Virtualisation

Professeur : Florent Gluck Assistant : Sebastien Chassot

April 13, 2022

Docker storage

Exercice 1

On désire comprendre comment fonctionne le mécanisme d'overlay du noyau Linux à l'aide d'un scénario empirique. Vous allez créer une archive dont le contenu sera utilisé comme couche basse en lecture seule pour overlayfs. Ensuite, vous utiliserez un répertoire dédié comme couche haute en lecture/écriture.

Commencez par créer les répertoires nécessaires, c'est à dire :

- Un répertoire utilisé comme couche basse (*lower*)
- Un répertoire utilisé comme couche haute (*upper*)
- Un répertoire présentant la vue unifiée (*merged*)
- Un répertoire utilisé comme couche de travail (*workdir*)

Pour la couche basse *read-only*, populez dans celle-ci une arborescence de base contenant l'outil Busybox disponible ici https://www.busybox.net/. Voici comment compiler et déployer Busybox :

1. Décompressez l'archive du projet Busybox
2. A la racine du projet, exécutez `make menuconfig`
3. Sélectionnez avec la touche "Enter" l'entrée `Settings` → `Destination path for 'make install' (NEW)`
   - Spécifiez le répertoire où installer Busybox sur votre système ; à vous de choisir un répertoire accessible par votre utilisateur (non root).
4. Sortez du menu en veillant à sauvergarder la configuration et exécutez `make` pour compiler Busybox
5. Exécutez `make install` pour installer Busybox dans le répertoire de destination choisi dans la configuration

Téléchargez donc le code source de la dernière version de Busybox, compilez-le et installez-le dans le répertoire correspondant à votre couche basse.

Maintenant que vous avez l'arborescence nécessaire et la couche basse prête, vous pouvez, similairement à ce ce que réalise Docker, mettre en place une couche *container* accessible en lecture/écriture (*read/write*) basée sur la couche basse grâce au système de fichiers overlayfs. Pour obtenir cette vue unifiée de ces deux systèmes de fichiers dans le répertoire `merged`, réalisez un *mount* en précisant le système de fichiers overlayfs selon la syntaxe vue en cours :

```
sudo mount -t overlay overlay -o lowerdir=lower,upperdir=upper,workdir=
   workdir merged
```

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`)

Exercice 2

Ici, on s'intéresse à mieux comprendre comment Docker utilise les couches d'une image.

Exécutez un container `Fedora:26`, 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 quelles sont leurs chemins absolus dans le système de fichiers ?
- Quel est le chemin absolu de la couche *container writable* ?
- En inspectant le système de fichiers sur le serveur, listez le contenu complet de la couche *container writable* (info: `tree` est très pratique pour visualiser les arborescences)
- Sur le serveur, quel est le chemin absolu du répertoire correspondant au système de fichiers racine du container ?

Sur le serveur, déplacez-vous dans le répertoire correspondant au système de fichiers racine 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 utilisez 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 ? Que pouvez-vous également remarquer ?

Exercice 3

Dans cet exercice on désire modifier un container et rendre ces modifications persistantes, en les *committant* 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 instancié depuis l'image que vous venez de télécharger.

On désire rajouter les éditeurs de texte `vim` et `nano` ainsi que les outils `wget` pour télécharger des fichiers via HTTP, `file` pour déterminer la nature des fichiers et `tree` qui est très pratique pour visualiser les arborescences de fichiers.

Une fois ces programmes installés, supprimez les répertoires `/home`, `/opt` et `/mnt`, puis sortez du container.

Listez alors les différences entre votre nouvelle image et l'image utilisée pour l'instantation du container.

- 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 une nouvelle image appelée `mydebian:11.3`. Détruisez alors le container debian précédent étant donné qu'il n'est plus utile, puis inspectez l'historique de l'image `mydebian:11.3` 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 en MB de ces/cette nouvelle(s) couche(s) ?

Enfin, instanciez un nouveau container à partir de votre nouvelle image `mydebian10` et vérifiez que ce nouveau container contient bien les changements effectués (programmes ajoutés + répertoires supprimés).

Exercice 4

Vous allez maintenant déployer un serveur web et voir comment mapper un port de la machine hôte (serveur) vers un container.

Pour ceci, exécutez un nouveau container avec la commande `docker run -d -p8011:80 --name websrv01 hepia/docker_ex04:latest`, puis sur votre machine cliente ouvrez un navigateur pointant sur l'URL de votre serveur Docker sur le port 8011. A noter que vous pouvez également utiliser un outil comme `curl` à la place du navigateur.

Pour obtenir des informations sur le container Nginx, veuillez consulter la page de l'image sur le Docker Hub ici.

- Expliquez en détails ce que réalise la commande `docker run -d ...` précédente.

Affichez la page "home", ainsi que les pages `images` et `ex04`.

Depuis le client Docker, listez le fichier se trouvant dans le répertoire `/usr/share/nginx/html/ex04` du container.

- De quel fichier s'agit-il ?
- Que réalise la commande `docker logs websrv01` ?

Vous allez maintenant voir comment il est possible de partager des fichiers entre la machine cliente et le container grâce à l'utilisation de volumes.

Créez un volume vide nommé `myvol` (cf. `docker help volume`).

- Sur quelle machine et à quel chemin dans le système de fichiers se trouve votre volume `myvol` fraîchement créé ?

Exécutez un deuxième container, comme précédemment, mais nommé `websrv02` et écoutant sur le port 8012, et montez le volume `myvol` dans le répertoire `/usr/share/nginx/html/ex04` de ce nouveau container.

Naviguez sur la page ex04 servie par `websrv02` (port 8012).

- Pourquoi n'y a-t-il aucune différence par rapport à la même page servie par le container `websrv01` (port 8011) ?

Ajoutez un fichier de votre choix au volume `myvol` en utilisant `docker cp` pour le copier dans le container `websrv02` (rappel : le volume est monté dans le répertoire `/usr/share/nginx/html/ex04` du container).

Naviguez à nouveau sur la page ex04 servie par `websrv02`.

- Quels fichiers pouvez-vous voir cette fois-ci ?

Vous allez maintenant essayer de mettre en place un répertoire partagé entre la machine cliente et le container à l'aide du mécanisme de *bind mount*.

Créez un répertoire, p.ex `mydir`, et ajoutez y un fichier, p.ex. `bind.txt`. Exécutez ensuite un troisième container comme precédemment, mais nommé `websrv03`, sur le port 8013 et partagez via un *bind mount*, le répertoire précédent (`mydir`) dans le répertoire `/usr/share/nginx/html/ex04` du container.

- Expliquez pourquoi ce partage via un *bind mount* ne fonctionne pas.
- Comment serait-il donc possible de faire fonctionner ce partage, toujours via un *bind mount* ?