FONCTIONNEMENT ET MANIPULATION DES VOLUMES DANS DOCKER

Les volumes

Introduction

Comme vous le savez déjà, les données dans un conteneur sont éphémères. Il faut donc trouver un moyen à notre solution "**comment sauvegarder les données d'un conteneur**". Nous avions vu une méthode qui n'est pas très recommandée, qui consistait à transformer notre conteneur en image et de relancer un nouveau conteneur basé sur cette nouvelle image. Je vous ai aussi parlé d'une autre méthode qui repose sur les volumes, et ça tombe bien car nous allons voir cette méthode plus en détail sur ce chapitre.

Pourquoi les données d'un conteneur sont éphémères ?

Afin de comprendre ce qu'est un volume Docker, nous devons d'abord préciser le fonctionnement normal du **système de fichiers dans Docker**.

En effet, une image Docker se compose d'un ensemble de **layers (calques) en lecture seule**. Lorsque vous lancez un conteneur à partir d'une image, Docker ajoute au sommet de cette pile de layers un nouveau **layer en lecture-écriture**. Docker appelle cette combinaison de couches un "**Union File System**".

Voyons voir comment le moteur Docker gère les modifications de vos fichiers au sein de votre conteneur :

- Lors d'une modification de fichier, Docker crée une copie depuis les couches en lecture seule vers le layer en lecture-écriture.
- Lors d'une création de fichier, Docker crée le fichier que sur le layer en lecture-écriture, et ne touche pas au layer en lecture seule.
- Lors d'une suppression de fichier, Docker supprime le fichier que sur le layer en lecture-écriture, et si il est déjà éxistant dans le layer en lecture seule alors il le garde.

Les données dans le layer de base sont en lecture seule, elles sont donc protégées et intactes par toutes modifications, seul le layer en lecture-écriture est impacté lors de modifications de données.

Lorsqu'un conteneur est supprimé, le layer en lecture-écriture est supprimé avec. Cela signifie que toutes les modifications apportées après le lancement du conteneur auront disparus avec.

La création des volumes

Afin de pouvoir sauvegarder (persister) les données et également partager des données entre conteneurs, Docker a développé le concept de volumes. Les volumes sont des répertoires (ou des fichiers) qui ne font pas partie du système de fichiers Union mais qui existent sur le système de fichiers hôte.

En outre, les volumes constituent souvent le meilleur choix pour persistance des données pour le layer en lecture-écriture, car un volume n'augmente pas la taille des conteneurs qui l'utilisent et son contenu existe en dehors du cycle de vie d'un conteneur donné.

Créer et gérer des volumes

Contrairement à un montage lié, vous pouvez créer et gérer des volumes en dehors de la portée de tout conteneur.

Pour **créer un volume**, nous utiliserons la commande suivante :

```
docker volume create <VOLUMENAME>
```

Soit :

```
docker volume create volume-test
```

Pour **lister les volumes** :

```
docker volume ls
```

Résultat :

```
DRIVER              VOLUME NAME
local               0af7c41b62cf782b4d053e03b4b11575078bb01bbda90edfa73fbc88ac1703ec
...
local               volume-test
```

Pour **récolter des informations sur un volume**, il faut utiliser la commande suivante :

```
docker volume inspect volume-test
```

Résultat sous format JSON:

```
[
    {
        "CreatedAt": "2019-07-03T10:03:20+02:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/volume-test/_data",
        "Name": "volume-test",
        "Options": {},
        "Scope": "local"
    }
```

```
]
```

Dans ce résultat, on peut remarquer que toutes les nouvelles données de notre conteneur seront stockées sur le point de montage `/var/lib/docker/volumes/volume-test/_data`.

Pour **supprimer un volume** :

```
docker volume rm volume-test
```

Démarrer un conteneur avec un volume

Si vous démarrez un conteneur avec un volume qui n'existe pas encore, Docker le créera pour vous.

Pour démarrer un conteneur avec un volume, il faut utiliser l'option `-v` de la commande `docker run`.

Pour ce chapitre, nous allons créer une petite image pour tester cette option, commencez d'abord par créer un Dockerfile avec le contenu suivant :

```
FROM alpine:latest

RUN mkdir /data

WORKDIR /data
```

Ensuite buildez notre image

```
docker build -t vtest .
```

la commande suivante va créer et monter le volume `data-test` dans le dossier `/data` du conteneur.

```
docker run -ti --name vtest_c -v data-test:/data vtest
```

Ouvrez un autre terminal et dans celui-ci inspectez le nouveau volume :

```
docker volume inspect data-test
```

Résultat sous format JSON:

```
[
    {
        "CreatedAt": "2019-07-03T10:28:55+02:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/data-test/_data",
        "Name": "data-test",
        "Options": null,
        "Scope": "local"
    }
]
```

Nous allons essayer de voir en temps réel le contenu de ce volume, pour cela utilisez la commande suivante sur votre nouveau terminal :

```
sudo watch -n 1 ls /var/lib/docker/volumes/data-test/_data
```

Résultat :

```
sEvery 1,0s: ls /var/lib/dock...  localhost.localdomain: Wed Jul  3 10:48:52 2019
```

Pour le moment le dossier est vide, maintenant retournez vers le terminal avec le shell du conteneur et créez dans le dossier **/data** un fichier avec le texte suivant :

```
echo "ceci est un test" > test.txt
```

Si vous retournez sur le nouveau terminal, vous verrez dessus votre nouveau fichier :

```
sEvery 1,0s: ls /var/lib/dock...  localhost.localdomain: Wed Jul  3 10:48:59 2019

test.txt
```

Maintenant, je vais quitter mon conteneur avec la commande `exit` et le supprimer :

```
docker rm -f vtest_c
```

Je vais recréer un nouveau conteneur, pour vérifier que les données ont bien été sauvegardées :

```
docker run -ti --name vtest_c -v data-test:/data vtest
```

Dans ce même nouveau conteneur, vérifie je le contenu du fichier crée précédemment :

```
cat test.txt
```

Résultat :

```
ceci est un test
```

Cool, nos données sont maintenant bien sauvegardées .

Amélioration de notre image LAMP

Dans le chapitre précédent, nous avions créé une image avec stack LAMP, malheureusement, c'est pas top niveau persistance de données car, lors d'un redémarrage du conteneur, nous allons rencontrer les deux problèmes suivants :

- Les données de notre base de données ne sont pas sauvegardées.
- Les modifications des sources de notre application ne seront pas appliquées.

Pour résoudre ce problème, nous allons utiliser les volumes !

Commencez par télécharger le projet , en cliquant ici. Dans ce projet, j'ai gardé le même Dockerfile, mais j'ai juste changé les fichiers sources en rajoutant un formulaire.

Désarchivez le fichier zip et buildez votre image :

```
docker build -t my_lamp .
```

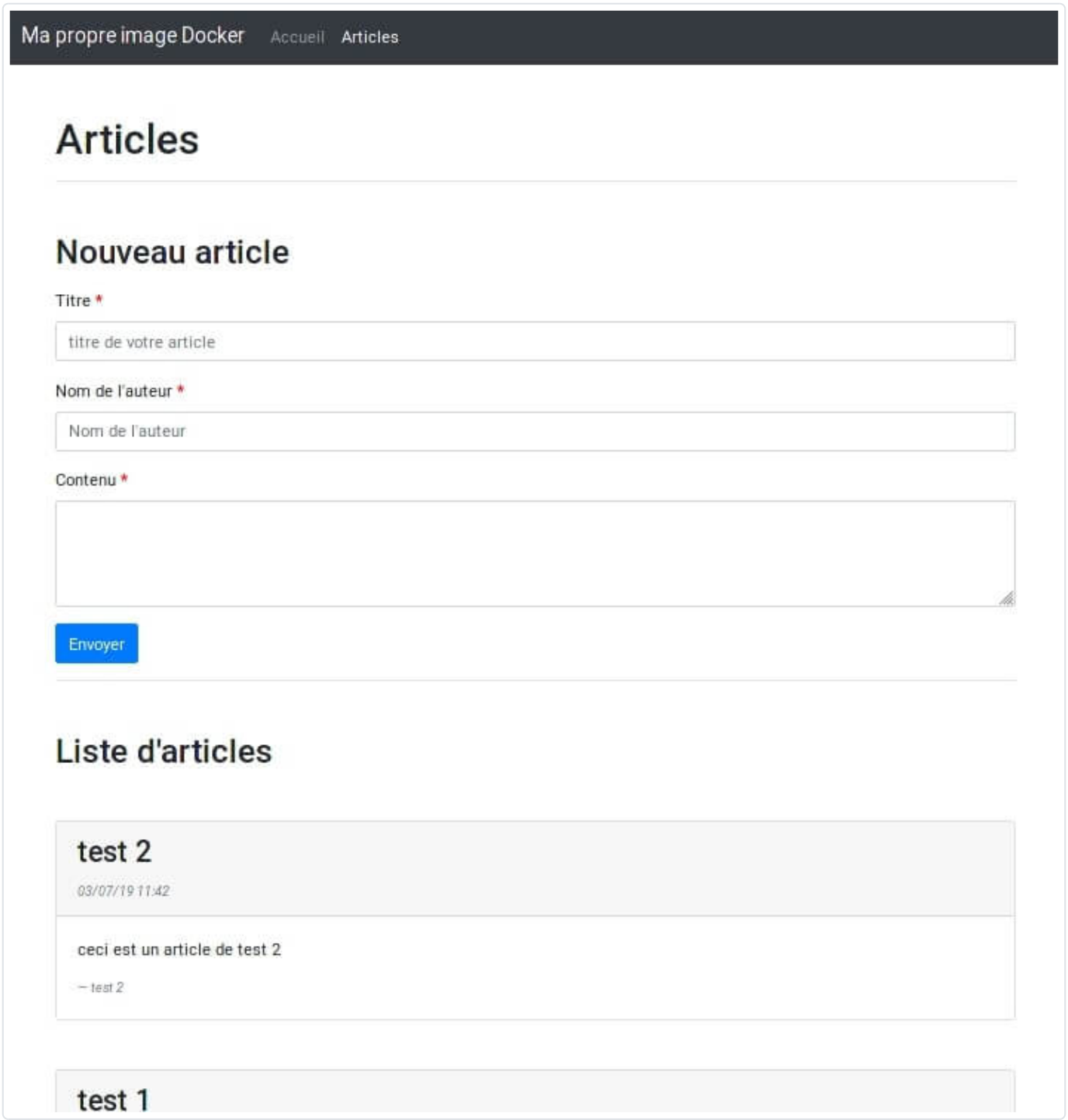
Concernant la base de données, nous allons créer un volume nommé "mysqldata" qui sera par la suite basé sur le dossier **/var/lib/mysql** du conteneur:

```
docker volume create --name mysqldata
```

Pour les sources de mon application, je vais faire les choses différemment. Je vais juste changer le dossier source du volume (les volumes nous donne cette possibilité). Lançons donc notre conteneur :

```
docker run -d --name my_lamp_c -v $PWD/app:/var/www/html -v mysqldata:/var/lib/mysql
```

La commande `$PWD` prendra automatiquement le chemin absolu du dossier courant, donc faites bien attention à exécuter votre image depuis le dossier du projet où mettez le chemin complet si vous souhaitez lancer votre commande depuis n'importe quelle autre chemin.



Vous pouvez dès à présent modifier vos sources, depuis votre conteneur ou votre machine local et vos changements seront permanents et immédiatement traités par l'interpréteur php. Les données de votre base de données seront aussi automatiquement sauvegardées dans le volume mysqldata.

Bon bah bravo, nous avons atteint notre objectif !

Conclusion

Avec les volumes, nous avons pu créer une image assez stable et éxploitable dans un environnement de production. Vous l'avez sûrement remarquer mais notre commande `docker run` commence à devenir vraiment longue et nous n'avons pas encore résolu le problème qui est de séparer notre conteneur d'application web de notre conteneur de base de données. Et c'est pour ces raisons que dans le prochain chapitre, nous verrons le docker-compose.yml, c'est un fichier qui va nous permettre de définir le comportement de nos conteneurs et d’exécuter des applications Docker à conteneurs multiples.

Comme d'habitude, voici un petit récapitulatif des commandes liées aux volumes :

```
## Créer une volume
docker volume create <VOLUME NAME>

# Lister les volumes
docker volume ls

## Supprimer un ou plusieurs volume(s)
docker volume rm <VOLUME NAME>
    -f ou --force : forcer la suppression

## Récolter des informations sur une volume
docker volume inspect <VOLUME NAME>

## Supprimer tous les volumes locaux inutilisés
docker volume prune
    -f ou --force : forcer la suppression

## Supprimer un conteneur Docker avec le/les volumes associés
docker rm -v <CONTAINER_ID ou CONTAINER_NAME>
    -f ou --force : forcer la suppression
    -v ou --volume : supprime les volumes associés au conteneur
```