

TP N°11 BLOC N°2



Déploiement du service web Nginx dans un conteneur géré à l'aide des composants logiciels de Docker

SOMMAIRE

Déploiement du service web Nginx dans un conteneur géré à l'aide des composants logiciels de DOCKER

Etape 1 : Création de la VM "SRVDOCKER" et installation des composants logiciels de Docker.....Page 1

Etape 2 : Configuration de Docker et test du conteneur de démonstration "HelloWorld".....Page 2

Etape 3 : Gestion des images et des conteneurs sur Docker.....Page 3

Etape 4 : Exposition d'un conteneur "Nginx" au travers du réseau Docker.....Page 4

Etape 5 : La gestion des volumes et la persistance des données.....Page 5

ETAPE 1 : Création de la VM "SRVDOCKER" et installation des composants logiciels de Docker

1.) Pour commencer ce TP sur le déploiement du service web Nginx dans un conteneur à l'aide de DOCKER, nous allons commencer par créer la machine (SRVDOCKER)



2.) Petit truc à préciser il existe deux possibilités pour installer les composants logiciels de Docker. Mais nous le ferons sur les serveurs de Debian.

Pour commencer mettez à jour votre système Debian, Pour cela saisissez la commande :

- `apt update && apt upgrade -y`

Puis installez les paquets "ca-certificates, curl, gnupg et lsb-release" pour cela saisissez la commande :

- `apt install ca-certificates, curl, gnupg et lsb-release -y`

3.) Après avoir installé les paquets nécessaires, nous allons créer un répertoire pour les clés de sécurité et ainsi télécharger de façon sécurisée les composants logiciels, pour cela saisissez la commande :

- `sudo install -m 0755 -d /etc/apt/keyrings`

Definition de la commande "sudo install -m 0755 -d /etc/apt/keyrings" :

- `install` : Indique qu'on veut créer un répertoire) `"/etc/apat/keyring"`
- `-m 0755` : Indique les droits sur le dossier : propriétaire = lecture/écriture/exécution // autres = lecture/exécution
- `-d` : Permet de créer un dossier
- `/etc/apt/keyrings` : fichier en question

Rôle : Cette commande crée le répertoire `/etc/apt/keyrings` avec des droits d'accès spécifiques, le tout en utilisant les privilèges administrateur.

4.) Après avoir créé le répertoire `"/etc/apt/keyrings"` Nous allons télécharger et ajouter la clé GPG de Docker, Pour cela saisissez la commande :

- `curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg`

Definition de la commande “curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg” :

- curl : Outil de transfert de données via URL (comme un navigateur en ligne de commande)
- -f (--fail) : Échoue silencieusement si le serveur retourne une erreur HTTP (ex: 404)
- -s (--silent) : Mode silencieux, n’affiche pas la barre de progression ni les messages inutiles
- -S (--show-error) : Affiche les erreurs même en mode silencieux (utile pour le débogage)
- -L (--location) : Suit les redirections (si l’URL change, curl va à la bonne destination)
- https://download.docker.com/linux/debian/gpg : URL où se trouve la clé GPG publique officielle de Docker pour Debian
- gpg : Exécute la commande avec les privilèges administrateur (nécessaire pour écrire dans /etc/apt/keyrings/)
- --dearmor : Convertit une clé GPG du format ASCII(lisible) vers le format binaire (format que apt comprend)
- -o /etc/apt/keyrings/docker.gpg : Spécifie le fichier de sortie où sera enregistrée la clé binaire

Rôle : Cette commande va télécharger la clé GPG officielle de Docker, la convertir en format binaire, et l’enregistrer dans un fichier sécurisé que le gestionnaire de paquets apt utilisera pour authentifier les paquets Docker.

5.) Maintenant nous allons modifier les droits d’accès au fichier “docker.gpg” en ajoutant le droit en lecture et à tout le monde, pour cela saisissez la commande :

- **sudo chmod a+r /etc/apt/keyrings/docker.gpg**

6.) Nous allons créer un fichier nommé “docker.list” à l’emplacement “/etc/apt/sources.list.d/”, pour cela saisissez la commande :

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

7.) Editons le fichier “docker.list” en saisissant la commande :

- **cat /etc/apt/sources.list.d/**

```
root@tepson:/home/tepson# cat /etc/apt/sources.list.d/docker.list
deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian bookworm stable
root@tepson:/home/tepson#
```

Dans le fichier docker.list, on remarque que le dépôt officiel Docker est correctement configuré avec l’URL https://download.docker.com/linux/debian, l’architecture amd64, la distribution bookworm et le canal stable. On voit également que la clé GPG située dans /etc/apt/keyrings/docker.gpg est utilisée pour sécuriser et authentifier les paquets Docker.

Role :

- **deb** : Indique qu'il s'agit d'un dépôt contenant des paquets binaires précompilés (par opposition à deb-src pour les codes sources).
- **[arch=amd64]** : Précise l'architecture du système : amd64 pour les processeurs 64 bits (standard sur PC). Cela évite de télécharger des paquets pour d'autres architectures.
- **signed-by=/etc/apt/keysring/docker.gpg** : Spécifie le chemin vers la clé GPG qui sera utilisée pour signer et vérifier l'authenticité des paquets. C'est le fichier que nous avons créé avec curl et gpg --dearmor.
- **https://download.docker.com/linux/debian** : L'URL du dépôt officiel de Docker pour Debian. apt ira chercher les paquets à cette adresse.
- **bookworm** : Le nom de code de notre version de Debian (ici Debian 12 Bookworm). apt utilise cette information pour sélectionner les paquets compatibles.
- **stable** : La branche du dépôt : stable signifie que vous recevrez les versions stables et testées de Docker (recommandé). D'autres branches possibles : test ou nightly.

8.) Actualisons notre fichiers sources en saisons la commande "apt update"

9.) Maintenant installons Docker Ce et ses composant logiciel, pour cela saisissez la commande :

- **apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin**

10.) Vérifions l'état du service Docker engine, en saisisant la commande "systemctl status docker"

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enabled)
   Active: active (running) since Thu 2026-04-23 16:16:38 CEST; 31s ago
 TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Main PID: 2928 (dockerd)
     Tasks: 9
    Memory: 30.7M
       CPU: 717ms
    CGroup: /system.slice/docker.service
            └─2928 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

11.) Configurons le démarrage automatique de Docker au lancement de notre serveur, en saisisant les commande :

- **systemctl enable docker.service**
- **systemctl enable containerd.service**

12.) Arrêtons, démarrons puis redémarrons le service docker engine pour cela saisissez la commande :

- **systemctl stop docker**
- **systemctl start docker**
- **systemctl restart docker**

13.) Si vous voulez observez la version de docker saisissez la commande :

- `docker --version`

```
[root@tepson:/home/tepson# docker --version
Docker version 29.4.1, build 055a478
```

Commande pour la version du composant de docker :

- `docker compose version`

```
[root@tepson:/home/tepson# docker compose version
Docker Compose version v5.1.3
root@tepson:/home/tepson#
```

ATTENTION : SI UN MESSAGE D'ERREUR APPÂRAIT, RELANCEZ VOTRE SERVEUR DOCKER

14.) saisons les commandes “`docker ps`” puis “`docker ps -a`”

```
[root@tepson:/home/tepson# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
[root@tepson:/home/tepson# docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
root@tepson:/home/tepson#
```

bien que ici la commande montre des colonne mais “`docker ps`” lister les tous les conteneurs présents sur le système qui sont **ACTIF**, et la commande “`docker ps -a`” va lister les conteneurs présents sur le système qui sont soit **EN COURS D'EXECUTION, ARRETE, EN PAUSE, CREER MAIS JAMAIS UTILISER** .

Rôle des colonne :

- **CONTAINER ID** : Représente l'Identifiant unique du conteneur
- **IMAGE** : Indique l'image utilisé par le conteneur
- **COMMAND** : Représente la commande lancé au démarrage
- **CREATED** : Indique depuis combien de temps que le conteneur existe
- **STATUS** : Indique si le conteneur tourne (UP) ou arrêté (EXITED)
- **PORTS** : Indique le ports ouvert vers la machien sur ver le conteneur
- **NAMES** : Indique le nom du conteneurs

ETAPE 2 : Configuration de Docker et test du conteneur de démonstration "HelloWorld"

Pour commencer cette étape 2 qui consistera à configurer docker faire des test du conteneur de démonstration (HelloWorld), nous allons le compte utilisateur tepson afin qu'il accès aux services de docker en mode "moindre privilèges".

15.) Pour cela ajoutons l'utilisateur tepson au groupe utilisateur docker, pour cela saisissez la commande :

- `sudo usermod -aG docker tepson`

16.) Maintenant quitter le mode administrateur et maintenant lançons le conteneur de démonstration "HelloWorld", pour cela saisissez la commande :

- `docker run hello-world`

ATTENTION : SI UN MESSAGE D'ERREUR APPÂRAIT, RELANCEZ VOTRE SERVEUR DOCKER

17.) Pour testez au bon fonctionnement que tepson fait partie du groupe utilisateur docker, du conteneur "HelloWorld", saisissez les commande :

- `$docker ps`
- `$docker ps -a`

REMARQUE : A FAIRE SUR LE COMPTE TEPSON ET NON ROOT

```
[tepson@tepson:~]$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
[tepson@tepson:~]$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
1d023adbfd98   hello-world  "/hello"                2 hours ago  Exited (0) 2 hours ago  eager_einstein
tepson@tepson:~$
```

Nous avons fait la remarque précédemment que "docker ps" liste les conteneurs **ACTIF** et "docker ps -a" les conteneur **EN COURS D'EXECUTION, ARRETE, EN PAUSE, CREER MAIS JAMAIS UTILISE.est** Ici on constate que notre conteneur avec l'image "Hello-world" est (ont suppose) en arret.

18.) Maintenant depuis le compte root, nous allons créer un fichier json que nous allons appelé "daemon.json" dans l'emplacement "/etc/docker", saisissez la commande :

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

Personnalisons la configuration de "Docker"

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "data-root": "/var/lib/docker",
  "live-restore": true
}
```

19 & 20.) Petite rappel un fichier JSON (JavaScript Object Notation) est un format léger et lisible qui structure des données sous forme de paires clé/valeur.

Rôle du contenu du fichier "daemon.json"

- **log-driver** : Va définir le type de stockage des logs. "json-files" est le format standard (fichiers JSON lisibles)
- **log-opts** : Est une option supplémentaire pour gérer les logs (taille maximale).
- **max-size** : La taille maximale d'un fichier de log. Ici c'est 10m = 10 mégaoctets
- **max-file** : Représente le nombre maximum de fichiers de log (rotation). Ici 3 fichiers
- **data-root** : Le dossier où Docker stocke les images, conteneurs, volumes (ici /var/lib/docker).
- **live restore** : Si true, les conteneurs restent actifs même si le démon Docker redémarre

Finalement à quoi sert ce fichier daemon.json ?

Le fichier daemon.json est un fichier de configuration au format JSON qui permet d'adapter le comportement du démon Docker. Il offre un contrôle précis sur les logs, le stockage, et la résilience des conteneurs. Dans un environnement de conteneurisation, cela garantit une meilleure gestion des ressources, une supervision simplifiée.

21.) Maintenant redémarrons docker, pour cela saisissez la commande :

- **systemctl restart docker**

22.) Maintenant nous allons configurer les interfaces réseaux physiques et logiques des conteneurs sur Docker. Listons les interfaces réseaux disponibles sur notre "SRVDOCKER".

```
[root@tepson:/home/tepson# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:89:e9:9b brd ff:ff:ff:ff:ff:ff
    inet 10.1.5.228/24 brd 10.1.5.255 scope global dynamic enp0s3
        valid_lft 82484sec preferred_lft 82484sec
    inet6 fe80::a00:27ff:fe89:e99b/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 62:1c:3e:89:29:90 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::601c:3eff:fe89:2990/64 scope link
        valid_lft forever preferred_lft forever
root@tepson:/home/tepson#
```

Ici on constate 3 interfaces :

- **docker0** : bridge virtuel créé par Docker
- **lo** : Interface local de la machine
- **enp0s3** : réseau de l'hôte

De part l'interface docker0, on conclue et confirme que Docker a configuré son réseau interne

ETAPE 3 : Gestion des images et des conteneurs sur Docker

Dans cette étape 3, nous allons gérer des images et des conteneurs sur Docker

23.) Listons les images stockées en local, pour cela saisissez la commande

```
root@tepson:/home/tepson# docker images
```

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
hello-world:latest	f9878146db2e	25.9kB	9.49kB	U

```
root@tepson:/home/tepson#
```

On constate que la commande `docker images` affiche la liste des images téléchargées localement avec leur nom, leur version (tag), leur identifiant, leur date de création et leur taille. Dans notre TP, on voit les images `hello-world`, `ubuntu` et `nginx`.

24.)

- **IMAGE** : nom de l'image
- **ID** : Identifiant de l'image
- **DISK USAGE** : Taille de l'image sur le disque
- **CONTENT SIZE** : Taille de ce qui se trouve dans l'image

25.) Après avoir lister les images stockées en local dans l'étape précédente, nous allons rechercher les images basées sur le système Linux Ubuntu, pour cela saisissez la commande :

- `docker search ubuntu`

```
[tepson@tepson:~]$ docker search ubuntu
```

NAME	DESCRIPTION	STARS	OFFICIAL
ubuntu	Ubuntu is a Debian-based Linux operating sys...	17813	[OK]
ubuntu/squid	Squid is a caching proxy for the Web. Long-t...	125	
ubuntu/nginx	Nginx, a high-performance reverse proxy & we...	141	
ubuntu/bind9	BIND 9 is a very flexible, full-featured DNS...	117	
ubuntu/kafka	Apache Kafka, a distributed event streaming ...	60	
ubuntu/zookeeper	ZooKeeper maintains configuration informatio...	14	
ubuntu/apache2	Apache, a secure & extensible open-source HT...	103	
ubuntu/prometheus	Prometheus is a systems and service monitori...	80	
ubuntu/jre	Chiselled Java runtime based on Ubuntu. Long...	25	
ubuntu/dotnet-aspnet	Chiselled Ubuntu runtime image for ASP.NET a...	26	
ubuntu/mysql	MySQL open source fast, stable, multi-thread...	74	
ubuntu/python	A chiselled Ubuntu rock with the Python runt...	33	
ubuntu/postgres	PostgreSQL is an open source object-relatio...	44	
ubuntu/redis	Redis, an open source key-value store. Long...	25	
ubuntu/dotnet-runtime	Chiselled Ubuntu runtime image for .NET apps...	24	
ubuntu/dotnet-deps	Chiselled Ubuntu for self-contained .NET & A...	16	
ubuntu/grafana	Grafana, a feature rich metrics dashboard & ...	12	
ubuntu/memcached	Memcached, in-memory keyvalue store for smal...	6	
ubuntu/jdk	Ubuntu rock for OpenJDK	0	
ubuntu/loki	Grafana Loki, a log aggregation system like ...	2	
ubuntu/mimir	Ubuntu ROCK for Mimir, a horizontally scalab...	0	
ubuntu/rabbitmq	Ubuntu-based RabbitMQ image - an open source...	0	
ubuntu/opentelemetry-collector	This is a rock for the OpenTelemetry Collect...	0	
ubuntu/git-sync	Clone a git repo and keep it in sync with th...	3	
ubuntu/grafana-agent	Ubuntu ROCK for Grafana Agent, an open-sourc...	0	

```
tepson@tepson:~$
```

On constate que la commande `docker search ubuntu` affiche toutes les images liées à Ubuntu disponibles sur Docker Hub, avec leur popularité (stars) et leur statut officiel.

L'image officielle ubuntu est la plus téléchargée et recommandée

26.) Maintenant nous allons télécharger la dernière version du système ubuntu, saisissez la commande :

- `docker pull ubuntu:latest`

```
tepson@tepson:~$ docker images
```

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
hello-world:latest	f9078146db2e	25.9kB	9.49kB	U
ubuntu:latest	c4a8d5503dfb	119MB	31.7MB	

```
tepson@tepson:~$
```

Après le téléchargement, on constate que l'image "ubuntu:latest" est maintenant présent dans la liste des images locale, avec une taille de 31 Mo.

27.) Vérifions qu'aucun conteneur autre que "HelloWorld" soit présent sur SRVDOCKER, pour vérifier saisissez la commande :

- `docker ps` et `docker ps -a`

```
tepson@tepson:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1d023adbfd98	hello-world	"/hello"	7 hours ago	Exited (0)	7 hours ago	eager_einstein

```
tepson@tepson:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1d023adbfd98	hello-world	"/hello"	7 hours ago	Exited (0)	7 hours ago	eager_einstein

```
tepson@tepson:~$
```

On constate qu'aucun conteneur n'est en cours d'exécution (`docker ps`), mais le conteneur hello-world apparaît dans `docker ps -a` avec le statut Exited, car il s'est arrêté après avoir affiché son message

28.) Créons un nouveau conteneur, le conteneur "srvtest" qui se créera à partir de l'image "Ubuntu", Pour cela saisissez la commande :

- `docker run -tid --name srvtest ubuntu:latest`

Role de chaque partie de la commande "docker run -tid --name srvtest ubuntu:latest" :

- `docker run` : crée et démarre un conteneur (create + start)
- `-t` : permet d'avoir un Shell interactif
- `-i` : Garde
- `-d` : Lance le conteneur en arrière-plan. Le terminal ne reste pas bloqué sur le conteneur.
- `--name` : donne un nom au conteneur (dans mon cas srvtest)
- `ubuntu:latest` : Le nom de l'image

29.) Vérifions que notre conteneur est en cours d'exécution

```
tepson@tepson:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f1233288f03d	ubuntu:latest	"/bin/bash"	11 minutes ago	Up 11 minutes		srvtest

```
tepson@tepson:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f1233288f03d	ubuntu:latest	"/bin/bash"	12 minutes ago	Up 12 minutes		srvtest
1d023adbfd98	hello-world	"/hello"	8 hours ago	Exited (0)	8 hours ago	eager_einstein

```
tepson@tepson:~$
```

La commande `docker ps` affiche le conteneur `srvtest` avec le statut `Up`, car il exécute la commande `"/bin/bash"` qui reste active en arrière-plan. La commande `docker ps -a` montre également le conteneur `hello-world` avec le statut `Exited (0)`, ce qui signifie qu'il s'est arrêté normalement après son exécution. Les colonnes `CREATED` et `STATUS` précisent depuis quand les conteneurs existent et leur état actuel, tandis que `COMMAND` indique le programme exécuté au démarrage. Ces informations permettent de savoir quels conteneurs tournent, lesquels sont arrêtés, et pourquoi.

30.) accédons au terminal au conteneur `srvtest`, pour cela saisissez la commande :

- `docker exec -it srvtest bash`

```
[tepson@tepson:~$ docker exec -it srvtest bash  
root@f1233288f03d:/#
```

On constate qu'on s'est connecté en mode interactif à l'intérieur du conteneur `srvtest`. Une fois connecté, le prompt change, indiquant qu'on est dans l'environnement isolé du conteneur.

saisissons la commande `"hostname"`, voyons ce qui se passe.

```
[root@f1233288f03d:/# hostname  
f1233288f03d
```

cette commande indique l'ID du conteneur

observons pour la commande `"cat /etc/os-release"`.

```
[root@f1233288f03d:/# cat /etc/os-release  
PRETTY_NAME="Ubuntu 24.04.4 LTS"  
NAME="Ubuntu"  
VERSION_ID="24.04"  
VERSION="24.04.4 LTS (Noble Numbat)"  
VERSION_CODENAME=noble  
ID=ubuntu  
ID_LIKE=debian  
HOME_URL="https://www.ubuntu.com/"  
SUPPORT_URL="https://help.ubuntu.com/"  
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"  
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"  
UBUNTU_CODENAME=noble  
LOGO=ubuntu-logo  
root@f1233288f03d:/#
```

Sur les 2 premières lignes on observe le nom de l'OS utilisé c'est à dire `ubuntu`, cela confirme et indique que le conteneur est bien basé sur une distribution Ubuntu. La 1ère ligne indique et précise la version Ubuntu utilisée (ici `24.04 .4 LTS`).

Quittons le terminal du conteneur srvtest

31.) Maintenant nous allons arrêter l'exécution du conteneur "srvtest" pour cela saisissez la commande :

- `docker stop srvtest`

Vérifions qu'il est bien arrêté en saisissant les commandes :

- `docker ps` et `docker ps -a`

```
tepson@tepson:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
tepson@tepson:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED        STATUS              PORTS   NAMES
f1233288f03d   ubuntu:latest   "/bin/bash"   39 minutes ago   Exited (137) 15 seconds ago           srvtest
1d023adbfd98   hello-world     "/hello"      8 hours ago     Exited (0) 8 hours ago                eager_einstein
tepson@tepson:~$
```

Après avoir exécuté `docker stop srvtest`, la commande `docker ps` n'affiche plus le conteneur (car il est arrêté), mais la commande `docker ps -a` confirme qu'il est bien arrêté avec le statut `Exited (137)`, preuve qu'il ne tourne plus.

32.) Maintenant nous allons créer un nouveau conteneur que nous allons appeler "srvprod" qui sera basé sur l'image de la dernière version d'Ubuntu, pour cela saisissez la commande :

- `docker run -tid --name srvprod ubuntu:latest`

Vérifions son état en saisissant la commande :

- `docker ps`

```
tepson@tepson:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED        STATUS       PORTS   NAMES
01211c177342   ubuntu:latest   "/bin/bash"   About a minute ago   Up About a minute           srvprod
tepson@tepson:~$
```

On constate que le conteneur `srvprod` est bien actif, de part le fait que son statut est `UP`.

Vérifions également le nom du système et la version de Ubuntu installé, pour cela saisissez la commande :

- `cat /etc/os-release`

```
[tepson@tepson:~]$ docker exec -it srvprod bash
root@01211c177342:/# cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.4 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.4 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
root@01211c177342:/#
```

On constate que pareil que pour le conteneur srvtest, le conteneur srvprod utilise l'os Ubuntu avec la version (24.04.4 LTS)

33.) Maintenant relançons le conteneur srvtest, pour cela saisissez la commande :

- `docker start srvtest`

Observons les conteneurs actifs (en cours d'exécution), pour cela saisissez la commande :

- `docker ps`

```
[tepson@tepson:~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
01211c177342	ubuntu:latest	"/bin/bash"	21 minutes ago	Up 21 minutes		srvprod
f1233288f03d	ubuntu:latest	"/bin/bash"	About an hour ago	Up 11 minutes		srvtest

```
tepson@tepson:~$
```

34.) Maintenant nous allons tenter de supprimer l'image Docker de la dernière version d'Ubuntu, pour cela saisissez la commande :

- `docker rmi ubuntu:latest`

```
[tepson@tepson:~]$ docker rmi ubuntu:latest
Error response from daemon: conflict: unable to delete ubuntu:latest (must be forced) - container f1233288f03d is using its referenced image c4a8d5503dfb
tepson@tepson:~$
```

On constate que la commande `docker rmi ubuntu:latest` échoue, car le conteneur srvtest (même arrêté) utilise encore cette image.

35.) Pour supprimer les images de Ubuntu et les conteneurs basés dessus c'est à dire srvtest et srvpro, il faudrait comment par arrêter les deux conteneurs avant en saisissant la commande :

- `docker stop srvtest srvprod`

Puis supprimer les conteneurs :

- `docker rm srvtest srvprod`

puis supprimer les images Ubuntu :

- **`docker rmi ubuntu:latest`**

ETAPE 4 : Exposition d'un conteneur "Nginx" au travers du réseau Docker

Dans cette étape 4, nous allons exposer un conteneur au travers du réseau Docker

36.) tout d'abord nous allons rechercher les images contenant les composants logiciels de "Nginx", pour cela saisissez la commande :

- `docker pull nginx`

```
[root@tepson:/home/tepson# docker search nginx
NAME                                DESCRIPTION                                STARS    OFFICIAL
nginx                               Official build of Nginx.                  21257    [OK]
nginx/nginx-ingress                NGINX and NGINX Plus Ingress Controllers fo... 120
nginx/nginx-prometheus-exporter    NGINX Prometheus Exporter for NGINX and NGIN... 51
nginx/unit                         This repository is retired, use the Docker o... 66
nginx/nginx-ingress-operator       NGINX Ingress Operator for NGINX and NGINX P... 3
nginx/nginx-quic-qns               NGINX QUIC interop                        1
nginx/nginxaas-loadbalancer-kuber... 1
nginx/unit-preview                 Unit preview features                     0
bitnami/nginx                     Bitnami Secure Image for nginx            203
bitnamicharts/nginx               Bitnami Helm chart for NGINX Open Source    3
ubuntu/nginx                      Nginx, a high-performance reverse proxy & we... 141
kasmweb/nginx                     An Nginx image based off nginx:alpine and in... 9
rancher/nginx                     3
linuxserver/nginx                 An Nginx container, brought to you by LinuxS... 236
dtagdevsec/nginx                  T-Pot Nginx                              0
chainguard/nginx                   Build, ship and run secure software with Cha... 5
paketobuildpacks/nginx            0
vmware/nginx                      3
gluufederation/nginx              A customized NGINX image containing a consu... 1
antrea/nginx                      Nginx server used for Antrea e2e testing    0
cleanstart/nginx                  Secure by Design, Built for Speed, Hardened ... 0
activestate/nginx                 ActiveState's customizable, low-to-no vulner... 0
intel/nginx                       0
docksal/nginx                     Nginx service image for Docksal           1
geokrety/nginx                    Our customized nginx image                0
root@tepson:/home/tepson#
```

voici les image de conteannt les composans logiciels de Ngix. Mais celle qui nous intéresse c'est la toute 1er

37.) Téléchargeons l'image qui contient uniquement "Ngix", pour cela saisissez la commande :

- `docker pull nginx`

38.) Vérifions la présence de Nginx dans la liste des image Docker, pour cela saisissez a commande :

- `docker images | grep nginx`

```
[root@tepson:/home/tepson# docker images | grep nginx
WARNING: This output is designed for human readability. For machine-readable output, please use --format.
nginx:latest        6e23479198b9        240MB        65.8MB
root@tepson:/home/tepson#
```

39.) Maintenant créons et lançons le conteneur "srvweb", pour cela saisissez la commande :

- `docker run -d --name srvweb -p 8080:80 nginx`

Ici cette commande permet de créer et lancer le conteneur srvweb basé sur l'image Nginx, avec exposition des ports.

Rôle de chaque partie de la commande :

- `docker run` : Permet de créer et démarrer un conteneur
- `-d` : permet de lancer le conteneur en arrière-plan
- `--name srvweb` : Le nom du conteneur (srvweb)
- `-p 8080:80` : La Redirection de ports : le ports 8080 de l'hôte est lié au port du conteneur
- `nginx` : l'image docker utilisé (Nginx)

40.) Vérifions que le conteur srvweb est bien en cours d'exécution

```
root@tepson:/home/tepson# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
cb84f8d94a53   nginx         "/docker-entrypoint..." 8 hours ago    Up 8 hours    0.0.0.0:8080->80/tcp, [::]:8080->80/tcp  srvweb
01211c177342   ubuntu:latest "/bin/bash"             10 hours ago    Up 10 hours                               srvprod
f1233288f03d   ubuntu:latest "/bin/bash"             11 hours ago    Up 10 hours                               srvtest
root@tepson:/home/tepson#
```

On constate que le conteneur est bien en cours d'exécution car le statut du conteneur est UP (Actif)

41.) SaAfin d'observer le contenu du fichier de configuration du conteneur srvweb, saisissez la commande :

- `docker inspect srvweb | more`

```
[root@tepson:/home/tepson# docker inspect srvweb | more
```

```
[
  {
    "Id": "cb84f8d94a534aaf91b1838e8af70bb35ae8fc8be9d6c92cf29a091cccf50447",
    "Created": "2026-04-24T03:17:07.963038049Z",
    "Path": "/docker-entrypoint.sh",
    "Args": [
      "nginx",
      "-g",
      "daemon off;"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 3408,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2026-04-24T03:17:08.320591176Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "sha256:6e23479198b998e5e25921dff8455837c7636a6711a04a635cf1bb363d199dc",
    "ResolvConfPath": "/var/lib/docker/containers/cb84f8d94a534aaf91b1838e8af70bb35ae8fc8be9d6c92cf29a091cccf50447/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/cb84f8d94a534aaf91b1838e8af70bb35ae8fc8be9d6c92cf29a091cccf50447/hostname",
    "HostsPath": "/var/lib/docker/containers/cb84f8d94a534aaf91b1838e8af70bb35ae8fc8be9d6c92cf29a091cccf50447/hosts",
    "LogPath": "/var/lib/docker/containers/cb84f8d94a534aaf91b1838e8af70bb35ae8fc8be9d6c92cf29a091cccf50447/cb84f8d94a534aaf91b1838e8af70bb35ae8fc8be9d6c92cf29a091cccf50447-json.log",
    "Name": "/srvweb",
    "RestartCount": 0,
    "Driver": "overlayfs",
    "Platform": "linux",
    "MountLabel": "",
    "ProcessLabel": "",
    "AppArmorProfile": "docker-default",
    "ExecIDs": null,
    "HostConfig": {
      "Binds": null,
      "ContainerIDFile": "",
      "LogConfig": {
        "Type": "json-file",
        "Config": {
          "max-file": "3",
          "max-size": "10m"
        }
      },
      "NetworkMode": "bridge",
      "PortBindings": {
        "80/tcp": [
          {
            "HostIp": "",
            "HostPort": "8080"
          }
        ]
      },
      "RestartPolicy": {
        "Name": "no",
        "MaximumRetryCount": 0
      },
      "AutoRemove": false,
      "VolumeDriver": "",
      "VolumesFrom": null,
      "ConsoleSize": [

```

```

        64,
        207
    ],
    "CapAdd": null,
    "CapDrop": null,
    "CgroupnsMode": "private",
    "Dns": null,
    "DnsOptions": [],
    "DnsSearch": [],
    "ExtraHosts": null,
    "GroupAdd": null,
    "IpcMode": "private",
    "Cgroup": "",
    "Links": null,
    "OomScoreAdj": 0,
    "PidMode": "",
    "Privileged": false,
    "PublishAllPorts": false,
    "ReadonlyRootfs": false,
    "SecurityOpt": null,
    "UTSMode": "",
    "UsernsMode": "",
    "ShmSize": 67108864,
    "Runtime": "runc",
    "Isolation": "",
    "CpuShares": 0,
    "Memory": 0,
    "NanoCpus": 0,
    "CgroupParent": "",
    "BlkioWeight": 0,
    "BlkioWeightDevice": [],
    "BlkioDeviceReadBps": [],
    "BlkioDeviceWriteBps": [],
    "BlkioDeviceReadIOps": [],
    "BlkioDeviceWriteIOps": [],
    "CpuPeriod": 0,
    "CpuQuota": 0,
    "CpuRealtimePeriod": 0,
    "CpuRealtimeRuntime": 0,
    "CpusetCpus": "",
    "CpusetMems": "",
    "Devices": [],
    "DeviceCgroupRules": null,
    "DeviceRequests": null,
    "MemoryReservation": 0,
    "MemorySwap": 0,
    "MemorySwappiness": null,
    "OomKillDisable": null,
    "PidsLimit": null,
    "Ulimits": [],
    "CpuCount": 0,
    "CpuPercent": 0,
    "IOMaximumIOps": 0,
    "IOMaximumBandwidth": 0,
    "MaskedPaths": [
        "/proc/acpi",
        "/proc/asound",
        "/proc/interrupts",
        "/proc/kcore",
        "/proc/keys",
        "/proc/latency_stats",
        "/proc/sched_debug",
        "/proc/scsi",

```

42 & 43.) Maintenant affichons les informations des configuration IP de Docker, pour cela saisissez la commande :

- **docker network inspect bridge | more**

```
root@tepson:/home/tepson# docker network inspect bridge | more
[
  {
    "Name": "bridge",
    "Id": "a2a827048b4f8a7030b6032e70640eed1fccaf9f448d99b64dc2c937d91c6eca",
    "Created": "2026-04-23T21:01:31.060509714+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv4": true,
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {},
    "Containers": {
      "01211c177342d37f1ed74bc47a98c73a1b4a815d752ffa804dcdfd131475ee78": {
        "Name": "srvprod",
        "EndpointID": "fcc3c4ffc66ebdca4ae985dbfd29500fe2aaf5a3291203d4c0b635fb8da98a6",
        "MacAddress": "f2:76:16:fd:a1:50",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      },
      "cb84f8d94a534aaf91b1838e8af70bb35ae8fc8be9d6c92cf29a091ccccf50447": {
        "Name": "srvweb",
        "EndpointID": "09307532e62be18d73e5d08379309dc906fdb84e5e29ca5897676c21aa8c10b4",
        "MacAddress": "4e:05:b8:81:d4:b5",
        "IPv4Address": "172.17.0.4/16",
        "IPv6Address": ""
      },
      "f1233288f03de28354b52501a548982cbbd3c9340a0f2b4509c1fc0d1ce74b75": {
        "Name": "srvtest",
        "EndpointID": "6ba989cad35e54dae9abaa28d4726c4698ac20f4e71dce71167df61f566067d4",
        "MacAddress": "22:d6:f1:37:f8:e3",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
      }
    },
    "Status": {
      "IPAM": {
        "Subnets": [
          {
            "172.17.0.0/16": {
              "IPsInUse": 6,

```

On constate que l'adresse ip attribué par docker sur le réseau bridge pour le conteneur "srvweb" est 172.17.0.4 ont peut voir également la passerelle qui est 172.17.0.1 avec un masque en /16

En faite le réseau docker0 agit comme un pont virtuel entre l'hôte et les conteneurs

44.) Accédons au conteneur "srvweb" en ligne via le bash, pour cela saisissez la commande :

- **docker exec -it srvweb bash**

```
[root@tepson:/home/tepson# docker exec -it srvweb bash
root@cb84f8d94a53:/#
```

Maintenant nous allons modifier la page d'accueil par défaut de Nginx via le fichier index.html

Tout d'abord mettons a jour notre conteneur et ainsi installer également les paquet a jour qui nous seront utiles, pour cela saisissez la commande :

- apt update -y

ont constate que nano n'est pas télécharger par défaut, pour y remédier saisissez la commande :

- apt install nano -y

Éditons le fichier index.html en saisissant la commande :

- nano /usr/share/nginx/html/index.html

```
GNU nano 0.4 /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to 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>Welcome to nginx!</h1>
<p>If you see this page, nginx is successfully installed and working.
Further configuration is required for the web server, reverse proxy,
API gateway, load balancer, content cache, or other features.</p>

<p>For online documentation and support please refer to
<a href="https://nginx.org/">nginx.org</a>.<br/>
To engage with the community please visit
<a href="https://community.nginx.org/">community.nginx.org</a>.<br/>
For enterprise grade support, professional services, additional
security features and capabilities please refer to
<a href="https://f5.com/nginx">f5.com/nginx</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Remplacer le code html par celui-ci :

<!DOCTYPE html>

<html>

<head>

<title>Les Jardins de Saint-Eloi</title>

</head>

<body>

<h1>Bienvenue sur le site des Jardins de Saint-Eloi !</h1>

</body>

</html>

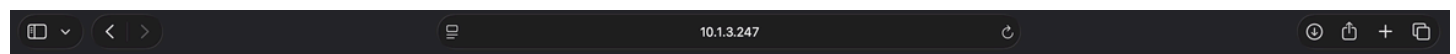
```
GNU nano 8.4 /usr/share/nginx/html/index.html
!DOCTYPE html>
<html>
<head>
  <title>Les Jardins de Saint-Eloi</title>
</head>
<body>
  <h1>Bienvenue sur le site des Jardins de Saint-Eloi !</h1>
</body>
</html>
```

45.) Maintenant depuis votre machine hôte (physique), saisissez dans votre navigateur, l'adresse ip de votre de votre (SRVDOCKER) avec le port 8080 :

- http://Adreese_Ip_SRVDOCKER:8080

pour moi c'est :

- <http://10.1.3.247:8080>



Bienvenue sur le site des Jardins de Saint-Eloi !

46.) En réaccédant au conteneur avec `docker exec -it srvweb bash`, on vérifie que le fichier `index.html` contient toujours le texte personnalisé. Le site web affiche bien "Bienvenue sur le site des Jardins de Saint-Eloi", confirmant que la modification est active.

ETAPE 5 : La gestion des volumes et la persistance des données

Dans cette dernière étape nous allons gerer les volumes et la persistance des données

47.) commençons par arrêter et supprimer le conteneur srvweb en saisissant les commande :

- `docker stop srvweb puis docker rm srvweb`

48.) Maintenant créons un conteneur appelé "srvweb" basé sur Nginx". On profitera pour créer un volume représenté par le répertoire `"/var/www/html"`, pour garantir la persistance des données du site web, pour cela saisissez la commande :

- `docker run -d --name srvweb -p 8080:80 -v /var/www/html:/usr/share/nginx/html nginx`

```
root@tepson:/home/tepson# docker run -d --name srvweb -p 8080:80 -v /var/www/html:/usr/share/nginx/html nginx
0824ed73bab7d803b20cf66f1bbc0ed80b7c9dcae0f8ff253a4b2f2c8ae93351
root@tepson:/home/tepson#
```

49.) Maintenant créons le fichier "index.html" dans le volume créé sur la machine hôte, pour cela saisissez la commande :

- `nano /var/www/html/index.html`

50. Maintenant depuis votre machine hôte (physique), saisissez dans votre navigateur, l'adresse ip de votre de votre (SRVDOCKER) avec le port 8080 :

- `http://Adreese_Ip_SRVDOCKER:8080`

pour moi c'est :

- `http://172.20.10.5:8080`



***L'ADRESSE A CHANGÉ CAR JE ME SUIS CONNECTÉ AU PARTAGE DE CONNEXION DE MON TELEPHONE**

51.) Arrêtons et supprimer le conteneur maintenant pour cela saisissez la commande :

- **docker stop srvweb puis docker rm srvweb**

52.) Recréer le conteneur avec la même commande.



on constate que le site est toujours accessible bien que on est supprimer le conteneur et recréer

53.) listons le conteneur le contenu du fichier "index.html". d'abord aller dans le conteneur en saisissant la commande :

- `docker exec -it srvweb bash`

déplacer vous dans le répertoire, `"/usr/share/nginx/html"`, pour cela saisissez la commande :

- `cd /usr/share/nginx/html`

listons le contenu du répertoire en saisissant la commande `"ls -la"`

```
[root@4e76fa5ad286:/usr/share/nginx/html# ls -la
total 12
drwxr-xr-x 2 root root 4096 Apr 24 13:50 .
drwxr-xr-x 3 root root 4096 Apr 22 01:22 ..
-rw-r--r-- 1 root root 169 Apr 24 13:50 index.html
```

maintenant affichons le contenu du fichier "index.html"

```
[root@4e76fa5ad286:/usr/share/nginx/html# cat index.html
<!DOCTYPE html>
<html>
<head>
    <title>Les Jardins de Saint-Eloi</title>
</head>
<body>
    <h1>Bienvenue sur le site des Jardins de Saint-Eloi !</h1>
</body>
</html>
root@4e76fa5ad286:/usr/share/nginx/html#
```

En accédant au conteneur avec `docker exec -it srvweb bash`, puis en allant dans `/usr/share/nginx/html`, on constate que le fichier `index.html` est bien présent et contient le texte personnalisé. Cela prouve que le volume monté entre l'hôte (`/var/www/html`) et le conteneur fonctionne correctement : les fichiers sont persistants et partagés.

CONCLUSION

On constate que le fichier `index.html` est présent dans `/usr/share/nginx/html` du conteneur et qu'il contient la page personnalisée. Le volume assure donc la persistance des données.

AU REVOIR