Des fondamentaux de la conteneurisation aux bases de Docker

Philippe HORTOLLAND

- Séminaire RAISIN -

18 juin 2026

Sommaire

Objectifs de la présentation :

- ✓ Comprendre les bases des **conteneurs** et de **Docker**
- ✓ Analyser un **Dockerfile** (exemple : **n8n**)
- ✓ Installer **Docker** dans nos environnements **(Windows, macOS et Linux)**
- ✓ Lancer nos premiers **conteneurs** (exemple : **Ollama**)
- ✓ Utiliser **Docker-Compose** pour orchestrer plusieurs conteneurs (exemple : **GLPI et phpMyAdmin**)

- ✓ Connaître les commandes essentielles de **Docker** et **Docker Compose**

Partie 1 : les conteneurs

Qu'est-ce qu'un **conteneur** ?

Définition :

- Un **conteneur** est une **enveloppe virtuelle** qui permet de distribuer une application avec tous les éléments dont elle a besoin pour fonctionner.

- Contrairement à une **machine virtuelle**, le **conteneur** n'intègre pas de **noyau (kernel)**, il s'appuie directement sur le noyau de l'ordinateur sur lequel il est déployé.

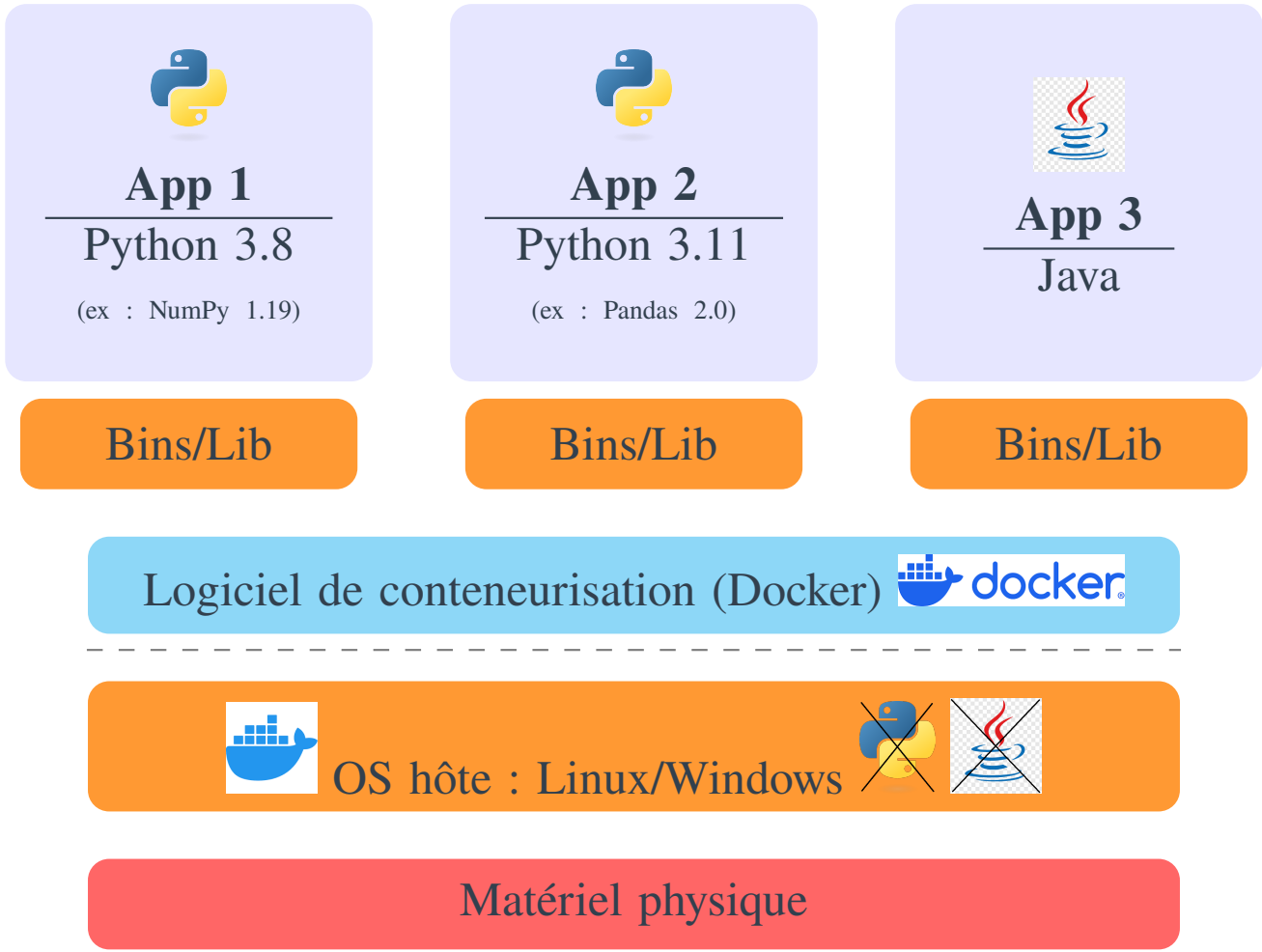
Source : `https://fr.wikipedia.org/wiki/Conteneur_(informatique)`

Propriétés des conteneurs

Propriétés des conteneurs :

- **Isolation** : Chaque conteneur fonctionne de manière isolée.
- **Légèreté** : Les conteneurs ne nécessitent pas de système d'exploitation dédié, ce qui les rend **plus rapides à démarrer** et **moins gourmands en ressources**.
- **Portabilité** : Un conteneur peut être déployé sur n'importe quel système prenant en charge une technologie de conteneurisation (**Docker**, **Podman**, **LXC**, etc.) : **Si tout fonctionne localement, cela fonctionnera partout !**
- **Reproductibilité** : Grâce aux **images** de conteneurs, il est possible de recréer **exactement le même environnement** sur différentes machines.
- **Scalabilité** : Facile à dupliquer, à orchestrer (avec **Kubernetes**, **Docker Swarm**, etc.) et à intégrer dans des pipelines **CI/CD**.

Principe d'encapsulation d'applications avec les conteneurs



(Chaque conteneur isole une version d'application et ses dépendances)

Notions de **virtualisation lourde** et de **virtualisation légère**

Caractéristiques des machines virtuelles (virtualisation lourde) :

- Ressources du **système hôte** (**CPU/GPU/RAM**) **totalement dédiées** (nécessité de recréer un système complet).
- **Avantage** : Sans partage du **kernel**, il est possible d'installer un **système Windows ou BSD** sur un système hôte Linux.

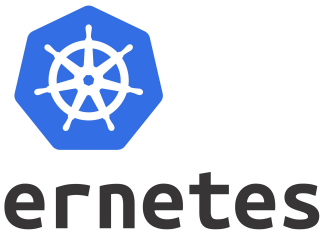
Caractéristiques des conteneurs (virtualisation légère) :

- Technologie permettant d'**isoler une application** en **partageant les ressources du système hôte** : plus léger, plus facile à contrôler.
- Notions de **Cgroups** et **Namespaces** (analogie : colocation).
  - **Cgroups** : Contrôle et limite les ressources (**CPU**, **GPU**, etc.).
  - **Namespace** : Isole les conteneurs les uns des autres.

Solutions associées aux **conteneurs**

Quelques solutions associées à la technologie des conteneurs :

- **Docker** : **Plateforme la plus répandue** pour créer, gérer et déployer des conteneurs.
- **Podman** : Alternative à Docker (n'utilise pas de **démon**, mieux sécurisé).
- **LXC** : **Conteneurs Linux natifs**.
- **Kubernetes** : **Orchestration de conteneurs à grande échelle**.



Partie 2 : Qu'est-ce que Docker ?

Docker : Moteur de conteneurisation

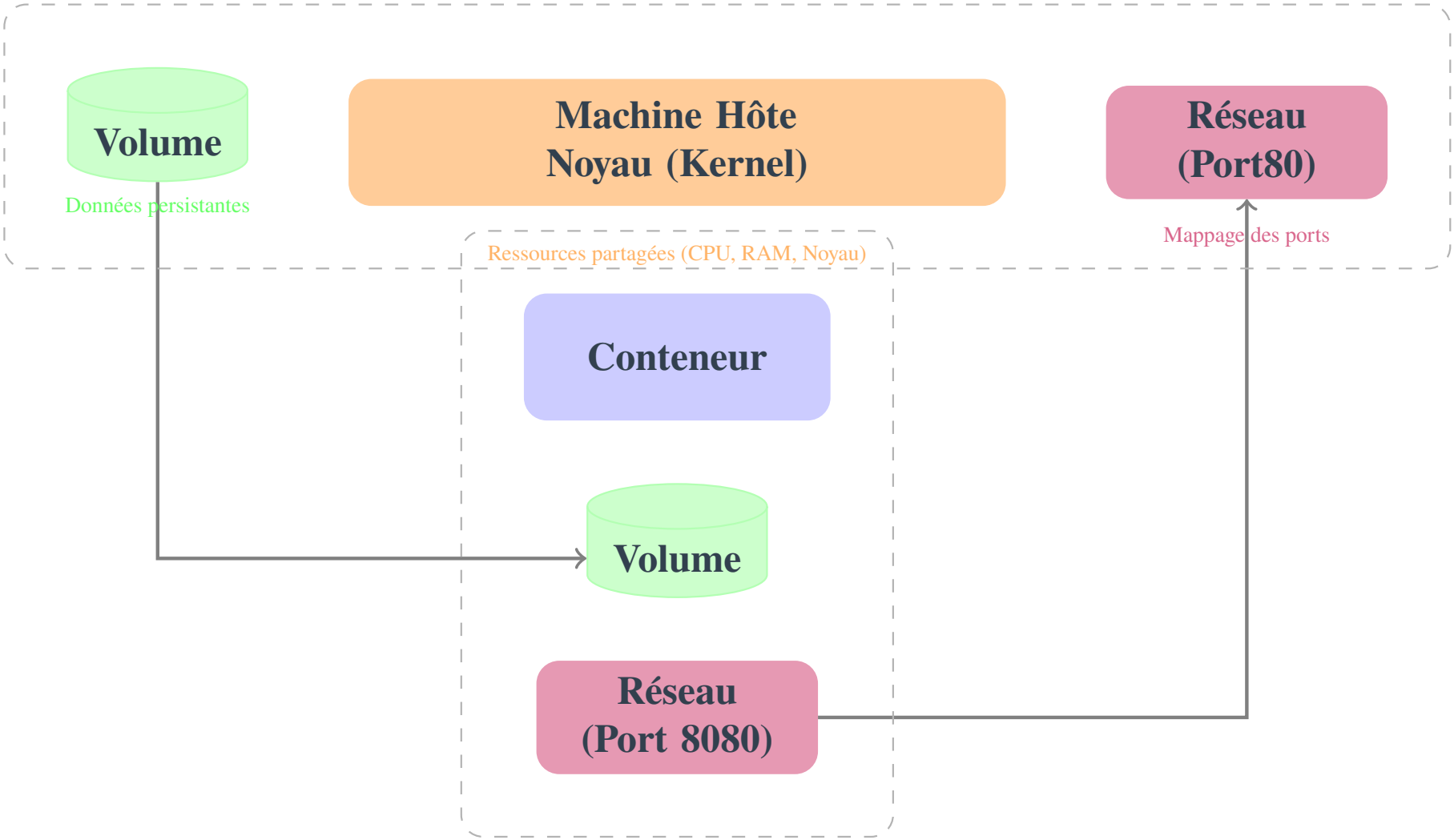
Caractéristiques principales :

(les conteneurs existent depuis bien plus longtemps que Docker.)

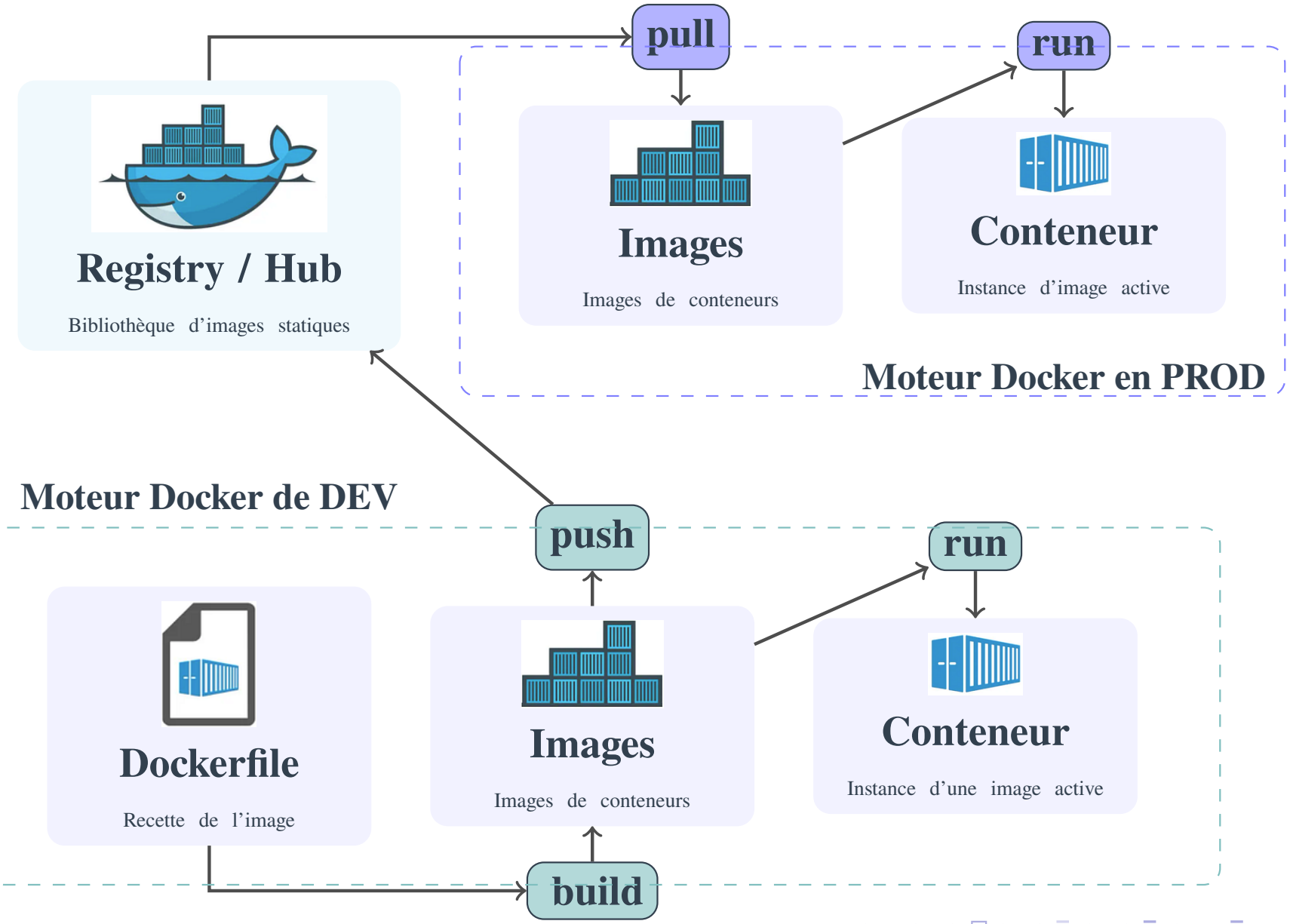
- **Docker** est un **Moteur de conteneurisation** (le plus utilisé)
- Solution **Open Source** (Auteur : Solomon Hykes, franco-américain) - Distribuée par l'entreprise **Docker Inc.** (depuis mars 2013).
- Dans la vision Docker, un conteneur ne fait tourner **qu'un seul processus**.
- **Particularité des conteneurs Docker** : Ils sont **immuables** (**stateless**) et ne stockent pas de données persistantes.
  Ils reprennent leur **état initial** à chaque redémarrage.

➤ **Lorsqu'un conteneur utilise une base de données, on utilise un volume persistant pour stocker les données**

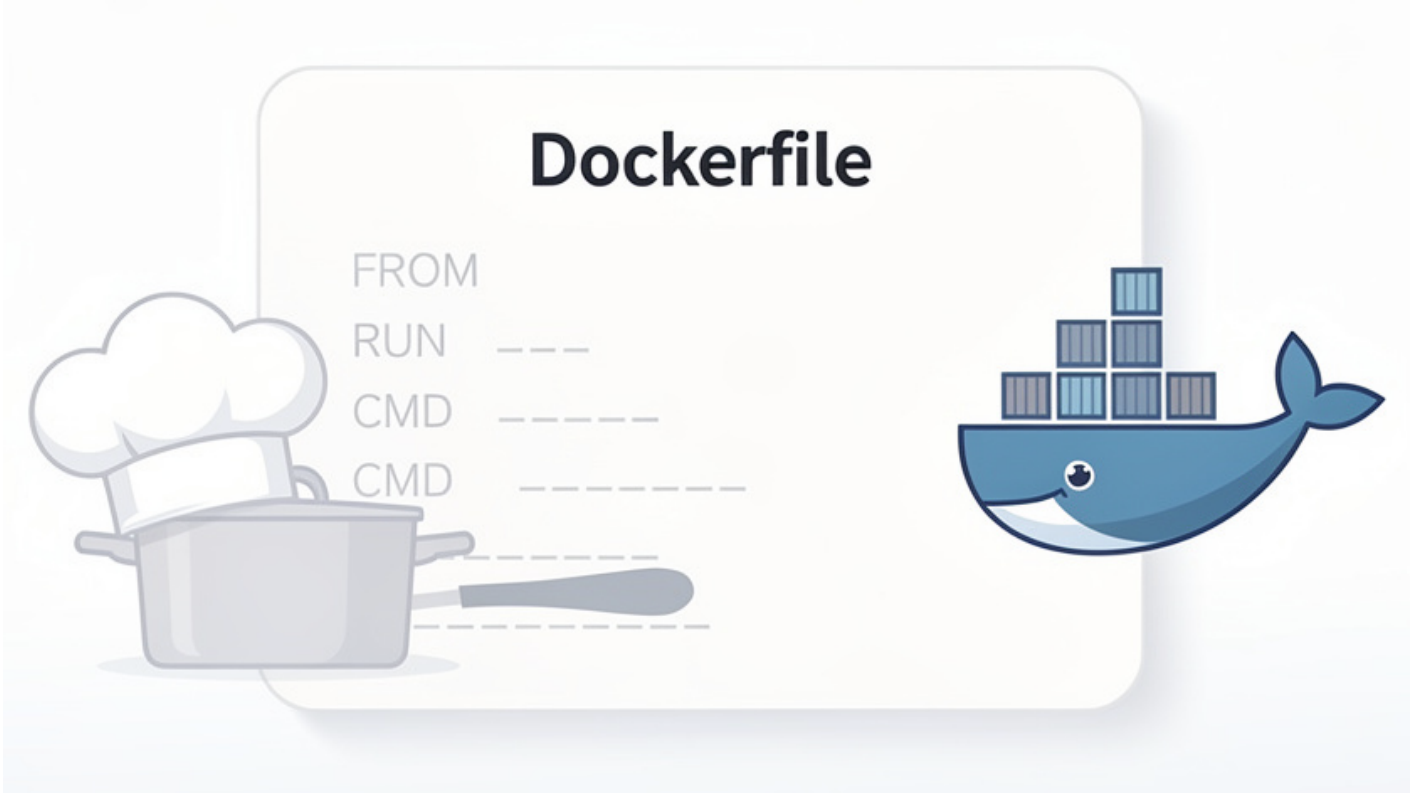
Fonctionnement de Docker : Volumes, Réseau, Ressources



Cycle de vie des conteneurs Docker



Partie 3 : Comprendre un Dockerfile



Comprendre un **Dockerfile**

- Un **Dockerfile** est comme une **recette de cuisine** permettant de concevoir une **image Docker**.
- La conception d'un **Dockerfile** relève davantage des compétences des équipes **DevOps** que des **administrateurs système et réseau (ASR)**.
- Il est tout de même utile de savoir **lire et modifier** un **Dockerfile**.
- Les recettes des **Dockerfiles** se trouvent généralement sur **GitHub** ou des environnements similaires :

✗ **Exemple avec n8n** :
`https://github.com/n8n-io/n8n/blob/master/docker/images/n8n/Dockerfile`

Structure d'un Dockerfile - Ex : n8n - Les instructions

Code | Blame | 38 lines (30 loc) · 1.26 KB

```
1   ARG NODE_VERSION=24.14.1
2   ARG N8N_VERSION=snapshot
3
4   # Rebuild native addons for the container platform. The base image has no
5   # package manager or build tools, so we compile in a throwaway builder stage.
6   FROM node:${NODE_VERSION}-alpine3.22 AS builder
7   COPY ./compiled /usr/local/lib/node_modules/n8n
8   RUN apk add --no-cache python3 make g++ && \
9       cd /usr/local/lib/node_modules/n8n && \
10      npm rebuild sqlite3 isolated-vm
11
12  FROM n8nio/base:${NODE_VERSION}
13
14  ARG N8N_VERSION
15  ARG N8N_RELEASE_TYPE=dev
16  ENV NODE_ENV=production
17  ENV N8N_RELEASE_TYPE=${N8N_RELEASE_TYPE}
18  ENV SHELL=/bin/sh
19
20  WORKDIR /home/node
21
22  COPY --from=builder /usr/local/lib/node_modules/n8n /usr/local/lib/node_modules/n8n
23  COPY docker/images/n8n/docker-entrypoint.sh /
24
25  RUN ln -s /usr/local/lib/node_modules/n8n/bin/n8n /usr/local/bin/n8n && \
26      mkdir -p /home/node/.n8n && \
27      chown -R node:node /home/node && \
28      rm -rf /root/.npm /tmp/*
29
30  EXPOSE 5678/tcp
31  USER node
32  ENTRYPOINT ["tini", "--", "/docker-entrypoint.sh"]
33
34  LABEL org.opencontainers.image.title="n8n" \
35        org.opencontainers.image.description="Workflow Automation Tool" \
36        org.opencontainers.image.source="https://github.com/n8n-io/n8n" \
37        org.opencontainers.image.url="https://n8n.io" \
38        org.opencontainers.image.version=${N8N_VERSION}
```

Structure d'un Dockerfile - Ex : n8n - Notion de couches

Code Blame 38 lines (30 loc) · 1.26 KB

```
1   ARG NODE_VERSION=24.14.1
2   ARG N8N_VERSION=snapshot
3
4   # Rebuild native addons for the container platform. The base image has no
5   # package manager or build tools, so we compile in a throwaway builder stage.
6   FROM node:${NODE_VERSION}-alpine3.22 AS builder
7   COPY ./compiled /usr/local/lib/node_modules/n8n
8   RUN apk add --no-cache python3 make g++ && \
9       cd /usr/local/lib/node_modules/n8n && \
10      npm rebuild sqlite3 isolated-vm
11
12  FROM n8nio/base:${NODE_VERSION}
13
14  ARG N8N_VERSION
15  ARG N8N_RELEASE_TYPE=dev
16  ENV NODE_ENV=production
17  ENV N8N_RELEASE_TYPE=${N8N_RELEASE_TYPE}
18  ENV SHELL=/bin/sh
19
20  WORKDIR /home/node
21
22  COPY --from=builder /usr/local/lib/node_modules/n8n /usr/local/lib/node_modules/n8n
23  COPY docker/images/n8n/docker-entrypoint.sh /
24
25  RUN ln -s /usr/local/lib/node_modules/n8n/bin/n8n /usr/local/bin/n8n && \
26      mkdir -p /home/node/.n8n && \
27      chown -R node:node /home/node && \
28      rm -rf /root/.npm /tmp/*
29
30  EXPOSE 5678/tcp
31  USER node
32  ENTRYPOINT ["tini", "--", "/docker-entrypoint.sh"]
33
34  LABEL org.opencontainers.image.title="n8n" \
35        org.opencontainers.image.description="Workflow Automation Tool" \
36        org.opencontainers.image.source="https://github.com/n8n-io/n8n" \
37        org.opencontainers.image.url="https://n8n.io" \
38        org.opencontainers.image.version=${N8N_VERSION}
```

Les instructions Dockerfile

Instruction	Description	Exemple
`FROM`	Image de base	`FROM ubuntu:22.04`
`RUN`	Exécute une commande	`RUN apt update`
`COPY`	Copie des fichiers	`COPY app.py /app/`
`ADD`	Copie/extrait archives	`ADD file.tar.gz /`
`WORKDIR`	Répertoire de travail	`WORKDIR /app`
`ENV`	Variable d'environnement	`ENV VAR=value`
`CMD`	Commande par défaut	`CMD ["python", "app.py"]`
`ENTRYPOINT`	Point d'entrée	`ENTRYPOINT ["python"]`
`EXPOSE`	Ports à exposer	`EXPOSE 8080`
`VOLUME`	Point de montage	`VOLUME /data`

Build d'un **Dockerfile**

Important :

- La commande `RUN` définit le nombre de **couches** (**layers**) créées dans une **image Docker**.

Avantage :

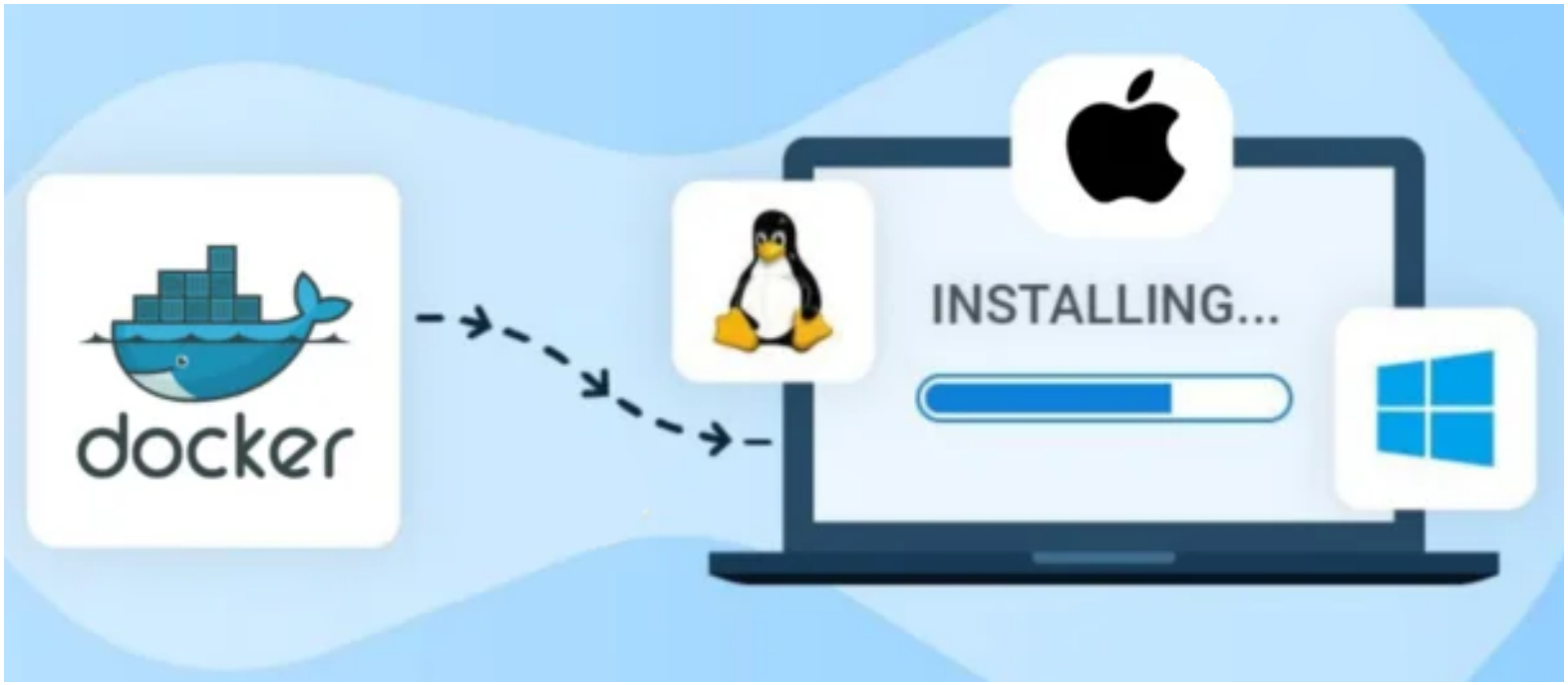
- Lors de la mise à jour d'une image, **seules les couches modifiées sont téléchargées**, ce qui évite de télécharger une image complète à chaque fois.

Commande pour créer une image Docker à partir d'un Dockerfile :

```
docker build -t docker-build .
```

- L'argument `-t` permet de **donner un nom** à l'image Docker.
- Le point final `.` représente le répertoire où se trouve le **Dockerfile** (ici à la racine du projet) : **à ne pas oublier !**

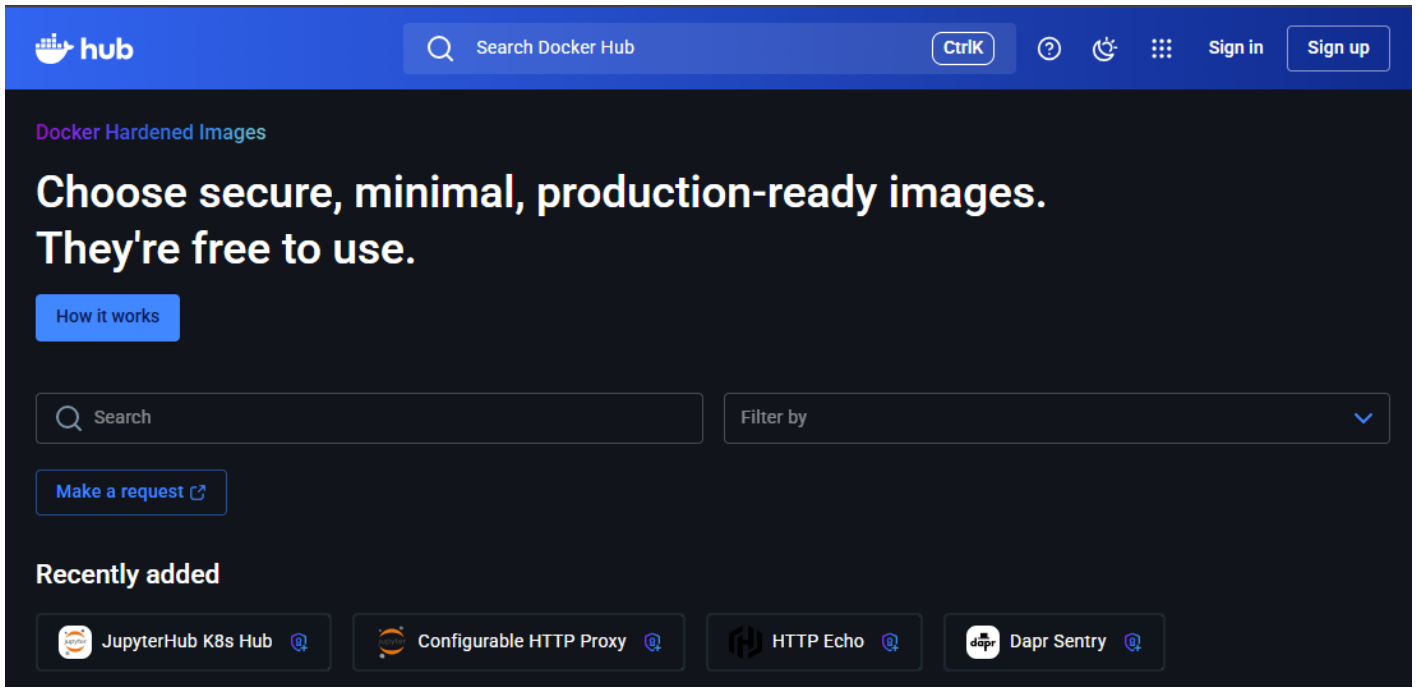
Partie 4 : Installation de Docker



Connexion au **Docker Hub**

Docker Hub :

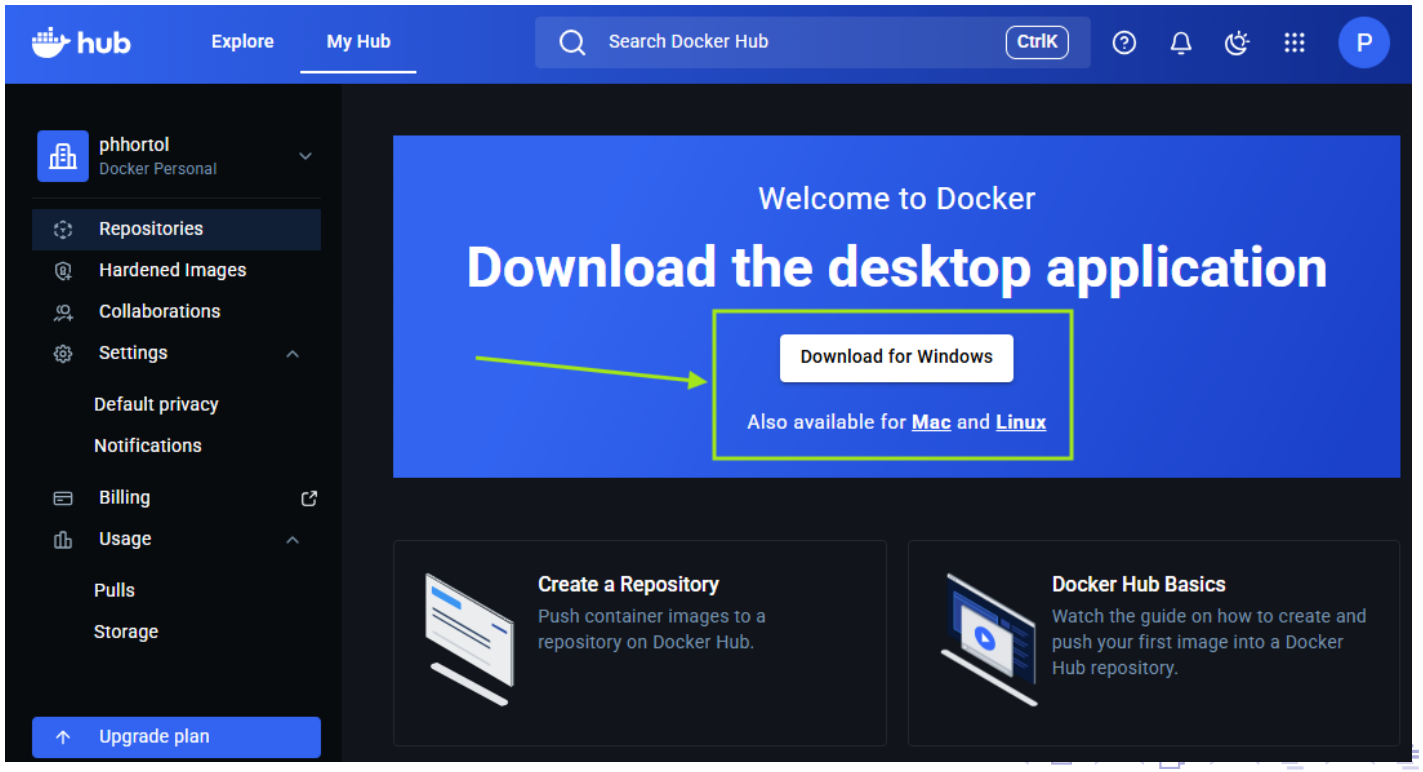
- Site web : https://hub.docker.com/
- La création d'un compte est **gratuite** et **recommandée**.
- Docker Hub propose un large **catalogue d'images**.



Installation de **Docker**

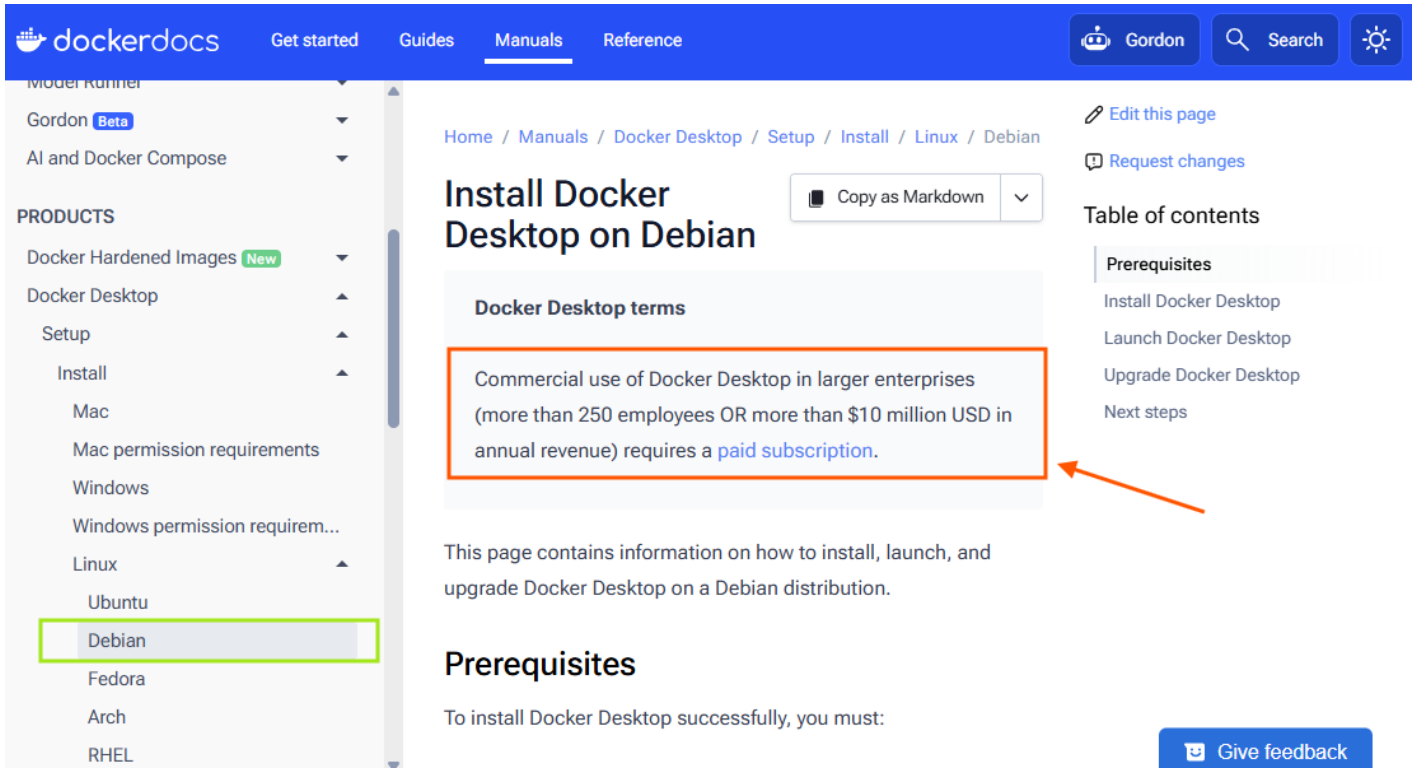
Des versions disponibles pour toutes les plateformes :

- La connexion au compte donne accès au téléchargement de **Docker Desktop** pour **macOS** et **Windows** (**DEV**).
- **Docker CE** et **Docker Enterprise** (**Linux**).
- Dans nos contextes **ASR**, nous utilisons (en **PROD**) la **version communautaire (CE Linux)** de Docker.



Installation de **Docker** (version community) sous **Linux**

- Ici, la création d'un compte sur **hub.docker** n'est pas nécessaire.
- Plusieurs solutions sont proposées pour installer **Docker sous Linux**.
- Il est recommandé d'installer Docker via le **dépôt apt**.



Points clés de l'installation (1/2)

En quelques étapes : Code à copier-coller directement depuis le site !

https://docs.docker.com/engine/install/debian/#install-using-the-repository

- Ajout de la **clé GPG** du **dépôt Docker**.
- Ajout du **dépôt officiel Docker** à la liste des **sources APT**.

```
# Add Docker's official GPG key:
sudo apt update
sudo apt 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:
sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/debian
Suites: $(. /etc/os-release && echo "$VERSION_CODENAME")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt update
```

Points clés de l'installation (2/2)

- Installation des paquets **Docker** ; on en profite pour installer les paquets **Docker Compose** :

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

- **Docker** est maintenant installé ! **Le service démarre automatiquement** après l'installation.

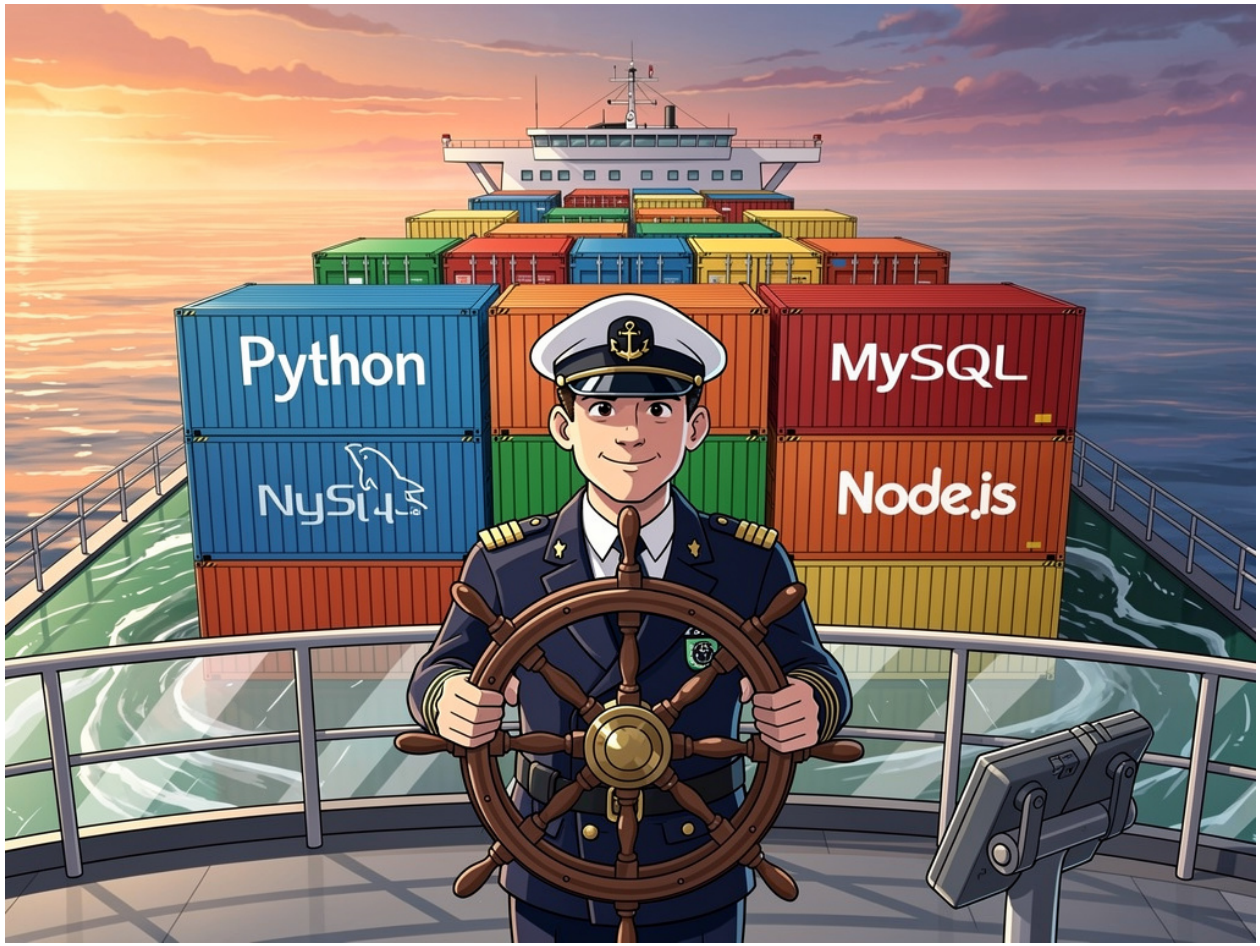
- Pour vérifier que **Docker** est en cours d'exécution, utilisez cette commande :

```
sudo systemctl status docker
```

- Une fois **Docker** installé, il faut **accorder les accès** à l'utilisateur l'autorisant à discuter avec le **démon (daemon)** :

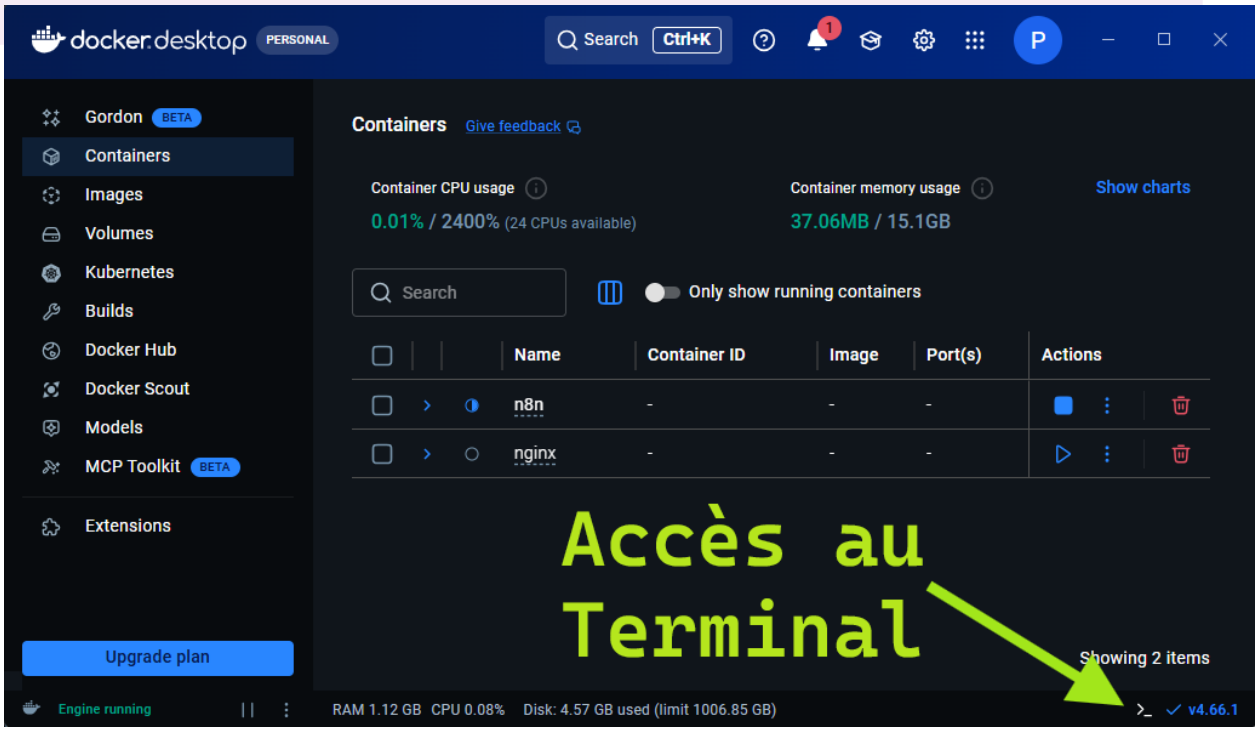
```
sudo usermod -aG docker utilisateur
```

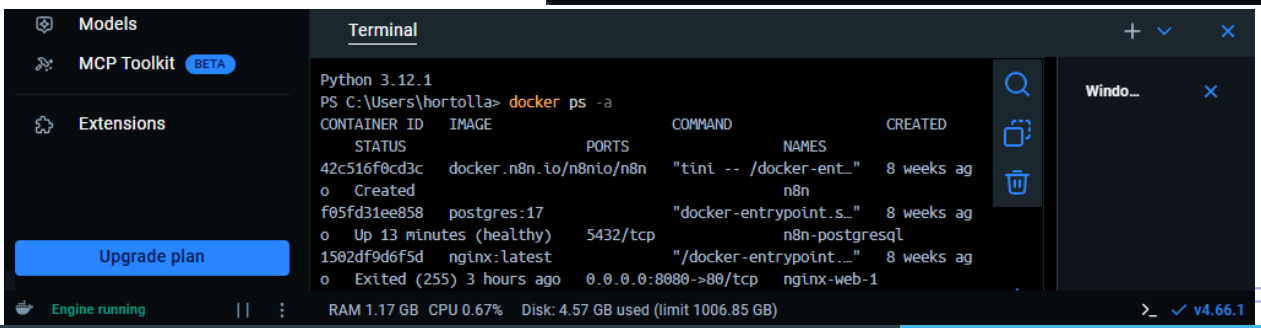
Partie 5 : Prise en main de Docker



Prise en main de **Docker**

- On manipulera de préférence **Docker** à partir du **terminal**.
- Via **Docker Desktop** (**macOS** et **Windows**) ou **Shell** sous **Linux**.

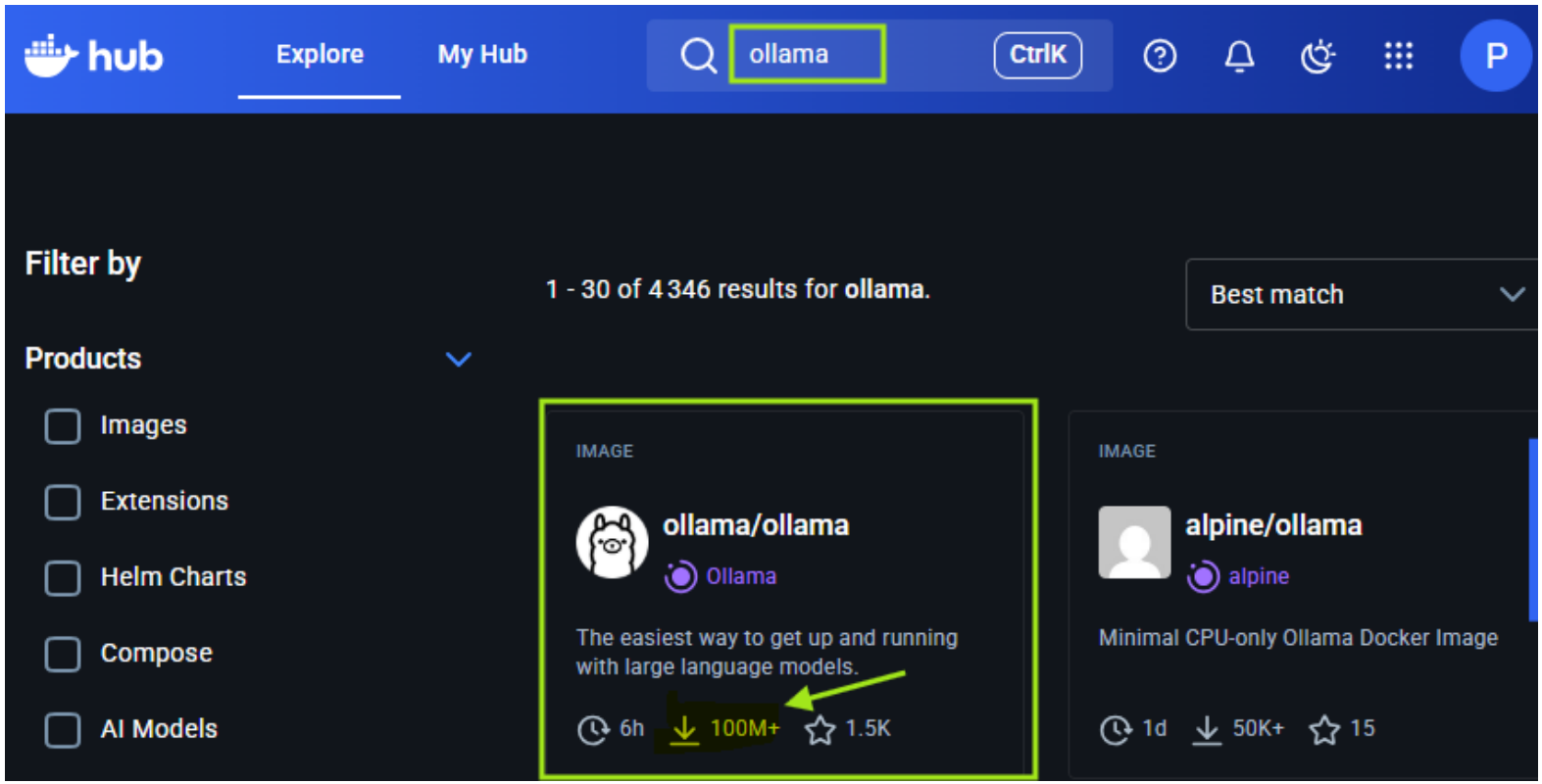




Pull d'une **image Docker**

Cas pratique : Exploitation d'une **image** à partir de **DockerHub** :

- Exemple avec l'application **Ollama** !
- Plusieurs images sont proposées, on repère facilement l'**image officielle d' ollama** par son **nombre de téléchargements** !



Pull d'une **image Docker**

- Choisir de préférence l'**image la plus récente** (**latest**) sauf cas particuliers !
- La commande `docker pull ollama/ollama` permet de télécharger l'**image Docker officielle** d'**Ollama** depuis le **registre Docker Hub** vers notre **machine locale**.


```
PS C:\Users\hortolla> docker pull ollama/ollama
Using default tag: latest
latest: Pulling from ollama/ollama
b40150c1c271: Extracting [==================================================>    ]  27.85MB/29.73MB
2d0e9b7d523c: Downloading [=====================>                              ]  46.57MB/102.4MB
ed7bdd433780: Download complete
f79a5fdbd072: Downloading [>                                                  ]  22.46MB/3.618GB
```

Quelques commandes de base : exemple avec Ollama

- On **vérifie la présence de l'image** dans notre environnement :

  `docker images`

- Pour **supprimer cette image**, on utiliserait la commande :

  `docker rmi ollama/ollama`

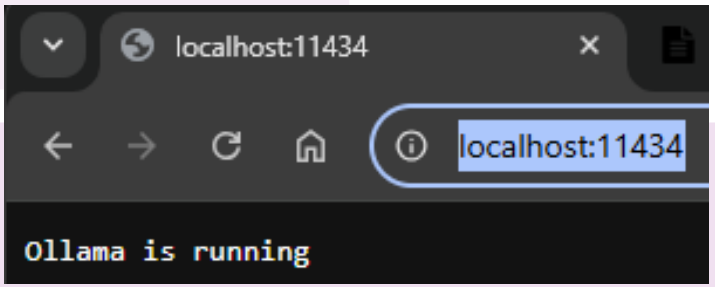
```
PS C:\Users\adminlocal> docker images
IMAGE                  ID             DISK USAGE   CONTENT SIZE   EXTRA
kerberos:latest        2adf94ae5c22       195MB             0B    U
ldap:latest            1f891aaf0a7b       161MB             0B    U
nginx:latest           22bd15417453       192MB             0B    U
ollama/ollama:latest   7e8ee88de3cc      6.55GB             0B
PS C:\Users\adminlocal> docker rmi 7e8ee88de3cc
Untagged: ollama/ollama:latest
Untagged: ollama/ollama@sha256:6077dbbd6508dce8973f8b91c30d8026b1279ab0483e15f0dfad469dba676c2f
Deleted: sha256:7e8ee88de3ccdb59b8eb64fb16eb6432da07a67986550a40a305416d509489bf
Deleted: sha256:0a5834342f5a35d355da960519520ee572bd2ec8e259f126201492f7597baa83
Deleted: sha256:56ea25ac9b5aeca92a1c0083fcdc6353fd8e8ade2bd76b5d5ddf076764b3a22c
Deleted: sha256:99e3ffe0ab7a4b2f0631a837a39d1f3c27fe55abcda8a652fd0913b2bde927a9
Deleted: sha256:538812a4b9bd45adaac2b5e5b967daa6999aa44eb110aa32ae7c69702b906475
```

Commandes de base Docker : exemple avec Ollama

Utilisation de la commande :

```
docker run -d -v ollama:/root/.ollama -p
 11434:11434 -name ollama ollama/ollama
```

- On peut vérifier le bon fonctionnement d'**Ollama** via un navigateur internet.



Options utilisées :

- `-d` : Libère le shell après le démarrage du conteneur
- `-p 11434:11434` : Mappe le port 11434 de l'hôte au port 11434 du conteneur
- `-v ollama:/root/.ollama` : Associe un volume persistant pour les données Ollama
- `-name ollama` : Nomme le conteneur "ollama"

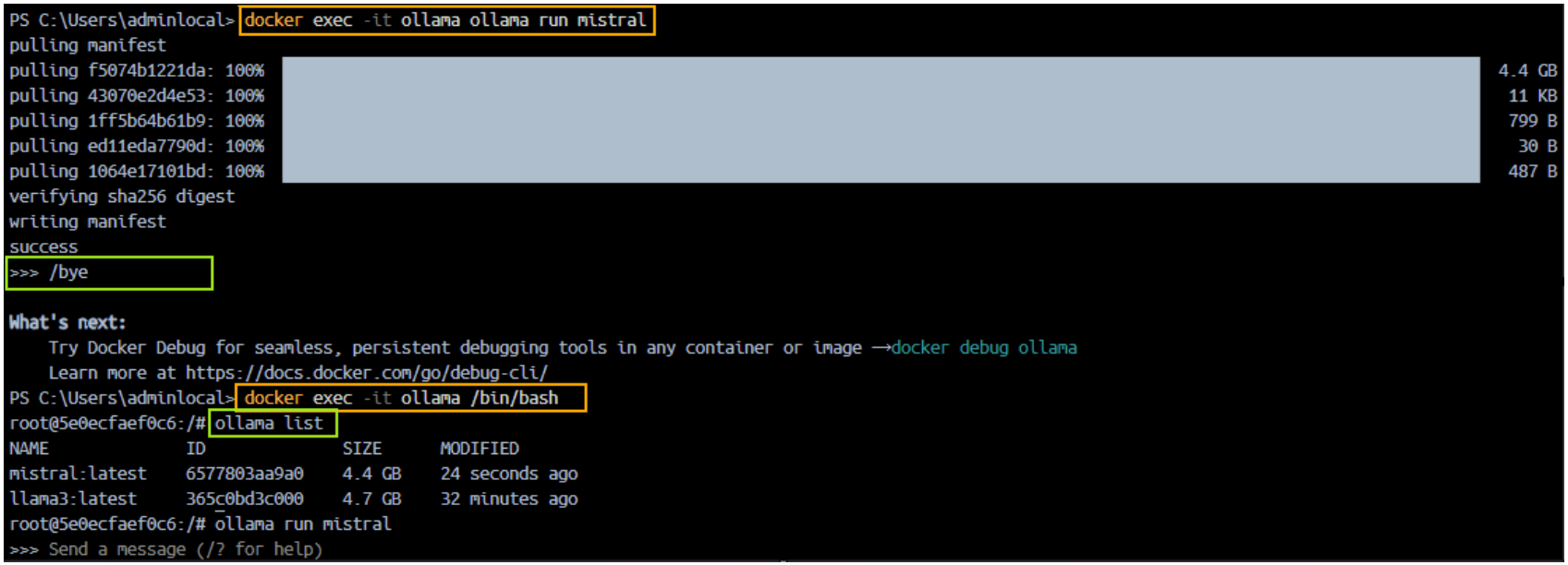
Commandes de base Docker : exemple avec Ollama

- `docker ps` : Liste les conteneurs Docker en cours d'exécution
- `docker ps -a` : Liste tous les conteneurs Docker (y compris ceux arrêtés)
- `docker stop ID` : Arrête un conteneur Docker spécifique
- `docker start ID` : Démarre un conteneur Docker spécifique
- `docker rm ID` : Supprime un conteneur Docker spécifique
- `docker system prune` : Nettoie les éléments inutilisés du système

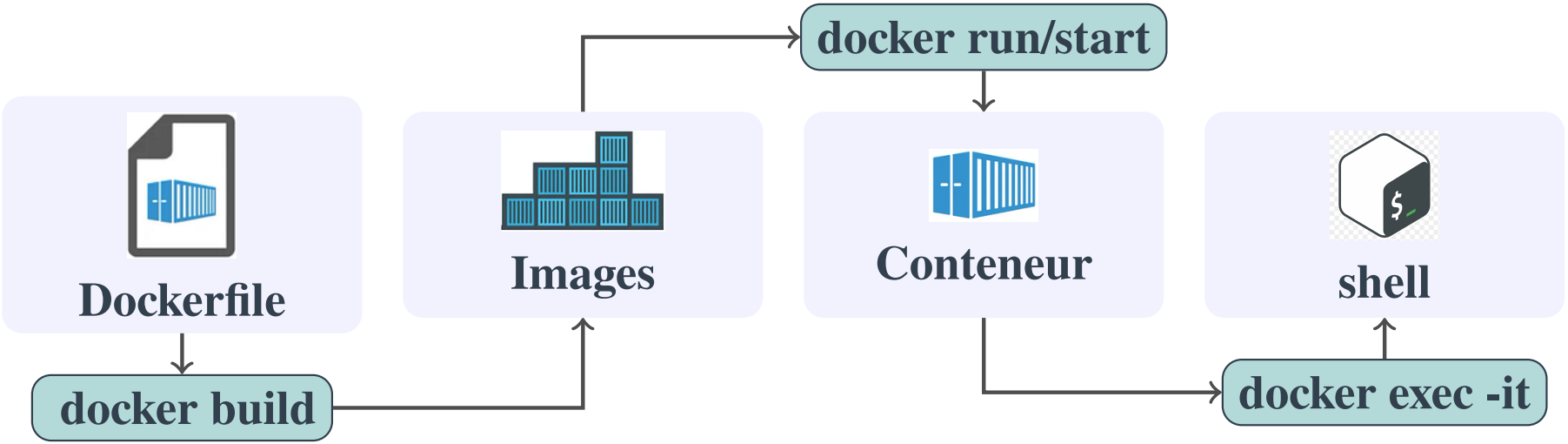
```
PS C:\Users\adminlocal> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                     PORTS
2f60335b0dee   ollama/ollama  "/bin/ollama serve"      25 hours ago   Up 25 hours                0.0.0.0:11434->11434/tcp, [::]:11434->11434/tcp
9502c9c5fdb7   nginx:latest   "/docker-entrypoint.…"   9 months ago   Exited (0) 9 months ago
688aa182b7eb   ldap           "/usr/local/bin/entr…"   15 months ago  Exited (1) 15 months ago
4e6c5968c115   kerberos       "/usr/sbin/krb5kdc -…"   15 months ago  Exited (0) 15 months ago
PS C:\Users\adminlocal> docker ps
CONTAINER ID   IMAGE          COMMAND             CREATED        STATUS        PORTS                                                NAMES
2f60335b0dee   ollama/ollama  "/bin/ollama serve" 25 hours ago   Up 25 hours   0.0.0.0:11434->11434/tcp, [::]:11434->11434/tcp   ollama
PS C:\Users\adminlocal> docker stop 2f60335b0dee
2f60335b0dee
PS C:\Users\adminlocal> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\adminlocal> docker rm 2f60335b0dee
2f60335b0dee
PS C:\Users\adminlocal> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                     PORTS     NAMES
9502c9c5fdb7   nginx:latest   "/docker-entrypoint.…"   9 months ago   Exited (0) 9 months ago              heuristic_lovelace
688aa182b7eb   ldap           "/usr/local/bin/entr…"   15 months ago  Exited (1) 15 months ago             quirky_snyder
4e6c5968c115   kerberos       "/usr/sbin/krb5kdc -…"   15 months ago  Exited (0) 15 months ago             distracted_einstein
```

Commandes avancées Docker : Exemple avec Ollama

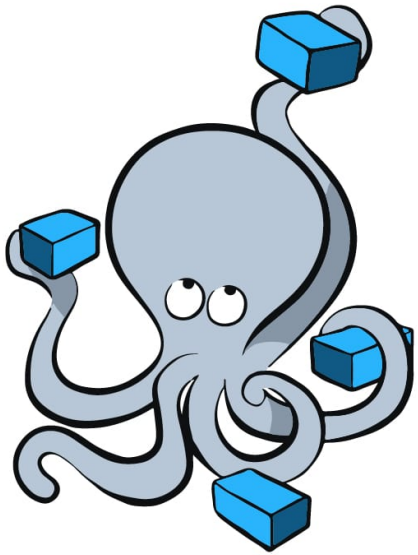
- `docker exec -it ollama ollama run mistral` : Exécute le modèle Mistral dans le conteneur Ollama en mode interactif
- `docker exec -it ollama /bin/bash` : Ouvre un terminal Bash interactif dans le conteneur Ollama



Rappel : Cycle de vie des conteneurs Docker



Partie 6 : Docker Compose



docker
Compose

À quoi ça sert ?

- Pour la mise en place d'applications faisant appel à plusieurs conteneurs, la **solution d' orchestration** **Docker Compose** entre en jeu.
- Exemple : l'installation de **WordPress**, **eLabFTW**, **GLPI**. . . c'est généralement le cas d'applications faisant appel à des **bases de données**.
- **Docker Compose** est un outil écrit en **Python** qui permet de définir plusieurs conteneurs dans un fichier unique :

`docker-compose.yml`

Installation de Docker Compose

- **Docker Compose** est installé généralement **en même temps** que **Docker** sur les postes **Linux**.
- **Docker Compose** est **intégré nativement** avec **Docker Desktop** (**macOS** et **Windows**).
- Commande pour **vérifier** la présence et la version de **Docker Compose** :

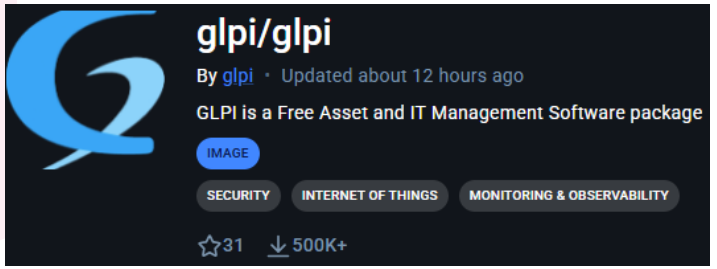
```
docker-compose --version
```

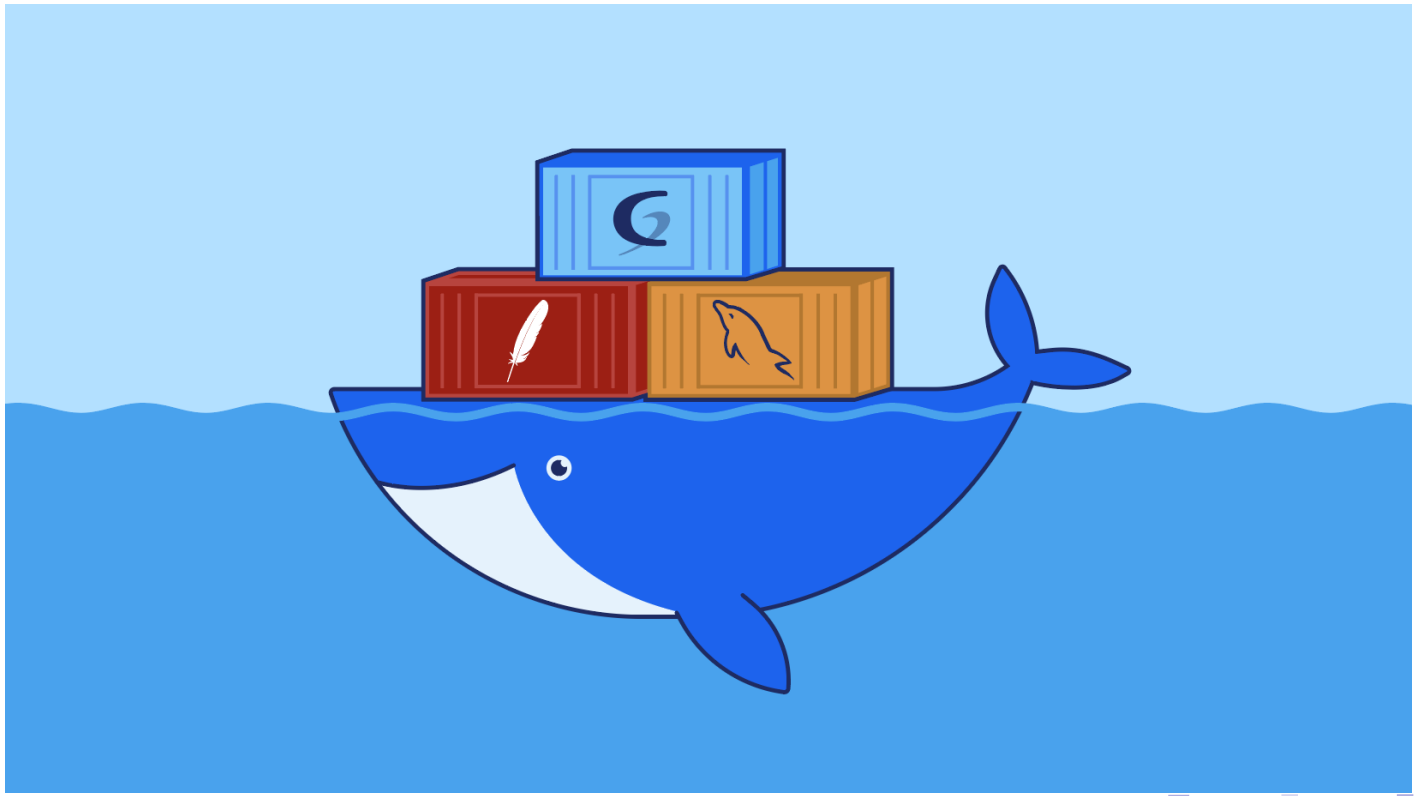
Analyse du **docker-compose.yml** fourni par **GLPI**

Structure d'un fichier docker-compose.yml

- Exemple avec **GLPI** :

https://hub.docker.com/r/glpi/glpi





Analyse du docker-compose.yml fourni par GLPI

```yaml
name: glpi
services:
  glpi:
    image: "glpi/glpi:latest"
    restart: "unless-stopped"
    volumes:
       # Using a named volume avoids permission issues on host (automatically managed by Docker)
       - glpi_data:/var/glpi
    env_file: .env # Pass environment variables from .env file to the container
    depends_on:
      - db
    ports:
      - "80:80"

  db:
    image: "mysql"
    restart: "unless-stopped"
    volumes:
       - db_data:/var/lib/mysql
    environment:
      MYSQL_RANDOM_ROOT_PASSWORD: "yes"
      MYSQL_DATABASE: ${GLPI_DB_NAME}
      MYSQL_USER: ${GLPI_DB_USER}
      MYSQL_PASSWORD: ${GLPI_DB_PASSWORD}

volumes:
   glpi_data:
   db_data:
```

Principales commandes de Docker Compose

✓ Attention : toujours se situer dans le dossier contenant le fichier docker-compose.yml

- `docker-compose build` : Reconstruit les images des services
- `docker-compose up -d` : Lance la création de l'ensemble des conteneurs en arrière-plan
- `docker-compose restart` : Redémarre les conteneurs
- `docker-compose stop` : Arrête les conteneurs sans les supprimer
- `docker-compose down` : Arrête et supprime les conteneurs, réseaux et volumes
- `docker-compose logs` : Affiche l'ensemble des logs des conteneurs
- docker-compose logs -f –tail 10 : Logs en temps réel (10 dernières lignes)

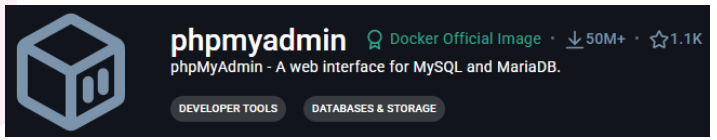
Ajout de phpMyAdmin à docker-compose.yml

Structure du fichier docker-compose.yml

- Pour **phpMyAdmin** :

  https://hub.docker.com/-/phpmyadmin

- On ajoute les **dépendances** et **ports**.




```yaml
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  restart: "unless-stopped"
  environment:
    PMA_HOST: db
    PMA_PORT: 3306
    PMA_ARBITRARY: 1
  depends_on:
    - db
  ports:
    - "82:80"
```

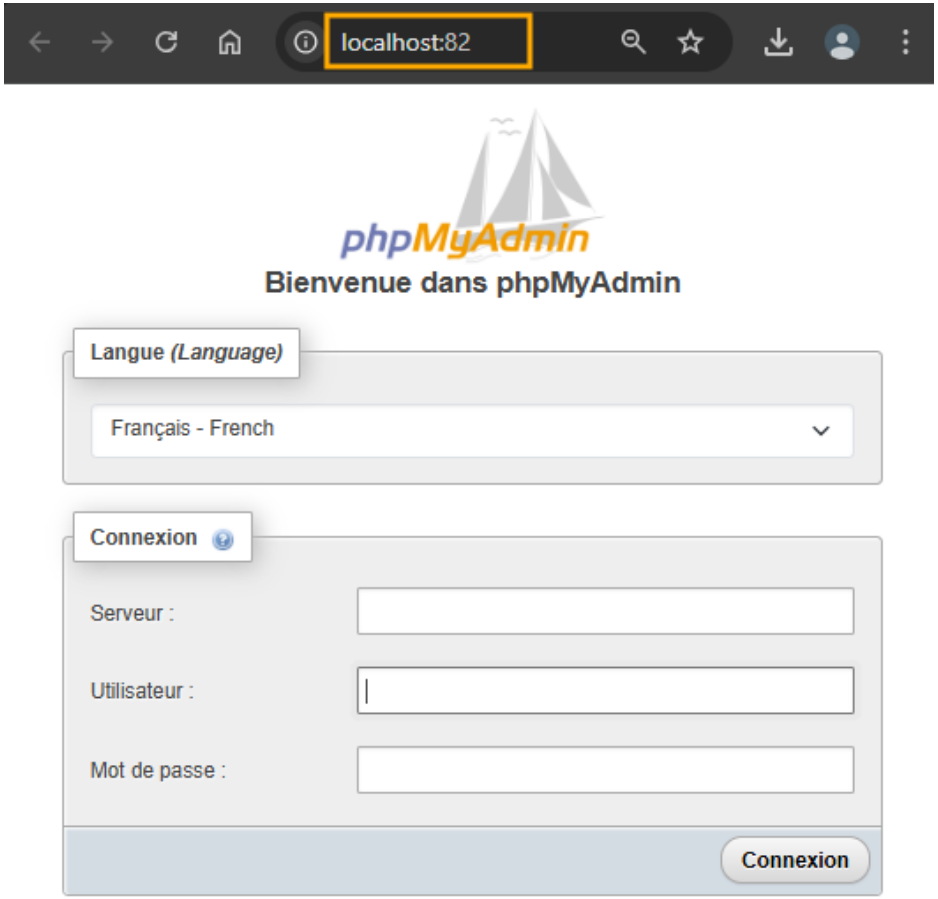
Forme finale du fichier docker-compose.yml

```
services:
  glpi:
    image: "glpi/glpi:latest"
    restart: unless-stopped
    volumes:
      - "./storage/glpi:/var/glpi:rw"
    env_file: .env
    depends_on:
      - db
    ports:
      - "81:80"

  db:
    image: "mysql:8.0"
    restart: unless-stopped
    volumes:
      - "./storage/mysql:/var/lib/mysql"
    environment:
      MYSQL_RANDOM_ROOT_PASSWORD: "yes"
      MYSQL_DATABASE: ${GLPI_DB_NAME}
      MYSQL_USER: ${GLPI_DB_USER}
      MYSQL_PASSWORD: ${GLPI_DB_PASSWORD}
    expose:
      - "3306"

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    restart: unless-stopped
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
      PMA_ARBITRARY: 1
    depends_on:
      - db
    ports:
      - "82:80"
```

Test des applications GLPI et phpMyAdmin localement



Partie finale

Conclusion : Conteneurisation, Docker et Docker Compose

Les 3 notions à retenir

- **Conteneurs** : Isolation légère, portabilité et reproductibilité pour vos applications.
- **Docker** : Moteur standard pour créer, gérer et déployer des conteneurs avec simplicité.
- **Docker Compose** : Orchestration multi-conteneurs via un fichier `docker-compose.yml`.

Cas d'usage : Plusieurs cas pratiques seront présentés à la suite de cette présentation !

Merci de votre attention !

Des questions ?