

# Installation & Configuration

- [Installation Docker Engine & Docker Compose](#)
- [Création d'un Cluster Docker Swarm](#)
- [Commandes Docker](#)

# Installation Docker Engine & Docker Compose

## Prérequis

Avant de commencer, assurez-vous que votre système répond aux exigences suivantes :

1. **Système d'exploitation**: Une VM Debian.
2. **Droits d'administrateur**: Vous devez avoir accès à un compte utilisateur avec des droits `sudo` pour exécuter des commandes d'installation et de configuration.
3. **Connexion Internet**: Pour télécharger Docker et ses dépendances.
4. **Espace disque et mémoire**: Assurez vous d'avoir suffisamment d'espace disque et de mémoire pour Docker et les conteneurs que vous prévoyez d'exécuter.

## Installation de Docker Engine sur Debian

### Étape 1: Mise à jour du système

Commencez par mettre à jour votre liste de paquets et mettez à niveau les paquets existants pour vous assurer que votre système est à jour.

```
sudo apt-get update  
sudo apt-get upgrade -y
```

### Étape 2: Installer les paquets nécessaires

Avant d'installer Docker Engine pour la première fois sur une nouvelle machine hôte, vous devez configurer le dépôt **apt de Docker**. Ensuite, vous pourrez installer et mettre à jour Docker à partir de ce dépôt.

```
# Add Docker's official GPG key:
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/debian \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

## Étape 3: Installation de docker

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

*Pour vérifier son fonctionnement :*

```
sudo docker run hello-world
```

*Pour désinstaller docker :*

```
sudo apt-get purge docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin docker-ce-
rootless-extras
```

Si l'installation s'est bien déroulée, vous verrez un message indiquant que votre installation de Docker fonctionne correctement.

## Post-installation (facultatif mais recommandé)

### Configurer Docker pour l'utiliser sans sudo (facultatif)

Par défaut, l'exécution des commandes Docker nécessite des privilèges administratifs (sudo).

Pour éviter d'avoir à taper `sudo` chaque fois que vous utilisez la commande `docker`, ajoutez votre utilisateur au groupe Docker.

```
sudo usermod -aG docker $USER
```

Après avoir exécuté cette commande, vous devrez vous déconnecter et vous reconnecter pour que les modifications prennent effet.

## Démarrer Docker au démarrage

Pour vous assurer que Docker démarre automatiquement à chaque démarrage du système :

```
sudo systemctl enable docker
```

Source : [Docker Official](#)

## Descriptif

Ces commandes, exécutées dans l'ordre, préparent un système Debian pour une installation sécurisée et vérifiée de Docker Engine directement depuis les sources officielles de Docker, assurant ainsi que vous obtenez la version la plus récente et authentique de Docker.

### Ajout de la clé GPG officielle de Docker

`sudo apt-get update` : Cette commande met à jour l'index des paquets disponibles et leurs versions, mais sans installer ou mettre à jour aucun paquet. Cela assure que vous avez les dernières informations sur les paquets disponibles avant d'installer des dépendances.

`sudo apt-get install ca-certificates curl` : Installe `ca-certificates` et `curl` sur votre système.

`ca-certificates` : Ce paquet est nécessaire pour que votre système puisse vérifier les certificats SSL des sites web (ce qui est crucial pour des téléchargements sécurisés).

`curl` : Un outil en ligne de commande utilisé pour transférer des données avec des URL. Ici, il sert à télécharger la clé GPG de Docker depuis Internet.

`sudo install -m 0755 -d /etc/apt/keyrings` : Crée le répertoire `/etc/apt/keyrings` s'il n'existe pas déjà, avec des permissions telles que le propriétaire (root) peut lire, écrire et exécuter, tandis que les membres du groupe et les autres utilisateurs peuvent seulement lire et exécuter. Ce répertoire est utilisé pour stocker les clés GPG de confiance pour APT.

`sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc` : Télécharge la clé GPG officielle de Docker et la sauvegarde dans le fichier `/etc/apt/keyrings/docker.asc`. L'option `-fsSL` rend `curl` silencieux sauf en cas d'erreur, suit les redirections, et permet le protocole SSL.

`sudo chmod a+r /etc/apt/keyrings/docker.asc` : Modifie les permissions du fichier de clé GPG pour permettre à tous les utilisateurs de le lire. Cela est nécessaire pour que APT puisse vérifier les paquets signés par cette clé.

## Ajout du dépôt Docker aux sources APT

La commande `echo` suivie de `sudo tee /etc/apt/sources.list.d/docker.list > /dev/null` : Cette commande complexe ajoute la ligne de dépôt officiel de Docker aux sources APT du système.

`echo` génère la ligne de texte qui définit le dépôt.

Le dépôt est configuré pour utiliser l'architecture système (`arch=$(dpkg --print-architecture)`), signé par la clé précédemment téléchargée (`signed-by=/etc/apt/keyrings/docker.asc`), pointe vers le dépôt Docker (`https://download.docker.com/linux/debian`), et spécifie la version du système d'exploitation (`$(. /etc/os-release && echo "$VERSION_CODENAME")`) et le canal `stable`.

`sudo tee /etc/apt/sources.list.d/docker.list > /dev/null` écrit la sortie de `echo` dans le fichier `/etc/apt/sources.list.d/docker.list` (créant ou écrasant le fichier si nécessaire), avec les droits d'administration. L'envoi de la sortie vers `/dev/null` empêche l'affichage de la ligne dans la console.

`sudo apt-get update` : Exécute de nouveau `apt-get update` pour que APT prenne en compte le nouveau dépôt ajouté. Après cette commande, le système est prêt à installer Docker Engine depuis le dépôt officiel ajouté.

## Installation de Docker & Docker Compose

`docker-ce` : "Docker Community Edition" est le logiciel de base de Docker qui permet de créer, déployer et gérer des conteneurs. C'est le cœur de Docker, fournissant l'environnement d'exécution des conteneurs.

`docker-ce-cli` : "Docker Community Edition Command Line Interface" est l'interface en ligne de commande pour Docker. Elle permet aux utilisateurs d'interagir avec Docker et de gérer les conteneurs Docker à travers une variété de commandes.

`containerd.io` : Containerd est un environnement d'exécution de conteneurs disponible sous forme de daemon pour Linux et Windows. Il s'agit d'un composant de bas niveau pour la gestion du cycle

de vie des conteneurs, responsable de l'exécution des conteneurs, de la gestion de leur stockage d'images, de la mise en réseau, et plus encore. Docker utilise Containerd sous le capot.

[docker-buildx-plugin](#) : Docker Buildx est un plugin Docker qui fournit des fonctionnalités de build avec BuildKit, permettant la construction de l'image Docker avec un support amélioré pour la multi-architecture, le caching, et plus encore. Cela permet de créer des images Docker de manière plus efficace et avancée, y compris la possibilité de créer simultanément des images pour différentes architectures à partir d'un seul poste de commande.

[docker-compose-plugin](#) : Docker Compose est un outil pour définir et gérer des applications multi-conteneurs avec Docker. Le plugin [docker-compose-plugin](#) intègre Docker Compose directement dans l'écosystème Docker, permettant aux utilisateurs de définir et d'exécuter des applications multi-conteneurs à l'aide de fichiers YAML.

Cela simplifie le processus de configuration, d'exécution, et de gestion des applications conteneurisées en utilisant une syntaxe déclarative.

# Création d'un Cluster Docker Swarm

Créer un cluster Docker Swarm est un processus direct qui peut être accompli en quelques étapes.

Ce tutoriel vous guidera à travers la création d'un cluster Swarm, depuis les prérequis jusqu'à la promotion de nœuds en tant que managers ou workers, en passant par la configuration initiale.

## Prérequis

1. **Machines Virtuelles (VMs) ou Serveurs Physiques** : Vous aurez besoin d'au moins une machine pour agir en tant que manager du Swarm et de deux autres machines comme nœuds workers. Trois machines sont un bon point de départ pour un environnement de production minimal pour la tolérance aux pannes. Ces machines doivent être configurées avec Docker.
2. **Docker Installé** : Chaque machine doit avoir Docker installé. Vous pouvez trouver les instructions d'installation sur le site officiel de Docker pour diverses distributions Linux, Windows, et macOS.
3. **Réseau** : Les machines doivent être capables de communiquer entre elles à travers les ports utilisés par Docker Swarm (TCP port 2377 pour la communication entre les managers, et TCP et UDP ports 7946, UDP port 4789 pour le trafic overlay network).

## Étape 1 : Initialisation du Swarm

Choisissez une de vos machines pour être le manager initial du Swarm. Connectez-vous à cette machine, puis exécutez :

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

Remplacez `<MANAGER-IP>` par l'adresse IP de la machine que vous souhaitez utiliser comme manager. Cette commande initialise un nouveau Swarm et configure la machine actuelle en tant que manager du Swarm.

## Étape 2 : Ajout de Nœuds au Swarm

Après avoir initialisé le Swarm, vous obtiendrez un message contenant un token pour joindre d'autres nœuds au Swarm en tant que workers.

Sur chaque machine destinée à être un worker, exécutez la commande fournie, qui ressemblera à ceci :

```
docker swarm join --token SWMTKN-1-xxxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx <MANAGER-IP>:2377
```

- Remplacez `<MANAGER-IP>` par l'adresse IP du manager Swarm. Cette commande permet à la machine d'agir comme un nœud worker dans le Swarm.

## Étape 3 : Vérification de l'État du Swarm

Pour vérifier que les nœuds ont bien rejoint le Swarm, exécutez la commande suivante sur le manager :

```
docker node ls
```

Cela affichera la liste de tous les nœuds dans le Swarm, y compris leur rôle (manager ou worker).

### Exemple

```
root@EVA-00:~# docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
1p14w09a3a1ysibjegktesxwr *	EVA-00	Ready	Active	Reachable	20.10.24+dfsg1
7tnbhyvlbmuv27dnk8d3oqf9b	EVA-01	Ready	Active	Leader	20.10.24+dfsg1
5mxg7b4mnkjd3z8vcwo9ygsdy	EVA-02	Ready	Active		20.10.24+dfsg1

## Étape 4 : Promotion d'un Nœud Worker en Manager

Si vous souhaitez augmenter la tolérance aux pannes de votre cluster Swarm en ajoutant un manager supplémentaire, vous pouvez promouvoir un nœud worker. Sur un nœud manager, exécutez :

```
docker node promote <WORKER-NODE-ID>
```

Remplacez `<WORKER-NODE-ID>` par l'ID du nœud que vous souhaitez promouvoir, que vous pouvez trouver en exécutant `docker node ls`.



Source :

[Docker Swarm Official](#)

## Conclusion

Vous avez maintenant un cluster Docker Swarm fonctionnel composé d'un manager et de plusieurs nœuds workers. Vous pouvez déployer des applications sur ce cluster en utilisant des services Docker Swarm, ce qui permet de gérer facilement la disponibilité, le scaling, et les mises à jour de vos applications conteneurisées.

Ce tutoriel couvre les bases, mais Docker Swarm offre de nombreuses autres fonctionnalités et options de configuration pour optimiser et sécuriser votre cluster.

# Commandes Docker

Voici une liste des commandes Docker les plus couramment utilisées et leurs fonctions.

Notez que cette liste n'est pas exhaustive, car Docker offre un large éventail de commandes pour diverses opérations, mais cela couvrira les plus fondamentales.

## Gestion des Conteneurs

- **docker run** : Lance un conteneur à partir d'une image spécifiée. Vous pouvez utiliser diverses options avec cette commande pour contrôler le comportement du conteneur (comme **-d** pour le mode détaché, **-p** pour la publication des ports, etc.).
- **docker ps** : Liste tous les conteneurs en cours d'exécution. Utilisez l'option **-a** pour voir tous les conteneurs, y compris ceux qui sont arrêtés.
- **docker stop** : Arrête un ou plusieurs conteneurs en cours d'exécution.
- **docker start** : Démarre un ou plusieurs conteneurs arrêtés.
- **docker restart** : Redémarre un ou plusieurs conteneurs.
- **docker rm** : Supprime un ou plusieurs conteneurs.
- **docker create** : Crée un nouveau conteneur mais ne le démarre pas immédiatement.
- **docker exec** : Exécute une commande dans un conteneur en cours d'exécution.

- **docker logs** : Affiche les logs d'un conteneur.
- **docker pause** / **docker unpause** : Met en pause les processus dans un conteneur, et les reprend ensuite.

## Gestion des Images

- **docker images** ou **docker image ls** : Liste les images Docker disponibles localement.
- **docker pull** : Télécharge une image ou un dépôt d'images depuis un registre.
- **docker push** : Envoie une image ou un dépôt vers un registre.
- **docker build** : Crée une image à partir d'un Dockerfile.
- **docker rmi** : Supprime une ou plusieurs images Docker.
- **docker history** : Montre l'historique des modifications d'une image.

## Réseau et Stockage

- **docker network create** / **docker network rm** : Crée ou supprime des réseaux.
- **docker network ls** : Liste les réseaux Docker.

- `docker volume create` / `docker volume rm` : Crée ou supprime des volumes pour la persistance des données.
- `docker volume ls` : Liste les volumes Docker.

## Docker Compose (outil distinct)

- `docker-compose up` : Démarre et exécute l'ensemble d'une application définie par un fichier `docker-compose.yml`.
- `docker-compose down` : Arrête et supprime les ressources spécifiées dans le fichier `docker-compose.yml`.

## Informations et Configuration

- `docker version` : Affiche la version de Docker qui est installée.
- `docker info` : Affiche des informations sur le système Docker, y compris le nombre de conteneurs et d'images, les informations sur le réseau et le stockage.
- `docker login` : Permet de s'authentifier sur un registre Docker.
- `docker system prune` : Supprime les conteneurs, les réseaux, les images (non utilisées par au moins un conteneur), et le cache de build inutilisés.

# Gestion des Stacks

- **docker stack ls** : Liste toutes les stacks déployées dans le Swarm. Une stack est un groupe de services qui sont liés entre eux et qui partagent des dépendances, définis dans un fichier `docker-compose.yml` ou équivalent.
- **docker stack deploy** : Déploie une nouvelle stack ou met à jour une stack existante. C'est la commande utilisée pour lancer une application conteneurisée sur un Swarm en utilisant un fichier de composition Docker Compose. Elle prend souvent l'option `-c` pour spécifier le fichier de composition.
- **docker stack rm** : Supprime une stack du Swarm. Cela arrête et supprime tous les services associés à la stack, mais conserve les volumes de données, à moins que des options spécifiques ne soient utilisées pour les supprimer également.

## Gestion des Services dans une Stack

*Ces commandes offrent une interface de haut niveau pour la gestion des applications distribuées sur un cluster Docker Swarm, simplifiant le déploiement et la maintenance d'applications conteneurisées à grande échelle.*

*Elles sont particulièrement utiles pour les opérations de déploiement continu et pour la gestion de l'infrastructure en tant que code, permettant aux développeurs et aux administrateurs système de décrire l'état souhaité de leur application dans des fichiers de composition et de laisser Docker s'occuper du déploiement et de la gestion.*

- **docker stack services** : Affiche les services dans une stack. Cette commande fournit une vue d'ensemble des services qui composent la stack, y compris leur état, le nombre de répliques, et les ports exposés.
- **docker stack ps** : Liste les tâches (containers) d'une stack. Cela permet de voir le détail de chaque conteneur lancé par les services de la stack, y compris leur état et sur quel nœud

du Swarm ils sont déployés.

## Inspection et Configuration

- **`docker stack inspect`** : Affiche des informations détaillées sur une stack, en format JSON. Cela inclut la configuration des services, les réseaux utilisés, et d'autres paramètres.

## Gestion d'un cluster Swarm

Les commandes `docker swarm` sont utilisées pour gérer et orchestrer un cluster Docker Swarm, qui est un groupe de machines Docker (nœuds) fonctionnant ensemble dans un mode de cluster.

Docker Swarm fournit des fonctionnalités d'orchestration de conteneurs natives, permettant de déployer, mettre à l'échelle et gérer des applications conteneurisées sur plusieurs hôtes Docker.

## Initialisation et Gestion du Swarm

- **`docker swarm init`** : Initialise un nouveau cluster Swarm sur l'hôte. Cette commande transforme la machine Docker sur laquelle elle est exécutée en un manager Swarm, ce qui lui permet de gérer et d'orchestrer le Swarm. Lors de l'initialisation, vous pouvez spécifier divers paramètres, comme l'adresse IP à utiliser pour la communication inter-nœuds.
- **`docker swarm join`** : Rejoint une machine au Swarm, soit en tant que nœud worker, soit en tant que manager supplémentaire, en fonction des tokens et des options spécifiés. Cette commande nécessite l'adresse d'un nœud manager existant et un token d'adhésion approprié.
- **`docker swarm leave`** : Fait quitter le Swarm à un nœud, le retirant ainsi de l'ensemble du cluster. Peut être utilisée sur des workers comme sur des managers, avec des considérations spéciales pour assurer que le Swarm continue de fonctionner correctement sans ce nœud.

- **docker swarm update** : Met à jour la configuration du Swarm. Cette commande permet de modifier des paramètres du Swarm, tels que la politique de certificat ou les paramètres de dispatcher.

## Gestion et Sécurité des Nœuds

- **docker node ls** : Liste tous les nœuds dans le Swarm. Fournit des informations telles que l'ID, le nom, le rôle (manager ou worker), l'état, et la disponibilité.
- **docker node inspect** : Affiche des informations détaillées sur un ou plusieurs nœuds en format JSON. Utile pour obtenir des informations de configuration spécifiques ou l'état actuel d'un nœud.
- **docker node rm** : Supprime un nœud du Swarm. Il est important de noter que cette commande ne fait pas quitter le Swarm au nœud lui-même; elle doit être utilisée pour enlever un nœud qui a déjà quitté le Swarm ou qui est en panne.
- **docker node update** : Met à jour les paramètres d'un nœud. Cela peut inclure le changement de rôle d'un nœud (de worker à manager ou inversement) ou la modification de sa disponibilité.

## Gestion des Tokens du Swarm

- **docker swarm join-token** : Gère les tokens utilisés pour rejoindre le Swarm. Vous pouvez créer de nouveaux tokens ou afficher les tokens existants pour les rôles de manager ou de worker, facilitant ainsi l'ajout sécurisé de nouveaux nœuds au cluster.

## Sécurité et Configuration

- **docker swarm ca** : Affiche et met à jour le certificat d'autorité de certification (CA) du Swarm. Permet de gérer la façon dont les certificats sont émis et renouvelés au sein du Swarm pour sécuriser la communication inter-nœuds.

## Promouvoir un Nœud Worker en Manager

- **`docker node promote`** : Cette commande est utilisée pour promouvoir un ou plusieurs nœuds workers à un rôle de manager dans le Swarm. Promouvoir un nœud en manager augmente le nombre de nœuds capables de gérer le cluster, ce qui peut améliorer la tolérance aux pannes et la disponibilité du cluster. La commande nécessite l'ID ou le nom du nœud (ou des nœuds) à promouvoir.

## Rétrograder un Nœud Manager en Worker

- **`docker node demote`** : Utilisez cette commande pour rétrograder un ou plusieurs nœuds managers à un rôle de worker. Rétrograder un manager en worker peut être nécessaire pour réduire le nombre de managers et simplifier la gestion du cluster, ou lorsqu'un nœud manager n'est plus nécessaire ou ne doit plus avoir le rôle de gestion. Tout comme pour la promotion, cette commande prend l'ID ou le nom du nœud (ou des nœuds) à rétrograder.

### *Considérations Importantes*

**Équilibrage des Rôles** : Lors de la promotion ou de la rétrogradation des nœuds, il est crucial de maintenir un équilibre entre le nombre de managers et de workers dans votre Swarm. Un nombre optimal de managers assure une bonne tolérance aux pannes sans introduire de complexité ou de latence de gestion inutiles.

**Sécurité et Fiabilité** : Les managers maintiennent l'état du cluster et prennent des décisions de gestion importantes. Il est donc essentiel de s'assurer que les nœuds promus en managers sont sécurisés et fiables.