Docker

Vos applications dans des boîtes pour la sécurité, la propreté, la fiabilité, la portabilité et pleins de bonnes raisons :)

- [Docker] Installation
- [Docker] Cheat-sheet
- [Docker] Compose
- [Docker] Swarm

[Docker] Installation



Installation

Linux

```
curl -fsSL get.docker.com | bash
```

Debian

```
apt-get update && apt-get install ca-certificates curl gnupg -y && install -m 0755 -d
/etc/apt/keyrings && curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -
o /etc/apt/keyrings/docker.gpg && chmod a+r /etc/apt/keyrings/docker.gpg && echo "deb
[arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian "$(. /etc/os-release && echo "$VERSION_CODENAME")"
stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null && apt-get update && apt-get
install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y
```

Ubuntu

```
apt-get update && apt-get install ca-certificates curl gnupg && install -m 0755 -d
/etc/apt/keyrings && curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -
o /etc/apt/keyrings/docker.gpg && chmod a+r /etc/apt/keyrings/docker.gpg && echo "deb
[arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu "$(. /etc/os-release && echo "$VERSION_CODENAME")"
stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null && apt update && apt install
```

```
docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

CentOS / Alma Linux / Rocky Linux

```
yum install -y yum-utils && yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo && yum install -y docker-ce docker-ce-
cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Arch Linux / Manjaro

```
pacman -Sy docker && systemctl enable --now docker.service
```

Alpine Linux

```
apk policy docker && apk add docker && rc-update add docker boot && service docker start
```

Mettre l'utilisateur dans le groupe docker

Pour mettre l'utilisateur dans le groupe docker et ainsi pouvoir lancer des conteneurs avec d'autres utilisateurs que root :

```
usermod -aG docker <USER>
```

Puis :

```
newgrp docker
```

> Relancez votre session pour prendre en compte les changements.

[Docker] Cheat-sheet

Introduction

Cette page répertorie des commandes docker pour administrer vos conteneurs.

Commandes de base

Démarrer un conteneur

```
docker run -d -p <HOST:CT> --name <CT_NAME> <IMG>
```

Arrêter un conteneur

```
docker stop <CT_NAME>
```

Supprimer un conteneur

```
docker rm <CT_NAME>
```

> Le conteneur doit être arrêté pour le supprimer.

Redémarrer un conteneur

```
docker restart <CT_NAME>
```

Images

Télécharger une image Docker

```
docker pull <IMG>
```

Lister les images présente sur le système

```
docker images
```

Supprimer une image docker

```
docker rmi <IMG>
```

Supprimer les images inutilisées

```
docker images -f dangling=true
```

Réseaux

Création de réseau Docker

Pour créer un réseau simple sur Docker, il faut exécuter la commande suivante :

```
docker network create --driver <DRVR_TYPE> <NET_NAME>
```

Types de drivers	Description	Commande
bridge (default)	Créer un réseau Naté et isolé pour les conteneurs.	docker network create --driver bridge <NET_NAME>
host	Utilise directement le réseau de l'hôte	docker run --network host --name <CT_NAME> <IMG>
overlay	Permet aux conteneurs qui ne sont pas sur le même hôte d'appartenir à un même réseau (très utilisé avec Docker Swarm).	docker network create --driver overlay <NET_NAME>
macvlan	Donne une adresse MAC aux conteneur ce qui les fait apparaître comme de véritables appareils du réseau de l'hôte.	docker network create -d macvlan --subnet=<NET_ID/MASK> --gateway=<IP> -o parent=<IFACE> <NET_NAME>

Lister les réseaux existants

```
docker network ls
```

Afficher le détail d'un réseau

```
docker network inspect <NET_NAME>
```

Supprimer un réseau

```
docker network rm <NET_NAME>
```

Connecter un conteneur à un réseau

```
docker network connect <NET_NAME> <CT_NAME>
```

Déconnecter un conteneur d'un réseau

```
docker network disconnect <NET_NAME> <CT_NAME>
```

Créer un réseau isolé

```
docker network create --driver bridge --subnet <NET_ID/MASK> isolated_network
```

Volumes

Créer un volume

```
docker volume create <VOL_NAME>
```

Attacher un volume à un conteneur

```
docker run -d -v <VOL_NAME>:/data --name <CT_NAME> <IMG>
```

> Les volumes docker sont stockés dans **/var/lib/docker/volumes** .

Faire du bind mount

Ce type de montage spécifie l'emplacement du volume sur le système hôte :

```
docker run -d -v /root/my_app/data:/data --name <CT_NAME> <IMG>
```

Lister les volumes

```
docker volume ls
```

Afficher les détails d'un volume

```
docker volume inspect <VOL_NAME>
```

Créer un volume en lecture seule

```
docker run -d -v /root/my_app/data:/data:ro --name <CT_NAME> <IMG>
```

Supprimer un volume

```
docker volume rm <VOL_NAME>
```

Nettoyer les volumes non utilisés

```
docker volume prune
```

Copier un fichier de l'hôte sur un volume

```
docker cp /path/to/file.txt <CT_NAME>:/data/file.txt
```

Faire une sauvegarde d'un volume

```
docker run --rm --volumes-from <CT_NAME> -v $(pwd):/backup ubuntu tar cvzf
/backup/backup.tar.gz /data
```

Dockerfile

Le **Dockerfile** est un fichier permettant de créer une image docker pour votre application :

```
# Utilise l'image Python comme base
FROM python:3.8

# Définit l'espace de travail
WORKDIR /app
```

```
# Copie les fichiers de l'application dans l'espace de travail
COPY . /app

# Installe les dépendances
RUN pip install -r requirements.txt

# Expose le port 5000 du conteneur
EXPOSE 5000

# Lance le script au démarrage de l'application.
CMD ["python", "app.py"]
```

Build une image à partir d'un Dockerfile

```
docker build -t <APP_NAME:latest> .
```

Sécurité

Lancer un conteneur avec les droits d'un utilisateur non-root

```
docker run -u <UID>
```

Créer un secret

```
echo "<SECRET>" | docker secret create <SECRET_NAME> -
```

Ajouter un health check sur un conteneur

Cette option permet d'éteindre un conteneur automatique si celui-ci est en panne (pour éviter les problèmes de sécurité et les comportements inattendus) :

```
docker run --name <CT_NAME> --health-cmd="curl --fail http://localhost:80/health || exit 1" -d
<CT_IMG>
```

Débogage

Ouvrir un shell dans un conteneur

```
docker exec -it <CT_NAME> /bin/bash
```

Vérifier l'état du service

```
systemctl status docker
```

Vérifier le fonctionnement de docker

```
docker run hello-world
```

Afficher les conteneurs actifs

```
docker ps
```

> Vous pouvez ajouter l'option **-a** pour afficher tous les conteneurs.

Afficher les logs d'un conteneur

```
docker logs <CT_NAME>
```

Afficher la configuration d'un conteneur

```
docker inspect <CT_NAME>
```

Afficher les statistiques des conteneurs

```
docker stats
```

Écouter les événements docker

```
docker events
```

Afficher les changements dans le système de fichier d'un conteneur

```
docker diff <CT_NAME>
```

Mettre à niveau un conteneur

Arrêt du conteneur

```
docker container stop <CONTAINER_NAME>
```

Recherche du nom de l'ancienne image

```
docker ps -f name=<CONTAINER_NAME> --format "{{.Image}}"
```

Suppression du conteneur

```
docker container rm <CONTAINER_NAME>
```

Suppression de l'ancienne image

```
docker rmi <CONTAINER_NAME>:<CONTAINER_OLD_VERSION>
```

Redéploiement du conteneur

```
docker run -d --name <CONTAINER_NAME> <IMG_REPOS/IMG>:<NEW_VERSION>
```

[Docker] Compose

Introduction

Les **stacks** docker (ensemble de conteneurs) peuvent être gérées à travers compose ou docker-compose selon les appellations.



Installation

Vous avez deux manières d'installer docker compose, soit via le plugin (recommandé) :

```
apt install -y docker-compose-plugin
```

Ou en téléchargeant le binaire :

```
curl -L https://github.com/docker/compose/releases/download/v2.22.0/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose && chmod +x /usr/local/bin/docker-compose
```

> Selon le type d'installation, il faudra utiliser la commande **docker compose** ou lacommande **docker-compose**.

Utilisation

Fichier compose.yml

Voici un exemple de fichier **compose.yml** pour déployer un serveur web php.

```
version: '3.9'
services:
  php: php:apache
  container_name: php
  environment:
  - PUID=1000
  - GUID=1000
  - TZ=Europe/Paris
  ports:
  - '80:80'
  - '443:443'
  volumes:
  - /website:/var/www/html
```

Lancer la stack

```
docker compose up -d
```

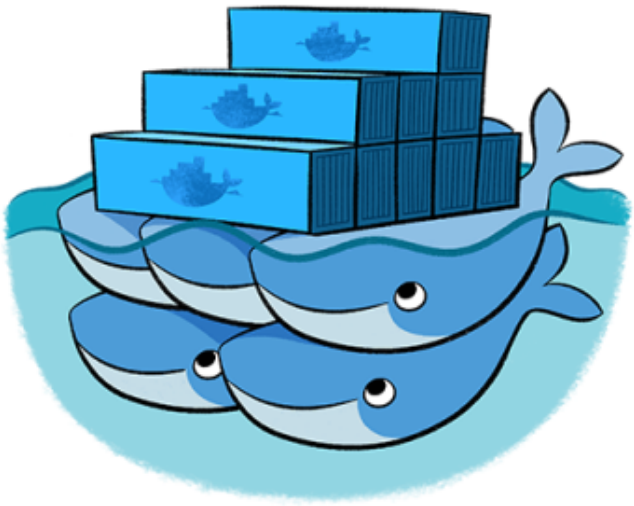
[Docker] Swarm

Introduction

Docker Swarm permet de faire de la répartition de charge et de la HA dans vos infrastructures Docker.

Pour cela, on va mettre en place un cluster de noeuds avec des managers et des workers (masters/slaves).





Prérequis

- Avoir Docker installé sur tous les noeuds du cluster

Installation

Tout d'abord, rendez-vous sur votre **Manager** pour créer le cluster :

```
docker swarm init --advertise-addr <manager-IP>
```

Ensuite, il faut vous connecter sur chaque **Workers** pour les faire rejoindre votre cluster :

```
docker swarm join --token <TOKEN>
```

Le token a été affiché lors de la création du cluster sur le Manager sinon vous pouvez l'afficher avec cette commande :

```
docker swarm join-token worker
```

Et pour faire rejoindre des managers :

```
docker swarm join-token manager
```

Manuel

Afficher les membres du cluster

```
docker node ls
```

Déployer un conteneur dans le cluster

```
docker service create --name webserver -p 8080:80 nginx
```

Déployer une stack dans le cluster

```
docker stack deploy --compose-file compose.yml <STACK_NAME>
```

Afficher les services

```
docker service ls
```

Scale un service

```
docker service scale <SERVICE_NAME>=3
```

Supprimer une stack

```
docker stack rm <STACK_NAME>
```

Sortir un noeud du cluster

Sur le noeud :

```
docker swarm leave
```

> S'il s'agit du dernier noeud Manager du cluster, spécifier **--force** pour supprimer le cluster.

Et depuis le Manager :

```
docker node rm <NODE_ID>
```