



Inserted_image_20250221181711.webp

Faire persister des données

Prérequis

Docker Desktop lancé, et les commandes des deux chapitres précédents en tête. Travaillez dans un terminal Linux ou macOS, pas dans PowerShell.

Préparez un dossier de travail, on en aura besoin à partir de la Partie 3 :

```
mkdir -p ~/tp-donnees && cd ~/tp-donnees
```

Ce qu'on cherche à obtenir

Le cours affirme qu'un chemin monté n'est plus une couche : ce qui s'écrit là n'entre ni dans la couche inscriptible, ni dans l'image. Il annonce même comment le vérifier — `docker diff` ne doit rien y voir. C'est exactement ce que vous allez faire.

Vous allez constater la perte, puis la réparer de trois façons différentes, et découvrir au passage que deux syntaxes presque identiques produisent des comportements opposés.

Toujours `--help` d'abord

Une famille de commandes entière apparaît dans ce TP :

```
docker volume --help
```

Partie 1 — Constater la perte

Avant de réparer, il faut voir le problème. On va écrire quelque chose dans un conteneur, puis le détruire.

Lancez un conteneur qui reste en vie, et écrivez-y un fichier :

```
docker run -dit --name sans-volume alpine sh
docker exec sans-volume sh -c 'mkdir -p /data && echo "relevé n°1" >
/data/mesures.txt'
docker exec sans-volume cat /data/mesures.txt
```

Regardez où ce fichier a atterri :

```
docker diff sans-volume
```

- Retrouvez-vous `/data/mesures.txt` dans la liste ? Avec quelle lettre ?
- D'après le chapitre précédent, dans quelle couche se trouve-t-il ?

Détruisez le conteneur et recréez-le à l'identique :

```
docker rm -f sans-volume
docker run -dit --name sans-volume alpine sh
docker exec sans-volume cat /data/mesures.txt
```

- Que répond la dernière commande ?
- Le conteneur est pourtant « le même ». D'où repart-il, exactement ?

Nettoyez :

```
docker rm -f sans-volume
```

Validation — vous avez vu le fichier apparaître dans `docker diff`, puis disparaître avec le conteneur.

Partie 2 — Un volume nommé

On refait exactement la même chose, en branchant un stockage extérieur sur `/data`.

Créez le volume, puis regardez ce que Docker en dit :

```
docker volume create donnees
docker volume ls
docker volume inspect donnees
```

- Que contient le champ `Mountpoint` ?
- Ce chemin existe-t-il sur votre machine ? Essayez `ls` dessus.

Si `ls` échoue, c'est normal

Sur macOS et Windows, ce chemin n'existe pas sur votre machine : il est **dans la VM Linux** qui fait tourner Docker. C'est précisément l'intérêt d'un volume nommé — vous n'avez pas à savoir où il est rangé, ni à pouvoir y accéder.

Lancez maintenant un conteneur qui monte ce volume. Le cours en a donné deux écritures : on commence par la plus explicite, `--mount`, où chaque élément porte son nom.

```
docker run --help | grep -i mount
```

Elle attend une suite de clés séparées par des virgules. Trois suffisent ici : le **type** de montage, la **source** et la **destination** — reprenez la forme vue en cours.

Lancez un conteneur Alpine qui reste en vie, nommé `avec-volume`, avec le volume `donnees` monté sur `/data`. Puis écrivez-y le même fichier qu'en Partie 1 :

```
docker exec avec-volume sh -c 'echo "relevé n°1" > /data/mesures.txt'
docker exec avec-volume cat /data/mesures.txt
```

La vérification qui compte

```
docker diff avec-volume
```

- Le fichier `/data/mesures.txt` apparaît-il cette fois ?
- Comparez mot pour mot avec la sortie de la Partie 1. Qu'est-ce qui a changé ?

C'est toute la démonstration

`docker diff` liste ce qui a changé dans la **couche inscriptible** du conteneur. Votre fichier n'y figure pas : il n'y est jamais passé. Le montage l'a envoyé directement dans le volume, à côté de l'empilement des couches — exactement ce que montre le schéma du cours.

Le fichier survit-il ?

```
docker rm -f avec-volume
docker volume ls
```

- Le conteneur a disparu. Et le volume ?

Relancez un conteneur neuf, avec le même volume monté au même endroit, puis lisez le fichier.

- Est-il toujours là ? Avec son contenu ?

Deux conteneurs, un volume

Lancez un **second** conteneur montant le même volume, nommé `lecteur`, puis faites écrire l'un et lire l'autre :

```
docker exec avec-volume sh -c 'echo "relevé n°2" >> /data/mesures.txt'
docker exec lecteur cat /data/mesures.txt
```

- Le second conteneur voit-il ce que le premier vient d'écrire ?
- Rappelez-vous la Partie 1 du TP précédent : deux conteneurs d'une même image partageaient-ils quoi que ce soit ?

Nettoyez les deux conteneurs, mais **gardez le volume** :

```
docker rm -f avec-volume lecteur
```

Validation — `docker diff` n'a jamais montré le fichier, le volume existe toujours dans `docker volume ls`, et son contenu a survécu à deux destructions de conteneur.

Partie 3 — Un bind mount

Cette fois, c'est vous qui choisissez l'emplacement : un dossier de votre machine.

Préparez le dossier et un fichier dedans :

```
cd ~/tp-donnees
mkdir -p site
echo "<h1>Première version</h1>" > site/index.html
```

Lancez Nginx en montant votre dossier `site` à la place de sa racine web : `/usr/share/nginx/html`. Cette fois c'est la forme courte, `-v` — la seconde écriture du cours, celle que vous croiserez partout dans la documentation et les exemples en ligne :

```
docker run -d --name web -p 8080:80 -v ??? : ??? nginx
```

- Retrouvez-vous, dans cette seule chaîne, les trois éléments que `--mount` vous faisait nommer en Partie 2 ?

Ouvrez <http://localhost:8080>.

- Voyez-vous votre page, ou celle de Nginx ?

Maintenant, modifiez le fichier **depuis votre machine**, sans toucher au conteneur :

```
echo "<h1>Deuxième version</h1>" > site/index.html
```

Rechargez la page.

- Avez-vous eu besoin de redémarrer le conteneur ? De reconstruire quoi que ce soit ?

Essayez dans l'autre sens, en écrivant depuis le conteneur :

```
docker exec web sh -c 'echo "<p>écrit depuis le conteneur</p>" >>
/usr/share/nginx/html/index.html'
cat site/index.html
```

- Le fichier de votre machine a-t-il changé ?

C'est le même répertoire des deux côtés

Il n'y a pas de copie, pas de synchronisation : les deux chemins désignent les mêmes fichiers. C'est ce qui fait du bind mount l'outil du développement — on édite avec son éditeur habituel, le conteneur voit les modifications immédiatement.

Ce qu'un bind mount recouvre

Le dossier `/usr/share/nginx/html` n'était pas vide dans l'image : il contenait la page d'accueil de Nginx. Où est-elle passée ?

```
docker exec web ls /usr/share/nginx/html
```

- Que contient ce dossier ? Retrouvez-vous `50x.html`, livré avec l'image ?
- Le fichier a-t-il été supprimé de l'image, à votre avis ?

Vérifiez-le sur un conteneur sans montage :

```
docker run --rm nginx ls /usr/share/nginx/html
```

- Conclusion ?

Nettoyez :

```
docker rm -f web
```

Validation — votre page s'affiche, vos modifications depuis l'hôte apparaissent sans redémarrage, et vous savez dire où est passée la page d'accueil de Nginx.

Partie 4 — La différence qui surprend

Même conteneur, même chemin de destination, une seule chose change : un volume nommé à la place du dossier de votre machine.

```
docker run -d --name web2 -p 8080:80 -v site-web:/usr/share/nginx/html nginx
```

Ouvrez <http://localhost:8080>.

- Que voyez-vous ? Comparez avec ce qu'aurait donné un dossier vide monté au même endroit.

Regardez ce que contient le volume :

```
docker exec web2 ls /usr/share/nginx/html
```

- D'où sortent ces fichiers ? Vous ne les y avez pas mis.

Le pré-remplissage

Quand un volume nommé **vide** est monté sur un chemin qui contient déjà quelque chose dans l'image, Docker y recopie ce contenu à la première utilisation. Un bind mount ne fait jamais ça : il recouvre, point.

Deux syntaxes presque identiques, deux comportements opposés. C'est une des confusions les plus fréquentes avec Docker.

Vérifiez que la recopie n'a lieu qu'une fois. Modifiez la page, détruisez le conteneur, recréez-le avec le même volume :

```
docker exec web2 sh -c 'echo "<h1>Ma page</h1>" > /usr/share/nginx/html/index.html'
docker rm -f web2
docker run -d --name web2 -p 8080:80 -v site-web:/usr/share/nginx/html nginx
```

Rechargez la page.

- Votre version a-t-elle survécu, ou Docker a-t-il recopié la page d'origine par-dessus ?
- Qu'en déduisez-vous sur la condition du pré-remplissage ?

Nettoyez le conteneur :

```
docker rm -f web2
```

Validation — vous savez expliquer pourquoi un volume nommé affiche la page de Nginx là où un dossier vide ne montrerait rien.

Partie 5 — tmpfs, ce qui ne doit rien laisser

Un troisième mécanisme, pour les données qui ne doivent surtout pas persister.

Cherchez l'option dans l'aide :

```
docker run --help | grep -i tmpfs
```

Lancez un conteneur Alpine avec un `tmpfs` monté sur `/scratch`, nommé `ephemere`, puis écrivez-y un fichier :

```
docker exec ephemere sh -c 'echo "secret déchiffré" > /scratch/cle.txt'
docker exec ephemere cat /scratch/cle.txt
```

Vérifiez la nature de ce montage, puis cherchez le fichier dans la couche inscriptible :

```
docker exec ephemere df -h /scratch
docker diff ephemere
```

- Quel type de système de fichiers `df` annonce-t-il pour `/scratch` ?
- Le fichier apparaît-il dans `docker diff` ? Comme pour le volume, ou différemment ?

Redémarrez le conteneur, sans le détruire :

```
docker restart ephemere
docker exec ephemere cat /scratch/cle.txt
```

- Le fichier a-t-il survécu ? Rappelez-vous où il était stocké.

Nettoyez :

```
docker rm -f ephemere
```

Validation — vous savez dire ce qui distingue un `tmpfs` d'un volume : les deux échappent à la couche inscriptible, un seul survit à un redémarrage.

Partie 6 — Sauvegarder et restaurer un volume

Votre volume `donnees` contient toujours les relevés de la Partie 2. On va les archiver, tout détruire, puis tout remettre.

Un volume n'est pas un fichier : on ne peut pas le copier directement. Le motif consiste à lancer un conteneur jetable qui monte **deux** choses : le volume d'un côté, un dossier de votre machine de l'autre — puis à archiver l'un vers l'autre.

```
cd ~/tp-donnees
docker run --rm -v donnees:/data -v "$PWD":/save \
  alpine tar -czf /save/donnees.tgz -C /data .
ls -lh donnees.tgz
```

- L'archive existe-t-elle sur votre machine ?
- Le conteneur qui l'a créée existe-t-il encore ? Pourquoi ?

Détruisez le volume et vérifiez qu'il a bien disparu :

```
docker volume rm donnees
docker volume ls
```

Restaurez-le. C'est le même motif, dans l'autre sens — à vous d'adapter la commande d'archivage en commande d'extraction :

```
docker volume create donnees
```

`tar -czf` crée une archive. Pour l'extraire, remplacez `-c` par `-x`. Le reste de la commande ne change pas, y compris le `-C /data` qui dit où travailler.

Vérifiez la restauration :

```
docker run --rm -v donnees:/data alpine cat /data/mesures.txt
```

- Retrouvez-vous les deux relevés de la Partie 2 ?

Validation — `docker volume ls` affiche à nouveau `donnees`, et son contenu est identique à celui d'avant la suppression.

Partie 7 — Ranger derrière soi

Ce TP a laissé des conteneurs, deux volumes, un dossier et une archive.

Faites le ménage vous-même. Vous avez toutes les commandes : celles des chapitres précédents pour les conteneurs et les images, celles de ce TP pour les volumes.

- Supprimez les conteneurs encore présents.
- Supprimez les volumes `donnees` et `site-web`.
- Supprimez le dossier `~/tp-donnees` et son contenu.
- **Gardez** les images : elles resserviront au chapitre suivant.

⚠ Un volume supprimé ne revient pas

Il n'y a ni corbeille, ni confirmation différée. C'est la raison d'être de la Partie 6 : avant de supprimer un volume de production, on l'archive.

Validation — `docker volume ls` ne contient plus aucun volume de ce TP, et `docker ps -a` plus aucun conteneur.

Mémo des commandes

- **Volumes** : `docker volume create`, `ls`, `inspect`, `rm`, `prune`
- **Monter, forme explicite** : `--mount type=volume,src=nom,dst=/chemin`
- **Monter, forme courte** : `-v nom:/chemin` (volume), `-v /hote:/chemin` (bind mount), `--tmpfs /chemin`
- **Vérifier** : `docker diff`, `docker exec ... ls`, `docker volume inspect`
- **Sauvegarder** : un conteneur jetable montant le volume et un dossier de l'hôte

Pour finir

Vous avez vérifié le point central du chapitre avec un outil du chapitre précédent : ce qui est écrit dans un montage n'apparaît pas dans `docker diff`, parce qu'il n'est jamais passé par la couche inscriptible.

Vous avez aussi rencontré la confusion la plus coûteuse de Docker : sur un chemin déjà peuplé, un volume nommé se pré-remplit et un bind mount recouvre. La syntaxe est presque la même, le résultat est

l'inverse.