Chapitre 1 - Prise en main

Qu'est ce que Docker ?

Docker est une plateforme open-source qui permet aux développeurs de construire, déployer et exécuter des **conteneurs**.

Les conteneurs combinent :

- Le code source de l'application
- Les bibliothèques de l'OS
- Les dépendances nécessaires pour exécuter l'application dans n'importe quel environnement.

Photo de José M. Alarcón sur Unsplash

> La métaphore avec un porte conteneur est intéressante.
> Les conteneurs se ressemblent d'extérieur. Ils possèdent les mêmes dimensions et les mêmes dispositifs d'attache et d'empilement.
> Depuis l'intérieur d'un conteneur fermé, impossible d'accéder à l'extérieur.

Le terme Docker fait généralement référence à Docker Engine, le moteur d'exécution pour la construction et l'exécution des conteneurs.
Docker fait également référence à la société qui vend la version commerciale de Docker ou au projet open source Docker.

Docker a été créé en 2013 avant de devenir rapidement un incontournable dans le monde du développement logiciel.

Conteneurisation vs virtualisation

La conteneurisation de Docker est souvent comparée à de la virtualisation.

Le principe de base est le même : l'application conteneurisée est isolée du reste de la machine et ne peut pas sortir du conteneur, tout comme un système d'exploitation virtualisé ne peut pas sortir de sa machine virtuelle.

> La conteneurisation va plus loin : Le conteneur n'embarque pas de système d'exploitation mais il isole l'application du reste de la machine hôte : processus, système de fichiers, réseau, ressources CPU et RAM : tout est délimité et isolé du reste.

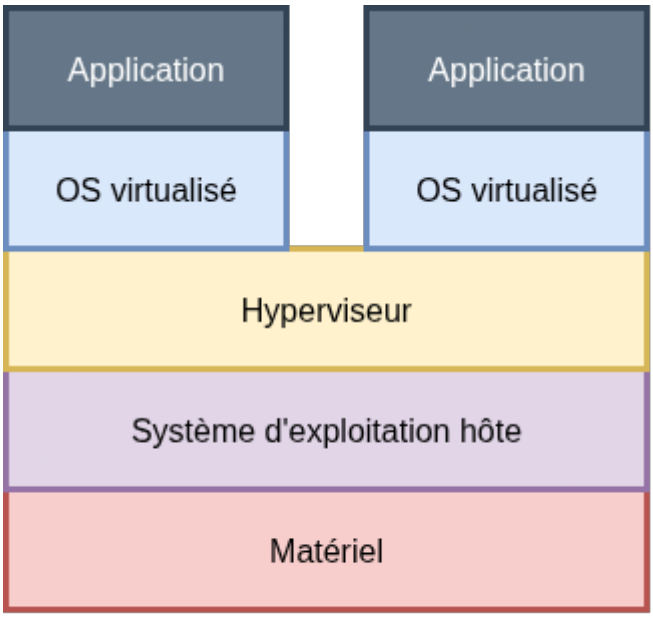
La conteneurisation offre tous les avantages de la virtualisation, notamment l'isolation et l'évolutivité, ainsi que d'autres avantages :

- **Poids**
  Contrairement aux machines virtuelles, les conteneurs n'embarquent pas le système d'exploitation. La taille d'un conteneur se mesure généralement en Mo (les VM en Go).

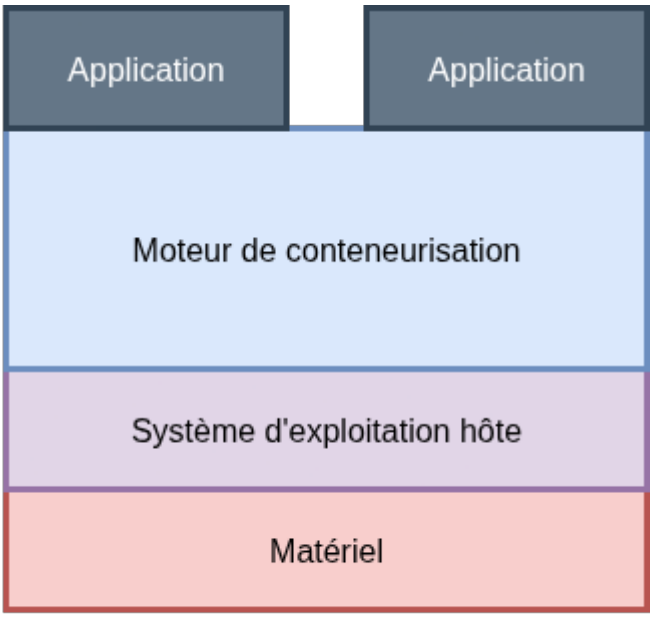
- **Portabilité**
  Les applications conteneurisées peuvent être écrites une seule fois et exécutées n'importe où.
  Par rapport aux machines virtuelles, les conteneurs sont plus rapides et plus faciles à déployer, déplacer, orchestrer, etc..

- **Efficacité**
  Avec les conteneurs, les développeurs peuvent exécuter plusieurs copies d'une application sur une même machine, sans atteindre les limites liées à l'utilisation de plusieurs machines virtuelles simultanées.

La différence peut être schématisée ainsi :



Virtualisation



Conteneurisation

Installation et hello-world

Docker Engine et Docker Desktop

Docker est nativement compatible avec Linux. Il peut être installé et piloté en ligne de commande.

Sous Windows, l'utilisation de Docker Desktop est recommandée, **même si elle n'est pas obligatoire**.
Il est tout à fait possible d'installer Docker dans WSL, sans Docker Desktop.

> Pour la suite de ce cours, nous n'utiliserons que la ligne de commande.

Installation

Linux

La documentation officielle propose une procédure d'installation pour la plupart des distributions : Install | Docker Docs

Un exemple sous Debian :

```
# suppression d'anciennes installations
for pkg in docker.io docker-doc docker-compose podman-docker containerd runc; do sudo apt-get remove $pkg; done

# Ajout du repository Docker officiel (clé PGP)
sudo apt-get update
sudo apt-get 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

# Ajout du repository Docker officiel (source APT)
echo \
```

```
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Installation des paquets
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin

# Démarrage automatique avec systemd (généralement automatique)
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

Par défaut, docker n'est pas utilisable depuis un compte non root. La documentation détaille comment rendre accessible le démon Docker : □□ Post-installation steps | Docker Docs

```
# Create the docker group.
sudo groupadd docker

# Add your user to the docker group.
sudo usermod -aG docker $USER

# Add without reboot
newgrp docker
```

Windows

Docker préconise l'installation de □□ Docker Destkop.

Sous le capot, Docker Desktop installe Docker Engine dans l'environnement virtualisé WSL ou Hyper-V sous Windows.

Il est tout à fait possible d'installer manuellement WSL sous Windows puis de suivre la procédure d'installation de Docker pour un environnement Linux.

MacOS

Voir documentation : □□ Docker Desktop Mac

Machine virtuelle Debian

> En cas de difficulté, une machine virtuelle Debian 13 prête à l'emploi est disponible ci-dessous.

Machine virtuelle Debian 13

Hello World

Il est possible de tester l'installation avec la commande suivante :

```
docker run hello-world
```

```
debian@debian:/$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:ec153840d1e635ac434fab5e377081f17e0e15afab27beb3f726c3265039cfff
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/

debian@debian:/$
```

- `hello-world` est une image docker permettant de vérifier son bon fonctionnement.

Commandes de base

Exécuter un conteneur : **docker run**

Commande de base :

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Le nom de l'image docker à utiliser doit être spécifié avec le paramètre `image`. Quelques exemples :

- Distributions Linux : `debian`, `ubuntu`, `alpine`
- Logiciels et CMS : `wordpress`, `prestashop`
- Langages de programmation : `php`, `python`
- Bases de données : `mysql`, `postgresql`

> Il existe actuellement 11 millions d'image (août 2025) sur le Docker Hub (que nous approfondirons plus tard).

La commande **docker run** propose de nombreux paramètres dont voici les principaux :

Paramètre	Description	Exemple

`--name`	Attribuer un nom au conteneur. Si il n'est pas défini, Docker générera un nom aléatoire à partir d'un dictionnaire de mots.	`docker run ubuntu --name=test1`
		`docker run ubuntu --name=test2`
		`docker run ubuntu --name=test3`

`-ti`	Les paramètres **-t** et **-i** permettent d'ouvrir un terminal interactif dans le conteneur : **-t** alloue un pseudo-terminal, et **-i** garde l'entrée ouverte.	`docker run -ti ubuntu` `# je suis redirigé dans la console du conteneur`

Paramètre	Description	Exemple
`-d`, `--detached`	Lance le conteneur en arrière-plan, en mode **détaché**.	`docker run -d ubuntu` `# le conteneur se lance en arrière-plan`

`--rm`	Supprimer le conteneur lorsqu'il est arrêté.	`docker run --rm -ti ubuntu`
		`# le conteneur se lance dans la console`
		`# Il sera supprimé après la sorite (exit)`

-e, --env	Passer une variable d'environnement à l'application conteneurisée	`docker run -e VAR1=value1` `# la variable VAR1 sera définie dans le conteneur`

```
debian@debian:~$ docker run -ti --name=test1 ubuntu
root@81aae328d11d:/# echo "Je suis dans mon conteneur"
Je suis dans mon conteneur
root@81aae328d11d:/# exit
exit
debian@debian:~$ docker run -d --name=test2 ubuntu
85d0402c2f0d48a7ec7dbc73cf8dc65fb66d3079e5ceb8c46a77580985569859
debian@debian:~$ echo "Mon conteneur test2 a été lancé en arrière-plan"
Mon conteneur test2 a été lancé en arrière-plan
debian@debian:~$
```

> Nous approfondirons plus tard d'autres paramètres de la commande **docker run**.

La commande `docker run` se termine par la commande à lancer dans le conteneur ainsi que ses paramètres. L'ajout de cette commande est optionnel, il n'est pas toujours nécessaire de l'indiquer.

Certaines images ne lancent pas automatiquement d'invite de commande `bash`.
Par exemple, l'image officielle python démarre sur une invite de commande python :

```
debian@debian:~$ docker run -ti python
Python 3.13.5 (main, Jul 22 2025, 04:26:21) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> str("Je suis en python ici car je n'ai pas précisé de commande.")
"Je suis en python ici car je n'ai pas précisé de commande."
>>> exit
debian@debian:~$ docker run -ti python bash
root@aafde6d872d3:/# echo "Maintenant je suis en bash dans mon conteneur python"
Maintenant je suis en bash dans mon conteneur python
root@aafde6d872d3:/# python --version
Python 3.13.5
```

Lister les conteneurs : **docker ps**

La commande `docker ps` liste les conteneurs démarrés sur la machine. Le paramètre `--all`, `-a` permet d'afficher les conteneurs stoppés.

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

```
debian@debian:~$ docker run -tid --name=test_1 ubuntu
8343b7be708201965140558 38b7130c9db59a2f04bb13ad062d7e1182d375a35
debian@debian:~$ docker run -ti --name=test_2 ubuntu
root@b5909c1ce8eb:/# exit
exit
debian@debian:~$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
8343b7be7082   ubuntu    "/bin/bash"   30 seconds ago   Up 30 seconds             test_1
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                      PORTS     NAMES
b5909c1ce8eb   ubuntu    "/bin/bash"   12 seconds ago   Exited (0) 4 seconds ago              test_2
8343b7be7082   ubuntu    "/bin/bash"   32 seconds ago   Up 32 seconds                         test_1
debian@debian:~$
```

- `test_1` est lancé en mode détaché (`-d`), il tourne en arrière-plan.
- `test_2` est lancé en mode attaché, il s'arrête lorsque je le quitte avec `exit`.

> Astuce : il est possible de se détacher d'un conteneur sans l'arrêter avec **CTRL+P** puis **CTRL+Q** !

```
debian@debian:~$ docker run -ti --name=test_3 ubuntu
root@53272edb658e:/# debian@debian:~$
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                      PORTS     NAMES
53272edb658e   ubuntu    "/bin/bash"   10 seconds ago   Up 10 seconds                         test_3
b5909c1ce8eb   ubuntu    "/bin/bash"   4 minutes ago    Exited (0) 4 minutes ago              test_2
8343b7be7082   ubuntu    "/bin/bash"   4 minutes ago    Up 4 minutes                          test_1
debian@debian:~$
```

Après avoir pressé **CTRL+P** puis **CTRL+Q**, je suis détaché du conteneur `test_3` qui tourne encore en arrière-plan.

Arrêt d'un conteneur : **docker stop** et **docker start**

```
docker start MON_CONTENEUR  # démarre le conteneur
docker stop MON_CONTENEUR # arrête le conteneur
docker rm MON_CONTENEUR # supprime le conteneur
```

```
debian@debian:~$ docker run -tid --name=toto alpine
353bc1b72eb8bc941e689c5f880054cc3290500ff84b8050f0a3dd15e9f1533e
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
353bc1b72eb8   alpine    "/bin/sh"   4 seconds ago   Up 3 seconds             toto
debian@debian:~$ docker stop toto
toto
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS                       PORTS     NAMES
353bc1b72eb8   alpine    "/bin/sh"   23 seconds ago   Exited (137) 4 seconds ago             toto
debian@debian:~$ docker start toto
toto
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS         PORTS     NAMES
353bc1b72eb8   alpine    "/bin/sh"   37 seconds ago   Up 7 seconds             toto
debian@debian:~$ docker stop toto
toto
debian@debian:~$ docker rm toto
toto
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
debian@debian:~$
```

Visualiser les logs : **docker logs**

```
docker logs MON_CONTENEUR
docker logs -f MON_CONTENEUR # -f pour --follow, permet de visualiser en direct le log.
```

```
debian@debian:~$ docker run -tid --name=test alpine date
d3ae72c6496375046f6101aeb036170f111d41fec491396c10c3a0a0466493f4
debian@debian:~$ docker logs test
Sat Aug  2 21:11:37 UTC 2025
debian@debian:~$
```

La commande date est lancée au démarrage du conteneur détaché, je ne vois pas le résultat.
J'y accède plus tard avec la commande `docker logs`.

S'attacher à un conteneur : **docker attach**

```
docker attach MON_CONTENEUR
```

```
debian@debian:~$ docker run -tid --name=test_debian debian
a436317b1b38823d1709539da831b8a9b45e5300eed1f42bce54fb9ff035ba92
debian@debian:~$ docker attach test_debian
root@a436317b1b38:/# echo "Je suis dans mon conteneur"
```

Le conteneur a été lancé en arrière-plan (mode détaché) avec `-d`. Je peux y accéder plus tard.

Exécuter une commande : **docker exec**

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

La syntaxe de cette commande ressemble à `docker run`. La différence est qu'ici, la commande est lancée dans un conteneur déjà en cours d'exécution.

La commande `docker exec` est utile pour lancer une invite de commande (`bash`) dans un conteneur qui n'en propose pas forcément. Exemple avec python :

```
debian@debian:~$ docker run -tid --name=test_python python
be97292531739fa1634c0d3c24f28b4afff0a4a983aa51ac5f0faba79c1b9b91
debian@debian:~$ docker attach test_python
>>> str('Je suis en Python après un docker attach')
'Je suis en Python après un docker attach'
>>> read escape sequence
debian@debian:~$ docker exec -ti test_python bash
root@be9729253173:/# echo "Et maintenant je suis en bash, dans le conteneur"
Et maintenant je suis en bash, dans le conteneur
root@be9729253173:/# exit
exit
debian@debian:~$ docker logs test_python
Python 3.13.5 (main, Jul 22 2025, 04:26:21) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> str('Je suis en Python après un docker attach')
'Je suis en Python après un docker attach'
debian@debian:~$
```

Je lance un conteneur python sans préciser de commande. La commande par défaut de l'image docker est utilisée.
Lorsque je m'attache au conteneur avec `docker attach`, je suis dans une invite de commande python.
Je me détache du conteneur et je lance une invite de commande `bash` avec `docker exec`.

La commande**docker exec** exécute un nouveau processus dans le conteneur. La sortie de ce processus n'est pas visible dans les logs du conteneur !

Images et système de fichiers

Images Docker

Une image Docker est un **fichier immuable** qui contient tout le nécessaire pour exécuter une application dans un conteneur.
Elle contient le code source, les bibliothèques, les dépendances, les variables d'environnement, les configurations, etc.

L'image Docker est une photographie figée d’un environnement logiciel, garantissant que l'application s’exécutera systématiquement de la même manière, indépendamment de son environnement.

Une image Docker est construite à partir d'un fichier de configuration appelé **Dockerfile.** Ce fichier décrit étape par étape comment assembler l'image.

> Nous reviendrons plus tard sur le Dockerfile.

Une fois construite, une image peut être stockée localement ou poussée vers un registre Docker (comme Docker Hub ou un registre privé), où elle peut être versionnée et partagée.

Pour identifier une image Docker, on utilise la syntaxe suivante :

```
propriétaire/nom:tag
```

- **propriétaire :** correspond à l'utilisateur ou à l'organisation qui possède l'image sur le registre.
- **nom :** le nom de l’image, comme `nginx`, `ubuntu`, etc.
- **tag :** une étiquette qui permet de versionner l'image. Par défaut, si aucun tag n'est spécifié, Docker utilise `latest`.
  Les tags peuvent représenter des versions (1.0, v2.3.4) ou des environnements (dev, prod).

> Certaines images "officielles" ne possèdent pas de propriétaire, donc il n'est pas nécessaire de le préciser.

Quelques exemples :

```
nginx:latest # image officielle de Nginx, version la plus récente.
thibaud/appli:1.2.0 # image personnalisée appartenant à l'utilisateur thibaud, version 1.2.0.
```

Registry Docker Hub

Comme expliqué ci-dessus, les images Docker doivent être stockées sur un **registre**.

Un registre peut être **public** ou **privé**, par exemple au sein d'une entreprise afin d'y stocker une application propriétaire.

Le registre public et par défaut est le Docker Hub. Docker Hub

La commande `docker pull` permet de télécharger une image depuis un registre, par défaut le registre public **Docker Hub**.

```
debian@debian:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
b71466b94f26: Pull complete
Digest: sha256:7c06e91f61fa88c08cc74f7e1b7c69ae24910d745357e0dfe1d2c0322aaf20f9
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
debian@debian:~$ docker pull ubuntu:latest
latest: Pulling from library/ubuntu
Digest: sha256:7c06e91f61fa88c08cc74f7e1b7c69ae24910d745357e0dfe1d2c0322aaf20f9
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
debian@debian:~$ docker pull ubuntu:24.04
24.04: Pulling from library/ubuntu
Digest: sha256:7c06e91f61fa88c08cc74f7e1b7c69ae24910d745357e0dfe1d2c0322aaf20f9
Status: Downloaded newer image for ubuntu:24.04
docker.io/library/ubuntu:24.04
debian@debian:~$
```

Actuellement, l'image `latest` d'Ubuntu correspond à la version 24.04 sur le Docker Hub. Ces trois identifiants retournent la même image, qui n'est téléchargée qu'une seule fois : `ubuntu`, `ubuntu:latest`, `ubuntu:24.04`.

À l'inverse, si je lance un `docker run` sur une image qui n'est pas déjà stockée localement, le téléchargement est automatique depuis le Docker Hub :

```
debian@debian:~$ docker run -ti ubuntu:18.04
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
7c457f213c76: Pull complete
Digest: sha256:152dc042452c496007f07ca9127571cb9c29697f42acbfad72324b2bb2e43c98
Status: Downloaded newer image for ubuntu:18.04
root@c59694091d14:/#
root@c59694091d14:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.6 LTS (Bionic Beaver)"
```

La commande `docker image` permet de gérer les images stockées localement.

```
docker image ls # lister les images
docker image rm ubutu:18.04 # supprimer l'image ubuntu taggée 18.04
```

```
debian@debian:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
debian        latest    047bd8d81940   3 days ago    120MB
hello-world   latest    1b44b5a3e06a   6 days ago    10.1kB
ubuntu        24.04     e0f16e6366fe   2 weeks ago   78.1MB
ubuntu        latest    e0f16e6366fe   2 weeks ago   78.1MB
ubuntu        18.04     f9a80a55f492   2 years ago   63.2MB
debian@debian:~$ docker image rm ubuntu:18.04
Untagged: ubuntu:18.04
Untagged: ubuntu@sha256:152dc042452c496007f07ca9127571cb9c29697f42acbfad72324b2bb2e43c98
Deleted: sha256:f9a80a55f492e823bf5d51f1bd5f87ea3eed1cb31788686aa99a2fb61a27af6a
Deleted: sha256:548a79621a426b4eb077c926eabac5a8620c454fb230640253e1b44dc7dd7562
debian@debian:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
debian        latest    047bd8d81940   3 days ago    120MB
hello-world   latest    1b44b5a3e06a   6 days ago    10.1kB
ubuntu        24.04     e0f16e6366fe   2 weeks ago   78.1MB
ubuntu        latest    e0f16e6366fe   2 weeks ago   78.1MB
debian@debian:~$
```

Système de fichiers

L'arborescence des fichiers dans un conteneur Docker est **isolé** du reste de la machine.

Un conteneur Docker contient une arborescence Linux. Exemple ci-dessous :

```
debian@debian:~$ docker run -ti ubuntu bash
root@a9a69b04bd71:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

L'arborescence ci-dessus n'est accessible que depuis le conteneur.

- Le conteneur n'a pas accès à l'arborescence de la machine hôte.

- La machine hôte n'a pas accès à l'arborescence du conteneur.

```
debian@debian:~$ echo "Docker c'est pratique" > /home/debian/test.txt
debian@debian:~$ cat /home/debian/test.txt
Docker c'est pratique
debian@debian:~$
debian@debian:~$ docker run -ti debian bash
root@fa14d7193dc3:/# cd /home
root@fa14d7193dc3:/home# ls
root@fa14d7193dc3:/home# mkdir debian
root@fa14d7193dc3:/home# echo "Docker c'est fabuleux" > /home/debian/test.txt
root@fa14d7193dc3:/home# cat /home/debian/test.txt
Docker c'est fabuleux
root@fa14d7193dc3:/home#
root@fa14d7193dc3:/home# exit
exit
debian@debian:~$ cat /home/debian/test.txt
Docker c'est pratique
debian@debian:~$
```

Volatilité

Contrairement à une **image** docker qui est figée, le contenu d'un container est **volatile**.

> Lorsqu'un container est supprimé, tous les fichiers modifiés sont perdus !

Exemple :

```
debian@debian:~$ docker run -ti --name=test debian
root@a4a1252f5a65:/# echo "Coucou Docker !" > /root/coucou.txt
root@a4a1252f5a65:/# cat /root/coucou.txt
Coucou Docker !
root@a4a1252f5a65:/# exit
exit
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NAMES
a4a1252f5a65   debian    "bash"    36 seconds ago   Exited (0) 6 seconds ago              test
debian@debian:~$ docker rm test
test
debian@debian:~$ docker run -ti --name=test debian
root@78368e00949e:/# cat /root/coucou.txt
cat: /root/coucou.txt: No such file or directory
root@78368e00949e:/# exit
exit
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NAMES
78368e00949e   debian    "bash"    24 seconds ago   Exited (1) 2 seconds ago              test
debian@debian:~$
```

Les deux conteneurs nommés `test` à partir de l'image `debian` ne sont pas les mêmes : ils n'ont pas le même identifiant.

Les données du premier conteneur sont perdues lors de sa suppression.

> Nous verrons plus loin les **volumes Docker** qui permettent de stocker des données persistantes.

Isolation : un exemple avec fastfetch

> L'hôte et le conteneur **partagent** quelques éléments tels que le noyau Linux, le CPU, la RAM disponible et l'espace disque.
> À l'inverse, la distribution Linux, les paquets installés et le système de fichier sont **isolés**.

Les volumes

Utilisation des volumes

Les volumes sont des répertoires accessibles depuis le conteneur et l'hôte.
Les volumes sont persistants, contrairement aux fichiers des conteneurs docker.

Les volumes sont très utiles avec Docker, ils peuvent servir à stocker, par exemple :

- Les données d'une base de données : il existe des images Docker pour la plupart des SGBD
- Les fichiers de configuration d'une application
- Les fichiers utilisateurs d'une application web (uploads)
- etc.

Une application dockerisée correctement configurée doit stocker ses données dans des volumes.

> L'avantage des volumes est la sauvegarde : seules les données **utiles** s'y trouvent et il est possible de dupliquer une application à partir d'une image docker et des volumes associés.

Un volume est accessible depuis le container à partir de son point de montage. Exemple :

```
debian@debian:~$ pwd
/home/debian
debian@debian:~$ mkdir test && echo "Les volumes Docker c'est pratique !" > test/contenu.txt
debian@debian:~$ docker run -ti -v /home/debian/test:/toto ubuntu
root@2b038b1f509b:/# cd /toto
root@2b038b1f509b:/toto# cat contenu.txt
Les volumes Docker c'est pratique !
root@2b038b1f509b:/toto# echo "Les volumes Docker c'est super !" > contenu.txt
root@2b038b1f509b:/toto# exit
exit
debian@debian:~$ cat test/contenu.txt
Les volumes Docker c'est super !
debian@debian:~$
```

Le dossier `/home/debian/test` de l'hôte est monté dans `/toto` au sein du conteneur avec le paramètre `-v`. L'hôte et le conteneur ont accès au répertoire.

```
-v /home/debian/test:/toto
```

Généralement, les images Docker d'applications indiquent les répertoires dans lesquels elles stockent leurs données utiles, par exemple :

- **MariaDB** stocke les fichiers de bases de données dans `/var/lib/mysql` (documentation).
- **Odoo** stocke des données dans `/var/lib/odoo` (documentation).
- **Redmine** stocke les fichiers uploadés dans `/usr/src/redmine/files` (documentation).

Volumes mappés vs. Volumes managés

Il existe deux types de volumes Docker : mappés et managés. Ils diffèrent par leur mode de gestion et leur usage :

- **Les volumes mappés** relient un dossier de l'hôte à un dossier du conteneur, ce qui permet un accès direct aux fichiers et facilite le développement.
  Les fichiers dépendent du système de fichiers de l'hôte, ce qui réduit la portabilité et peut poser des problèmes en production.

- **Les volumes managés** sont créés et gérés par Docker. Ils sont stockés dans un emplacement interne et peuvent être utilisés par plusieurs conteneurs. Ils offrent une meilleure isolation et une meilleure portabilité.
  Leur usage est idéal pour la persistance des données (bases de données, application, etc.), mais ils nécessitent des commandes Docker pour y accéder.

> L'exemple ci-dessus est un volume mappé.

Gestion des volumes managés

Les volumes managés docker sont gérés avec la commande `docker volume`.

```
docker volume create mon_volume # créer un volume
docker volume ls # lister les volumes
docker volume rm mon_volume # supprimer un volume
```

Lors de l'exécution d'un conteneur, il faut aussi utiliser `-v` pour monter un volume managé. À l'inverse d'un volume mappé, il faut indiquer le nom du volume plutôt que son chemin.

```
docker run -v mon_volume:/home/thibaud/volume ubuntu
# Le contenu du volume mon_volume sera monté dans /home/thibaud/volume au sein du conteneur
```

Le volume est indépendant du conteneur auquel il est attaché :

```
debian@debian:~$ docker volume create mon_volume
mon_volume
debian@debian:~$ docker run -ti --rm -v mon_volume:/home/toto ubuntu
root@592d0e235bd4:/# cd /home/toto/
root@592d0e235bd4:/home/toto# echo "Je suis dans un volume !" > contenu.txt
root@592d0e235bd4:/home/toto# ls
contenu.txt
root@592d0e235bd4:/home/toto# exit
exit
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
debian@debian:~$ docker volume ls
DRIVER    VOLUME NAME
local     mon_volume
debian@debian:~$ docker run -ti -v mon_volume:/var/test debian
root@213af518f0de:/# cd /var/test/
root@213af518f0de:/var/test# cat contenu.txt
Je suis dans un volume !
root@213af518f0de:/var/test#
```

- Le volume `mon_volume` est attaché à un conteneur `ubuntu` dans `/home/toto` et le fichier `contenu.txt` y est créé.
- Lorsque le conteneur est arrêté, il est automatiquement supprimé puisqu'il a été lancé avec `--rm`.
- Le volume `mon_volume` persiste et son contenu est préservé.
- Un nouveau conteneur `debian` est lancé et le volume `mon_volume` y est attaché dans `/var/test` : le fichier `contenu.txt` est disponible.

Une subtilité des volumes managés est **la priorité des données** si l'un des deux répertoires est vide lors du lancement d'un conteneur :

Volume	Point de montage (conteneur)	Action

Si vide	Si non vide	Les données initialement dans le conteneur sont copiées dans le volume.
Si non vide	Si vide	Le contenu du volume est monté dans le conteneur, comme dans l'exemple ci-dessus.
Si non vide	Si non vide	Le contenu du répertoire dans le conteneur est écrasé par celui du volume.

La gestion des ports

Redirection des ports

La plupart des applications dockerisées écoutent sur des ports spécifiques. Par exemple : serveurs web, bases de données, API, etc.

Pour les rendre accessibles depuis l'extérieur, **il faut rediriger ces ports** avec l'option `--port`, `-p`.
Cette option permet de lier un port du conteneur à un port de la machine hôte.

Dans le détail, lorsqu'une requête est envoyée à un port de la machine hôte, Docker la redirige automatiquement vers le port correspondant à l'intérieur du conteneur.

La syntaxe de `-p` est la suivante :

```
-p [IP:]PORT_HOTE:PORT_CONTENEUR
```

- `PORT_CONTENEUR` : le port sur lequel l'application à l'intérieur du conteneur écoute.
- `PORT_HOTE` : le port de la machine hôte sur lequel les requêtes externes seront reçues.
- `IP` **(facultative)** : permet de restreindre l'accès à une IP spécifique, par exemple `127.0.0.1`.

> La redirection de ports docker permet de répondre à des **cas d'usages fréquents** : contrôle d'accès aux applications, gestion des conflits de port avec des applications similaires, configuration d'architectures réseaux complexes, etc.

Exemples avec strm/helloworld-http

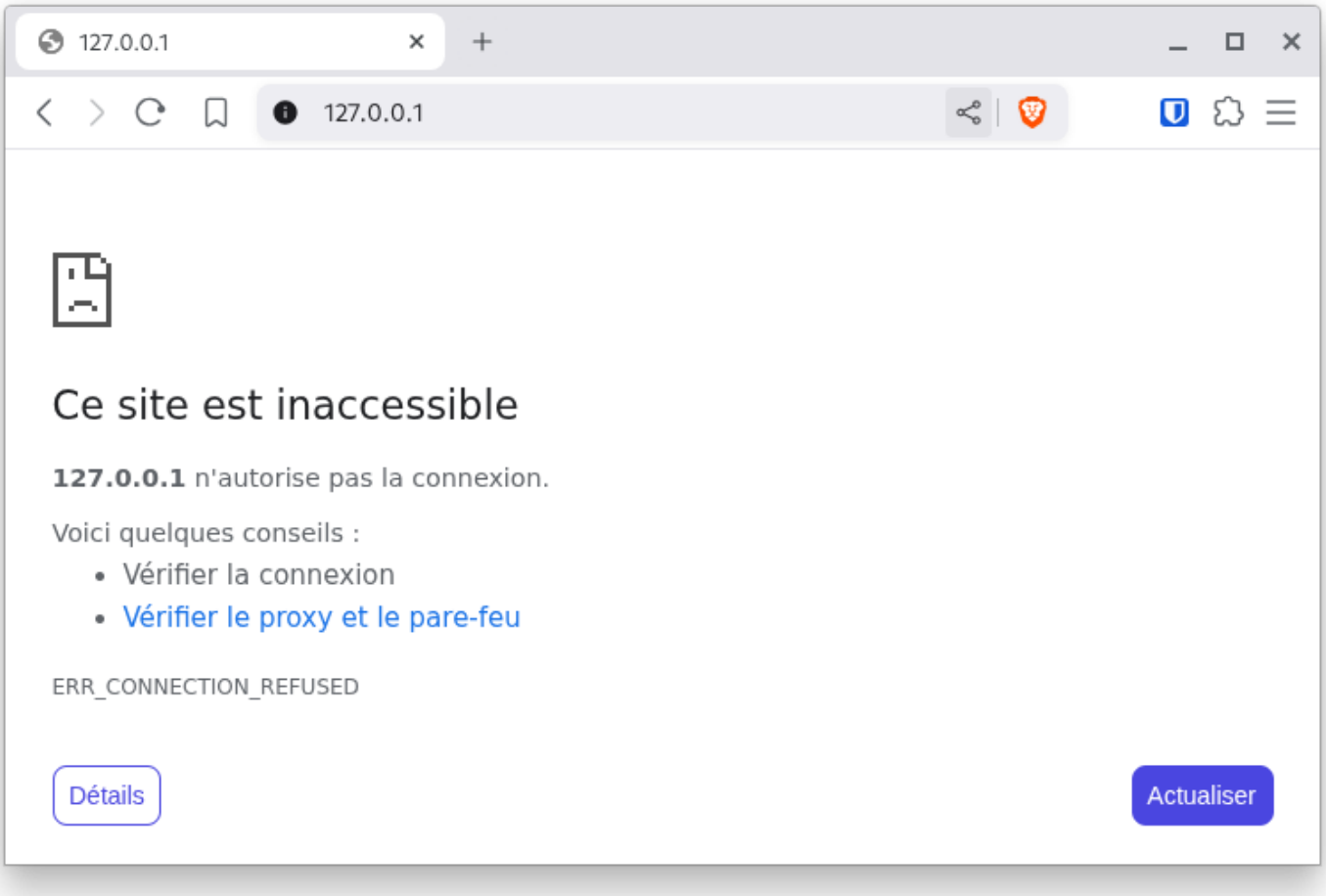
L'image strm/helloworld-http permet d'afficher un message test et écoute sur le port `80`, port `HTTP` par défaut.

Une fois lancé, un `docker ps -a` nous confirme que le port `80` du conteneur est ouvert.

```
debian@debian:~$ docker run -ti --rm --name=test_web_1 strm/helloworld-http
Serving HTTP on 0.0.0.0 port 80 ...
```

```
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE                  COMMAND      CREATED          STATUS          PORTS     NAMES
0047600361d4   strm/helloworld-http   "/main.sh"   33 seconds ago   Up 32 seconds   80/tcp    test_web_1
debian@debian:~$
```

Néanmoins, le port `80` de la machine hôte **ne répond pas** :



> Le port `80` de la machine hôte **n'est pas automatiquement redirigé** vers celui du conteneur.

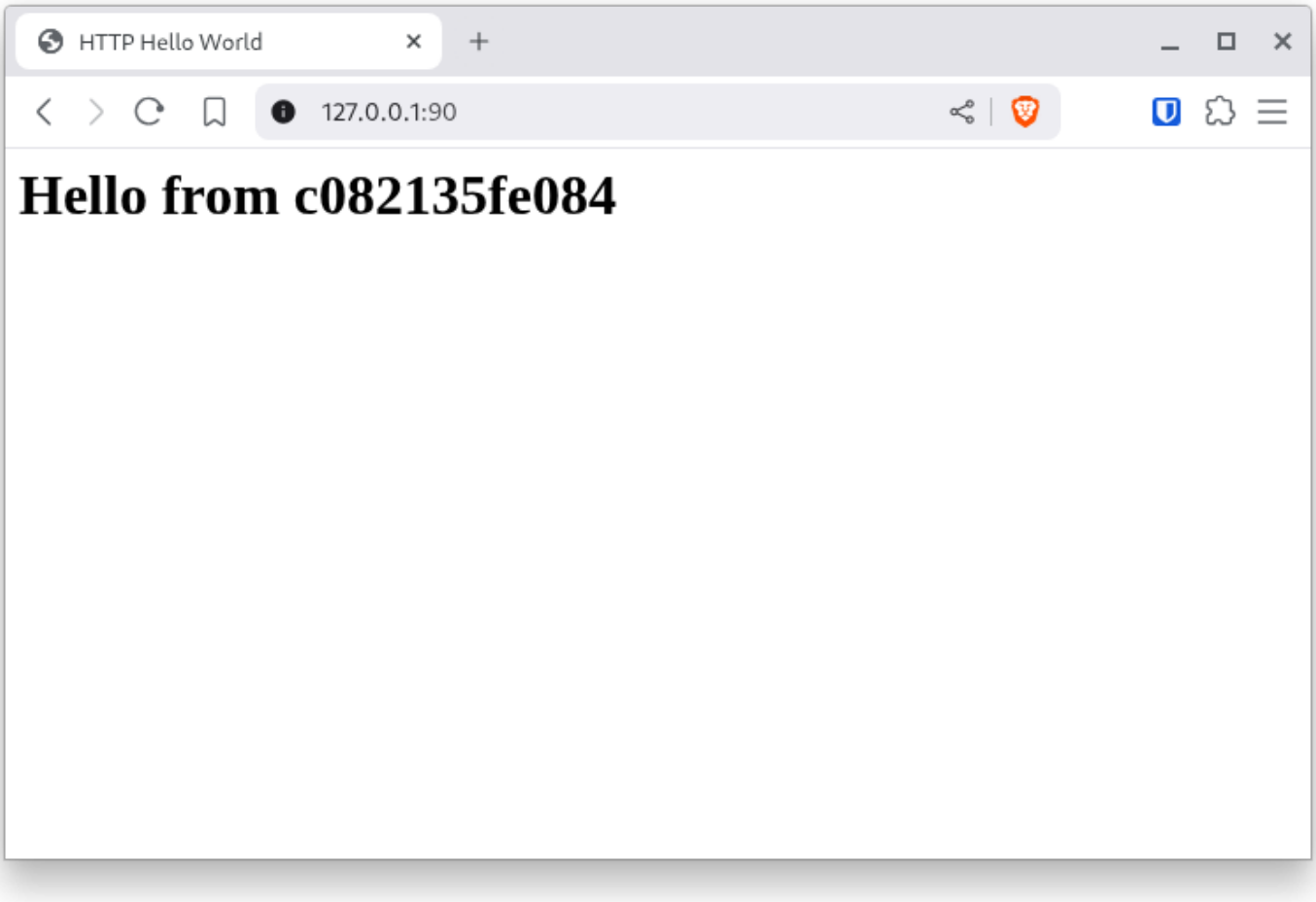
Il faut lancer le container avec l'argument `-p`. Il est tout à fait possible d'indiquer un autre port pour l'hôte, par exemple `90`.

```
debian@debian:~$ docker run -ti --rm -p 90:80 --name=test_web_2 strm/helloworld-http
Serving HTTP on 0.0.0.0 port 80 ...
```

Un `docker ps -a` nous confirme que le port `90` de la machine hôte est redirigé vers le port `80` du conteneur.

```
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE                 COMMAND      CREATED         STATUS         PORTS                                   NAMES
c082135fe084   strm/helloworld-http  "/main.sh"   3 seconds ago   Up 3 seconds   0.0.0.0:90->80/tcp, [::]:90->80/tcp   test_web_2
debian@debian:~$
```

La page web est accessible depuis `127.0.0.1:90`. Elle est également accessible depuis le LAN, toujours sur le port `90`.



Il est possible de n'écouter que depuis l'IP `127.0.0.1` avec `-p 127.0.0.1:90:80`.

```
debian@debian:~$ docker run -d -p 127.0.0.1:90:80 --name=test_web_3 strm/helloworld-http
53ec777e586ab6c00b200a5c64bf7aa69e5facc83402ffdb4ae57117bbb90238
debian@debian:~$ docker ps -a
CONTAINER ID   IMAGE                 COMMAND      CREATED         STATUS         PORTS                     NAMES
53ec777e586a   strm/helloworld-http  "/main.sh"   3 seconds ago   Up 3 seconds   127.0.0.1:90->80/tcp   test_web_3
debian@debian:~$
```

> Cela peut-être utile pour faire fonctionner une application derrière un reverse proxy (traefik, apache, nginx, etc.) afin de lui attribuer un nom de domaine.

TP : Application PHP

Objectif

L'objectif est de lancer un script PHP dans **deux conteneurs avec des versions PHP distinctes**.
Il faudra au préalable lancer un conteneur temporaire afin de **télécharger le script PHP dans un volume managé**.

> Aucune connaissance de PHP n'est requise.

Consignes

1 - Récupération du script PHP dans un volume docker

- Créez un volume docker managé
- Lancez un `bash` dans un conteneur à partir de l'image `debian` et mappez le volume sur `/tp_docker`
- Exécutez les commandes suivantes afin de récupérer le fichier `index.php` depuis le conteneur

```
cd /tp_docker
apt update && apt-get install wget -y
wget https://formation-tfrichet-assets.s3.fr-par.scw.cloud/docker-tp-1/index.php
chmod 777 ./
```

2 - Exécution avec PHP 8.2

- Lancez un conteneur à partir de l'image `php:8.2-apache`.
- Redirigez le port `1082` de l'hôte vers le port `80` du conteneur.
- Mappez votre volume vers `/var/www/html`.
- Lancez le conteneur et ouvrez un navigateur web sur http://127.0.0.1:1082.

  > Remplacez 127.0.0.1 par l'IP de votre hôte docker si nécessaire.
- Un formulaire doit s'afficher, renseignez votre nom et validez.

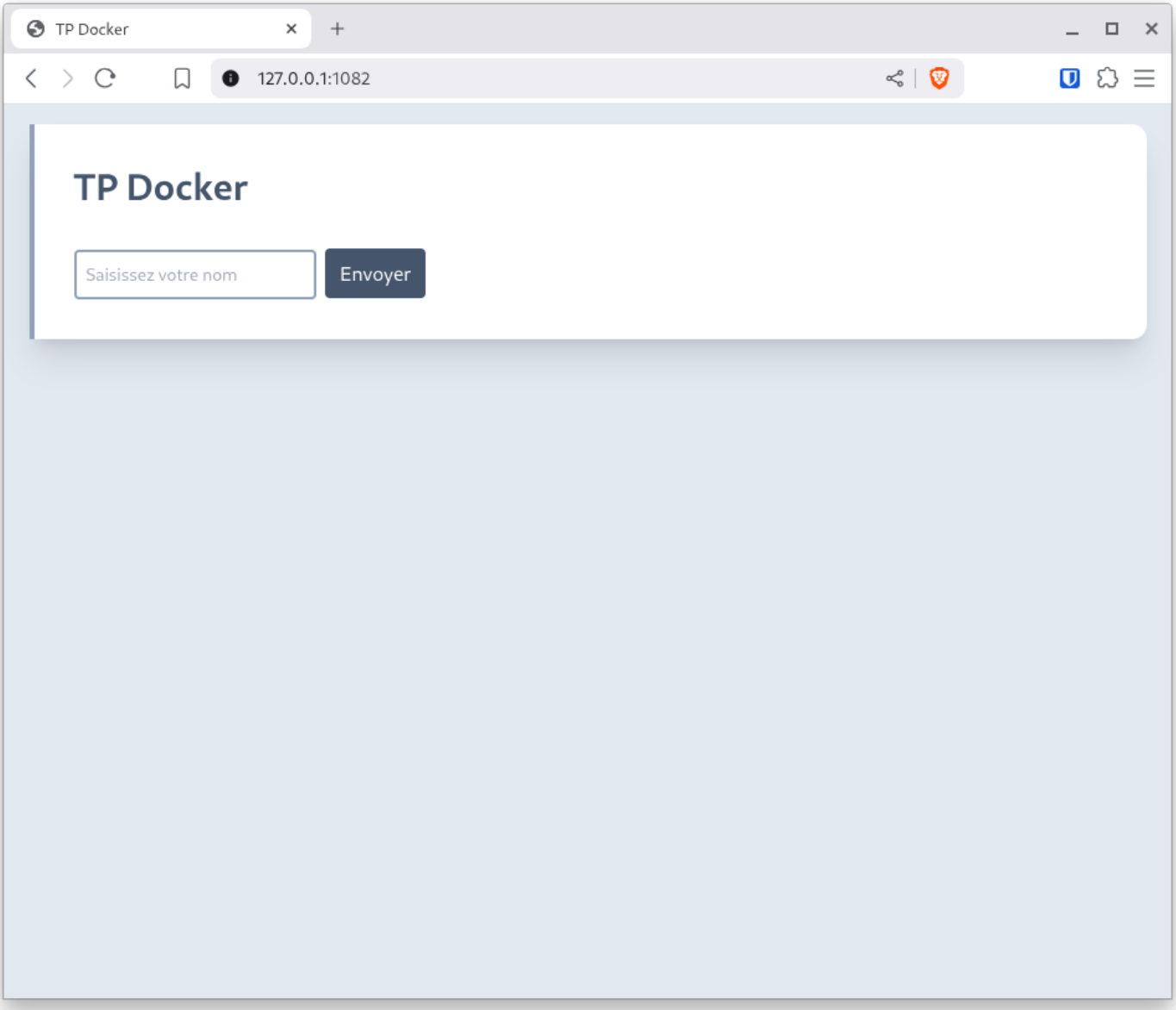
- Des informations telles que la date, un UUID généré et des informations systèmes s'affichent.

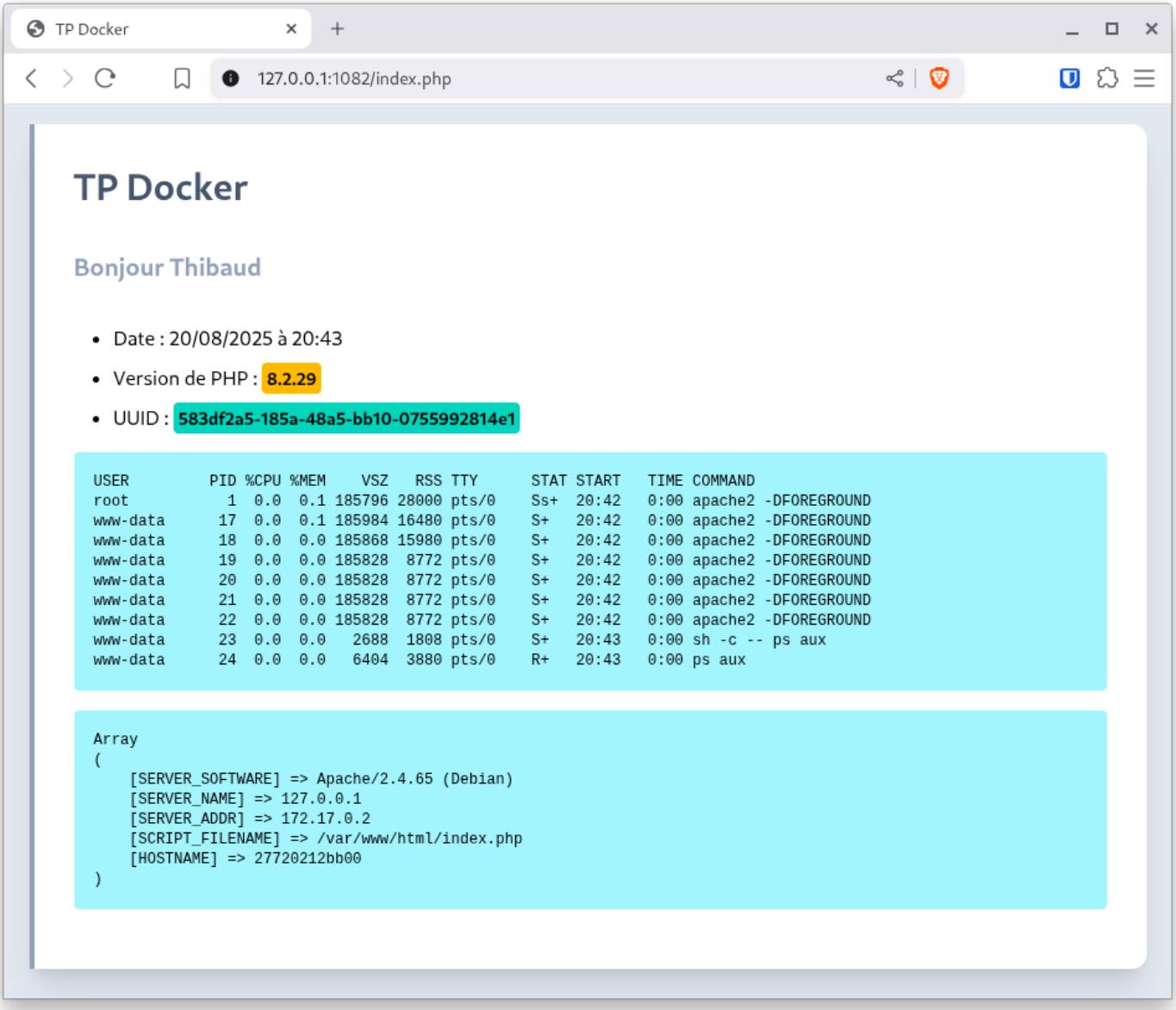
3 - Exécution avec PHP 8.3

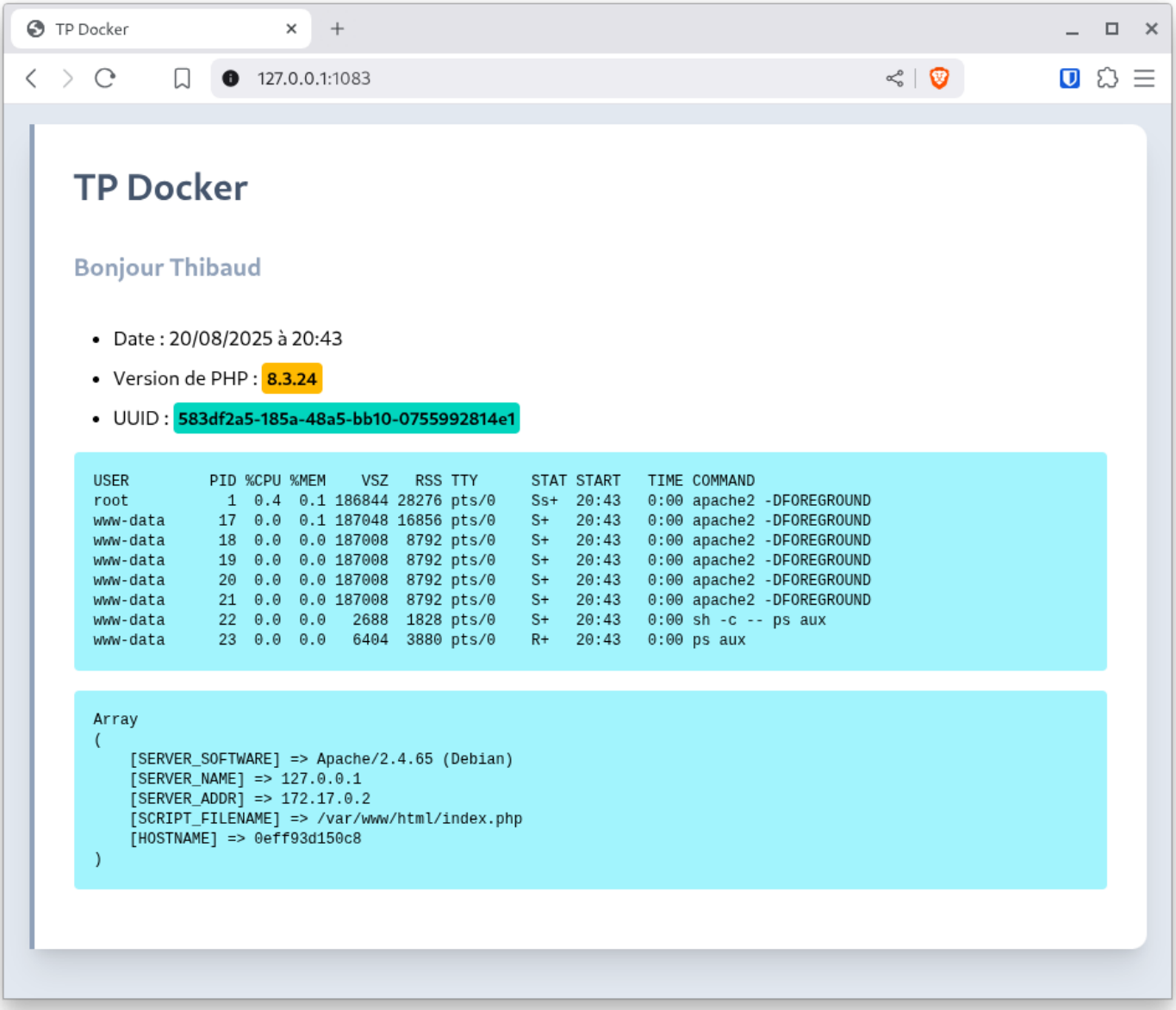
- Lancez un autre conteneur lié au volume managé.
- Utilisez cette fois-ci l'image `php:8.3-apache` et redirigez le port `1083` de l'hôte vers le port `80` du conteneur.
- Rendez-vous sur http://127.0.0.1:1083.
- Votre nom et l'UUID s'affichent, la version de PHP est désormais en 8.3.

Résultat attendu

> Les 3 captures d'écran suivantes sont attendues pour valider le TP.


TP Docker
127.0.0.1:1082
TP Docker
Saisissez votre nom
Envoyer


TP Docker
127.0.0.1:1082/index.php
TP Docker
Bonjour Thibaud
Date : 20/08/2025 à 20:43
Version de PHP : 8.2.29
UUID : 583df2a5-185a-48a5-bb10-0755992814e1
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.1 185796 28000 pts/0 Ss+ 20:42 0:00 apache2 -DFOREGROUND
www-data 17 0.0 0.1 185984 16480 pts/0 S+ 20:42 0:00 apache2 -DFOREGROUND
www-data 18 0.0 0.0 185868 15980 pts/0 S+ 20:42 0:00 apache2 -DFOREGROUND
www-data 19 0.0 0.0 185828 8772 pts/0 S+ 20:42 0:00 apache2 -DFOREGROUND
www-data 20 0.0 0.0 185828 8772 pts/0 S+ 20:42 0:00 apache2 -DFOREGROUND
www-data 21 0.0 0.0 185828 8772 pts/0 S+ 20:42 0:00 apache2 -DFOREGROUND
www-data 22 0.0 0.0 185828 8772 pts/0 S+ 20:42 0:00 apache2 -DFOREGROUND
www-data 23 0.0 0.0 2688 1808 pts/0 S+ 20:43 0:00 sh -c -- ps aux
www-data 24 0.0 0.0 6404 3880 pts/0 R+ 20:43 0:00 ps aux
Array
(
[SERVER_SOFTWARE] => Apache/2.4.65 (Debian)
[SERVER_NAME] => 127.0.0.1
[SERVER_ADDR] => 172.17.0.2
[SCRIPT_FILENAME] => /var/www/html/index.php
[HOSTNAME] => 27720212bb00
)


TP Docker
127.0.0.1:1083
TP Docker
Bonjour Thibaud
Date : 20/08/2025 à 20:43
Version de PHP : 8.3.24
UUID : 583df2a5-185a-48a5-bb10-0755992814e1
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.4 0.1 186844 28276 pts/0 Ss+ 20:43 0:00 apache2 -DFOREGROUND
www-data 17 0.0 0.1 187048 16856 pts/0 S+ 20:43 0:00 apache2 -DFOREGROUND
www-data 18 0.0 0.0 187008 8792 pts/0 S+ 20:43 0:00 apache2 -DFOREGROUND
www-data 19 0.0 0.0 187008 8792 pts/0 S+ 20:43 0:00 apache2 -DFOREGROUND
www-data 20 0.0 0.0 187008 8792 pts/0 S+ 20:43 0:00 apache2 -DFOREGROUND
www-data 21 0.0 0.0 187008 8792 pts/0 S+ 20:43 0:00 apache2 -DFOREGROUND
www-data 22 0.0 0.0 2688 1828 pts/0 S+ 20:43 0:00 sh -c -- ps aux
www-data 23 0.0 0.0 6404 3880 pts/0 R+ 20:43 0:00 ps aux
Array
(
[SERVER_SOFTWARE] => Apache/2.4.65 (Debian)
[SERVER_NAME] => 127.0.0.1
[SERVER_ADDR] => 172.17.0.2
[SCRIPT_FILENAME] => /var/www/html/index.php
[HOSTNAME] => 0eff93d150c8
)