0. présentation de Docker et Docker Desktop

🧩 1. Qu’est-ce que Docker ?

Docker est une plateforme permettant de créer, exécuter et gérer des applications dans des conteneurs. Un conteneur est un environnement isolé, léger et reproductible qui contient tout ce dont une application a besoin pour fonctionner.

Idée clé

Docker garantit que ton application fonctionne de la même manière partout, que ce soit sur ton PC, un serveur ou le cloud.

🏗️ 2. Pourquoi Docker existe ?

Les applications modernes sont composées de nombreux services.
Les environnements diffèrent d’une machine à l’autre.
Les installations manuelles sont longues et sources d’erreurs.
Docker résout ces problèmes en fournissant un environnement standardisé, portable et automatisé.

📦 3. Les composants essentiels de Docker

Images Docker — modèles contenant le code et les dépendances.
Conteneurs — instances exécutées à partir d’images.
Docker CLI — interface en ligne de commande pour piloter Docker.
Docker Compose — outil pour lancer plusieurs services ensemble.
Volumes — stockage persistant pour les données.
Réseaux Docker — communication entre conteneurs.
Docker Hub — bibliothèque d’images en ligne.

🧬 4. Comment fonctionne Docker ?

Docker utilise une technologie appelée containerisation, basée sur des fonctionnalités du noyau Linux (namespaces, cgroups). Contrairement aux machines virtuelles, Docker ne virtualise pas un système complet : → il partage le noyau de l’hôte, ce qui le rend beaucoup plus léger et rapide.

🎯 5. Docker Desktop :

Docker Desktop est un logiciel qui permet de créer, gérer et exécuter facilement des conteneurs Docker sur Windows, macOS ou Linux, grâce à une interface graphique intuitive et un moteur Docker intégré.

☸️ 6. Docker et Kubernetes

Docker peut être utilisé avec Kubernetes, un orchestrateur permettant de :
déployer des applications à grande échelle,
gérer des clusters,
automatiser le redémarrage, la mise à jour et la montée en charge.
Docker Desktop peut activer Kubernetes en un clic pour les démonstrations.

📌 7. Conclusion

Docker est devenu un standard incontournable pour le développement, le déploiement et la gestion d’applications modernes. Il simplifie la vie des développeurs, des administrateurs système et des enseignants grâce à sa flexibilité et sa portabilité.

1. Qu’est-ce que Docker Desktop ?

Une application tout-en-un qui installe :

- le moteur Docker (Docker Engine),
- Docker CLI,
- Docker Compose,
- une interface graphique,
- une VM Linux (sur Windows/macOS),
- des outils de développement intégrés.

Il simplifie :

- la gestion des conteneurs,
- la gestion des images,
- les volumes,
- les réseaux,
- les stacks Compose.

Il permet d’utiliser Docker sans ligne de commande obligatoire, même si la CLI reste disponible.

🖥️ Docker CLI

Interface en ligne de commande permettant de gérer les conteneurs, images, volumes et réseaux.

🧬 Docker Compose

Outil permettant de définir et lancer plusieurs services via un fichier docker-compose.yml.
Idéal pour les applications multi-conteneurs (ex : base de données + backend + frontend).

📦 Images Docker

Modèles immuables contenant tout le nécessaire pour exécuter une application. Elles servent de base pour créer des conteneurs.

🗂️ Volumes Docker

Espaces de stockage persistants permettant de conserver les données même si le conteneur est supprimé.

🧱 Stacks Compose

Ensemble de services définis dans un fichier Compose et déployés comme une application complète.
Permet de gérer plusieurs conteneurs comme un seul bloc.

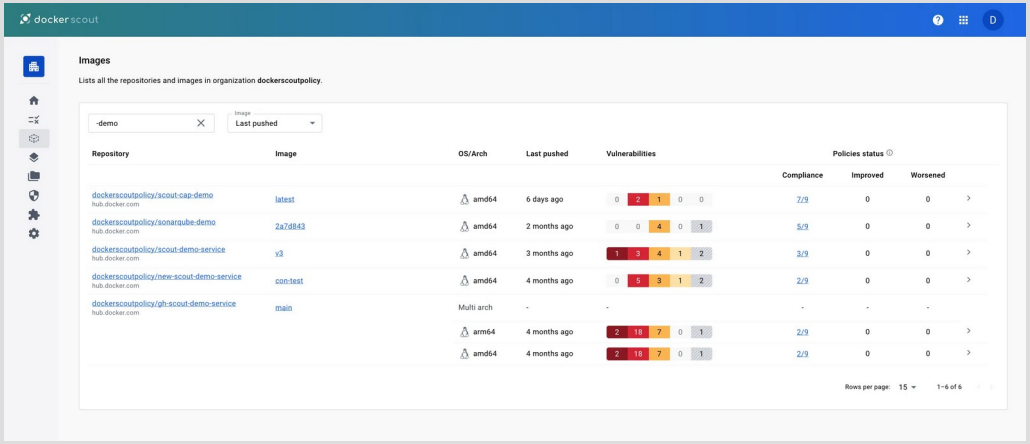
☸️ Kubernetes

Orchestrateur de conteneurs intégré à Docker Desktop.
Il permet de déployer, scaler et gérer des applications complexes en production.

🌐 Docker Hub

Registre officiel d’images Docker. Permet de télécharger, publier et partager des images avec la communauté ou une équipe.

2. Interface et fonctionnalités principales



🗂️ Volumes

Visualisation des volumes persistants.

Suppression sécurisée.

📌 Tableau de bord (Dashboard)

Vue d'ensemble des conteneurs en cours d'exécution.

Démarrage/arrêt/restart en un clic.

Logs accessibles graphiquement.

🌐 Réseaux

Liste des réseaux Docker (bridge, host, overlay…).

📦 Images

Téléchargement depuis Docker Hub.

Suppression, renommage, inspection.

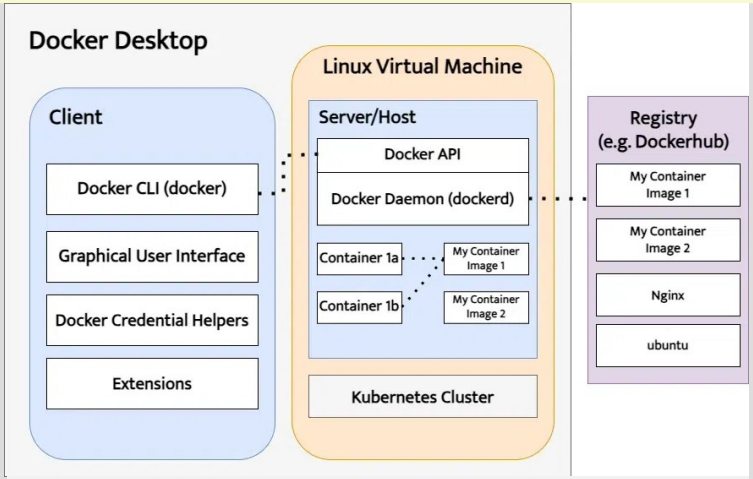
🧬 Docker Compose

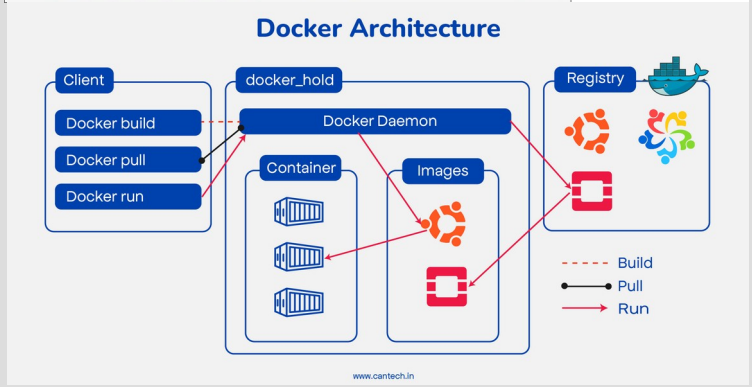
Lancement de stacks complètes.

Visualisation des services.

Logs par service.

3. Architecture de Docker Desktop





Sur Windows/macOS :

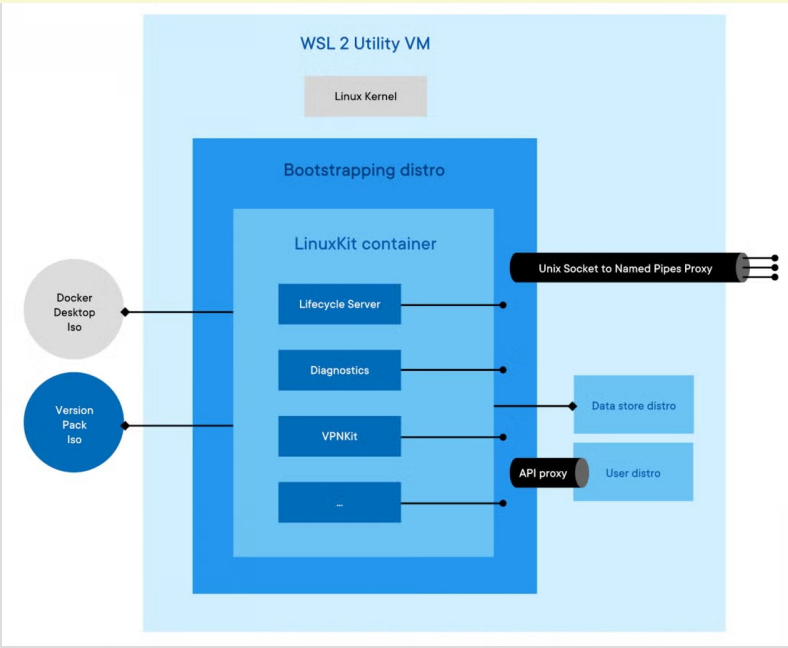
Docker Desktop utilise une VM Linux (WSL2 ou Hyper-V).

Les chemins Windows sont montés dans la VM, puis dans les conteneurs.

C'est pour cela que tu vois des chemins comme /data/compose/4/data dans Portainer.

Sur Linux :

Pas de VM : Docker Desktop agit comme une surcouche graphique.



4. Pourquoi utiliser Docker Desktop ?

Installation simple et rapide.

- Interface graphique intuitive.
- Gestion centralisée des conteneurs.
- Intégration avec :
- Kubernetes (optionnel),
- Docker Hub,
- extensions (Portainer, Dev Environments…).

Parfait pour :

- les développeurs,
- les enseignants,
- les étudiants,
- les tests rapides,
- les environnements de démonstration.

🧪 Cas d’usage concrets

Tester un serveur web en local.

Déployer une base de données (PostgreSQL, MariaDB…).

Lancer des applications complètes via Docker Compose.

Utiliser des outils comme Portainer, InvenTree, Nextcloud, Home Assistant.

Simuler des environnements réseau pour TP (ce que tu fais déjà).

🔐 Limitations et points d’attention

Consommation RAM/CPU due à la VM.

Performances parfois inférieures à Linux natif.

Nécessite Windows 10/11 Pro pour Hyper-V (sinon WSL2).

Certaines fonctionnalités avancées nécessitent un compte Docker.

4.1. Comprendre les images Docker et leurs actions

🧩 Qu’est-ce qu’une image Docker ?

Une image Docker est un modèle ou un plan de construction d’un conteneur.
Elle contient :
le programme
les dépendances
la configuration
le système de fichiers nécessaire
➡️ Une image = un modèle statique ➡️ Un conteneur = une instance vivante de ce modèle

🖥️ Où voir les images dans Docker Desktop ?

Dans Docker Desktop :
Aller dans l’onglet Images
Chaque image apparaît avec :
son nom
sa taille
son tag (ex : latest)
les actions disponibles

🎯 À retenir

Une image est un modèle
Un conteneur est créé à partir de cette image
Docker Desktop permet :
de télécharger,
de lancer,
de inspecter,
et de supprimer les images
Les images ne changent jamais : elles sont immuables

🛠️ Actions disponibles sur une image

▶️ Run

Crée un conteneur à partir de l’image. ➡️ C’est l’action principale.

🔄 Pull (depuis Docker Hub)

Télécharge l’image depuis Docker Hub. ➡️ Permet d’obtenir la dernière version.

🗑️ Delete

Supprime l’image du système. ➡️ Libère de l’espace disque. ⚠️ Impossible si un conteneur utilise encore cette image.

📄 View Details

Affiche les informations techniques :
tags
taille
couches (layers)

4.2. Comprendre les conteneurs et leurs actions

🧩 Qu’est-ce qu’un conteneur ?

Un conteneur Docker est une application en cours d’exécution, isolée du système, créée à partir d’une image.
Il contient :
le programme
ses dépendances
sa configuration
son propre système de fichiers temporaire
➡️ Un conteneur = une instance vivante d’une image.

🚦 Cycle de vie d’un conteneur

Dans Docker Desktop, un conteneur peut être :
Running → il fonctionne
Stopped → il est arrêté mais existe encore
Exited → il s’est terminé automatiquement
Deleted → il disparaît complètement

🎯 À retenir

Un conteneur est éphémère
Les actions Start / Stop / Restart / Delete gèrent son cycle de vie
Les données ne survivent pas sans volume ou bind mount
Docker Desktop permet de tout gérer sans ligne de commande

🛠️ Actions disponibles dans Docker Desktop

▶️ Start / Run

Lance le conteneur. ➡️ L’application démarre.

⏸️ Stop

Arrête le conteneur. ➡️ L’application s’arrête mais les fichiers internes restent (sauf si pas de volume).

🔄 Restart

Arrête puis relance le conteneur. ➡️ Attention : les données internes sont perdues si aucun volume n’est utilisé.

📄 Logs

Affiche les messages produits par l’application. ➡️ Très utile pour comprendre ce qui se passe.

🧭 Inspect

Montre les détails techniques : ports, volumes, variables, configuration.

🗑️ Delete

Supprime le conteneur. ➡️ Les données internes disparaissent (sauf si volume).

5. Introduction à l’installation sous Windows

- Docker Desktop permet d’utiliser Docker facilement sur Windows.

- Installation simple mais nécessite quelques prérequis

Objectif : préparer un environnement fonctionnel pour exécuter des conteneurs.

Prérequis système

- Windows 10/11 64 bits

- Version Pro / Enterprise / Education recommandée

- WSL2 activé (Windows Subsystem for Linux)

- Virtualisation activée dans le BIOS

- Compte administrateur Windows

- Connexion Internet pour télécharger Docker Desktop

Télécharger Docker Desktop

Aller sur le site officiel : Docker.com
Télécharger Docker Desktop for Windows
Fichier obtenu : Docker Desktop Installer.exe
Taille approximative : 500 Mo

Activer WSL

Ouvrir Powershell en mode administrateur
Executer : **wsl.exe –install**
Redemarer windows

Installer Docker Desktop

Double cliquer sur le fichier Docker Desktop Installer.exe
Suivre la procédure d’installation
Le procéssus d’installation redémarera votre ordinateur

Créer un compte docker personnel

Ouvrir docker desktop
Cliquer sur create account
choisir personnel et saisir le formulaire
Valider l’adresse e-mail

test docker desktop

Lancer Docker Desktop depuis le menu Démarrer
Attendre que le moteur Docker soit actif
Vérifier que Engine running en bas a gauche (moteur opérationnel)

6. Utiliser Docker Desktop

📦 Images Docker

Modèles immuables contenant tout le nécessaire pour exécuter une application.

🌐Accéder à la bibliothèque d'images

Dans Docker Desktop :
Aller dans l'onglet Docker Hub
Cette page présente les images officiel pour doccker
Une barre de recherche apparaît en haut

🔍 Rechercher l'image "hello-world"

Dans la barre de recherche, taper : hello-world
Sélectionner l'image officielle (publisher : library/hello-world)

⬇️ Télécharger l'image "hello-world"

Cliquer sur Pull pour télécharger l'image
Docker Desktop télécharge automatiquement l'image depuis Docker Hub.

🔍 Retrouver l'image "hello-world"

Aller dans l'onglet Images
Cette page présente les images officiel télécharger localement

📦Lancer l'image dans un container

🔍 Retrouver l'image "hello-world"

Aller dans l'onglet Images

▶️ Lancer l'image

Trouver hello-world dans la liste
Cliquer sur Run

🔍 Retrouver le container utilisant "hello-world"

Aller dans l'onglet Containers
Trouver le container qui utilise l'image Hello-Word
Cliquer sur le Name du container pour obtenir ces détails

⚠️ Avertissement important

Un conteneur Docker n'est pas une machine virtuelle : ➡️ tout ce qui est stocké à l'intérieur du conteneur est perdu si tu :
redémarres le conteneur
arrêtes puis relances le conteneur
supprimes le conteneur
recrées le conteneur à partir de l'image

🎯 Pourquoi ?

Les conteneurs sont éphémères : ils sont faits pour être recréés facilement.

💾 Solution

Pour conserver les données, il faut utiliser un volume Docker

7. Comprendre l’usage des ports

🖥️ Pourquoi parler des ports ?

Par défaut un conteneur est isolé du monde extérieur. Pour le relier au monde extérieur, on utilise des ports de communication,
Par exemple un conteneur utilisant basé sur l’image Nginx contient un serveur web qui écoute sur le port 80 à l’intérieur du conteneur.
Pour y accéder depuis ton PC, Docker Desktop doit rediriger un port de ton ordinateur vers le port 80 du conteneur.

🔌 Principe de la redirection de ports

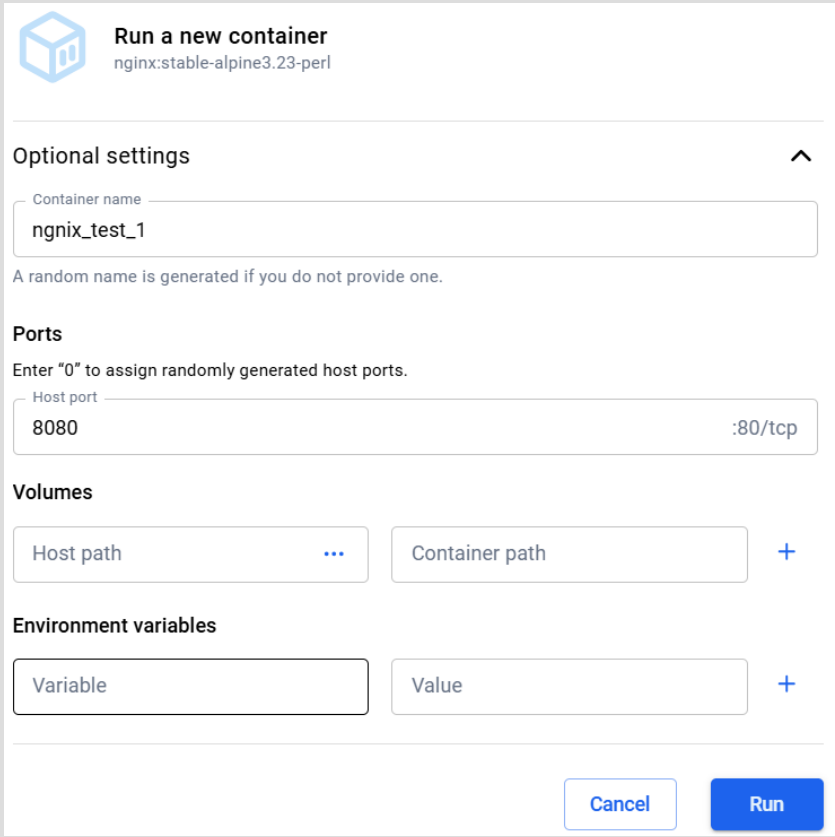
Port interne (container) : 80
Port externe (machine locale) : choisi par Docker Desktop (ex : 8080)
➡️ Cela crée un tunnel : localhost:8080 → conteneur:80

▶️ Comment configurer les ports dans Docker Desktop

Aller dans Docker Hub
rechercher l’image nginx
Cliquer sur Pull
Aller dans Images
Choisir l’image nginx
Cliquer sur Run
Dans la fenêtre de configuration :
Section Ports
Docker Desktop propose : Host port : 8080 → Container port : 80
Tu peux modifier le port externe si nécessaire

🌍 Tester dans le navigateur

Une fois le conteneur lancé :
👉 Ouvrir : http://localhost:8080 ➡️ Tu vois la page d’accueil Nginx Cela confirme que la redirection de ports fonctionne.



8. Comprendre les volumes Docker

📌 Pourquoi les volumes ?

Un conteneur est éphémère : ➡️ tout ce qui est stocké à l'intérieur est perdu si le conteneur est arrêté, redémarré ou supprimé.
Les volumes permettent de conserver les données en dehors du conteneur.

🧩 Qu'est-ce qu'un volume ?

Un volume est un espace de stockage persistant géré par Docker.
Il permet de sauvegarder :
fichiers de configuration
bases de données
contenus générés par l'application
logs, uploads, etc.
➡️ Même si le conteneur disparaît, le volume reste.

🖥️ Créer un volume dans Docker Desktop

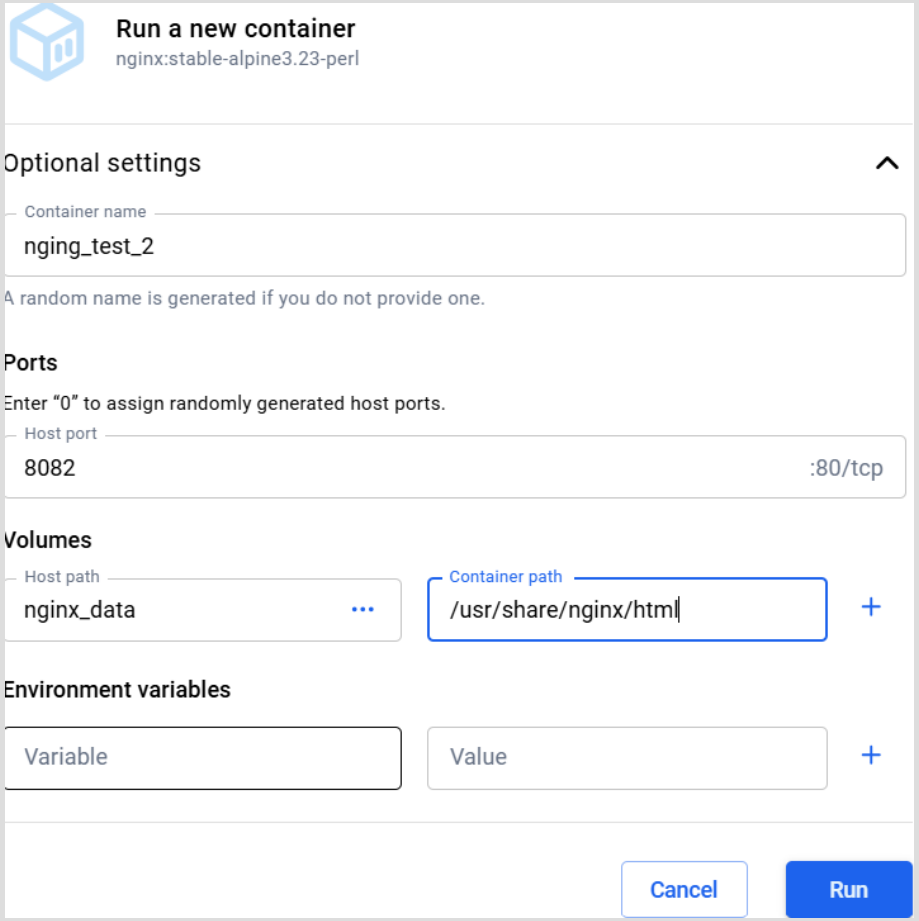
Ouvrir Docker Desktop
Aller dans l'onglet Volumes
Cliquer sur Create
Donner un nom au volume (ex : nginx_data)

🔗 Associer un volume à un conteneur (sans CLI)

Lors du lancement d'une image (ex : nginx) :
Aller dans Images
Cliquer sur Run
Dans la fenêtre de configuration :
Section Volumes
Cliquer sur Add Volume
Choisir un volume existant ou en créer un
Sélectionner le dossier interne du conteneur (ex : /usr/share/nginx/html)
➡️ Le volume devient le disque dur externe du conteneur.

🎯 À retenir

Sans volume → perte de données
Avec volume → données persistantes, même après suppression du conteneur



9. modifier les données d’un volume

🎯 Objectif
Comprendre comment modifier les fichiers du site web Nginx en passant par le volume Docker, sans toucher au conteneur.

📦 1. Où se trouve le contenu du site Nginx ?
Dans le conteneur Nginx, les fichiers web sont stockés dans :
`Code/usr/share/nginx/html`
`Ce dossier est monté sur un volume (ex : nginx-data) pour rendre les fichiers modifiables.`

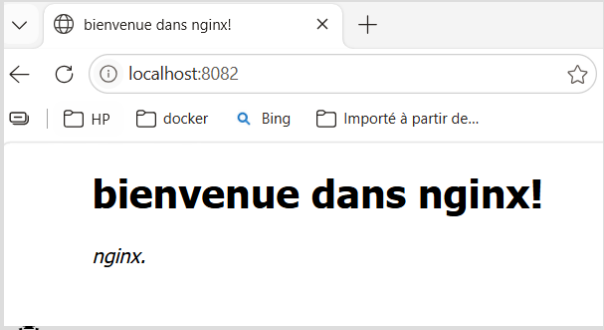
🗂️ 2. Ouvrir le volume dans Docker Desktop
Aller dans l’onglet Volumes
Sélectionner le volume nginx-data
Cliquer sur Explore (ou Browse) → Docker Desktop ouvre l’explorateur de fichiers du volume

✏️ 3. Modifier les fichiers du volume
Dans l’explorateur du volume :
Ouvrir le fichier index.html
Le modifier directement (ex : changer le texte, ajouter un titre, etc.)
Enregistrer les modifications
➡️ Le volume agit comme un dossier partagé entre ton PC et le conteneur.

🔄 4. Voir le résultat immédiatement
Ouvrir un navigateur :
`Codehttp://localhost:8082`
`→ La page affichée par Nginx reflète immédiatement les modifications du volume.`

🎯 À retenir
Le volume nginx-data contient les fichiers du site
Modifier le volume = modifier le site
Pas besoin de redémarrer le conteneur
Le contenu reste même si le conteneur est supprimé

```
<!DOCTYPE html>
<html>
<head>
<title>bienvenue dans nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>bienvenue dans nginx!</h1>
<p><em> nginx.</em></p>
</body>
</html>
```



10. Comprendre et utiliser un Bind Mount

🎯 Objectif
Permettre à l'utilisateur de lier un dossier de son ordinateur directement à un dossier interne du conteneur. ➡️ Idéal pour modifier des fichiers en temps réel (ex : site web Nginx).

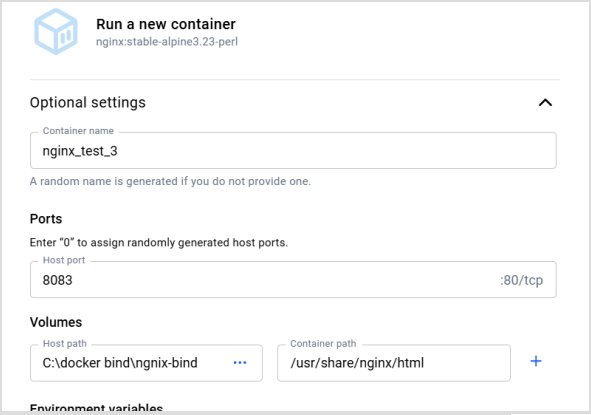
🧩 1. Qu'est-ce qu'un bind mount ?
Un bind mount est un lien entre :
un dossier de ton PC (Windows)
un dossier du conteneur Docker
➡️ Toute modification dans le dossier Windows apparaît instantanément dans le conteneur.

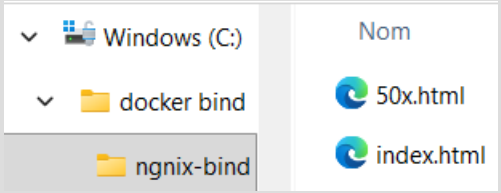
🖥️ 2. Quand utiliser un bind mount ?
Pour modifier un site web Nginx en direct
Pour développer une application sans reconstruire l'image
Pour partager des fichiers entre ton PC et un conteneur
Pour des TP où les élèves doivent éditer des fichiers facilement

🔗 3. Configurer un bind mount
Dans c : Créer un dossier docker bind
Puis dans ce dossier créer ngnix-bind
Dans Docker Desktop :
Aller dans Images
Cliquer sur Run pour l'image nginx
Dans la fenêtre de configuration, ouvrir la section Volumes
Cliquer sur Add Bind Mount
Dans Host Path : → choisir un dossier de ton PC (ex : c:\docker bind\ngnix-bind)
Dans Container Path : → indiquer le dossier interne du conteneur, par exemple :
`Code/usr/share/nginx/html`
➡️ `Le dossier Windows devient le dossier du site web Nginx.`

🌍 4. Tester le résultat
ajoute un fichier dans ton dossier Windows (ex : index.html)
Enregistrer
Ouvrir un navigateur : http://localhost:8083
➡️ Le site affiche immédiatement les modifications

.🎯 À retenir
Bind mount = dossier Windows ↔ dossier conteneur
Modifications en temps réel
Parfait pour le développement et les TP
Plus flexible qu'un volume, mais dépend du système hôte





11. choisir Volume ou Bind Mount

📦 1. Volume Docker

Définition : Espace de stockage géré par Docker, indépendant du système de fichiers Windows.
Caractéristiques :
Stockage persistant même si le conteneur est supprimé
Géré automatiquement par Docker
Idéal pour les données d'application (ex : bases de données, fichiers internes)
Accessible via l'onglet Volumes dans Docker Desktop
Peut être exploré et modifié via Explore
Avantages :
Sécurisé
Portable
Indépendant du système hôte
Parfait pour les environnements pédagogiques ou de production

📁 2. Bind Mount

Définition : Lien direct entre un dossier Windows et un dossier du conteneur.
Caractéristiques :
Le conteneur lit/écrit directement dans un dossier de ton PC
Idéal pour modifier des fichiers en temps réel (ex : site web Nginx)
Configurable dans Docker Desktop → Run → Volumes → Add Bind Mount
Dépend du chemin exact sur Windows
Avantages :
Parfait pour le développement
Modifications visibles instantanément
Très pratique pour les TP où les élèves éditent des fichiers

🧭 3. Quand utiliser quoi ?

Besoin		Solution recommandée
Conserver des données d'application	=	Volume
Modifier des fichiers depuis Windows	=	Bind Mount
Déploiement stable et portable	=	Volume
Développement, tests, TP	=	Bind Mount

🎯 Résumé

Volume → stockage persistant géré par Docker, fiable et portable
Bind Mount → lien direct avec un dossier Windows, idéal pour modifier des fichiers en direct
Les deux apparaissent dans Docker Desktop mais servent des usages différents

12. Comprendre le contenu du fichier docker-compose.yml

📌 Qu’est-ce que Docker Compose ?

Docker Compose permet de décrire une application complète dans un simple fichier texte nommé : docker-compose.yml

Docker Compose :

- récupère les images utiliser (pull)
  créer les containers
- créer les redirection de ports à ouvrir
- créer et attaches les volumes
- configure quelles variables

➡️ Compose = un plan complet pour créer plusieurs conteneurs automatiquement.

📌 Rôle du fichier docker-compose.ym

Un fichier docker-compose.yml sert à décrire une application complète : services, images, ports, volumes, réseaux, variables… ➡️ Docker Compose lit ce fichier et crée automatiquement les conteneurs nécessaires.

🧠 Éléments essentiels à connaître

services — chaque service = un conteneur
image — l’image Docker utilisée pour créer le conteneur
ports — permet d’accéder au service depuis l’extérieur
volumes — stockage persistant ou partage de fichiers
environment — configuration interne du conteneur
networks — communication entre services
volumes/networks (racine) — ressources partagées entre services

🧠 Éléments essentiels à connaître

services — chaque service = un conteneur
image — l’image Docker utilisée pour créer le conteneur
ports — permet d’accéder au service depuis l’extérieur
volumes — stockage persistant ou partage de fichiers
environment — configuration interne du conteneur
networks — communication entre services
volumes/networks (racine) — ressources partagées entre services

🧱 Structure générale d’un fichier Compose

```
version: "3.9"          # (optionnel) version du format Compose

services:               # Liste des services (conteneurs)
  nom_du_service:
    image: ...          # Image Docker à utiliser
    ports:              # Ports exposés
      - "8080:80"
    volumes:            # Volumes montés
      - ./data:/data
    environment:        # Variables d’environnement
      - TZ=Europe/Paris
    networks:           # Réseaux utilisés
      - mon_reseau
networks:               # (optionnel) Définition des réseaux
  mon_reseau:
    driver: bridge
volumes:                # (optionnel) Volumes persistants
  data_volume:
```

⚠️ Erreurs fréquentes dans un fichier docker-compose.yml

❌ 1. Mauvaise indentation (espaces vs tabulations)

Le YAML n'accepte pas les tabulations. ➡️ Utiliser 2 espaces par niveau. Une mauvaise indentation = erreur immédiate.

❌ 2. Chaînes non mises entre guillemets

Certains ports ou chemins doivent être entre guillemets :

```
ports:
  - "8080:80"
```

`Sans guillemets → interprétation incorrecte.`

❌ 3. Mauvais niveau de hiérarchie

Exemple d'erreur courante :

```
services:
  web:
  image: nginx   # ❌ Mauvais niveau
```

➡️ `image doit être sous web.`

❌ 4. Volumes mal définis

Deux erreurs fréquentes :
Volume non déclaré dans la section volumes:
Chemin local inexistant
Exemple correct :

```
volumes:
  data_volume:
```

❌ 5. Variables d'environnement mal formatées

Erreur :

```
environment:
  TZ Europe/Paris   # ❌
```

Correct :

```
environment:
  - TZ=Europe/Paris
```

❌ 6. Ports déjà utilisés

Si un autre service utilise déjà 8080, le conteneur ne démarre pas.
➡️ Vérifier avec docker ps ou changer le port externe.

❌ 7. Mauvaise version ou syntaxe obsolète

Certaines anciennes syntaxes (links:, version: obligatoire) ne sont plus nécessaires.
➡️ Utiliser les formats Compose récents.

🎯 À retenir

Le YAML est strict : indentation, guillemets, hiérarchie.
Les erreurs viennent souvent de petits détails.
Toujours valider le fichier avec : docker compose config

13. comment lancer un fichier docker-compose.yml dans le terminal intégré de Docker Desktop.

01 - Ouvrir le terminal Docker Desktop
Docker Desktop intègre un terminal prêt à l'emploi pour exécuter les commandes Docker et Compose.
Docker Desktop → Menu de gauche → Terminal
Vérifie que Docker Desktop est bien démarré
Le terminal s'ouvre dans un environnement Docker préconfiguré

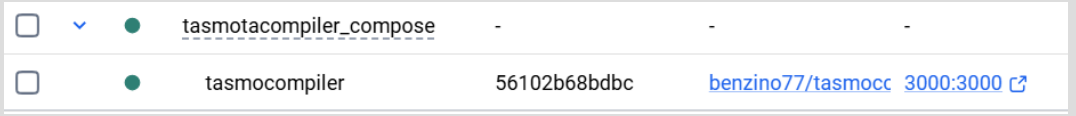
02 - Naviguer jusqu'au dossier contenant docker-compose.yml
La commande doit être exécutée dans le répertoire où se trouve ton fichier Compose.
Commande : cd chemin/vers/ton/projet
Utilise ls pour vérifier que docker-compose.yml est bien présent
Exemple : cd C:/docker_compose/mon-projet

03 - Lancer les services avec Docker Compose
Commande principale
Cette commande démarre tous les services définis dans le fichier Compose.
Commande : docker compose up -d
up démarre les conteneurs
-d les exécute en arrière-plan (mode détaché)
Docker Desktop affichera ensuite les conteneurs dans l'onglet Containers

04 - Vérifier que les conteneurs fonctionnent
Une fois lancés, tu peux contrôler l'état des services.
Commande : docker compose ps
Affiche la liste des conteneurs du projet
Vérifie que le statut est running
Tu peux aussi aller dans Docker Desktop → Containers

05 - Arrêter les services si nécessaire
Optionnel
Permet d'arrêter proprement tous les conteneurs du projet.
Commande : docker compose down
Arrête et supprime les conteneurs
Ne supprime pas les volumes sauf si tu ajoutes --volumes

🎯 À retenir
Docker créer un stack compose
Pour regrouper tous les containers du fichier docker-compose.yml



14. exemple d’installation simple : Tasmotacompiler

📌 Qu’est-ce que Tasmotacompiler ?

TasmoCompiler est une interface web simple qui vous permet de compiler un firmware Tasmota fantastique avec vos propres paramètres :

Pourquoi ?

Avoir une solution plus facile à utiliser (quelques clics seulement) et qui ne nécessite pas de connaissances sur l’installation d’un environnement de développement pour construire un firmware personnalisé.

Pour plus info sur Web GUI for custom Tasmota compilation · GitHub

📝 Structure du fichier docker-compose.yml

Exemple simple avec benzino77/tasmocompiler

```
version: "3.8"
services:
  tasmocompiler:
    image: benzino77/tasmocompiler:latest
    container_name: tasmocompiler
    ports:
      - "3000:3000"
    volumes:
      - tasmocompiler_data:/data
    restart: unless-stopped

volumes:
  tasmocompiler_data:
```

🗂️ créer l’emplacement des fichiers compose

Sur C : créer un dossier Docker_compose

Dans ce dossier créer un dossier pour contenir le fichier docker-compose,Yml

Dans notre exemple créer un dossier tasmocompiler_compose

⚠️ Avertissement important

Le nom des dossiers ne doit pas contenir d’espace

📝création du fichier docker-compose.yml

Dans le dossier, créer un nouveau fichier texte

Renommer le fichier texte en docker-compose.yml

Modifier le fichier avec le bloc-note

Copier ou saisie la structure du fichier

Enregistrer le fichier

🖥️ Comment l’utiliser dans Docker Desktop

Ouvrir Docker Desktop

Cliquer sur terminal dans la barre en bas a droite

Taper cd + path dossier ici cd C:\docker_compose\tasmotacompiler_compose

Taper la commande **docker compose up -d**

Le terminal exécute les taches demandés puis indique le résultat

✔️ **Image benzino77/tasmocompiler** **Pulled**
✔️ **Network tasmotacompiler_compose_default** **Created**
✔️ **Volume tasmotacompiler_compose_tasmocompiler_data** **Created**
✔️ **Container tasmocompiler** **Started**

🌍 Tester le résultat

Ouvrir un navigateur : http://localhost:3000

15.1 exemple d’installation complexe : Inventree

📦 InvenTree

InvenTree est un logiciel open-source de gestion d’inventaire conçu pour les environnements techniques, industriels et électroniques. Il permet de centraliser, organiser et suivre toutes les pièces, composants, stocks, fournisseurs et commandes d’un projet ou d’une entreprise.

Il se distingue par :

- une interface web moderne, simple et efficace
- une API REST complète pour l’intégration d’autres outils
- la gestion avancée des nomenclatures et des assemblages
- le suivi des stocks, des emplacements, des mouvements
- un système de plugins pour étendre les fonctionnalités
- une installation facile via Docker Compose

InvenTree est particulièrement apprécié dans les domaines :

- électronique / fabrication
- prototypage
- maintenance
- laboratoires techniques
- makerspaces

En résumé :

➡️ InvenTree est une solution puissante, flexible et moderne pour gérer efficacement un inventaire technique.

🏗️ 1. Architecture générale

InvenTree repose sur une architecture modulaire composée de trois blocs principaux :

Backend Django Gère la logique métier, les modèles, les API REST, l’authentification et les tâches internes.

Base de données PostgreSQL Stocke toutes les informations : pièces, stocks, fournisseurs, commandes, utilisateurs…

Cache Redis Accélère les opérations (sessions, notifications, tâches asynchrones).

📦 2. Les composants internes clés

🔹 Modèles (Models)

Définissent les objets métier : Part, StockItem, Supplier, BuildOrder, Location…

Représentent la structure de la base de données.

🔹 API REST

Accessible via /api/

Permet l’intégration avec d’autres systèmes (ERP, scripts Python, automatisations).

🔹 Interface Web (Frontend)

Basée sur Django + Bootstrap

Affiche les vues : inventaire, pièces, commandes, rapports…

🔹 Worker (Tâches asynchrones)

Exécute les tâches longues : génération de rapports PDF, synchronisation, import/export, notifications.

🗂️ 3. Organisation des données

Fichiers utilisateurs : images, pièces jointes, rapports → stockés dans /home/inventree/data

Base PostgreSQL : données structurées

Cache Redis : données temporaires

🔧 4. Extensions et plugins

InvenTree supporte un système de plugins permettant d’ajouter :

intégrations externes (ERP, fournisseurs, API)

automatisations

webhooks

scripts personnalisés

🎯 À retenir

InvenTree = Backend Django + PostgreSQL + Redis + Worker

Architecture modulaire, extensible et orientée API

Séparation claire entre données, fichiers, logique métier et tâches asynchrones

15.1 création des containers

📌 Objectif

Déployer InvenTree facilement grâce à un fichier docker-compose.yml

➡️ Tous les fichiers persistants sont stockés dans : C:\docker_compose\inventree

🗂️ 1. créer l'emplacement des fichiers compose

Sur C : créer un dossier **Docker_compose**
Dans ce dossier créer un dossier **inventree** pour contenir le fichier docker-compose,Yml
Dans le dossier **inventree**
créer un dossier **db-postgres**
créer un dossier **redis-cache**
créer un dossier **data**

⚠️ Avertissement important

Le nom des dossiers ne doit pas contenir d'espace

📝2. création du fichier docker-compose.yml

Dans le dossier **inventree**, créer un nouveau fichier texte
Renommer le fichier texte en docker-compose.yml
Modifier le fichier avec le bloc-notte
Copier ou saisie la structure du fichier
Enregistrer le fichier

▶️ 3. Lancer l'installation

Dans Docker Desktop → Terminal :

```
cd C:\docker_compose\inventree
docker compose up -d
```

Vérifier résultat

```
✔️ Image inventree/inventree:latest       Pulled
 ✔️ Image redis:7                          Pulled
 ✔️ Image postgres:15                      Pulled
 ✔️ Network inventree_default
Created
 ✔️ Container inventree-inventree-db-1
Started
 ✔️ Container inventree-inventree-cache-1
Started
 ✔️ Container inventree-inventree-server-1
Started
 ✔️ Container inventree-inventree-worker-1
Started
```

▶️ 4. Lancer l'installation de la base de donnée

Dans Docker Desktop → Terminal :
docker compose run --rm inventree-server invoke update

▶️ 5. Lancer la creation du SuperUser

Dans Docker Desktop → Terminal

```
docker compose run --rm \
  -e INVENTREE_SUPERUSER_NAME=adminuser \
  -e INVENTREE_SUPERUSER_EMAIL=admin@example.com \
  -e
INVENTREE_SUPERUSER_PASSWORD=MonSuperMotDePass
123! \
  inventree-server invoke superuser
```

🌍 Tester le résultat


```
Version: "3.9"

services:

  inventree-db:
    image: postgres:15
    environment:
      POSTGRES_DB: inventree
      POSTGRES_USER: inventree
      POSTGRES_PASSWORD: motdepassefort
    volumes:
      - ./db-postgres:/var/lib/postgresql/data

  inventree-cache:
    image: redis:7
    volumes:
      - ./redis-cache:/data

  inventree-server:
    image: inventree/inventree:latest
    depends_on:
      - inventree-db
      - inventree-cache
    ports:
      - "8081:8000"
    Environment:
      INVENTREE_SITE_URL: http://localhost:8081
      INVENTREE_DB_ENGINE: postgresql
      INVENTREE_DB_HOST: inventree-db
      INVENTREE_DB_NAME: inventree
      INVENTREE_DB_USER: inventree
      INVENTREE_DB_PASSWORD: motdepassefort
      INVENTREE_CACHE_HOST: inventree-cache
      INVENTREE_EXT_VOLUME: /home/inventree/data
    volumes:
      - ./data:/home/inventree/data

  inventree-worker:
    image: inventree/inventree:latest
    depends_on:
      - inventree-db
      - inventree-cache
    Environment:
      INVENTREE_SITE_URL: http://localhost:8001
      INVENTREE_DB_ENGINE: postgresql
      INVENTREE_DB_HOST: inventree-db
      INVENTREE_DB_NAME: inventree
      INVENTREE_DB_USER: inventree
      INVENTREE_DB_PASSWORD: motdepassefort
      INVENTREE_CACHE_HOST: inventree-cache
      INVENTREE_EXT_VOLUME: /home/inventree/data
    volumes:
      - ./data:/home/inventree/data
```

15.2 stack compose

⚠️ Avertissement important

Il a été crée un stack compose de 4 containers qui doivent fonctionner ensemble
Il est possible de les arrêter ensemble en cliquant sur le premier carré bleu

Il est possible de les redémarrer en cliquant sur le premier triangle blanc