Utilisation des volumes

Dans ce lab, nous allons illustrer la notion de volume. Nous verrons notamment comment définir un volume:

- dans un Dockerfile
- au lancement d'un container en utilisant l'option -v
- en utilisant la ligne de commande

Prérequis

Si vous utilisez Docker for Mac ou Docker for Windows, la plateforme Docker est installée dans une machine virtuelle tournant sur un hyperviseur léger (*xhyve* pour macOS, *Hyper-V* pour Windows). Pour effectuer ce lab il vous faudra utiliser la commande suivante pour accéder à un shell dans cette machine virtuelle.

```
$ docker run -it --privileged --pid=host debian nsenter -t 1 -m -u -n -i sh
```

Comme nous l'avons évoqué dans le cours, cette commande permet de lancer un shell dans un container basé sur debian, et faire en sorte d'utiliser les namespaces de la machine hôte (la machine virtuelle) sur laquelle tourne le daemon Docker.

Une fois que vous avez lancé ce container, vous pourrez naviguer dans le filesystem de la machine sur laquelle tourne le daemon Docker, c'est à dire l'endroit ou les images sont stockées.

Persistance des données dans un container

Nous allons illustrer pourquoi, par défaut, un container ne doit pas être utilisé pour persister des données.

En utilisant la commande suivant, lancez un shell intéractif dans un container basé sur l'image *alpine:3.8*, et nommé le *c1*.

```
$ docker container run --name c1 -ti alpine:3.8 sh
```

Dans ce container, créez le répertoire */data* et dans celui-ci le fichier *hello.txt*.

```
# mkdir /data && touch /data/hello.txt
```

Sortez ensuite du container.

```
# exit
```

Lors de la création du container, une layer read-write est ajoutée au dessus des layers read-only de l'image sous jacente. C'est dans cette layer que les changements que nous avons apportés dans le container ont étés persistés (création du fichier */data/hello.txt*). Nous allons voir comment cette layer est accessible depuis la machine hôte (celle sur laquelle tourne le daemon Docker) et vérifier que nos modifications sont bien présentes.

Utilisez la command *inspect* pour obtenir le path de la layer du container *c1*. La clé qui nous intéresse est *GraphDriver*.

```
$ docker container inspect c1
```

Nous pouvons scroller dans l'output de la commande suivante jusqu'à la clé *GraphDriver* ou bien nous pouvons utiliser le format Go templates et obtenir directement le contenu de la clé. Lancez la commande suivante pour n'obtenir que le contenu de *GraphDriver*.

```
$ docker container inspect -f "{{ json .GraphDriver }}" c1 | jq .
```

Vous devriez obtenir un résultat proche de celui ci-dessous.

```
{
  "Data": {
    "LowerDir": "/var/lib/docker/overlay2/d0ffe7...1d66-
init/diff:/var/lib/docker/overlay2/acba19...b584/diff",
    "MergedDir": "/var/lib/docker/overlay2/d0ffe7...1d66/merged",
    "UpperDir": "/var/lib/docker/overlay2/d0ffe7...1d66/diff",
    "WorkDir": "/var/lib/docker/overlay2/d0ffe7...1d66/work"
  },
  "Name": "overlay2"
}
```

Avec la commande suivante, vérifiez que le fichier *hello.txt* se trouve dans le répertoire référencé par *UpperDir*.

```
$ CONTAINER_LAYER_PATH=$(docker container inspect -f "{{ json
.GraphDriver.Data.UpperDir }}" c1 | tr -d '"')
$ find $CONTAINER_LAYER_PATH -name hello.txt
/var/lib/docker/overlay2/d0ffe7...1d66/diff/data/hello.txt
```

Supprimez le container *c1* et vérifiez que le répertoire spécifié par UpperDir n'existe plus.

```
$ docker container rm c1
$ ls $CONTAINER_LAYER_PATH
ls: cannot access '/var/lib/docker/overlay2/d0ffe7...1d66/diff': No such file or
directory
```

Cela montre que les données créées dans le container ne sont pas persistées et sont supprimées avec le container.

Définition d'un volume dans un Dockerfile

Nous allons maintenant voir comment les volumes sont utilisés pour permetttre de remédier à ce problème et permettre de persister des données en dehors d'un container.

Nous allons commencer par créer un Dockerfile basé sur l'image alpine et définir /data en tant que volume. Tous les élements créés dans /data seront persistés en dehors de l'union filesystem comme nous allons le voir.

Créez le fichier *Dockerfile* contenant les 2 instructions suivantes:

```
FROM alpine:3.8
VOLUME ["/data"]
```

Construisez l'image *imgvol* à partir de ce Dockerfile.

```
$ docker image build -t imgvol .
ending build context to Docker daemon  2.048kB
Step 1/2 : FROM alpine:3.8
```

```
 ---> 3f53bb00af94
Step 2/2 : VOLUME ["/data"]
 ---> Running in 7ee2310fca60
Removing intermediate container 7ee2310fca60
 ---> d8f6d5332181
Successfully built d8f6d5332181
Successfully tagged imgvol:latest
```

Avec la commande suivante, lancez un shell intéractif dans un container, nommé *c2*, basé sur l'image imgvol.

```
$ docker container run --name c2 -ti imgvol
```

Depuis le container, créez le fichier */data/hello.txt*

```
# touch /data/hello.txt
```

Sortons ensuite du container avec la commande CTRL-P suivie de CTRL-Q, cette commande permet de passer le process en tache de fond, elle n'arrête pas le container. Pour être sur que *c2* tourne, il doit apparaitre dans la liste des containers en execution. Vérifiez le avec la commande suivante:

```
$ docker container ls
```

Utilisez la commande *inspect* pour récuperer la clé Mounts afin d'avoir le chemin d'accès du volume sur la machine hôte.

```
$ docker container inspect -f "{{ json .Mounts }}"  c2 | jq .
```

Vous devriez obtenir un résultat similaire à celui ci-dessous (aux ID prêts).

```
[
  {
    "Type": "volume",
    "Name": "d071337...3896",
    "Source": "/var/lib/docker/volumes/d07133...3896/_data",
    "Destination": "/data",
    "Driver": "local",
```

```
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
]
```

Le volume */data* est accessible, sur la machine hôte, dans le path spécifié par la clé Source.

Avec la commande suivante, vérifiez que le fichier *hello.txt* est bien présent sur la machine hôte.

```
$ VOLUME_PATH=$(docker container inspect -f "{{ (index .Mounts 0).Source }}" c2)
$ find $VOLUME_PATH -name hello.txt
/var/lib/docker/volumes/cb5...f49/_data/hello.txt
```

Supprimez maintenant le container *c2*.

```
$ docker container stop c2 && docker container rm c2
```

Vérifier que le fichier *hello.txt* existe toujours sur le filesystem de l'hôte.

```
$ find $VOLUME_PATH -name hello.txt
/var/lib/docker/volumes/cb5...f49/_data/hello.txt
```

Cet exemple nous montre qu'un volume permet de persister les données en dehors de l'union filesystem et ceci indépendemment du cycle de vie d'un container.

Définition d'un volume au lancement d'un container

Précédemment nous avons défini un volume dans le *Dockerfile*, nous allons maintenant voir comment définir des volumes à l'aide de l'option -v au lancement d'un container.

Lancez un container avec les caractéristiques suivantes:

- basé sur l'image *alpine:3.8*
- nommé *c3*
- exécution en background (option -d)
- définition de */data* en tant que volume (option -v)

- spécification d'une commande qui écrit dans le volume ci-dessus

Pour ce faire, lancez la commande suivante:

```
$ docker container run --name c3 -d -v /data alpine sh -c 'ping 8.8.8.8 >
/data/ping.txt'
```

Inspectez le container et repérez notamment le chemin d'accès du volume sur la machine hôte.

```
$ docker inspect -f "{{ json .Mounts }}" c3 | jq .
[
  {
    "Type": "volume",
    "Name": "2ba36b...3ef2",
    "Source": "/var/lib/docker/volumes/2ba36b...3ef2/_data",
    "Destination": "/data",
    "Driver": "local",
    "Mode": "",
    "RW": true,
    "Propagation": ""
  }
]
```

Le volume est accessible via le filesystem de la machine hôte dans le path spécifié par la clé Source.

En utilisant la commande suivante, vérifiez ce que ce répertoire contient.

```
$ VOLUME_PATH=$(docker container inspect -f "{{ (index .Mounts 0).Source }}" c3)
$ tail -f $VOLUME_PATH/ping.txt
64 bytes from 8.8.8.8: seq=34 ttl=37 time=0.462 ms
64 bytes from 8.8.8.8: seq=35 ttl=37 time=0.436 ms
64 bytes from 8.8.8.8: seq=36 ttl=37 time=0.512 ms
64 bytes from 8.8.8.8: seq=37 ttl=37 time=0.487 ms
64 bytes from 8.8.8.8: seq=38 ttl=37 time=0.409 ms
64 bytes from 8.8.8.8: seq=39 ttl=37 time=0.438 ms
64 bytes from 8.8.8.8: seq=40 ttl=37 time=0.477 ms
...
```

Le fichier *ping.txt* est mis à jour régulièrement par la commande ping qui tourne dans le container.

Si nous stoppons et supprimons le container, le fichier *ping.txt* sera toujours disponible via le volume, cependant il ne sera plus mis à jour.

Supprimez le container *c3* avec la commande suivante.

```
$ docker container rm -f c3
```

Utilisation des volumes via la CLI

Les commandes relatives aux volumes ont été introduites dans Docker 1.9. Elles permettent de manager le cycle de vie des volumes de manière très simple.

La commande suivante liste l'ensemble des sous-commandes disponibles.

```
$ docker volume --help
```

La commande create permet de créer un nouveau volume. Créez un volume nommé *html* avec la commande suivante.

```
$ docker volume create --name html
```

Lister les volumes existants et vérifiez que le volume *html* est présent.

```
$ docker volume ls
DRIVER              VOLUME NAME
local               html
```

Comme pour les containers et les images (et d'autres primitives Docker), la commande inspect permet d'avoir la vue détaillée d'un volume. Inspectez le volume *html*

```
$ docker volume inspect html
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/html/_data",
```

```
        "Name": "html",
        "Options": {},
        "Scope": "local"
    }
]
```

La clé Mountpoint définie ici correspond au chemin d'accès de ce volume sur la machine hôte. Lorsque l'on crée un volume via la CLI, le path contient le nom du volume et non pas un identifiant comme nous l'avons vu plus haut.

Utilisez la commande suivante pour lancer un container basé sur *nginx* et monter le volume *html* sur le point de montage */usr/share/nginx/html* du container. Cette commande publie également le port 80 du container sur le port 8080 de l'hôte.

Note: */usr/share/nginx/html* est le répertoire servi par défaut par nginx, il contient les fichiers *index.html* et *50x.html*.

```
$ docker run --name www -d -p 8080:80 -v html:/usr/share/nginx/html nginx
```

Depuis l'hôte, regardez le contenu du volume *html*.

```
$ ls -al /var/lib/docker/volumes/html/_data
total 16
drwxr-xr-x 2 root root 4096 Jan 10 17:07 .
drwxr-xr-x 3 root root 4096 Jan 10 17:07 ..
-rw-r--r-- 1 root root  494 Dec 25 09:56 50x.html
-rw-r--r-- 1 root root  612 Dec 25 09:56 index.html
```

Le contenu du répertoire */usr/share/nginx/html* du container a été copié dans le répertoire */var/lib/docker/volumes/html/_data* de l'hôte.

Accédez à la page d'accueil en lançant un *curl* sur http://localhost:8080

```
$ curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
```

```
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Depuis l'hôte, nous pouvons modifier le fichier *index.html*.

```
cat<<END >/var/lib/docker/volumes/html/_data/index.html
HELLO !!!
END
```

Utilisez une nouvelle fois la commande *curl* pour vérifier que le container sert le fichier *index.html* modifié.

```
curl localhost:8080
HELLO !!!
```