

Données et persistance





Le problème

Gestion des données ?

Rappel : la couche inscriptible

- Tout ce qu'un conteneur écrit y atterrit
- Elle naît avec lui, et meurt avec lui
- Vous l'avez vue avec `docker diff`

“ Elle n'a jamais été conçue pour conserver quoi que ce soit. ”

Ce que ça implique

- `docker rm` emporte les données avec le conteneur
- Un conteneur recréé repart de l'image, donc de zéro
- Deux conteneurs de la même image ne partagent rien

Des questions de performance

- Chaque écriture passe par le copy-on-write : c'est plus lent
- Difficile de sauvegarder : les données sont enfouies dans le conteneur
- Impossible de partager : chaque conteneur a sa propre couche

Le besoin

- Des données qui **survivent** au conteneur
- Des données **partageables** entre plusieurs conteneurs
- Des données **sauvegardables** sans toucher au conteneur

Sortir les données du conteneur



Le principe du montage

- Docker branche un stockage extérieur sur un chemin du conteneur
- Pour le conteneur, c'est un répertoire comme un autre
- Pour Docker, c'est un objet à part, avec son cycle de vie

Ce chemin n'est plus une couche

- Ce qui s'écrit là n'entre ni dans la couche inscriptible, ni dans l'image
- `docker diff` ne le voit pas
- `docker commit` ne l'emporte pas

Monter, plutôt qu'écrire

Le conteneur ne voit qu'une seule arborescence

/etc

/usr

/app

/var/lib/data

empilement des couches

point de montage

Couche inscriptible du conteneur
éphémère

Couches de l'image — lecture seule

meurt avec le conteneur

Volume

géré par Docker
hors du conteneur

survit au conteneur

Trois conséquences

- Les données survivent à la destruction du conteneur
- Plusieurs conteneurs peuvent monter le même stockage
- Les écritures ne passent plus par le copy-on-write

Trois types de montage



Volume nommé

- Vous donnez un **nom**, Docker choisit où le ranger
- Il vit dans un espace géré par le démon
- Il existe indépendamment de tout conteneur

Bind mount

- Vous donnez un **chemin de votre machine**
- Docker le branche tel quel dans le conteneur
- Vous voyez des deux côtés les mêmes fichiers, en direct

tmpfs

- Un espace en **mémoire vive**, jamais écrit sur disque
- Donc bien plus rapide que n'importe quel disque
- Il disparaît à l'arrêt du conteneur

```
--tmpfs /tmp
```

Volume nommé ou bind mount

	Volume nommé	Bind mount
L'emplacement	Docker décide	Vous décidez
Portable	Oui	Non

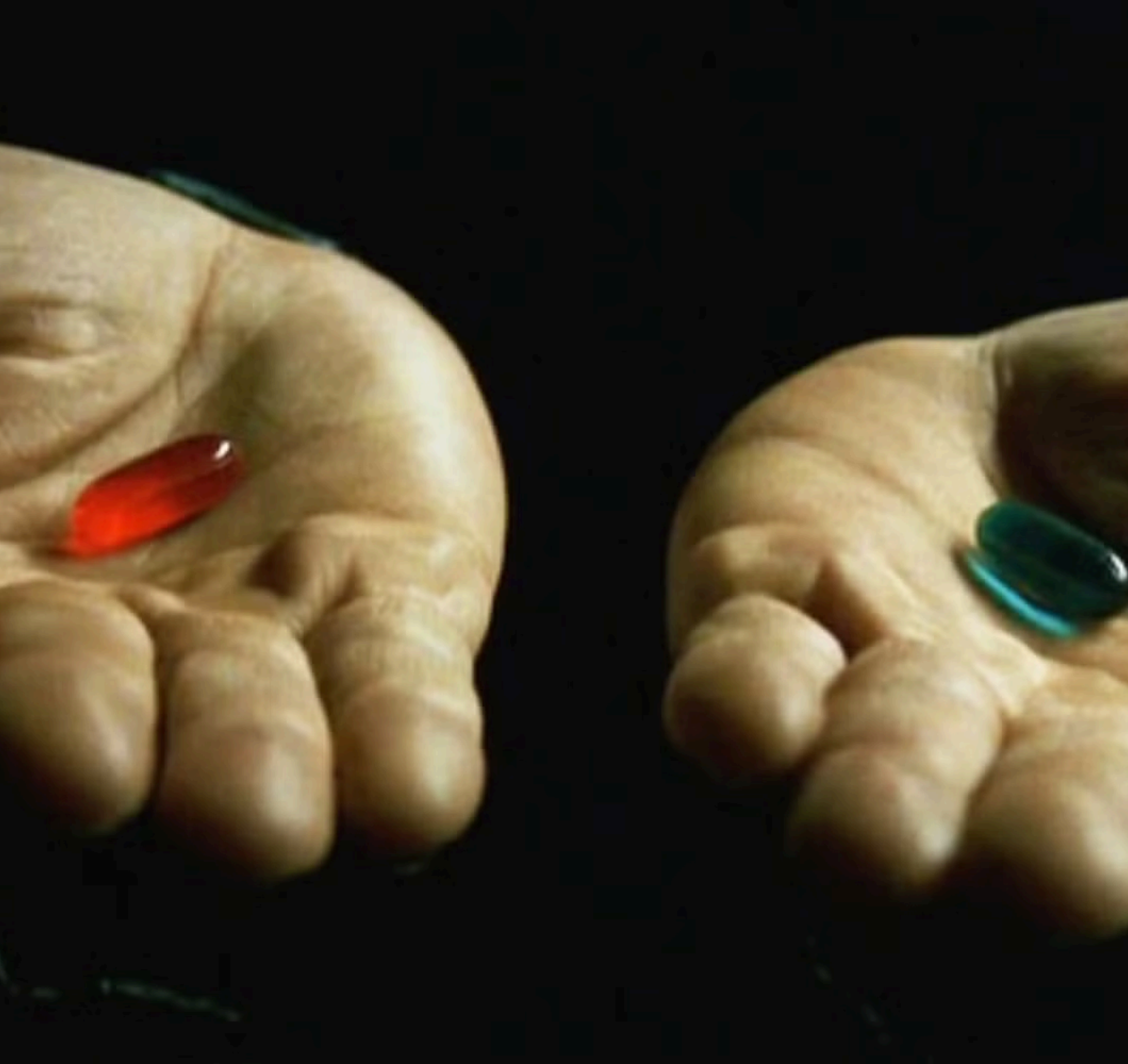
Deux écritures pour la même chose

```
# la forme courte  
-v donnees:/data
```

```
# la forme explicite  
--mount type=volume,src=donnees,dst=/data
```

Laquelle choisir

- `-v` : court, historique, tout dans une chaîne
- `--mount` : verbeux, explicite, une clé par option
- `--mount` : refuse ce qu'il ne comprend pas, `-v` devine (exemple : si le chemin destination n'existe pas)



Choisir

Volume nommé : le cas général

- Données d'application, bases de données
- Tout ce qui doit durer et qu'on ne lit jamais à la main
- Le seul choix portable d'une machine à l'autre

Bind mount : le développement

- Votre code source monté dans le conteneur
- Un fichier de configuration que vous éditez
- Un dossier de sortie que vous voulez consulter

“ On veut précisément que ce soit **votre** dossier, pas celui de Docker. ”

tmpfs : le sensible et le rapide

- Secrets déchiffrés le temps d'une exécution : rien n'atteint le disque
- Caches et fichiers temporaires : on y gagne en vitesse
- Tout ce qui ne doit laisser aucune trace

“ La contrepartie : ça consomme de la mémoire vive, et elle n'est pas extensible. ”



Le cycle de vie d'un volume

Les commandes

- `docker volume create` — le créer à l'avance
- `docker volume ls` — les lister
- `docker volume inspect` — savoir où il est rangé
- `docker volume rm` — le supprimer

Un volume survit à tout

- Il survit à l'arrêt du conteneur
- Il survit à la suppression du conteneur
- Il survit à la suppression de l'image

Les volumes anonymes

- Créés sans qu'on les demande, quand une image en déclare un
- Un nom illisible, aucune trace de leur origine
- Ils s'accumulent en silence

“ `docker rm -v` emporte les anonymes du conteneur, jamais les volumes nommés. ”

Sauvegarder un volume

- Un volume n'est pas un fichier : on ne le copie pas directement
- Le motif : un conteneur jetable qui monte le volume **et** un dossier de l'hôte

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

Restaurer

- Exactement le même motif, dans l'autre sens
- Le conteneur jetable désarchive vers le volume

```
docker run --rm \
  -v donnees:/data -v "$PWD":/save \
  alpine tar -xzf /save/data.tgz -C /data
```

Particularités

Un bind mount masque ce qu'il recouvre

- Le contenu de l'image à ce chemin devient invisible
- Il n'est pas effacé : il est caché, comme sous un whiteout
- Monter un dossier vide sur `/app` vide `/app` , du point de vue du conteneur

Un volume nommé vide se pré-remplit

- À sa première utilisation, Docker y recopie le contenu de l'image
- Les fois suivantes, non : le volume a désormais son propre contenu
- Un bind mount, lui, ne se pré-remplit **jamais**

“ Deux comportements opposés pour une syntaxe presque identique. ”

Sur Mac et Windows

- Le conteneur tourne dans une VM Linux, votre dossier est de l'autre côté
- Chaque accès à un bind mount traverse cette frontière
- Sur beaucoup de petits fichiers, c'est spectaculairement lent

“ Un volume nommé vit **dans** la VM. Il n'a pas ce problème. ”

En une phrase

“ Tout ce qu'un conteneur écrit meurt avec lui, sauf aux chemins où l'on a monté autre chose — et ce montage n'appartient ni à l'image, ni au conteneur. ”

À retenir (1/2)

- La couche inscriptible n'est pas un lieu de stockage
- Un montage perce l'empilement des couches, il n'en fait pas partie
- Volume nommé par défaut, bind mount quand l'emplacement compte

À retenir (2/2)

- Un volume survit à tout, sauf à sa suppression explicite
- Un volume vide se pré-remplit depuis l'image, un bind mount jamais
- Sur Mac et Windows, un bind mount traverse une VM : c'est lent

Données et persistance

