🐳 Docker — Docker Desktop

Objectifs

À la fin de cette page, tu seras capable de :

- Naviguer dans les différentes sections de Docker Desktop
- Superviser et gérer des conteneurs sans ligne de commande
- Inspecter les logs, les ressources et les variables d'environnement d'un conteneur
- Gérer les images et les volumes
- Retrouver l'équivalent CLI de chaque action graphique

---

5 notions-clés

1. **Dashboard** — Vue d'ensemble de tous les conteneurs et leur état en temps réel
2. **Containers** — Section principale pour gérer les conteneurs (démarrer, arrêter, inspecter, logs)
3. **Images** — Bibliothèque locale des images téléchargées ou construites
4. **Volumes** — Interface pour visualiser et supprimer les volumes persistants
5. **Settings** — Configuration des ressources allouées à Docker (RAM, CPU, WSL)

Présentation générale

Docker Desktop est l'application graphique officielle de Docker pour Windows et macOS. Elle te permet de faire tout ce qu'on peut faire en ligne de commande, mais via une interface visuelle.

À son lancement, une icône apparaît dans la **barre des tâches** (systray). Un clic droit dessus donne accès aux actions rapides ; un double-clic ouvre l'interface principale.

L'interface principale est divisée en une **barre de navigation gauche** avec les sections :

```
🐳 Logo Docker (Dashboard)
├── Containers
├── Images
├── Volumes
├── Builds
└── Extensions
    Settings (engrenage en bas)
```

Section : Dashboard (Accueil)

La page d'accueil affiche un **résumé rapide** de l'état de Docker :

- Nombre de conteneurs en cours d'exécution / arrêtés
- Espace disque utilisé par Docker
- Liens vers la documentation et Docker Hub

Quand l'utiliser : Pour un coup d'œil rapide avant de commencer à travailler.

Section : Containers

C'est la section la plus utilisée au quotidien. Elle liste **tous les conteneurs**, qu'ils soient en cours d'exécution ou arrêtés.

La liste des conteneurs

Chaque ligne affiche :

- **Nom** du conteneur
- **Image** utilisée (ex: `nginx:1.25`)
- **Statut** : `Running` (vert) / `Exited` (gris) / `Starting` (orange)
- **Ports** exposés (ex: `0.0.0.0:8080→80/tcp`)
- **Actions rapides** : ▶ Démarrer · ■ Arrêter · 🔄 Redémarrer · 🗑 Supprimer

Les projets **Docker Compose** sont regroupés sous leur nom de dossier, avec les services listés en dessous. Tu peux plier/déplier le groupe.

Les onglets d'un conteneur (clic sur le nom)

En cliquant sur le nom d'un conteneur, tu accèdes à sa fiche détaillée avec plusieurs onglets :

Onglet `Logs`

Affiche les sorties du conteneur en temps réel (équivalent de `docker logs -f`).

- Tu peux **filtrer** les logs par mot-clé
- Tu peux **copier** les logs
- L'option "**Timestamps**" affiche la date/heure de chaque ligne

Cas d'usage typique : MySQL met plusieurs secondes à démarrer. Tu viens ici pour voir quand il est prêt (`ready for connections`), ou pour comprendre pourquoi l'app PHP ne se connecte pas.

Onglet `Inspect`

Affiche les métadonnées complètes du conteneur en JSON :

- Politique de redémarrage (`RestartPolicy`)
- Variables d'environnement (`Env`)
- Volumes montés (`Mounts`)
- Réseau(x) (`Networks`)
- Commande de démarrage (`Cmd`)

Cas d'usage : Vérifier que le conteneur a bien reçu la bonne variable `DB_HOST`, ou qu'il est bien connecté au bon réseau.

Onglet `Bind mounts` / `Volumes`

Liste les volumes et bind mounts du conteneur :

- Le chemin sur ta machine hôte
- Le chemin correspondant dans le conteneur

Cas d'usage : Vérifier que le bind mount pointe bien vers le bon dossier de code.

Onglet `Exec`

Ouvre un **terminal interactif** directement dans le conteneur (équivalent de `docker exec -it nom bash`).

Tu peux :

- Naviguer dans les fichiers du conteneur
- Lancer des commandes PHP, MySQL, etc.
- Inspecter les fichiers de configuration

Cas d'usage : Se connecter au conteneur MySQL pour vérifier les tables, ou regarder les fichiers PHP tels que le conteneur les voit.

Onglet `Stats`

Affiche en temps réel :

- **CPU** utilisé (en %)
- **RAM** utilisée / allouée
- **Réseau** (octets envoyés / reçus)
- **Disque** (octets lus / écrits)

Cas d'usage : Détecter un conteneur qui consomme anormalement de la RAM ou du CPU.

Section : Images

Liste toutes les images Docker stockées localement.

Ce que tu vois

Colonne	Signification
Nom	Nom de l'image (ex: `nginx`)
Tag	Version (ex: `1.25`, `latest`)
Taille	Espace occupé sur le disque
Créée	Date de création ou de téléchargement
Utilisation	Indique si l'image est utilisée par un conteneur

Actions disponibles

- **Run** → Lance directement un conteneur depuis l'image (ouvre un formulaire pour configurer les ports, le nom, les variables...)
- **Pull** → Vérifie si une nouvelle version est disponible sur Docker Hub
- **Push** → Envoie l'image vers un registre (Docker Hub, registre privé)
- **Inspect** → Voir les métadonnées de l'image (couches, taille, commandes)
- **Delete** → Supprimer l'image (impossible si un conteneur l'utilise encore)

Cas d'usage : Tu veux supprimer des vieilles images pour libérer de l'espace. Docker Desktop affiche clairement celles qui ne sont plus utilisées.

Bouton "Clean up"

Docker Desktop propose régulièrement de nettoyer les images inutilisées. Le bouton **"Clean up"** (ou "Prune") liste les images sans conteneur associé et permet de les supprimer en un clic.

Section : Volumes

Liste tous les volumes Docker créés sur ta machine.

Ce que tu vois

Colonne	Signification
Nom	Nom du volume (ex: `monprojet_db_data`)
Taille	Espace occupé
Date	Dernière utilisation
Conteneurs	Conteneurs qui utilisent ce volume

Actions

- **Inspecter** → Voir le chemin réel sur le disque (`/var/lib/docker/volumes/...`)
- **Parcourir les fichiers** → Ouvrir un explorateur de fichiers dans le volume (**très utile !**)
- **Supprimer** → Supprime définitivement les données (impossible si le volume est en cours d'utilisation)

Cas d'usage : Tu veux vérifier que tes données MySQL ont bien été sauvegardées dans le volume, ou tu veux parcourir les fichiers d'un volume pour déboguer.

Section : Builds

Affiche l'**historique des builds** Docker (quand tu fais `docker build` ou `docker compose build`).

Pour chaque build :

- Durée
- Statut (succès / échec)
- Les étapes du Dockerfile avec leur durée individuelle

Cas d'usage : Identifier quelle étape du Dockerfile est lente pour l'optimiser.

Section : Extensions

Docker Desktop supporte des **extensions communautaires** qui ajoutent des fonctionnalités à l'interface. Quelques exemples utiles :

Extension	Utilité
Portainer	Interface complète de gestion Docker
Disk usage	Analyse détaillée de l'espace utilisé
Logs Explorer	Recherche avancée dans les logs
Resource Usage	Graphiques de consommation CPU/RAM

Section : Settings (Paramètres)

Accessible via l'engrenage ⚙️ en bas à gauche.

General

- **Start Docker Desktop when you sign in** → Lance Docker au démarrage de Windows
- **Open Docker Dashboard at startup** → Ouvre l'interface automatiquement
- **Use the WSL 2 based engine** → Normalement activé par défaut sous Windows 11

Resources → Advanced

Permet de limiter les ressources allouées à Docker :

Paramètre	Recommandation
CPUs	Laisser à 50% de tes cœurs (ex: 4 sur 8)
Memory	4 Go minimum pour PHP+MySQL, 8 Go pour des projets plus lourds
Disk image size	Augmenter si tu as beaucoup d'images

💡 Sous Windows avec WSL 2, la mémoire est gérée dynamiquement. Docker prend ce dont il a besoin et le rend au système. Tu peux créer un fichier `.wslconfig` pour définir un maximum.

Resources → File sharing

Sous Windows, les bind mounts nécessitent que le dossier soit **accessible par WSL**. Tous les dossiers sous `C:\Users\` sont partagés par défaut.

Si tu veux partager un dossier sur un autre disque (ex: `D:\projets`), ajoute-le ici.

Docker Engine

Affiche et permet d'éditer le fichier de configuration JSON du moteur Docker (`daemon.json`).

Cas d'usage avancé : Configurer un registre d'images privé en entreprise, ou activer des fonctionnalités expérimentales.

Équivalences : CLI ↔ Docker Desktop

Action CLI	Équivalent Docker Desktop
`docker ps`	Section **Containers** → liste des conteneurs
`docker ps -a`	Section **Containers** → affiche aussi les arrêtés
`docker logs -f nom`	Clic sur le conteneur → onglet **Logs**
`docker exec -it nom bash`	Clic sur le conteneur → onglet **Exec**
`docker inspect nom`	Clic sur le conteneur → onglet **Inspect**
`docker stats nom`	Clic sur le conteneur → onglet **Stats**
`docker start nom`	Bouton ▶ sur le conteneur
`docker stop nom`	Bouton ■ sur le conteneur
`docker restart nom`	Bouton 🔄 sur le conteneur
`docker rm nom`	Bouton 🗑 sur le conteneur
`docker images`	Section **Images**
`docker rmi nom`	Section **Images** → bouton **Delete**

Action CLI	Équivalent Docker Desktop
`docker pull nom`	Section **Images** → bouton **Pull**
`docker image prune`	Section **Images** → bouton **Clean up**
`docker volume ls`	Section **Volumes**
`docker volume inspect nom`	Section **Volumes** → clic sur le volume
`docker volume rm nom`	Section **Volumes** → bouton **Delete**
`docker system df`	Section **Images** ou **Volumes** (affichage des tailles)

Workflow typique avec Docker Desktop

Démarrage de projet

1. Ouvrir un terminal dans le dossier du projet
2. `docker compose up -d --build` dans le terminal
3. Ouvrir Docker Desktop → section **Containers** pour surveiller le démarrage
4. Cliquer sur le service `db` → onglet **Logs** pour voir quand MySQL est prêt
5. Tester l'application dans le navigateur

Déboguer une erreur de connexion

1. L'application PHP dit "Connexion impossible"
2. Docker Desktop → **Containers** → clic sur le conteneur `app`
3. Onglet **Inspect** → vérifier `DB_HOST`, `DB_NAME`, `DB_PASSWORD`
4. Onglet **Logs** → chercher un message d'erreur PHP
5. Onglet **Exec** → tester manuellement : `php -r "new PDO(...)"`

Libérer de l'espace disque

1. Docker Desktop → **Images**
2. Clic sur **Clean up** → voir les images non utilisées
3. Cocher et supprimer
4. Docker Desktop → **Volumes** → supprimer les volumes orphelins

---

Cours TTR Informatique — CEPES Jodoigne