Virtualisation des systèmes

Florent Glück

April 14, 2025

Docker storage

(1) Overlayfs

On désire comprendre plus en profondeur comment les couches d'un container fonctionnent. Pour cela, il nest nécessaire de comprendre le système de fichiers overlayfs du noyau Linux à l'aide d'une investigation empirique.

Le but ici est de créer un rootfs dont le contenu sera utilisé comme couche basse en lecture seule pour overlayfs, en combinaison avec une couche haute initialement vide. Pour simplifier, vous utiliserez donc une seule couche basse, mais il serait possible d'en avoir un nombre arbitraire.

Commencez par créer l'arborescence nécessaire à overlayfs, c'est à dire :

- Créez un premier répertoire vide nommé **lower** (nom arbitraire) ; celui-ci cera le répertoire racine de la couche basse utilisé par overlayfs
- Créez un deuxième répertoire vide nommé **upper** (nom arbitraire) ; celui-ci cera le répertoire racine de la couche haute utilisé par overlayfs
- Créez un troisième répertoire vide nommé **merged** (nom arbitraire) ; grâce à overlayfs, celui-ci cera le répertoire présentant la vue unifiée des couches basses et de la couche haute
- Créez un quatrième répertoire vide nommé **work** (nom arbitraire) ; celui-ci ne nous concerne pas ; il sera utilisé par le noyau comme répertoire de travail

La couche basse sera populée par un rootfs volontairement simple. Pour cela, vous allez utiliser le projet Busybox qui est beaucoup utilisé dans le domaine de l'embarqué. Celui-ci permet de générer un rootfs de très petite taille avec les outils de base nécessaires à un système Linux (shell, ls, cp, mv, rm, grep, top, etc.).

Le projet Busybox est disponible ici https://www.busybox.net/. Voici comment configurer, compiler et déployer Busybox :

1) Téléchargez l'archive du code source de la dernière version de Busybox
2) Décompressez l'archive
3) À la racine du projet, exécutez `make menuconfig`
4) Sélectionnez avec la touche "Enter" l'entrée `Settings → Destination path for 'make install' (NEW)`
   - spécifiez le répertoire où installer Busybox sur votre système ; celui-ci doit être le répertoire de la couche basse de votre overlayfs
   - au moment de l'installation de Busybox, l'arborescence du rootfs sera créée dans ce répertoire
5) Sortez du menu en veillant à sauvergarder la configuration
6) Exécutez `make` pour compiler les programmes du projet Busybox

7) Exécutez `make install` pour créeer l'arborescence du rootfs et installer les programmes compilés dans le répertoire de destination choisi dans la configuration

Si vous obtenez une erreur similaire à celle-ci lors de la compilation :

```
fatal error: curses.h: No such file or directory
```

Cela signifie qu'il vous manque les fichiers de développement pour la librairie ncurses. En effet, Busybox utilise la librairie ncurses pour l'affichage des menus. Installez alors le package `libncurses-dev` pour résoudre ce problème (le `dev` dans le nom du package signifie *development*).

Maintenant que vous avez la couche basse prête avec le rootfs et la couche haute vide et prête, vous pouvez, similairement à ce ce que réalise Docker, utiliser overlayfs pour présenter la couche **container** (*writable*) accessible en lecture/écriture (**read/write**) dans le répertoire `merged` créé précédemment.

Pour cela, réalisez un **mount** en précisant le système de fichiers overlayfs selon la syntaxe présentée dans les slides de cours.

Vérifiez que tout est monté correctement en inspectant la sortie de la commande `mount`, en particulier que la couche **merged** utilise bien le système de fichiers `overlay`.

Ajoutez/modifiez/supprimez des fichiers dans le répertoire `merged` et observez ce qu'il se passe dans les répertoires :

- `lower` (qui correspond à la couche basse)
- `upper` (qui correspond à la couche haute)
- `merged` (qui correspond à la vue unifiée)
- Quelle est l'analogie entre le scénario que vous avez réalisé ici et les containers Docker (image et couche **container**) ?
- Que se passe-t-il réellement lorsqu'un fichier de la vue unifiée (`merged`) est **ajouté** ?
- Que se passe-t-il réellement lorsqu'un fichier de la vue unifiée (`merged`) et existant dans la couche basse, est **modifié** ?
- Que se passe-t-il réellement lorsqu'un fichier de la vue unifiée (`merged`) et existant dans la couche basse, est **supprimé** ? (aide: affichez ce fichier avec `ls -l`, inspectez ces attributs et investiguez la commande `mknod`)

(2) Inspection de couches

Dans cet exercice, on s'intéresse à mieux comprendre comment les couches d'une image/ container sont gérées par Docker.

Démarrez un container `Fedora:43`, puis dans celui-ci :

- ajoutez un fichier à la racine
- supprimez un fichier, par exemple `/etc/issue`
- modifiez un fichier, par exemple en ajoutant une ligne à `/etc/passwd`

Utilisez `docker inspect` pour inspecter le contenu du container en exécution ci-dessus.

- Combien de couches basses (**lower layers**) contient ce container et quels sont leurs chemins absolus dans le système de fichiers sur la machine hôte ?
- Quel est le chemin absolu de la couche **container writable** ?
- En inspectant le système de fichiers sur la machine hôte, listez le contenu complet de la couche **container writable** (info: `tree` est très pratique pour visualiser les arborescences)
- Sur la machine hôte, quel est le chemin absolu du répertoire correspondant au **rootfs** du container ?

Sur la machine hôte, déplacez-vous dans le répertoire correspondant au **rootfs** du container. A l'intérieur de celui-ci, créez le fichier `Hello_from_the_outside`.

- Suite à l'opération ci-dessus, qu'observez-vous dans le container lorsque vous listez le contenu du répertoire racine ?

Précédemment, vous avez utilisé la commande docker `inspect` pour déterminer les différentes couches (overlayfs) du container.

- Est-ce qu'il est possible de simplement utiliser la commande `mount` sur le serveur pour obtenir les mêmes informations ? Justifiez votre démarche.

(3) Commit

Dans cet exercice, on désire modifier un container et rendre ces modifications persistantes, en les *commit* dans une nouvelle image.

Déterminez l'image Debian la plus récente (en terme de numéro de version), puis téléchargez là.

- Quelle est la version la plus récente que vous avez trouvée ?

Démarrez alors un container nommé `mydebian` instancié depuis l'image que vous venez de télécharger.

Dans ce container :

- Supprimez les répertoires `/opt` et `/mnt` et la commande `/usr/bin/uniq`
- Avec `dd`, créez le fichier `/home/random` de 8500 bytes à partir de `/dev/urandom`
- Ajouter la ligne ci-dessous à la fin du fichier `/etc/motd`:

```
Tagada tsoin tsoin
```

Une fois ces opérations réalisées, sortez du container.

A l'aide de `docker diff`, affichez les différences entre votre container `mydebian` et l'image utilisée pour son instantation.

- Quelle est la signification de la lettre située dans la première colonne des résultats de `docker diff` ?

A l'aide de la commande `docker commit`, committez alors ces changement en créeant la nouvelle image `debian:new`. Détruisez alors le container debian précédent étant donné qu'il n'est plus utile, puis inspectez l'historique de l'image `debian:new` que vous venez de créer.

- Combien de couches ont été ajoutées à l'image suite à vos modifications apportées au container ci-dessus ?
- Quelle est la taille de ces/cette nouvelle(s) couche(s) ?

Enfin, instanciez un nouveau container à partir de votre nouvelle image `debian:new` et vérifiez que ce nouveau container contient bien les changements effectués précédemment.

(4) Volume et bind mount

On désire enfin se familiariser avec l'utilisation de volumes et bind mounts.

Sur votre machine client, créez le volume `pics`, puis montez le dans le répertoire `/vol/pics` d'un container `ubuntu:24.04` créé pour l'occasion.

- Dans le container, quel est le contenu du volume `pics` ?

Depuis la machine cliente, copiez un répertoire contenant une série d'images dans le volume `pics` et vérifiez que vous pouvez accéder à ces images dans votre container.

Réalisez maintenant l'opération inverse : copiez le contenu du volume `pics` dans le répertoire courant de votre machine client.

- Est-ce que les opérations ci-dessus sur le volume `pics` fonctionnent correctement si le service `dockerd` s'exécute sur une machine différente du client `docker` ? Justifiez.

Réalisez maintenant les mêmes opérations que ci-dessus, mais plutôt que d'utiliser un volume, utilisez un **bind mount**.

- Est-ce que dans le cas d'un **bind mount**, les opérations ci-dessus fonctionnent correctement si le service `dockerd` s'exécute sur une machine différente du client `docker` ? Justifiez.