

*Antony Kervazo-Canut*

# Docker for Teenagers

**INTRODUCTION AUX  
CONTENEURS**



# SOMMAIRE

---

Introduction	3
Installation de Docker	4
Vérifications	5
Exécution d'un conteneur	6
Gestion des conteneurs	7
Gestion des images	8
DockerFile	9
Construction d'une image	10
Docker Hub	11
Gestion du réseau	12
Volumes	14
Bind Mount	15
Gestion des volumes	16
Sécurité	17
Maintenance & Surveillance	18
Monitoring & Logging	19
Navigation & Modifications	20
Docker Compose	21
Gestion de Docker Compose	22

# Introduction



Docker est une plateforme de conteneurisation qui permet de créer, déployer et gérer des applications virtualisées dans des conteneurs. Les conteneurs sont isolés les uns des autres et du système hôte, mais partagent le même système d'exploitation du noyau.

- **Portabilité:** Une fois qu'une application est dockerisée, elle peut être exécutée sur n'importe quel système qui supporte Docker sans modification.
- **Isolation:** Les applications fonctionnent dans des environnements isolés, ce qui réduit les conflits entre les applications.
- **Économie de ressources:** Les conteneurs partagent des ressources système, ce qui les rend plus efficaces en termes d'utilisation de la RAM et du CPU.
- **Simplicité et rapidité de déploiement:** Les conteneurs peuvent être créés, démarrés, arrêtés, et détruits en quelques secondes, permettant des déploiements rapides et faciles à gérer.

Docker utilise des conteneurs pour encapsuler les dépendances et les logiciels nécessaires pour exécuter une application, ce qui assure que celle-ci fonctionne de manière uniforme dans divers environnements de développement, de test et de production.

- **Images Docker:** Un modèle immuable contenant l'application et toutes ses dépendances. Les images sont utilisées pour créer des conteneurs.
- **Conteneurs Docker:** Instances exécutables d'images qui isolent l'application du système sous-jacent. Les conteneurs peuvent être ajoutés ou supprimés.
- **Dockerfile:** Un script de configuration utilisé pour automatiser le processus de création d'une image Docker.

# Installation de Docker



Docker tourne sur les systèmes Unix. Pour le faire tourner sur Windows il faudra donc passer par l'installation de WSL (Windows Subsystem for Linux).

<https://learn.microsoft.com/en-us/windows/wsl/install>

Une fois installé, vous pourrez alors utiliser Docker Desktop sur Windows.

Pour MacOS et Linux le processus est légèrement différent. A noter que sur MacOS, Docker Desktop sera installé par la commande et devra être lancé pour exécuter les conteneurs et les commandes.

```
# Sur MacOS, installer Docker via Homebrew
# Cette commande installe Docker, incluant Docker Engine,
# Docker CLI client, Docker Compose, Docker Desktop, et
# Kubernetes.
brew install --cask docker

# Sur Linux, installer Docker Engine
# Cette mise à jour de la liste des packages suivie de
# l'installation installe Docker Engine et ses composants.
sudo apt update && sudo apt install docker-ce docker-ce-cli
containerd.io
```

# Vérifications



Après avoir installé Docker sur votre système, qu'il soit sous Windows, macOS ou Linux, il est crucial de vérifier que tout fonctionne correctement et de procéder à quelques configurations initiales pour optimiser votre expérience avec Docker.

Pour les utilisateurs de MacOS et Windows, Docker Desktop doit indiquer que Docker est en cours d'exécution via son icône dans la barre de tâches ou la barre de menu.

```
● ● ●  
  
# Vérifier que le service Docker est actif  
# Cette commande est spécifique à Linux et affiche l'état du  
service Docker.  
systemctl status docker  
  
# Exécuter un conteneur de test pour vérifier le  
fonctionnement  
# Cette commande télécharge une image de test et exécute un  
conteneur qui imprime un message de confirmation.  
docker run hello-world
```

# Exécution d'un conteneur



Exécuter un conteneur Docker pour la première fois est une étape cruciale pour comprendre comment Docker encapsule les applications dans des environnements isolés.

```
● ● ●

# Télécharger l'image Nginx depuis Docker Hub
# Cette commande récupère la dernière version de l'image Nginx
# et la stocke localement.
docker pull nginx

# Exécuter un conteneur Nginx en arrière-plan (-d: détaché)
# -p 8080:80 : Mappe le port 80 du conteneur au port 8080 de
# l'hôte
# --name mon-serveur-web : Attribue le nom "mon-serveur-web" au
# conteneur pour une identification facile
docker run -d -p 8080:80 --name mon-serveur-web nginx

# Lister tous les conteneurs actifs
docker ps

# Cette commande arrête le conteneur nommé "mon-serveur-web".
docker stop mon-serveur-web

# Cette commande supprime le conteneur arrêté pour libérer des
# ressources.
docker rm mon-serveur-web
```

# Gestion des conteneurs



Après avoir lancé votre premier conteneur, il est essentiel de savoir comment gérer les conteneurs actifs, arrêtés, ou ceux nécessitant des modifications.

```
● ● ●

# Lister tous les conteneurs, actifs et inactifs
# L'option -a signifie "all", qui inclut les conteneurs
# inactifs dans la liste.
docker ps -a

# Redémarrer un conteneur
# Cette commande est utile pour appliquer des modifications ou
# des mises à jour.
docker restart mon-serveur-web

# Afficher les logs d'un conteneur
docker logs mon-serveur-web

# Cette commande fournit un JSON complet avec toutes les
# configurations et les états du conteneur spécifié.
docker inspect mon-serveur-web
```

Ces commandes constituent la base de la gestion des conteneurs avec Docker. En les maîtrisant, vous pourrez efficacement contrôler et maintenir vos environnements Docker.

# Gestion des images



Docker Hub est le registre public officiel de Docker où des milliers d'images sont disponibles pour être utilisées par la communauté. Ces images peuvent être des applications, des versions de système d'exploitation, ou des services configurés et prêts à l'emploi.

```
# Rechercher une image sur Docker Hub, ici nginx
docker search nginx

# Télécharger une image depuis Docker Hub
docker pull nginx

# Lister toutes les images Docker stockées localement
# Cette commande affiche une liste des images, leurs tags, la
# date de création, et la taille.
docker images

# Supprimer une image Docker
docker rmi nginx

# Inspecter une image Docker
# Cela retourne un JSON détaillant la structure et la
# configuration de l'image spécifiée.
docker inspect nginx
```

# DockerFile



Un Dockerfile est un script de configuration utilisé par Docker pour automatiser le processus de création d'images. En définissant une série d'instructions dans un Dockerfile, vous pouvez créer des images qui incluent votre application, ses dépendances, et toute configuration système nécessaire.

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

# Définit le répertoire de travail dans le conteneur
WORKDIR /app

# Copie les fichiers de l'application dans le conteneur
COPY . /app

# Installe les dépendances de l'application
RUN pip install -r requirements.txt

# Expose le port 5000 pour permettre l'accès à l'application
EXPOSE 5000

# Commande exécutée au démarrage du conteneur
CMD ["python", "app.py"]
```

Un Dockerfile typique commence par une instruction FROM, qui définit l'image de base. Ensuite, il contient diverses instructions pour copier les fichiers, installer les logiciels, exposer les ports, etc.

Astuce : la commande "docker init" s'adapte à votre projet pour vous construire un Dockerfile optimisé.

# Construction d'une image



Pour construire une image à partir de votre Dockerfile, placez-le dans le répertoire de votre projet. Après la construction, vous pouvez exécuter un conteneur basé sur votre nouvelle image pour tester si l'application fonctionne comme prévu.

```
# Construire une image Docker à partir d'un Dockerfile
# -t mon-app:latest : Tag l'image construite avec le nom 'mon-
# app' et le tag 'latest'.
# . : Indique le contexte de construction à Docker, ici le
# répertoire courant.
docker build -t mon-app:latest .

# Exécuter un conteneur à partir de l'image construite
# docker run -d -p 5000:5000 mon-app:latest
# -p 5000:5000 : Mappe le port 5000 du conteneur au port 5000
# de l'hôte
docker run -d -p 5000:5000 mon-app:latest
```

Visitez <http://localhost:5000> dans votre navigateur pour voir si l'application s'exécute correctement.

# Docker Hub



Une fois que vous avez créé une image Docker personnalisée qui fonctionne bien localement, vous pouvez la partager avec d'autres en la publiant sur Docker Hub. Docker Hub est une plateforme de partage d'images Docker qui permet aux utilisateurs de télécharger et de partager des images de manière publique ou privée.

```
● ● ●

# Se connecter à Docker Hub
# Suivez les instructions à l'écran pour entrer votre nom
# d'utilisateur et votre mot de passe de Docker Hub.
docker login

# Taguer l'image
# Le tag doit suivre le format username/repository:tag
# username correspond au nom d'utilisateur de dockerhub
# Remplacez 'image_locale' par le nom de votre image locale et
# 'latest' par le tag que vous souhaitez utiliser.
docker tag mon-app:latest monuser/monapp:latest

# Pousser l'image sur Docker Hub
# Assurez-vous que le nom d'utilisateur et le repository
# correspondent à ce que vous avez utilisé lors du tag.
docker push monuser/monapp:latest
```

Utilisez des tags spécifiques pour différentes versions de votre application au lieu de toujours utiliser latest, pour permettre aux utilisateurs de choisir une version stable.



# Gestion du réseau

Docker utilise des réseaux virtuels pour connecter les conteneurs entre eux et avec le monde extérieur tout en les isolant de manière sécurisée. La gestion de ces réseaux est essentielle pour assurer la communication correcte entre vos conteneurs et les autres services.

```
● ● ●

# Bridge : Le réseau par défaut pour les conteneurs Docker. Si
vous ne spécifiez pas un réseau, Docker connecte le conteneur
au réseau bridge par défaut.
# Utilisation : Idéal pour la communication entre les
conteneurs sur le même hôte.
docker network create --driver bridge mon_reseau_bridge

# Host : Supprime l'isolation entre les conteneurs et le
système hôte, en utilisant directement le réseau de l'hôte.
# Utile lorsque vous avez besoin de performances réseau
maximales et que l'isolation n'est pas une priorité.
docker run --network host --name mon_conteneur nginx

# Overlay : Permet aux conteneurs de différents hôtes de
communiquer comme s'ils étaient sur le même réseau.
# Utilisé dans les environnements de Docker Swarm pour
connecter des conteneurs à différents hôtes.
docker network create --driver overlay mon_reseau_overlay

# Macvlan : Permet aux conteneurs d'avoir une adresse MAC
unique, les faisant apparaître comme des périphériques
physiques sur le réseau.
# Utile dans des cas où les conteneurs doivent être considérés
comme des machines physiques séparées.
docker network create -d macvlan --subnet=192.168.1.0/24 --
gateway=192.168.1.1 -o parent=eth0 mon_reseau_macvlan
```



# Gestion du réseau

Pour compléter et enrichir la section sur la gestion du réseau dans Docker, explorons d'autres commandes essentielles qui peuvent être utiles pour la configuration et le diagnostic des réseaux de conteneurs.

```
● ● ●

# Lister tous les réseaux Docker
docker network ls

# Inspecter un réseau Docker
docker network inspect mon_reseau_bridge

# Supprimer un réseau Docker
docker network rm mon_reseau_bridge

# Connecter un conteneur existant à un réseau
docker network connect mon_reseau_bridge mon_conteneur

# Déconnecter un conteneur d'un réseau
docker network disconnect mon_reseau_bridge mon_conteneur

# Connecter un conteneur à un réseau avec un alias
# Cette option permet au conteneur d'être identifié avec un
# alias dans le réseau spécifié.
docker network connect --alias mon_alias mon_reseau_bridge
mon_conteneur
```

# Volumes



La persistance des données est cruciale pour de nombreuses applications, surtout celles qui stockent des états ou des informations utilisateurs, comme les bases de données. Docker gère le stockage persistant principalement à travers des volumes et des bind mounts.

Les volumes sont stockés dans une partie du système de fichiers gérée par Docker (/var/lib/docker/volumes sur Linux). Ils sont complètement gérés par Docker et sont isolés de l'hôte, ce qui les rend sûrs et flexibles. Ils sont idéal pour la production ou lorsque des données critiques doivent être conservées indépendamment du cycle de vie du conteneur.

```
# Créer un volume Docker
# Cette commande crée un volume nommé 'mon_volume' qui peut
# être attaché à des conteneurs.
docker volume create mon_volume

# Attacher le volume à un conteneur
# Le volume 'mon_volume' est monté dans le conteneur sous
# '/data'.
docker run -d -v mon_volume:/data --name mon_conteneur nginx
```

# Bind Mount



Un bind mount est un mappage d'un dossier ou fichier existant sur l'hôte dans un conteneur. Ce type de montage est dépendant du système de fichiers de l'hôte.

Utile pour le développement ou les situations où vous devez rapidement lier des données ou des configurations spécifiques à l'hôte dans un conteneur.



```
# Créer un bind mount entre l'hôte et un conteneur
# Mappe un chemin de l'hôte dans un conteneur. Les
# modifications sont réfléchies des deux côtés.
docker run -d -v /chemin/sur/hote:/chemin/dans/conteneur --name
mon_conteneur nginx
```

# Gestion des volumes



Pour fournir une vue plus complète et pratique de la gestion des volumes dans Docker, explorons davantage de commandes utiles qui peuvent être employées pour optimiser et sécuriser la gestion du stockage persistant.

```
● ● ●

# Montre tous les volumes créés dans Docker.
docker volume ls

# Inspecter un volume pour voir sa configuration détaillée,
# comme son emplacement sur le disque.
docker volume inspect mon_volume

# Créer un volume avec des options spécifiques
# Cette commande crée un volume avec le système de fichiers
# tmpfs, utile pour des données temporaires à haute performance.
docker volume create --driver local --opt type=tmpfs --opt
device=tmpfs --opt o=size=100m,uid=1000 mon_volume_avance

# L'option 'ro' monte le volume en lecture seule, empêchant le
# conteneur de modifier les données du volume.
docker run -d --name mon_conteneur -v mon_volume:/data:ro nginx

# Supprimer un volume Docker
# Assurez-vous que le volume n'est attaché à aucun conteneur
# actif avant de le supprimer.
docker volume rm mon_volume

# Supprime tous les volumes non attachés à des conteneurs.
docker volume prune

# Copier des fichiers du host vers un volume via un conteneur
docker cp /chemin/local/fichier.txt
mon_conteneur:/data/fichier.txt
```

# Sécurité



La sécurité des conteneurs est une priorité absolue dans la gestion des applications modernes. Docker fournit plusieurs mécanismes pour sécuriser vos conteneurs.



```
# Minimiser les privilèges des conteneurs
# Exécuter un conteneur en tant qu'utilisateur non-root
docker run -u 1000 nginx

# Créer un réseau Docker avec des restrictions
docker network create --driver bridge --subnet 172.28.0.0/16
isolated_network

# Utiliser Docker Secrets pour gérer les secrets
# Ne stockez pas de secrets, comme les mots de passe et les
# clés API, directement dans les images ou les scripts.
echo "mon_secret" | docker secret create mon_secret -
```

Mettez en place des audits de sécurité réguliers pour vérifier la configuration de vos hôtes Docker et de vos conteneurs.

# Maintenance & Surveillance



La surveillance active et la journalisation sont cruciales pour détecter les problèmes de performance ou de sécurité et intervenir rapidement. Utilisez des outils comme Prometheus, Grafana, ou ELK Stack pour le monitoring et le logging.

Effectuez des sauvegardes régulières des données importantes, en utilisant des volumes Docker pour stocker les données de manière à ce qu'elles soient séparées de la vie du conteneur.

Configurez des health checks pour vos conteneurs afin de s'assurer qu'ils fonctionnent correctement et de redémarrer automatiquement les services en cas de défaillance.

```
# Configuration d'un service de logging simple avec Docker
docker run -d --name mon_logging -p 5601:5601 -v
"/var/run/docker.sock:/var/run/docker.sock" mon_elk

# Créer un backup des données d'un volume
docker run --rm --volumes-from mon_conteneur -v $(pwd):/backup
ubuntu tar cvf /backup/backup.tar /data

# Ajouter un health check à un conteneur Docker
docker run --name mon_service --health-cmd="curl --fail
http://localhost:80/health || exit 1" -d mon_image
```

# Monitoring & Logging



La surveillance et le logging sont essentiels pour maintenir la santé et la performance des applications conteneurisées. Docker fournit plusieurs outils et intégrations pour aider à montrer l'état des conteneurs et à collecter les logs pour l'analyse.

```
● ● ●

# Afficher les statistiques de tous les conteneurs actifs
docker stats

# Écouter les événements Docker
docker events

# Surveiller l'utilisation de ressources par les conteneurs en
# temps réel (CPU, mémoire, réseau)
docker stats

# Inspecter un conteneur pour obtenir des informations
# détaillées
docker inspect mon_conteneur

# Afficher les logs d'un conteneur
docker logs mon_conteneur
```

# Navigation & Modifications



Naviguer dans un conteneur pour explorer son état actuel ou effectuer des modifications peut être crucial pour le développement et le débogage. Voici comment vous pouvez inspecter et modifier des conteneurs, et comment persister ces modifications dans une nouvelle image Docker.

```
# Démarrer un shell interactif dans un conteneur en cours
# d'exécution
docker exec -it mon_conteneur /bin/bash

# Afficher les modifications dans le système de fichiers d'un
# conteneur
docker diff mon_conteneur

# Créer une nouvelle image à partir des modifications d'un
# conteneur
docker commit mon_conteneur mon_nouvelle_image:tag
```

Si vous avez effectué des modifications dans un conteneur que vous souhaitez conserver, vous pouvez "commit" ces changements pour créer une nouvelle image Docker. Cela est utile pour le débogage ou la création rapide de configurations.

# Docker Compose



Docker Compose est un outil qui permet de définir et de gérer des applications multi-conteneurs. Avec Docker Compose, vous pouvez configurer les services, les volumes, les réseaux et les dépendances de vos applications dans un seul fichier YAML, simplifiant ainsi le processus de déploiement et de gestion.

```
● ● ● docker-compose.yml

version: '3.9'
services:
  web:
    image: nginx
    ports:
      - "80:80"
    deploy:
      replicas: 3
      update_config:
        parallelism: 2
        delay: 10s
    volumes:
      - web-data:/var/www/html
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example

volumes:
  web-data:
```

Cet exemple définit deux services, web et db. Le service web utilise l'image nginx et configure les politiques de déploiement, tandis que db utilise l'image postgres.

# Gestion de Docker Compose



Docker compose dispose de nombreuses commandes pour gérer les conteneurs. En voici les principales.

```
# Démarrer les services spécifiés dans le fichier docker-compose.yml en arrière-plan.  
docker compose up -d  
  
# Arrêter et supprimer les ressources (conteneurs, réseaux, volumes) créées.  
docker compose down  
  
# Affiche les projets Compose en cours d'exécution avec leur état.  
docker compose ls  
  
# Afficher les logs de tous les services.  
docker compose logs  
  
# Exécuter une commande unique dans un service.  
docker compose run webapp echo "Hello World"  
  
# Mettre à jour et reconstruire un service spécifique.  
docker compose up -d --no-deps webapp  
  
# Arrêter un service spécifique.  
docker compose stop webapp  
  
# Mise à l'échelle d'un service  
docker-compose up --scale web=3
```

# Dans la même collection

## Orchestration et Gestion de Conteneurs



## Infrastructure as Code



## Sécurité & Gestion des secrets



## Développement & CI/CD



↓ FOLLOW ME ↓



[ANTONYCANUT](#)



[ANTONY KERVAZO-CANUT](#)