ps <stack-name>`

List services: `docker service ls`

Delete stack: `docker stack rm <stack-name>`



149

Docker Stack

150

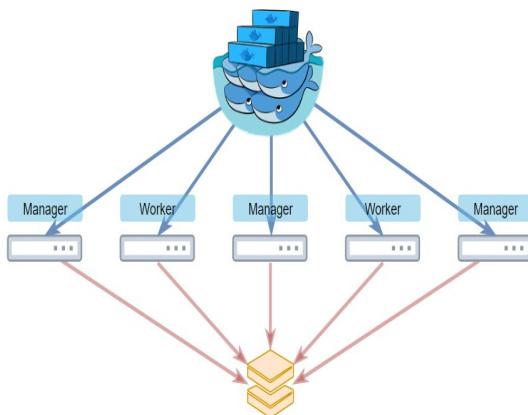
Docker Swarm : stockage

Stockage persistant

- Pour les conteneurs à hôte unique, les Bind Mounts et volumes fonctionnent bien, mais afin de partager de stockage sur plusieurs hôtes Docker, comme un cluster Swarm, nous avons besoin de distributed FS ou de Network FS.

Distributed/Network File System

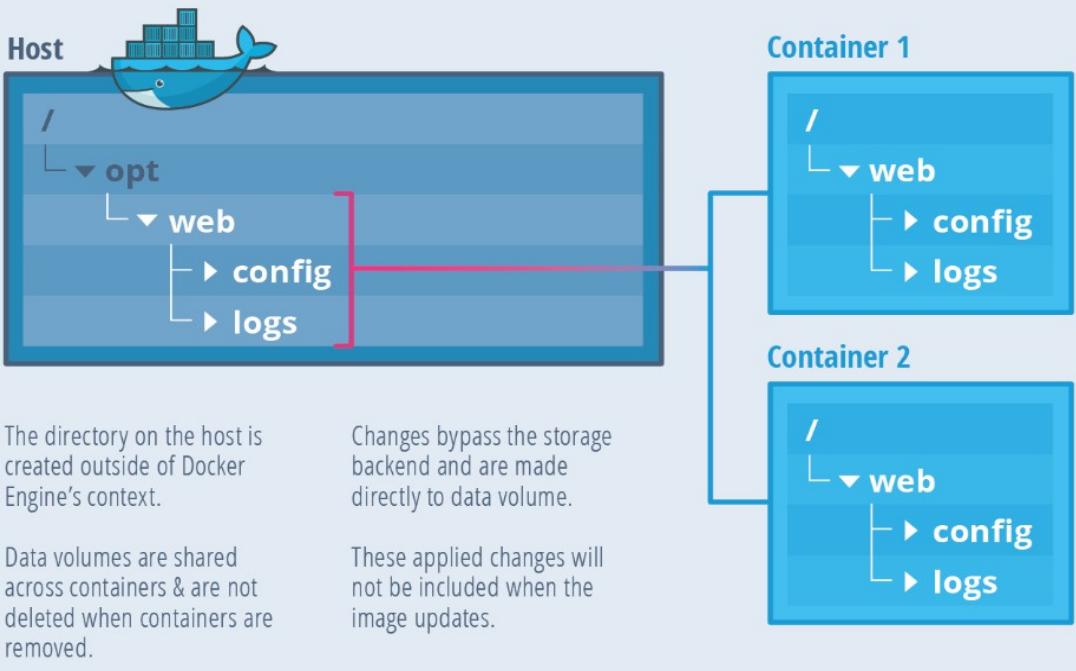
- Puisque les programmes exécutés sur votre cluster ne sont pas garantis de s'exécuter sur le nœud spécifique, les données ne peuvent pas être enregistrées dans un emplacement arbitraire.
- Si un programme tente de sauvegarder des données dans un fichier pour un usage ultérieur mais il est ensuite déplacé vers un nouveau nœud, le fichier ne se trouvera plus là où le programme l'attend.
- Distributed/Network FS permet aux applications de disposer d'un volume de stockage commun pour stocker et récupérer les données au-delà de leur cycle de vie.



151

Docker Swarm : stockage

Host-Based Persistence Shared Among Containers



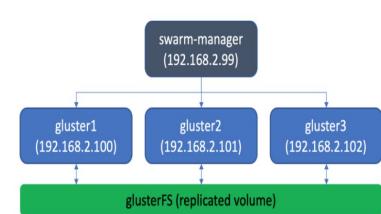
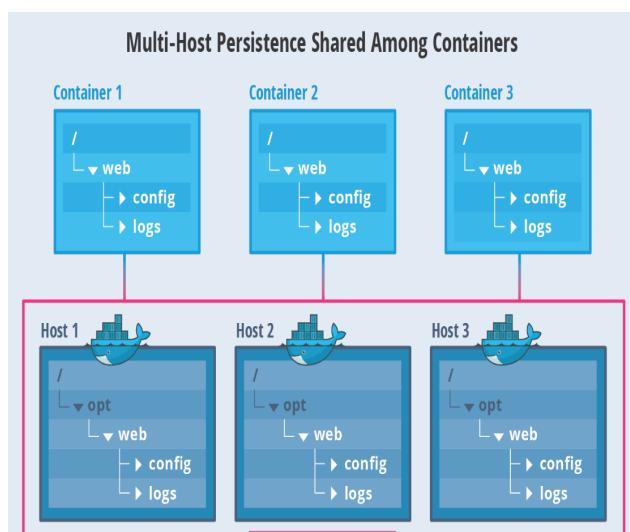
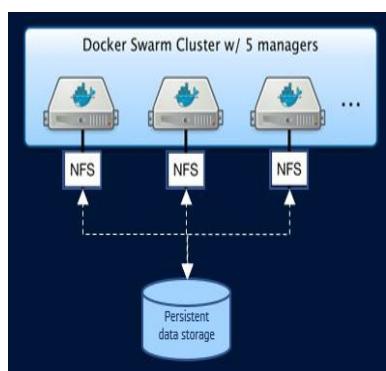
152

Docker Swarm : stockage

Persistent Storage across cluster

Requires distributed/network file storage

- NFS
- GlusterFS
- Ceph
- Convoy
- RexRay
- PortWorx
- StorageOS



GlusterFS

153