

Docker

Job 01 — Installation Docker CLI sur VM Debian

Configuration de la VM

- Debian 12 (mode console)
- 8 Go disque / 1 Go RAM / 1 vCPU

Commandes utilisées

```
sudo apt update && sudo apt upgrade -y
sudo apt install apt-transport-https ca-certificates curl gnupg
lsb-release -y

curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg
--dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo \
"deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/debian \
$(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

sudo apt install docker-ce docker-ce-cli containerd.io -y
```

Vérification

```
docker --version
```

tester si Docker est bien actif :

```
sudo systemctl status docker
```

Si ce n'est pas "active (running)", lance-le avec :

```
sudo systemctl start docker
```

Optionnel :

```
sudo usermod -aG docker $USER
```

Job 02 - Test Docker avec hello-world

Objectif: Tester que Docker est bien installé avec le conteneur "hello-world" et prendre en main les premières commandes de base.

1. Vérification de l'installation :

```
docker --version
docker info
```

2. Test de fonctionnement avec "hello-world" :

```
sudo docker run hello-world
```

```
dome@debian12-SI:~$ sudo docker run hello-world
[sudo] Mot de passe de dome :
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:7e1a4e2d11e2ac7a8c3f768d4166c2defeb09d2a750b010412b6ea13de1efb19
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

dome@debian12-SI:~$ █
```

Sortie attendue : message de bienvenue confirmant que Docker fonctionne.

```
dome@debian12-SI:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

dome@debian12-SI:~$
```

3. Commandes explorées :

docker images

```
dome@debian12-SI:~$ docker images
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Head "http://%2Fvar%2Frun%2Fdocker.sock/_ping": dial unix /var/run/docker.sock: connect: permission denied
dome@debian12-SI:~$ sudo docker images
[sudo] Mot de passe de dome :
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
hello-world         latest         74cc54e27dc4   2 months ago   10.1kB
```

docker ps -a

```
dome@debian12-SI:~$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS
NAMES
506c9da06526   hello-world   "/hello"       33 minutes ago Exited (0)    33 minutes ago
inspiring_bhaskara
dome@debian12-SI:~$
```

docker rm <container_id>dockers

docker rmi <image_id>

Commentaire

Tout fonctionne correctement, l'image `hello-world` a bien été téléchargée et exécutée. Le message de confirmation s'est bien affiché.

Job 03 - Utiliser un Dockerfile pour recréer le conteneur hello-world, mais cette fois à partir d'une image Debian minimale.



Contexte : : c'est quoi un Dockerfile ?

Un **Dockerfile** est un fichier texte contenant une série d'instructions utilisées par Docker pour construire une image personnalisée.

Il permet d'automatiser la création d'une image avec tout ce qu'elle doit contenir : système, dépendances, fichiers, etc.

Étapes détaillées à suivre :

1. Créer un dossier de travail :

```
mkdir job03-hello-from-docker  
cd job03-hello-from-docker
```

2. Écrire le Dockerfile :

Créer un fichier **Dockerfile** :

```
# Utiliser une image Debian minimale  
FROM debian:bullseye-slim
```

```
# Installer echo (inclus de base, donc ici pas besoin  
d'installation)
```

```
# Définir le point d'entrée du conteneur  
CMD echo "Hello from Docker!"
```

```
GNU nano 7.2 Dockerfile  
# Utiliser une image Debian minimale  
FROM debian:bullseye-slim  
  
# Installer echo (inclus de base, donc ici pas besoin d'installation)  
  
# Définir le point d'entrée du conteneur  
CMD echo "Hello from Docker!"
```

Ici, on se base sur une image Debian très légère et on configure simplement le conteneur pour afficher un message lors de l'exécution.

3. Construire l'image Docker :

```
docker build -t my-hello-debian .
```

```
dome@debian12-SI:~/job03-hello-from-docker$ sudo docker build -t my-hello-debian .
[+] Building 11.5s (3/4)                                docker:default
=> [internal] load build definition from Dockerfile      0.0s
=> => transferring dockerfile: 249B                      0.0s
=> [internal] load metadata for docker.io/library/debian:bullseye-slim 1.7s
=> [internal] load .dockerignore                        0.0s
=> => transferring context: 2B                             0.0s
=> [1/1] FROM docker.io/library/debian:bullseye-slim@sha256:e4b93db6aad977a95aa103917f 9.7s
=> => resolve docker.io/library/debian:bullseye-slim@sha256:e4b93db6aad977a95aa103917f 0.0s
=> => sha256:55147cbf65d4d152c070165835355b8ea7a090d48d81ba52cbeb9bbe 9.44MB / 30.25MB 9.7s
=> => sha256:e4b93db6aad977a95aa103917f3de8a2b16ead91cf255c3ccdb300c5d 4.54kB / 4.54kB 0.0s
=> => sha256:90e6d3bc46c9e60fd2285d7df3a931e6c9f106bfbca41772cf5c8779d 1.02kB / 1.02kB 0.0s
=> => sha256:65e31faf7b14478fdcce1f6dff03e269ca3a61cac0ead1941ee4ef40c8e31 453B / 453B 0.0s
```

- `-t my-hello-debian` : donne un nom à l'image.
- `.` : indique le dossier courant (où se trouve le Dockerfile).

4. Lancer un conteneur :

```
docker run --rm my-hello-debian
```

- `--rm` : supprime automatiquement le conteneur après exécution.
- Résultat attendu : `Hello from Docker!`

```
dome@debian12-SI:~/job03-hello-from-docker$ sudo docker run --rm my-hello-debian
Hello from Docker!
dome@debian12-SI:~/job03-hello-from-docker$
```

Job 04 - Créer une image Docker avec SSH

Créer une **image Docker avec SSH** (compte **root** / mot de passe **root123**) **sans utiliser une image SSH existante**, rediriger le port SSH vers un autre que 22 (ex. 2222), **lancer le conteneur** et **vérifier la connexion SSH**.

Étape 1 : Crée ton fichier Dockerfile

utiliser nano

nano Dockerfile



```
GNU nano 7.2 Dockerfile *
# 1. Image de base
FROM debian:latest

# 2. Mise à jour des paquets et installation de SSH
RUN apt update && apt install -y openssh-server

# 3. Création du mot de passe pour root
RUN echo 'root:root123' | chpasswd

# 4. Activation du SSH
RUN mkdir /var/run/sshd

# 5. Modification de la configuration SSH (autoriser root)
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

# 6. Redirection du port SSH (autre que 22, ici 2222)
EXPOSE 2222

# 7. Commande de lancement du serveur SSH
CMD ["/usr/sbin/sshd", "-D"]
```

Étape 2 : Construire l'image Docker

Dans ton terminal, place-toi dans le dossier contenant le **Dockerfile** puis exécute :

docker build -t ssh-image .

Cela crée une image nommée **ssh-image**.

```
dome@debian12-SI:~/job04-ssh-from-docker$ sudo docker build -t ssh-image .
[+] Building 1.2s (9/9) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile             0.0s
=> => transferring dockerfile: 595B                             0.0s
=> [internal] load metadata for docker.io/library/debian:latest 1.1s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                    0.0s
=> [1/5] FROM docker.io/library/debian:latest@sha256:18023f131f52fc3ea21973cabffe0b216 0.0s
=> CACHED [2/5] RUN apt update && apt install -y openssh-server 0.0s
=> CACHED [3/5] RUN echo 'root:root123' | chpasswd              0.0s
=> CACHED [4/5] RUN mkdir /var/run/sshd                        0.0s
=> CACHED [5/5] RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' 0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:73b68cfe25a0cefaeb095d75b5009c2bf94934269c89c769a9f6a054567 0.0s
=> => naming to docker.io/library/ssh-image                    0.0s
dome@debian12-SI:~/job04-ssh-from-docker$ ^C
```

Étape 3 : Lancer un conteneur avec redirection de port

`docker run -d -p 2222:22 --name ssh-container ssh-image`

Ici :

- `-d` : démarre en arrière-plan
- `-p 2222:22` : redirige le port 2222 de l'hôte vers le port 22 du conteneur
- `--name ssh-container` : donne un nom à ton conteneur

```
dome@debian12-SI:~/job04-ssh-from-docker$ sudo docker run -d -p 2222:22 --name ssh-container s
sh-image
b8fa052a56b6fc2d38f6566bb3dfe91b98bad17151cb31ad4a77a32ead7c50e0
dome@debian12-SI:~/job04-ssh-from-docker$
```

Étape 4 : Connexion SSH

Teste la connexion SSH avec cette commande depuis ta machine hôte :

(Pour le test sur la machine Debian (où on a créé docker run), tape simplement) :

`ssh root@localhost -p 2222`

```
dome@debian12-SI:~$ ssh root@localhost -p 2222
The authenticity of host '[localhost]:2222 ([::1]:2222)' can't be established.
ED25519 key fingerprint is SHA256:IO7XFFPZii8nz4mfw2Tz2aCAMPgxChvfAeopFWvqjhU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:2222' (ED25519) to the list of known hosts.
root@localhost's password:
Linux b8fa052a56b6 6.1.0-21-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.90-1 (2024-05-03) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@b8fa052a56b6:~# _
```

Pour arrêter le conteneur

```
sudo docker stop ssh-container
```

Cela met le conteneur en pause (il reste présent, mais arrêté).

```
dome@debian12-SI:~/job04-ssh-from-docker$ sudo docker stop ssh-container
ssh-container
dome@debian12-SI:~/job04-ssh-from-docker$
```

```
dome@debian12-SI:~$ ssh root@localhost -p 2222
ssh: connect to host localhost port 2222: Connection refused
dome@debian12-SI:~$ _
```

Pour le relancer plus tard

```
sudo docker start ssh-container
```

Job 05 - Automatiser et simplifier l'utilisation des commandes Docker

Automatiser et **simplifier l'utilisation des commandes Docker** en créant des **alias personnalisés** dans le fichier `~/ .bashrc`. Cela permet de taper des commandes plus courtes, plus simples à retenir.



Contexte : qu'est-ce qu'un alias ?

Un **alias**, c'est un raccourci que tu définis dans ton terminal. Par exemple, au lieu de taper `docker ps -a`, tu peux créer un alias appelé `dpsa` qui fera la même chose.

Ces alias sont placés dans le fichier `~/ .bashrc` (ou `~/ .zshrc` si tu utilises Zsh), ce qui les rend disponibles à chaque démarrage du terminal.

Étapes détaillées :

1. Ouvrir le fichier `~/ .bashrc`

```
nano ~/ .bashrc
```


2. Ajouter les alias suivants à la fin du fichier :

```
# Alias Docker utiles
alias d='docker'
alias dps='docker ps'
alias dpsa='docker ps -a'
alias di='docker images'
alias dstop='docker stop $(docker ps -q)'
alias drm='docker rm $(docker ps -a -q)'
alias drmi='docker rmi $(docker images -q)'
alias dstart='docker start'
alias dexec='docker exec -it'
alias dlogs='docker logs'
alias dvol='docker volume ls'
```

```
GNU nano 7.2 /home/dome/.bashrc *
#alias grep='grep --color=auto'
#alias fgrep='fgrep --color=auto'
#alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

# Alias Docker utiles
alias d='docker'
alias dps='docker ps'
alias dpsa='docker ps -a'
alias di='docker images'
alias dstop='docker stop $(docker ps -q)'
alias drm='docker rm $(docker ps -a -q)'
alias drmi='docker rmi $(docker images -q)'
alias dstart='docker start'
alias dexec='docker exec -it'
alias dlogs='docker logs'
alias dvol='docker volume ls'

^G Aide      ^O Écrire    ^W Chercher  ^K Couper    ^T Exécuter  ^C Emplacement
^X Quitter   ^R Lire fich.^_ Remplacer  ^U Coller    ^J Justifier ^_ Aller ligne
```

💡 Explications :

Alias	Commande équivalente	Fonction
<code>d</code>	<code>docker</code>	raccourci général
<code>dps</code>	<code>docker ps</code>	liste des conteneurs actifs
<code>dpsa</code>	<code>docker ps -a</code>	liste tous les conteneurs
<code>di</code>	<code>docker images</code>	liste des images disponibles
<code>dstop</code>	<code>docker stop \$(docker ps -q)</code>	arrête tous les conteneurs
<code>drm</code>	<code>docker rm \$(docker ps -a -q)</code>	supprime tous les conteneurs arrêtés
<code>drmi</code>	<code>docker rmi \$(docker images -q)</code>	supprime toutes les images
<code>dstart</code>	<code>docker start</code>	démarre un conteneur
<code>dexec</code>	<code>docker exec -it</code>	exécute une commande dans un conteneur actif
<code>dlogs</code>	<code>docker logs</code>	affiche les logs d'un conteneur
<code>dvol</code>	<code>docker volume ls</code>	liste les volumes

3. Recharger le fichier `.bashrc`

Après avoir sauvegardé avec `CTRL+O`, `Entrée`, puis `CTRL+X` :

```
source ~/.bashrc
```

Problème rencontré

le message d'erreur suivant indique que l'alias n'est pas reconnu :

```
dome@debian12-SI:~/job04-ssh-from-docker$ sudo dps
sudo: dps : commande introuvable
dome@debian12-SI:~/job04-ssh-from-docker$
```

👉 **Pourquoi ça ne marche pas ?** Les alias définis dans `.bashrc` **ne sont pas disponibles avec sudo** par défaut. Quand tu fais `sudo dps`, la commande `dps` est inconnue car `sudo` n'hérite pas de ton environnement utilisateur (et donc pas de tes alias).

Solution : Utiliser `sudo` sur la commande complète, pas sur l'alias

Utilise juste :

`dps`

... et non :

`sudo dps`

➡ La commande `docker` ne nécessite **pas sudo** si tu es dans le groupe `docker`. Si ce n'est pas encore fait, fais-le une fois :

`sudo usermod -aG docker $USER`

Ensuite **redémarre ta session** (ou fais `newgrp docker`) pour que ça prenne effet.

```
dome@debian12-SI:~/job04-ssh-from-docker$ sudo usermod -aG docker dome
dome@debian12-SI:~/job04-ssh-from-docker$ newgrp docker
dome@debian12-SI:~/job04-ssh-from-docker$ dps
CONTAINER ID    IMAGE          COMMAND        CREATED        STATUS        PORTS        NAMES
dome@debian12-SI:~/job04-ssh-from-docker$
```

et voilà!!!!

Job 06 - Comprendre et tester l'utilisation de volumes entre deux conteneurs ainsi que la gestion des volumes dans Docker.



Contexte : C'est quoi un volume dans Docker ?

Un **volume** est un espace de stockage **persistant** créé et géré par Docker. Contrairement à un simple système de fichiers dans le conteneur, un volume :

- n'est **pas supprimé automatiquement** quand le conteneur est supprimé.
- peut être **partagé entre plusieurs conteneurs**.
- est stocké par dans `/var/lib/docker/volumes/` par défaut.

Cela permet donc :

- de **conserver des données** même si le conteneur est supprimé.
 - de **partager des fichiers** entre deux conteneurs (par exemple, un conteneur qui écrit dans un fichier, un autre qui le lit).
-

Étape 1 : Créer un volume Docker

```
docker volume create volume-partage
```

Étape 2 : Lancer un premier conteneur (écrivain)

```
docker run -dit --name conteneur1 -v volume-partage:/data debian
```

Cela monte le volume `volume-partage` dans le dossier `/data` du conteneur.

Ensuite, tu peux te connecter au conteneur et écrire un fichier :

```
docker exec -it conteneur1 bash
echo "Salut depuis conteneur1" > /data/bonjour.txt
exit
```

```
dome@debian12-SI:~$ docker volume create volume-partage
volume-partage
dome@debian12-SI:~$ docker run -dit --name conteneur1 -v volume-partage:/data debian
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
7cd785773db4: Already exists
Digest: sha256:18023f131f52fc3ea21973cabffe0b216c60b417fd2478e94d9d59981ebba6af
Status: Downloaded newer image for debian:latest
6200d9617e942a2dd8547d74ea868be7a871a97e3d14267830e9c6cb4294d02b
```

Étape 3 : Lancer un second conteneur (lecteur)

```
docker run -dit --name conteneur2 -v volume-partage:/data debian
```

Puis vérifie que le fichier est bien présent :

```
docker exec -it conteneur2 bash
cat /data/bonjour.txt
exit
```

Tu devrais voir s'afficher :

```
Salut depuis conteneur1
```

✅ Tu viens de **partager un fichier entre deux conteneurs** grâce au volume !

Arborescence des fichiers créés

Côté hôte Debian (volume Docker) :

```
/var/lib/docker/volumes/
```

```
└─ volume-partage/
```

```
    └─ _data/
```

```
        └─ bonjour.txt ← Fichier créé par conteneur1 et lu par conteneur2
```

Bonus : Gestion des volumes

Lister les volumes :

```
docker volume ls
```

Inspecter un volume :

```
docker volume inspect volume-partage
```

Supprimer un volume (attention, cela efface les données) :

```
docker volume rm volume-partage
```

Job 07 - Mettre en place deux conteneurs Docker liés entre eux à l'aide de Docker Compose

Objectif :

- Un conteneur **Nginx**
- Un conteneur **FTP**
- Un **volume commun partagé** entre les deux (permettant de déposer un fichier HTML via FTP et de l'afficher via Nginx)

VOLUME	
Conteneur FTP	<-- partagé -->
/home/vsftpd	(volume)
Conteneur NGINX	/usr/share/nginx/html

Contexte:

Docker Compose, c'est quoi ?

C'est un outil qui permet de définir et de lancer plusieurs conteneurs Docker à l'aide d'un fichier YAML (docker-compose.yml). Plutôt que de lancer les conteneurs un par un à la main, on décrit tout dans un fichier et une commande suffit à tout démarrer (docker-compose up).

Volume commun

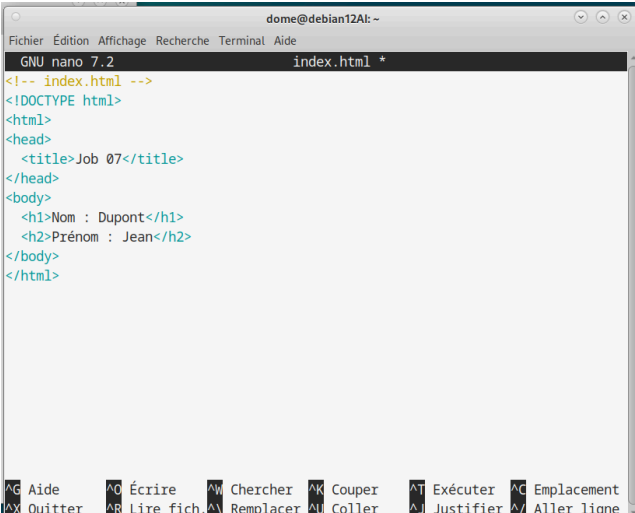
Un volume Docker permet de partager un dossier entre conteneurs. Cela veut dire que ce que tu envoies via FTP sera aussi visible côté Nginx.

Étapes:

1. Créer le fichier index.html

Crée un fichier index.html avec ton nom/prénom :

```
<!-- index.html -->
<!DOCTYPE html>
```



```
dome@debian12Alt: ~
GNU nano 7.2 index.html *
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Job 07</title>
</head>
<body>
  <h1>Nom : Dupont</h1>
  <h2>Prénom : Jean</h2>
</body>
</html>
```

```

<html>
<head>
  <title>Job 07 Docker</title>
</head>
<body>
  <h1>Nom : Dupont</h1>
  <h2>Prénom : Jean</h2>
</body>
</html>

```

2. Créer le fichier docker-compose.yml

Voici un exemple de fichier complet :

```

version: '3.8'

services:
  nginx:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - shared-data:/usr/share/nginx/html

  ftp:
    image: fauria/vsftpd
    ports:
      - "21:21"
      - "30000-30009:30000-30009"
    environment:
      - FTP_USER=user
      - FTP_PASS=pass123
      - PASV_MIN_PORT=30000
      - PASV_MAX_PORT=30009
      - PASV_ADDRESS=localhost
    volumes:
      - shared-data:/home/vsftpd

volumes:
  shared-data:

```



The screenshot shows a terminal window with the title 'dome@debian12A1: ~/Job07Docker'. The nano editor is open, editing 'docker-compose.yml'. The content of the file is as follows:

```

version: '3.8'

services:
  nginx:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - shared-data:/usr/share/nginx/html

  ftp:
    image: fauria/vsftpd
    ports:
      - "21:21"
      - "30000-30009:30000-30009"
    environment:
      - FTP_USER=user
      - FTP_PASS=pass123
      - PASV_MIN_PORT=30000
      - PASV_MAX_PORT=30009
      - PASV_ADDRESS=localhost
    volumes:
      - shared-data:/home/vsftpd

volumes:
  shared-data:

```

The bottom of the terminal shows the nano editor's command palette with options: Aide, Écrire, Chercher, Couper, Quitter, Lire fich., Remplacer, Coller.

Ce fichier définit :

- un serveur Nginx exposé sur le port 8080,
- un serveur FTP exposé sur les ports 21 + plage passive,
- un volume shared-data partagé entre les deux.

```
dome@debian12AI:~/Job07Docker$ ls
docker-compose.yml  index.html
dome@debian12AI:~/Job07Docker$
```

3. Démarrage des conteneurs

Dans ton terminal, place-toi dans le dossier job07 et exécute :

```
docker compose up -d
```

```
dome@debian12AI: ~/Job07Docker
Fichier Édition Affichage Recherche Terminal Aide
dome@debian12AI:~/Job07Docker$ sudo docker compose up -d
WARN[0000] /home/dome/Job07Docker/docker-compose.yml: the attribute `version` is obsolete,
it will be ignored, please remove it to avoid potential confusion
[+] Running 20/20
 ✓ nginx Pulled 202.6s
 ✓ 6e909acdb790 Pull complete 99.7s
 ✓ 5eaa34f5b9c2 Pull complete 200.9s
 ✓ 417c4bccf534 Pull complete 200.9s
 ✓ e7e0ca015e55 Pull complete 200.9s
 ✓ 373fe654e984 Pull complete 200.9s
 ✓ 97f5c0f51d43 Pull complete 200.9s
 ✓ c22eb46e871a Pull complete 201.0s
 ✓ ftp Pulled 211.5s
 ✓ 2d473b07cdd5 Pull complete 206.2s
 ✓ 33b95e46f70b Pull complete 209.5s
 ✓ e22029c8d9a7 Pull complete 209.8s
 ✓ e1871c5d8fc9 Pull complete 209.8s
 ✓ c17c1255c529 Pull complete 209.8s
 ✓ ddcbab051542 Pull complete 209.8s
 ✓ 1c68b0b593f1 Pull complete 209.8s
 ✓ dadb66293c59 Pull complete 209.8s
 ✓ 99a54b7a405b Pull complete 209.8s
 ✓ 200facf93d0a Pull complete 209.8s
 ✓ 16ecaf7d0305 Pull complete 209.9s
[+] Running 4/4
 ✓ Network job07docker_default Created 0.1s
 ✓ Volume "job07docker_shared-data" Created 0.0s
 ✓ Container job07docker-ftp-1 Started 1.2s
 ✓ Container job07docker-nginx-1 Started 0.9s
dome@debian12AI:~/Job07Docker$
```

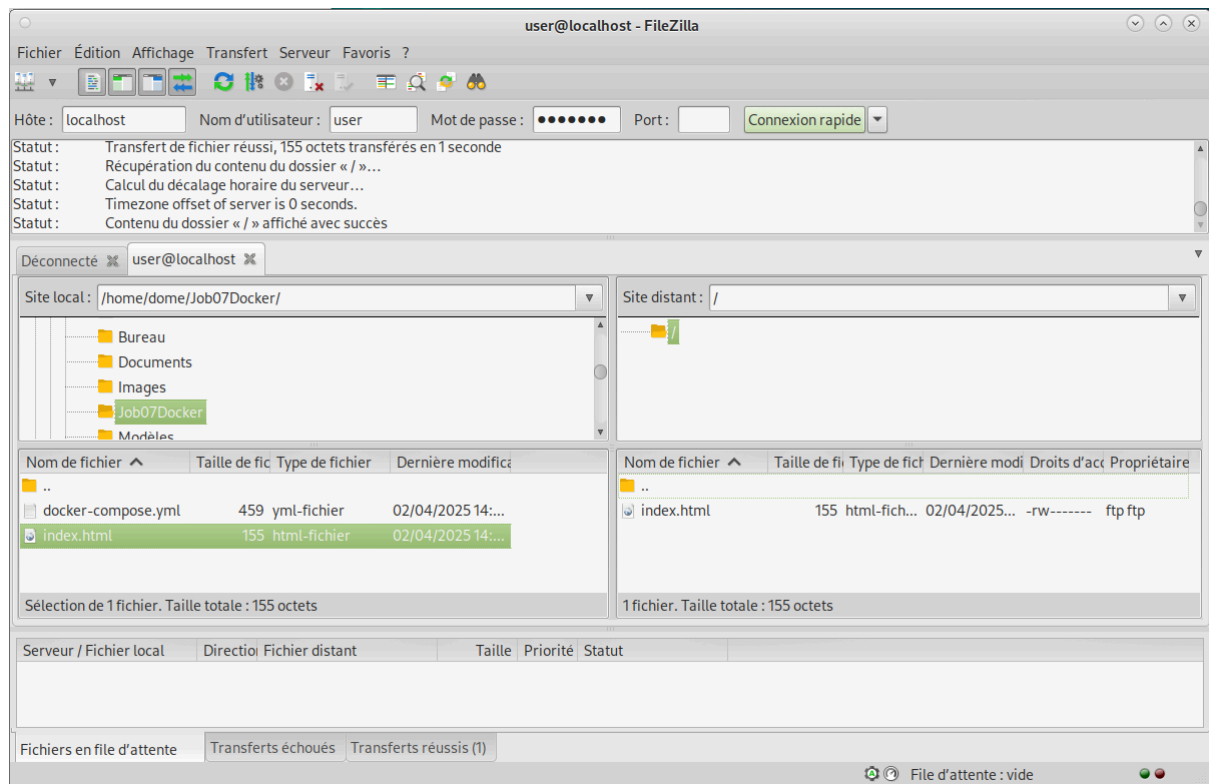
Cela va télécharger les images et démarrer les conteneurs en arrière-plan.

4. Connexion FTP (avec FileZilla)

Ouvre FileZilla sur ton PC :

- Hôte : localhost
- Utilisateur : user
- Mot de passe : pass123
- Port : 21

Une fois connecté, envoie ton fichier index.html dans le répertoire FTP (il sera automatiquement partagé avec Nginx).

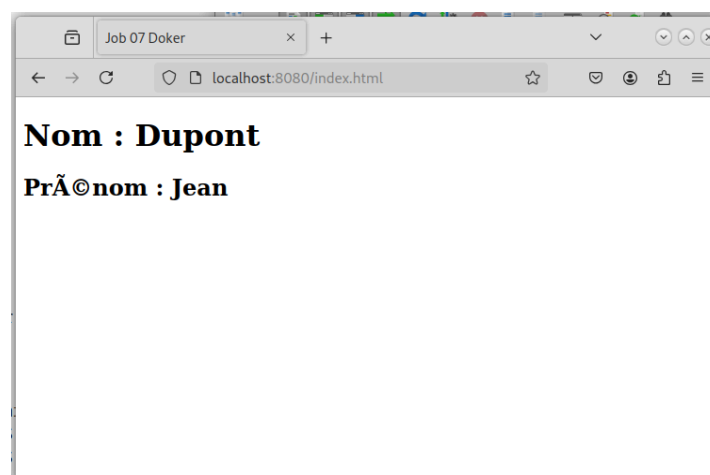


5. Vérification dans le navigateur

Ouvre ton navigateur et va sur :

<http://localhost:8080>

Tu devrais voir le fichier HTML s'afficher avec ton nom/prénom.



Pour couper les conteneurs on utilise la commande:

`sudo docker compose down -v`

Job 08 – Nginx personnalisé sans image existante

Objectif :

Créer un conteneur Docker avec Nginx sans utiliser d'image Nginx officielle, en partant de Debian.

Création des fichiers :

```
mkdir job08-nginx
cd job08-nginx
touch Dockerfile index.html
```

1. Contenu du `index.html` :

```
<html><body><h1>Job 08 - Mon Nginx Job 08 Docker
</h1></body></html>
```

Dockerfile :

```
# Étape 1 : partir d'une image Debian minimale
```

```
FROM debian:bookworm-slim
```

```
# Étape 2 : mettre à jour les paquets et installer nginx
```

```
RUN apt-get update && \
    apt-get install -y nginx && \
    apt-get clean
```

```
# Étape 3 : copier ta page HTML dans le bon dossier
```

```
COPY index.html /var/www/html/index.html
```

```
# Étape 4 : exposer le port 80 (celui de nginx)
```

```
EXPOSE 80
```

Étape 5 : démarrer nginx en avant-plan (important pour Docker)

```
CMD ["nginx", "-g", "daemon off;"]
```

Construction de l'image :

```
docker build -t mon-nginx-perso .
```

Lancement du conteneur :

```
docker run -d -p 8080:80 --name nginx-perso mon-nginx-perso
```

Vérification via navigateur : <http://localhost:8080>

Résultat attendu :

Affichage du message personnalisé dans le navigateur.



Job 09 - Mise en place d'un registry Docker local avec interface web (UI)

Objectif du Job

Mettre en place un registry Docker local permettant de stocker des images Docker en interne, puis ajouter une interface graphique (UI) pour le consulter via un navigateur web.

Difficultés rencontrées

- Problèmes d'affichage des images dans l'interface web (erreurs CORS).
- Non-affichage des images déjà poussées suite à une suppression du container registry.
- Problème de cache navigateur et de mauvaise configuration d'URL.

Ces difficultés ont permis de mieux comprendre le fonctionnement interne du registry et de son lien avec l'interface web.

1. Création du contai

```
docker run -d -p 5000:5000 --name registry registry:2
```

- `docker run` : lance un nouveau conteneur
- `-d` : mode détaché (en arrière-plan)
- `-p 5000:5000` : relie le port 5000 de la machine hôte au port 5000 du conteneur
- `--name registry` : nomme le conteneur "registry"
- `registry:2` : utilise l'image officielle du registre Docker, version 2

```
dome@debian12AI:~/Job08-Docker$ sudo docker run -d -p 5000:5000 --name registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
44cf07d57ee4: Downloading [=====> ] 3.136MB/3.418MB
bbbdd6c6894b: Download complete
8e82f80af0de: Downloading [=====> ] 2.595MB/6.404MB
3493bf46cdec: Download complete
6d464ea18732: Download complete
```

Après erreur CORS) Recréation du registry avec support CORS

```
docker rm -f registry
```

- Supprime le conteneur existant, forcément arrêté s'il est actif

```
docker run -d -p 5000:5000 \
--name registry \
-e REGISTRY_STORAGE_DELETE_ENABLED=true \
-e REGISTRY_HTTP_HEADERS_Access-Control-Allow-Origin='["http://192.168.40.142:8080"]' \
```

```
-e REGISTRY_HTTP_HEADERS_Access-Control-Allow-Methods=['GET', 'HEAD', 'OPTIONS']'  
\  
registry:2
```

- `-e VAR=valeur` : ajoute des variables d'environnement
- Ici, on autorise les requêtes venant de l'interface UI (port 8080)
- C'est essentiel pour que le navigateur ne bloque pas les appels (politique de sécurité CORS)

Configuration pour accepter les "insecure registries"

Docker, par défaut, n'autorise pas les connexions non sécurisées (HTTP) vers un registry local. Il faut modifier un fichier de configuration :

```
sudo nano /etc/docker/daemon.json
```

Et ajouter le contenu suivant :

```
{  
  
  "insecure-registries": ["localhost:5000"]  
  
}
```

Ensuite, redémarrer Docker :

```
sudo systemctl restart docker
```

2. Téléchargement de l'image `alpine` et envoi dans le registry

```
docker pull alpine
```

- Télécharge l'image minimale Alpine Linux depuis Docker Hub

```
docker tag alpine 192.168.40.142:5000/alpine
```

- Renomme localement l'image pour cibler notre registry local sur l'adresse IP

```
docker push 192.168.40.142:5000/alpine
```

- Envoie l'image dans le registry local

Remplacer `192.168.40.142` par l'adresse IP réelle de votre VM

```
dome@debian12AI:~/Job08-Docker$ docker pull alpine  
docker tag alpine localhost:5000/alpine  
docker push localhost:5000/alpine  
Using default tag: latest  
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.48/images/create?fromImage=alpine&tag=latest": dial unix /var/run/docker.sock: connect: permission denied  
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.48/images/alpine/tag?repo=localhost%3A5000%2Falpine&tag=latest": dial unix /var/run/docker.sock: connect: permission denied  
Using default tag: latest  
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.48/images/localhost:5000/alpine/push?tag=latest": dial unix /var/run/docker.sock: connect: permission denied  
dome@debian12AI:~/Job08-Docker$ sudo docker pull alpine  
Using default tag: latest  
latest: Pulling from library/alpine  
f18232174bc9: Pull complete  
Digest: sha256:a8560b36e8b210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3ef88c  
Status: Downloaded newer image for alpine:latest  
docker.io/library/alpine:latest  
dome@debian12AI:~/Job08-Docker$ sudo docker tag alpine localhost:5000/alpine  
dome@debian12AI:~/Job08-Docker$ sudo docker push localhost:5000/alpine  
Using default tag: latest  
The push refers to repository [localhost:5000/alpine]  
08000c18d16d: Pushed
```

Vérification via la commande `curl`

`curl http://localhost:5000/v2/_catalog`

- Permet de vérifier via l'API que l'image est bien enregistrée
- Une réponse attendue serait : `{ "repositories": ["alpine"] }`

3. Mise en place de l'interface graphique UI

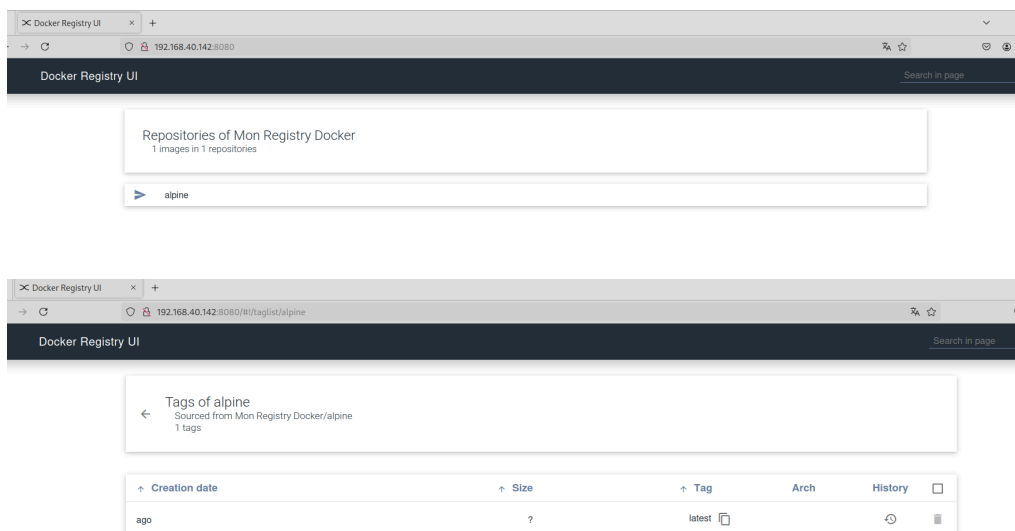
```
docker run -d -p 8080:80 \
  --name registry-ui \
  -e REGISTRY_URL=http://192.168.40.142:5000 \
  -e DELETE_IMAGES=true \
  -e REGISTRY_TITLE="Mon Registry Docker" \
  joxit/docker-registry-ui:2
```

- `-p 8080:80` : le port 80 de l'UI est exposé sur le port 8080 de la machine
- `REGISTRY_URL` : définit l'adresse du registry à interroger depuis l'interface
- `DELETE_IMAGES` : active la possibilité de supprimer des images via l'UI (optionnelle)
- `REGISTRY_TITLE` : titre affiché dans l'interface
- `joxit/docker-registry-ui:2` : image officielle de l'interface UI, version 2

L'interface est ensuite accessible depuis un navigateur à l'adresse :

`http://192.168.40.142:8080`

Capture d'écran



Job 10 - réaliser deux scripts bash :

1. Un script de désinstallation complète de Docker :

- Supprimer tous les conteneurs
- Supprimer toutes les images
- Supprimer tous les volumes
- Supprimer les paquets liés à Docker (Docker Engine, CLI...)
- Nettoyer le système

2. Un script d'installation automatique de Docker :

- Mise à jour du système
- Ajout du dépôt Docker
- Installation de Docker CLI
- Activation + démarrage du service Docker

Script 1 : Désinstaller Docker complètement

Nom du script : `uninstall_docker.sh`

Voici le script commenté ligne par ligne pour que tu comprennes :

```
#!/bin/bash

echo "Suppression des conteneurs..."
docker rm -f $(docker ps -aq) 2>/dev/null

echo " Suppression des images..."
docker rmi -f $(docker images -q) 2>/dev/null

echo "Suppression des volumes..."
docker volume rm $(docker volume ls -q) 2>/dev/null

echo "Arrêt de Docker..."
```

```
systemctl stop docker
```

```
echo " Suppression des paquets Docker..."  
apt-get purge -y docker-ce docker-ce-cli containerd.io  
docker-buildx-plugin docker-compose-plugin
```

```
echo " Suppression des dépendances inutilisées..."  
apt-get autoremove -y
```

```
echo " Suppression des fichiers de configuration..."  
rm -rf /var/lib/docker  
rm -rf /etc/docker  
rm -rf ~/.docker
```

```
echo " Docker a été supprimé proprement."
```

Ce script doit être exécuté avec les droits **root** :

```
sudo ./uninstall_docker.sh
```

```
dome@debian12-SI:~$ nano uninstall_docker.sh  
dome@debian12-SI:~$ chmod 775 uninstall_docker.sh  
dome@debian12-SI:~$ ./uninstall_docker.sh  
Suppression des conteneurs...  
fb2e01d9df11  
b90ce6072072  
48552b995371  
2651c530fadd  
b8fa052a56b6  
506c9da06526  
Suppression des images...  
Untagged: ssh-image:latest  
Untagged: ssh-images:latest  
Deleted: sha256:73b68cfe25a0cefaeb095d75b5009c2bf94934269c89c769a9f6a054567c4572  
Untagged: debian:latest  
Untagged: debian@sha256:18023f131f52fc3ea21973cabffe0b216c60b417fd2478e94d9d59981ebba6af  
Deleted: sha256:d844481994701734800e3e8708049b9585d1b81472f9856157fe17e45ab11f7a  
Untagged: my-hello-debian:latest  
Deleted: sha256:91397dc6980cb09461587f77fd16a1b0552ee0edc0acde73d04dd104bce00a89  
Untagged: alpine:latest
```

Script 2 : Installation automatique de Docker

Nom du script : `install_docker.sh`

Voici un script simple pour installer Docker sur Debian :

```
#!/bin/bash

echo " Mise à jour des paquets..."
apt-get update && apt-get upgrade -y

echo " Installation des dépendances nécessaires..."
apt-get install -y ca-certificates curl gnupg

echo " Ajout de la clé GPG officielle de Docker..."
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg
--dearmor -o /etc/apt/keyrings/docker.gpg

echo " Ajout du dépôt Docker dans APT..."
echo \
    "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg] \
    https://download.docker.com/linux/debian $(. /etc/os-release &&
echo "$VERSION_CODENAME") stable" \
    > /etc/apt/sources.list.d/docker.list

echo " Mise à jour des sources..."
apt-get update

echo " Installation de Docker Engine et CLI..."
apt-get install -y docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin

echo " Activation de Docker..."
systemctl enable docker
systemctl start docker
```

```
echo " Docker est installé et prêt à l'emploi."
```

Ce script aussi doit être exécuté avec les droits **root** :

```
sudo ./install_docker.sh
```

```
dome@debian12-SI:~$ nano install_docker.sh
dome@debian12-SI:~$ chmod 755 install_docker.sh
dome@debian12-SI:~$ sydo ./install_docker.sh
-bash: sydo : commande introuvable
dome@debian12-SI:~$ sudo ./install_docker.sh
Mise à jour des paquets...
Réception de :1 http://security.debian.org/debian-security bookworm-security InRelease [48,0 kB]
Atteint :2 http://deb.debian.org/debian bookworm InRelease
Réception de :3 http://deb.debian.org/debian bookworm-updates InRelease [55,4 kB]
Réception de :4 https://download.docker.com/linux/debian bookworm InRelease [47,0 kB]
Réception de :5 http://security.debian.org/debian-security bookworm-security/main Sources [148 kB]
Réception de :6 http://security.debian.org/debian-security bookworm-security/main amd64 Packages [251 kB]
Réception de :7 http://security.debian.org/debian-security bookworm-security/main Translation-en [151 kB]
700 ko réceptionnés en 1s (552 ko/s)
Lecture des listes de paquets... Fait
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Calcul de la mise à jour... Fait
Les paquets suivants seront mis à jour :
  liblzma5 xz-utils
2 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
```

Job 11 - Découvrir Portainer

Découvrir **Portainer**, un outil d'administration graphique pour Docker, **et refaire les Jobs 2 à 9 via l'interface web**. Il faut aussi **se renseigner sur les alternatives** à Portainer.

Installer Portainer sur ta VM Debian

Docker doit être déjà installé .

Voici les commandes à copier-coller :

```
docker volume create portainer_data
```

```
dome@debian12-SI:~$ docker volume create portainer_data
portainer_data
dome@debian12-SI:~$
```

```
docker run -d -p 8000:8000 -p 9443:9443 \
  --name portainer \
  --restart=always \
```

```
-v /var/run/docker.sock:/var/run/docker.sock \  
-v portainer_data:/data \  
portainer/portainer-ce:latest
```

```
dome@debian12-SI:~$ docker run -d -p 8000:8000 -p 9443:9443 \  
--name portainer \  
--restart=always \  
-v /var/run/docker.sock:/var/run/docker.sock \  
-v portainer_data:/data \  
portainer/portainer-ce:latest  
Unable to find image 'portainer/portainer-ce:latest' locally  
latest: Pulling from portainer/portainer-ce  
e2e06b27b87e: Pull complete  
1fed1531b45b: Pull complete  
04de093ad5ed: Downloading  7.674MB/18.85MB  
86a7cce72d42: Waiting  
e09df2601140: Waiting  
eae3ebf29ea8: Waiting  
c12aa3fbd31a: Waiting  
f111bda3f9a6: Waiting
```

Accéder à l'interface web

Dans ton navigateur, tape :

[https://\[IP_de_ta_VM\]:9443](https://[IP_de_ta_VM]:9443)

▼ New Portainer installation

Please create the initial administrator user.

Username

admin

Password

Confirm password

✕

⚠ The password must be at least 12 characters long.

Create user

☒ Allow collection of anonymous statistics. You can find more information about this in our [privacy policy](#).

> Restore Portainer from backup

⚠ Accepte le certificat non sécurisé (c'est normal pour un test local).

Tu devras créer un **mot de passe admin**. Retient-le bien !

Si un restart est necesaire taper la commande suivante dans le terminal

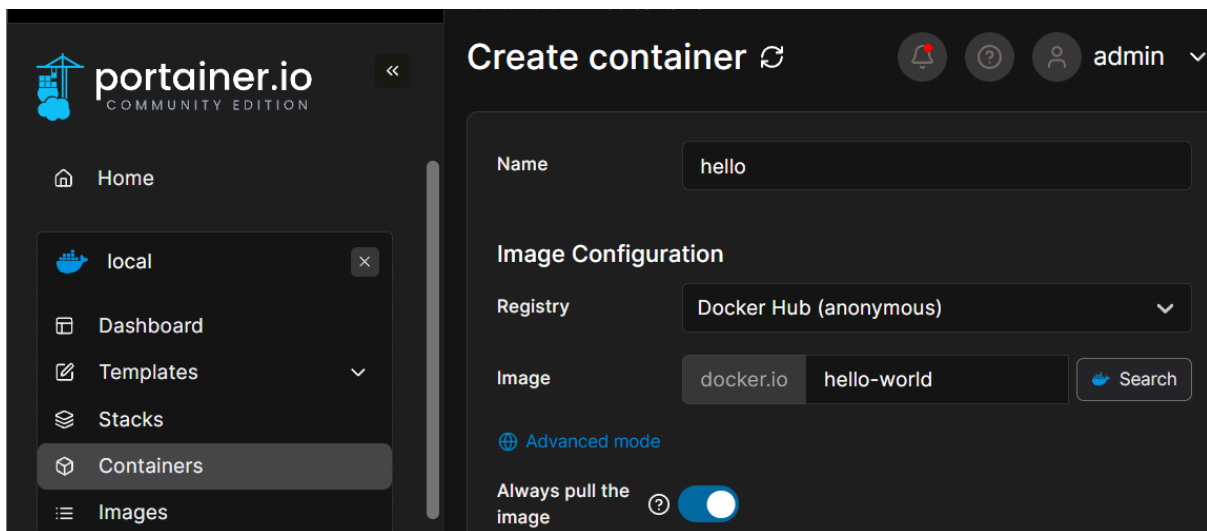
```
sudo docker restart portainer
```

Refaire les Jobs 2 à 9 dans Portainer

Job 2 – Hello World

- Va dans "Containers" > "Add container"

- Nom : `hello`
- Image : `hello-world`
- Clique sur "Deploy the container"
- Vérifie les logs pour voir le message "Hello from Docker"



```
This message shows that your installation appears to be working correctly.
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/
For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Job 3 – Dockerfile pour hello-world

- Crée un Dockerfile minimal en local (sur ta VM) avec :

```
FROM debian:bullseye-slim
```

```
CMD echo "Hello from my custom image!"
```

- Dans Portainer > "Images" > "Build a new image"
 - Upload le Dockerfile et construis l'image.
 - Lance le conteneur depuis cette image.
-


Job 4 – Conteneur SSH personnalisé

- Dockerfile à préparer :

```
FROM debian:bullseye
RUN apt update && apt install -y openssh-server
RUN echo 'root:root123' | chpasswd
RUN mkdir /var/run/sshd
EXPOSE 2222
CMD ["/usr/sbin/sshd", "-D"]
```

- Build dans Portainer (comme Job 3)
 - Lors du déploiement :
 - Map le port 2222 du conteneur vers, par exemple, 2222 sur l'hôte.
 - Teste via : `ssh root@IP -p 2222`
-

Job 5 – Alias Docker

 Ce job est spécifique au terminal. Il ne s'applique pas dans Portainer.

Job 6 – Volumes

- Dans "Volumes", crée un volume partagé : `shared-data`

- Dans deux conteneurs (ex. deux Debian), utilise `shared-data` comme volume monté sur `/data`
 - Crée un fichier dans `/data` dans un conteneur, vérifie depuis l'autre.
-

Job 7 – Nginx + FTP + Volume partagé

- Utilise **Stacks** dans Portainer pour importer un fichier `docker-compose.yml`
- Exemple de `docker-compose.yml` :

```
version: '3'
services:
  nginx:
    image: nginx
    ports:
      - "8080:80"
    volumes:
      - webdata:/usr/share/nginx/html

  ftp:
    image: fauria/vsftpd
    ports:
      - "21:21"
    environment:
      - FTP_USER=user
      - FTP_PASS=pass
    volumes:
      - webdata:/home/vsftpd

volumes:
  webdata:
```

- Envoie `index.html` via FileZilla.
- Affiche via `http://IP:8080`

Job 8 – Créer image Nginx sans image nginx

- Dockerfile à créer :

```
FROM debian
RUN apt update && apt install -y nginx
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- Build image depuis Portainer
 - Lance un conteneur en mappant le port 80
-

Job 9 – Registry local + UI

- Dans Portainer > "Stacks" > crée un `docker-compose.yml` avec :

```
version: '3'
services:
  registry:
    image: registry:2
    ports:
      - "5000:5000"
    volumes:
      - registry-data:/var/lib/registry

  ui:
    image: joxit/docker-registry-ui:static
    environment:
      - REGISTRY_TITLE=MyRegistry
      - REGISTRY_URL=http://registry:5000
    ports:
      - "8081:80"
    depends_on:
      - registry
```

```
volumes:  
  registry-data:
```

- Accède à ton UI de registre : <http://IP:8081>

Alternatives à Portainer

Voici quelques alternatives à Portainer à noter pour ton rapport :

Nom	Avantage principal
Rancher	Gère Docker et Kubernetes
DockStation	Interface riche et ergonomique (desktop)
LazyDocker	TUI dans le terminal, très léger
Kitematic	Ancien outil Docker officiel (déprécié)

Job 10 - Scripts d'automatisation

Objectif : Créer deux scripts bash, l'un pour désinstaller Docker complètement, l'autre pour l'installer.

Contenu :

- Script `uninstall_docker.sh` (avec commentaires)
- Script `install_docker.sh` (avec commentaires)

```
#!/bin/bash
```

```
echo "Mise à jour des paquets..."  
apt-get update && apt-get upgrade -y
```

```
echo "Installation des dépendances nécessaires..."  
apt-get install -y ca-certificates curl gnupg
```

```
echo "Ajout de la clé GPG officielle de Docker..."  
install -m 0755 -d /etc/apt/keyrings  
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg  
--dearmor -o /etc/apt/keyrings/docker.gpg
```

```
echo "Ajout du dépôt Docker dans APT..."  
echo \  
    "deb [arch=$(dpkg --print-architecture)  
signed-by=/etc/apt/keyrings/docker.gpg] \  
    https://download.docker.com/linux/debian $(. /etc/os-release &&  
echo "$VERSION_CODENAME") stable" \  
    > /etc/apt/sources.list.d/docker.list
```

```
echo "Mise à jour des sources..."  
apt-get update
```

```
echo "Installation de Docker Engine et CLI..."  
apt-get install -y docker-ce docker-ce-cli containerd.io  
docker-buildx-plugin docker-compose-plugin
```

```
echo "Activation de Docker..."  
systemctl enable docker  
systemctl start docker
```

```
echo "Docker est installé et prêt à l'emploi."
```