Table des matières

Official Docker CLI Cheat Sheet – free reference from Docker with common commands (build, run, stop, logs, images, etc.).

https://www.docker.com/resources/cli-cheat-sheet

Basic commands: images, containers, push/pull

- Dockerfile patterns
- Daily use commands

https://docs.docker.com/

https://hub.docker.com/search?badges=official

🧱 Qu’est-ce qu’un container (vraiment) ?

Un **container n’est PAS une VM**. 👉 C’est :

- un **processus Linux**
- isolé
- qui s’exécute sur le **noyau de l’hôte**
- avec son propre environnement

Docker = orchestration de processus isolés

🧪 Docker vs VM (résumé)

Docker	VM
Démarrage ms	Démarrage minutes
Partage kernel	Kernel dédié
Léger	Lourd
Processus	Machine complète

⚙️ Les 4 briques fondamentales

Images (ce qui est immuable) en lecture seule, composée de layers empilés, créée via un `Dockerfile`

```
Image
 ├── Layer base (alpine, debian…)
 ├── Layer dépendances
 ├── Layer appli
```

Container (image + runtime)

Namespaces (isolation)

Cgroups (limitation) (control groups limitent les ressources**)**

```
docker run --memory 512m --cpus="1.5" myapp
```

📦 Les volumes :

- ➔ ne sont pas dans les layers
- ➔ survivent au container
- ➔ sont gérés par Docker

```
docker volume create data
docker run -v data:/var/lib/db postgres
```

🔐 Sécurité (principes clés)

✔ Containers ≠ sandbox parfaite
✔ Partagent le noyau hôte
✔ Root dans container ≠ root sur l’hôte (mais attention)

Bonnes pratiques :

- utilisateur non-root
- images minimalistes
- `read-only`
- `cap-drop ALL`

🌐 Réseau des containers

Par défaut (`bridge`)

- IP privée
- NAT vers l’hôte
- DNS Docker

```
Container — docker0 — Host — Internet
```

Réseau custom

- DNS automatique
- communication par nom
- isolation logique

```
docker network create backend
curl http://api:8080
```

🔁 Ports (NAT)

```
-p 8080:80
```

➡ Port hôte → port container

➡ Le container n'expose rien par défaut

Exemple :

```
docker run -d \
  --name web \
  --restart unless-stopped \
  -p 8080:80 \
  --network frontend \
  nginx
```

📌 Quick Reference (Example Commands)

Here's a **mini cheat sheet** you can copy:

Images

```
docker build -t <name> .
docker images
docker rmi <image>
docker pull <image>
docker push <image>
```

Containers

```
docker run --name foo -d -p 80:80 <image>
docker ps           # running
docker ps -a        # all
docker start foo
docker stop foo
docker rm foo
docker logs -f foo
docker exec -it foo sh
```

Info & Cleanup

```
docker info
docker system prune
docker container stats
```

Construire une image

```
docker build -t nom-de-mon-image:tag .
```

-t : Permet de donner un nom (tag) à l'image.

. : Indique que le Dockerfile se trouve dans le répertoire courant.

Inspecter les métadonnées

```
docker inspect nom-de-mon-image
docker inspect -f '{{.NetworkSettings.IPAddress}}' nom-du-conteneur
```

Voir l'arborescence et les couches (History)

```
docker history nom-de-mon-image
# Une fois dedans :
ls -R /
```

```
dive nom-de-mon-image
```

Outil indispensable Dive, permet d'explorer visuellement chaque couche et de voir quels fichiers ont été ajoutés ou modifiés.

🚀 Build and push your first image

Click here https://docs.docker.com/get-started/introduction/build-and-push-first-image/

🚀 docker run - options essentielles

Syntaxe de base

```
docker run [OPTIONS] IMAGE [COMMAND]
```

```
-it                                     # interactif + terminal
docker run -it alpine sh

-d                                      Mode détaché (service) # exécution en arrière-plan
docker run -d nginx

--name my-container                     Nom du container

-p 8080:80                              Publication de ports
docker run -d -p 8080:80 nginx

--network my-net                        Réseau personnalisé
docker run -d --network backend api

--hostname my-host

-v /host/path:/container/path           Volume bind (host ↔ container)
docker run -v $(pwd):/app node

-v mydata:/var/lib/data                 Volume nommé (bonne pratique prod)

--read-only                             Lecture seule

-e ENV=prod                             Variables d'environnement
docker run -e POSTGRES_PASSWORD=secret postgres

--env-file .env                         Fichier .env

--restart always                        Cycle de vie Redémarrage automatique
--restart unless-stopped

--rm                                    Indispensable en prod Suppression automatique à l'arrêt

--memory 512m                           Ressources (limitation) Mémoire

--cpus="1.5"                            CPU

--user 1000:1000                        Utilisateur non-root

--cap-drop ALL                          Capacités Linux
--cap-add NET_ADMIN

--tmpfs /tmp                            Système de fichiers temporaire


   --(Options moins connues mais utiles)--
--pull always                           Toujours récupérer la dernière image
--log-driver                            json-file       Gestion des logs
--log-opt max-size=10m                  Rotation des logs
--health-cmd                            Healthcheck
--health-interval                       Intervalle de check
```

🧪 Debug & inspection

Variables, mounts, réseau

```
docker inspect <container>
```

Stats en temps réel

```
docker stats
```

Override du point d'entrée
--entrypoint sh

```
docker run -it --entrypoint sh myimage
```

🧠 Bonnes pratiques réseau (important)

✅ Utiliser **des réseaux custom**, pas `bridge` par défaut
✅ Ne pas exposer les ports inutilement
✅ Communication inter-containers via réseau Docker
❌ Éviter les IP fixes
❌ Éviter `--network host` (sauf cas très spécifique)

Annexes sommaire de cours

🐬 L’essentiel de Docker — Sommaire du cours

Introduction
- Pourquoi Docker ?
- Problèmes résolus par la conteneurisation
- À qui s’adresse ce cours (dev, ops, fullstack)

1. Découvrir Docker
- Comparer les solutions de virtualisation
  - VM vs Containers
  - Avantages et limites
- S’initier à l’architecture de Docker
  - Docker Engine
  - Images / Containers
  - Registry (Docker Hub)
- Comprendre le fonctionnement de Docker
  - Image vs container
  - Cycle de vie d’un container
  - Rôle du PID 1
- Quiz de validation

2. Installer Docker
- Installation sur Windows / macOS / Linux
- Docker Desktop vs Docker Engine
- Vérification de l’installation
- Premières commandes (`docker version`, `docker info`)
- Bonnes pratiques post-installation

3. Utiliser Docker au quotidien
- Commandes essentielles
  - `docker run`, `ps`, `stop`, `rm`
- Mode interactif vs détaché
- Gestion des logs
- Nettoyage de l’environnement Docker
- Bonnes pratiques de nommage

4. Aborder les conteneurs
- Qu’est-ce qu’un container (au sens système) ?
- Isolation : namespaces & cgroups
- Gestion des ressources (CPU, mémoire)
- Redémarrage et cycle de vie
- Debug d’un container en cours d’exécution

5. Employer les images Docker
- Structure d’une image Docker
- Les layers et le cache
- Dockerfile : bases essentielles
  - `FROM`, `RUN`, `COPY`, `CMD`, `ENTRYPOINT`
- Construire une image (`docker build`)
- Bonnes pratiques d’images (taille, sécurité)

6. Appréhender les volumes
- Problème de la persistance des données
- Volumes vs bind mounts
- Création et gestion des volumes
- Cas d’usage typiques (DB, logs)
- Bonnes pratiques de stockage

7. Mettre en œuvre le réseau
- Réseau Docker par défaut (`bridge`)
- Réseaux personnalisés
- DNS interne Docker
- Exposition de ports
- Communication inter-containers
- Sécurité réseau de base

8. Assimiler quelques bonnes pratiques
- Containers éphémères
- Images minimalistes
- Utilisateur non-root
- Gestion des secrets
- Quand passer à Docker Compose
- Erreurs courantes à éviter

Conclusion
- Ce qu’il faut retenir de Docker
- Quand Docker suffit
- Quand aller plus loin (Compose, Kubernetes)
- Ressources pour approfondir

🏗️ Sommaire : Docker Avancé & Orchestration Enterprise

Introduction
- Présentation des enjeux du déploiement de conteneurs en milieu professionnel.

1. Aborder Mirantis Container Runtime (MCR)
- Comprendre le moteur de conteneurisation de classe entreprise (ex-Docker Engine Enterprise).
- Installation et configuration de l'environnement.

2. Créer des applications multi-conteneurs
- Maîtriser l'interconnectivité entre les services.
- Gestion des volumes et de la persistance des données.

3. Créer un cluster SWARM
- Architecture d'un cluster : Nodes Managers vs Nodes Workers.
- Initialisation du cluster et gestion de la haute disponibilité.

4. Créer un projet Compose sur un cluster SWARM
- Déploiement de "Stacks" à l'aide de fichiers `docker-compose.yml`.
- Passage du développement à la production (Scaling des services).

5. Gérer Docker avec Universal Control Plane (UCP)
- Présentation de l'interface graphique de gestion Mirantis.
- Contrôle d'accès (RBAC) et monitoring du cluster.

6. Installer Docker Trusted Registry (DTR)
- Mise en place d'un registre privé d'images sécurisé.
- Gestion du cycle de vie des images (Scanning de vulnérabilités et signatures).

7. Découvrir la certification Docker Certified Associate (DCA)
- Présentation de l'examen et des compétences clés attendues.
- Conseils de préparation pour valider son expertise.

Conclusion
- **Synthèse :** Conclure sur Docker avancé (Vidéo).
- Perspectives sur l'évolution vers Kubernetes et les environnements Cloud.