Les volumes de conteneur Docker

Un volume Docker «vit» à l'extérieur du conteneur, sur la machine hôte.

À partir du conteneur, le volume agit comme un dossier que vous pouvez utiliser pour stocker et récupérer des données. Il s'agit simplement d'un point de montage vers un répertoire sur l'hôte. Il existe plusieurs façons de créer et de gérer des volumes Docker. Chaque méthode a ses avantages et ses inconvénients.

Utilisation de la commande "volume create" de Docker

Avantage Rapide et facile.

Inconvénient Le volume sur l'hôte est créé automatiquement par Docker et peut être difficile à localiser et à utiliser.

Créer et nommer un volume

La commande *docker volume create* créera un volume nommé. Le nom vous permet de localiser et d'affecter facilement des volumes Docker à des conteneurs.

Pour créer un volume, utilisez la commande:

```
[root@localhost ~]# docker volume create --name data-volume
```

Lancer un conteneur avec un volume

Par exemple, pour lancer un conteneur à partir de l'image centos que l'on nommera *my-volume-test* et mapper le volume *data-volume* au répertoire */data* du conteneur, la commande est:

```
[root@localhost ~]# docker run -it --name my-volume-test \
-v data-volume:/data centos /bin/bash
[root@9eb0efe02c40 /]# df -Th /data
Filesystem              Type  Size  Used Avail Use% Mounted on
/dev/mapper/centos-root xfs    17G  5.4G   12G  32% /data
```

Lister les volumes

Pour répertorier tous les volumes Docker sur le système, utilisez la commande:

```
[root@localhost ~]# docker volume ls
DRIVER              VOLUME NAME
local               data-volume
```

Inspecter un volume

Pour inspecter un volume nommé, et afficher des informations, y compris son point de montage (le répertoire où il "vit") sur le système hôte:

```
[root@localhost ~]# docker volume inspect data-volume
[
    {
        "Driver": "local",
```

```
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/data-volume/_data",
        "Name": "data-volume",
        "Options": {},
        "Scope": "local"
    }
]
```

Supprimer un volume

Remarque: vous ne pourrez pas supprimer un volume s'il est utilisé par un conteneur existant. Avant de supprimer le volume, vous devrez arrêter et supprimer le conteneur avec les commandes:

Par exemple, pour supprimer le volume *data-volume*, nous devons d'abord arrêter et supprimer le conteneur my-volume-test qui l'utilise:

```
[root@localhost ~]# docker stop my-volume-test
my-volume-test
[root@localhost ~]# docker rm my-volume-test
my-volume-test
```

Le volume *data-volume* peut ensuite être supprimé avec:

```
[root@localhost ~]# docker volume rm data-volume
data-volume
[root@localhost ~]# docker volume ls
DRIVER              VOLUME NAME
```

Créer un volume Docker et spécifier un répertoire hôte

Avantage: vous permet de mapper un dossier hôte spécifique sur un conteneur.

Inconvénients : impossible de créer un volume nommé comme avec la création de volume Docker . Ne peut pas être automatisé avec un Dockerfile.

Si vous souhaitez monter un répertoire spécifique sur votre ordinateur hôte en tant que volume Docker sur le conteneur, par exemple, pour lancer un nouveau conteneur et mapper le dossier */tmp/hostvolume* de l'hôte dans le dossier */containervolume* du conteneur, procédez comme suit:

```
[root@localhost ~]# mkdir /tmp/hostvolume
[root@localhost ~]# echo "Hello World" >> /tmp/hostvolume/host-hello.txt
```

Ensuite, lancez un conteneur nommé *my-directory-test* et mappez /tmp/hostvolume sur l'hôte à /containervolume sur le conteneur avec la commande:

```
[root@localhost ~]# docker run -it --name my-directory-test \
-v /tmp/hostvolume:/containervolume centos /bin/bash
[root@031f928f85a9 /]#
```

Une fois que vous êtes à l'invite de commandes du nouveau conteneur, listez les fichiers dans le volume partagé avec la commande:

```
[root@331cb6e93bbb /]# ls /containervolume/
host-hello.txt
```

Cela fonctionne également dans la direction opposée. Les fichiers que vous placez dans ce répertoire apparaîtront sur l'hôte. Vous pouvez tester cela à partir du conteneur en ajoutant un autre fichier au volume partagé:

```
[root@331cb6e93bbb /]# echo "Hello from the container." >>
/containervolume/container-hello.txt
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte. Une fois sur place, listez les fichiers du volume partagé avec la commande:

```
[root@localhost ~]# ls /tmp/hostvolume/
container-hello.txt  host-hello.txt
```

Créer un volume Docker à l'aide d'un Dockerfile

Avantage : vous permet d'automatiser le processus.

Inconvénient : impossible de créer un volume nommé comme avec la création de volume Docker . Impossible de spécifier un répertoire sur l'hôte.

Par exemple, pour créer un volume nommé /myvolume dans le conteneur, utilisez le Dockerfile suivant:

```
[root@localhost ~]# vim Dockerfile
FROM centos
VOLUME /myvolume
```

Ensuite, créez une image nommée dockerfile-volumetest à partir de ce Dockerfile avec la commande:

```
[root@localhost volume]# docker build -t dockerfile-volumetest .
Sending build context to Docker daemon 2.048 kB
Step 1/2 : FROM centos
 ---> 470671670cac
Step 2/2 : VOLUME /myvolume
 ---> Running in 3760bd4ff268
 ---> 88898f17d888
Removing intermediate container 3760bd4ff268
Successfully built 88898f17d888
```

Ensuite, lancez un conteneur nommé my-dockerfile-test à partir de cette image avec la commande:

```
[root@localhost volume]# docker run --name my-dockerfile-test -it
dockerfile-volumetest /bin/bash
[root@a3b0a8b8ebfd /]#
```

Une fois que vous êtes à l'invite de commandes du nouveau conteneur, créez un petit fichier de test dans le volume partagé avec la commande:

```
[root@a3b0a8b8ebfd /]# echo "Hello World" >> /myvolume/dockerfile-
container-hello.txt
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte.

Ensuite, trouvons le point de montage. Pour ce faire, utilisez la commande:

```
[root@localhost volume]# docker inspect my-dockerfile-test | grep -A 10
Mounts
        "Mounts": [
            {
                "Type": "volume",
                "Name":
"b89e7dca81a8b924cdce125d4151b73cc28d091b5016465893de1dcef5f1961d",
                "Source":
"/var/lib/docker/volumes/b89e7dca81a8b924cdce125d4151b73cc28d091b5016465893
de1dcef5f1961d/_data",
                "Destination": "/myvolume",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }

[root@localhost volume]# ls
/var/lib/docker/volumes/b89e7dca81a8b924cdce125d4151b73cc28d091b5016465893d
e1dcef5f1961d/_data
dockerfile-container-hello.txt
```

Partage de volumes entre conteneurs

Partager un volume sur l'hôte

Si vous créez un volume sur la machine hôte, il peut être utilisé par plusieurs conteneurs différents à la fois. Pour cet exemple, nous allons créer un répertoire sur l'hôte et l'utiliser comme volume partagé entre deux conteneurs.

Commencez par créer un répertoire à utiliser comme volume Docker, puis créez un petit fichier de test dans ce répertoire:

```
[root@localhost ~]# mkdir /tmp/webdata
[root@localhost ~]# echo "Hello from the host." >> /tmp/webdata/host-
hello.txt
```

Ensuite, lancez un conteneur nommé *sql-database* à partir de l'image officielle PostgreSQL, et mappez */webdata* sur l'hôte vers */data* sur le conteneur avec la commande:

```
[root@localhost ~]# docker run -it --name sql-database -v
/tmp/webdata:/data postgres /bin/bash
root@2ea7e51c8678:/#
```

Une fois que vous êtes à l'invite de commande du nouveau conteneur, vérifiez que le volume partagé soit correctement configuré :

```
root@55c90d07c903:/# ls /data/
host-hello.txt
```

Ajoutons un fichier à ce volume partagé avec la commande:

```
root@55c90d07c903:/# echo "Hello from the sql-database container." >>
/data/sql-hello.txt
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte.

Lancez maintenant un conteneur nommé webapp à partir de l'image officielle PHP + Apache et mappez */webdata* sur l'hôte vers */var/www/html* sur le conteneur.

```
[root@localhost ~]# docker run -it --name webapp -v
/tmp/webdata:/var/www/html php:5.6-apache /bin/bash
root@7f6e0540c434:/var/www/html#
```

Une fois que vous êtes à l'invite de commande du nouveau conteneur, vérifiez que le volume partagé est correctement configuré :

```
root@7f6e0540c434:/var/www/html# ls /var/www/html
host-hello.txt  sql-hello.txt
```

Ajoutons également un fichier depuis ce conteneur:

```
root@7f6e0540c434:/var/www/html# echo "Hello from the webapp container." >>
/var/www/html/webapp-hello.txt
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte. Une fois sur la machine hôte, vous verrez les trois fichiers répertoriés :

```
[root@localhost ~]# ls /tmp/webdata
host-hello.txt  sql-hello.txt  webapp-hello.txt
```

Maintenant que les deux conteneurs partagent un répertoire qui "vit" sur l'hôte, les données peuvent être transférées instantanément entre les trois emplacements simplement en les déplaçant vers ce répertoire.

Création d'un bind mount avec `docker volume`

La création d'un bind mount (un volume qui a un répertoire explicitement déclaré sous-jacent) est facile lors de l'utilisation de docker run :

```
docker run -v /var/app/data:/data:rw my-container1
```

Maintenant, chaque fois que vous écrivez dans le répertoire `data` du conteneur, vous écrivez également `dans /var/app/data`.

Vous pouvez faire la même chose avec l'option `--mount`.

```
docker run --mount type=bind,source=/var/app/data,target=/data my-
container2
```

Parfois, vous souhaiterez peut-être créer un bind mount *indépendant* d'un conteneur.

```
docker volume create \
--driver local \
-o o=bind \
-o type=none \
-o device=/var/app/data \
example-volume
```

Vous pouvez créer un conteneur qui utilise ce volume :

```
docker run -v example-volume:/data my-container3
```

Utilisation d'un conteneur comme volume de données partagé

Vous pouvez également configurer un conteneur distinct en tant que volume de données partagé.

Remarque: Cela fonctionnera, que le conteneur cible soit en cours d'exécution ou non. Les volumes Docker ne sont jamais supprimés et persistent même après l'arrêt du conteneur.

Pour cet exemple, nous allons créer un conteneur de données appelé *data-storage* qui servira de volume de données, et deux autres conteneurs qui le partageront comme volume de stockage.

Tout d'abord, lancez le conteneur de stockage de données à partir de l'image officielle de CentOS 7:

```
[root@localhost ~]# docker run -it -v /shared-data --name data-storage
centos /bin/bash
[root@20eeae8a28a3 /]#
```

Ajoutez ensuite un petit fichier dans le dossier */shared-data* :

```
[root@20eeae8a28a3 /]# echo "Hello from the data-storage container." >>
/shared-data/data-storage-hello.txt
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte.

Maintenant, lancez le conteneur d'application à partir de l'image Python officielle et montez le conteneur de stockage de données en tant que volume:

```
[root@localhost ~]# docker run -it --name app --volumes-from data-storage
python /bin/bash
```

Listez les fichiers dans le volume partagé avec la commande:

```
root@e9e8ea297f34:/# ls /shared-data
data-storage-hello.txt
```

Comme vous pouvez le voir, le dossier /shared-data a été monté à partir du dossier /shared-data sur le conteneur de stockage de données et contient le fichier data-storage-hello.txt .

Ajoutons-en un à partir de ce conteneur:

```
root@e9e8ea297f34:/# echo "Hello from the app container." >> /shared-
data/app-hello.txt
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte.

Enfin, lancez le conteneur Web à partir de l'image Apache officielle et montez le conteneur de stockage de données en tant que volume:

```
[root@localhost ~]# docker run -it --name web --volumes-from data-storage
httpd /bin/bash
```

Listez les fichiers dans le volume partagé avec la commande:

```
root@c05f311bd4ea:/usr/local/apache2# ls /shared-data/
app-hello.txt  data-storage-hello.txt

root@c05f311bd4ea:/usr/local/apache2# exit
```

Monter un volume en lecture seule

Tout au long de ce Lab, nous avons monté des volumes avec l'accès en lecture-écriture par défaut. Si vous souhaitez restreindre un conteneur à un accès en lecture seule à un volume, ajoutez simplement *:ro* au volume de conteneur spécifié dans l' instruction -v.

Par exemple, commencez par créer un volume sur l'hôte nommé *limited-access*:

```
[root@localhost ~]# docker volume create --name limited-access
limited-access
```

Exécutez ensuite un conteneur à partir de l'image centos nommé *allowed-to-write* et mappez le volume *limited-access* en tant que volume normal (lecture-écriture):

```
[root@localhost ~]# docker run -it --name allowed-to-write -v limited-
access:/data centos /bin/bash
[root@8154f26c537d /]#
```

Une fois que vous êtes à l'invite de commande de ce conteneur, créez un fichier de test avec la commande:

```
[root@8154f26c537d /]# echo "Hello from the container that is allowed to
write." >> /data/hello.txt
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte.

Ensuite, exécutez un conteneur à partir de l'image centos nommé *not-allowed-to-write* et mappez le volume *limited-access* en tant que volume en lecture seule:

```
[root@localhost ~]# docker run -it --name not-allowed-to-write -v limited-
access:/data:ro centos /bin/bash
```

Si vous essayez de créer un fichier de test dans le volume partagé avec une commande comme celle-ci:

```
[root@bec74d05cf4e /]# echo "Hello from the container that is not allowed
to write." >> /data/no-access.txt
bash: /data/no-access.txt: Read-only file system

[root@bec74d05cf4e /]# exit
```

Nettoyage:

```
[root@localhost ~]# docker container stop $(docker container ls -q)
[root@localhost ~]# docker container prune
[root@localhost ~]# docker volume prune
```