TP Docker

Docker est une plateforme qui permet de développer, expédier et exécuter des applications dans des conteneurs. Ces conteneurs offrent une solution légère et portable, encapsulant une application avec toutes ses dépendances pour qu'elle puisse s'exécuter de manière cohérente dans n'importe quel environnement.

Dans ce TP, nous allons explorer les fondamentaux de Docker, comprendre son fonctionnement et apprendre à créer, gérer et déployer des conteneurs. Nous aborderons les concepts clés tels que les images Docker, les conteneurs, les Dockerfiles et les réseaux Docker.

L'objectif de ce TP est de vous fournir les compétences nécessaires pour utiliser Docker dans vos projets de développement, en vous permettant de créer des environnements de développement cohérents et reproductibles, de simplifier le déploiement de vos applications, et de faciliter leur scalabilité.

Pour ce TP vous utiliserez la VM Docker sur eCloud, instanciée précédemment pour le TP Git.

C’est parti !

Premières commandes Docker :

```
docker version
docker info
docker ps (-a pour aussi voir les conteneurs qui ne sont pas en exécution)
```

- Donnez l’interprétation des commandes.

Les images présentes actuellement :

```
docker images
```

Récupération d’une image nginx :

```
docker pull nginx
```

Lancement d’un conteneur :

```
docker run --detach --name www nginx
```

Refaites les commandes `docker ps` et `docker images`.

Commandes d’arrêt et de suppression :

```
docker stop nom/ID
docker rm nom/ID (option -f si en fonctionnement)
docker container prune : suppression de tous les conteneurs stoppés
```

Remarques :

- L'option **--rm** de docker run permet une suppression automatique du container une fois celui-ci arrêté
- La commande **docker container start** permet de redémarrer un container précédemment arrêté
- La commande **docker container restart** est équivalent à docker stop puis docker start

Vous n'aviez peut-être pas vérifié mais votre container créé n'est pas accessible via le port 80, ce port est fermé sur le VLAN des instances eCloud.

Supprimez ce container puis créez le de nouveau en redirigeant ce port 80 vers le port 4208 :

```
docker run -d --name www -p 4208:80 nginx
```

Maintenant vous pouvez vérifier sur votre navigateur web.

Vous pouvez vous connecter à ce container via la commande :

```
docker exec -it www bash
```

Vous pouvez aussi copier un fichier depuis/vers un container (faites vous un dossier docker dans lequel vous ferez vos différentes manips) :

```
docker cp www:/usr/share/nginx/html/index.html .
```

Modifiez ce fichier index.html, copiez le sur le container puis relancez le container (docker restart), puis vérifiez.

Remarques :
Toutes les modifications apportées à un container durant son fonctionnement seront irrémédiablement perdues une fois celui-ci supprimé, sauf si on commit l'état au sein d'une nouvelle image par la commande :
docker container commit www nginx-du-dim
Vous pouvez vérifier la présence de cette nouvelle image avec `docker images` .

Maintenant vous pouvez commit cette version avec l'index.html que vous venez de modifier :

```
docker commit www loginagalan-nginx
```

- Validez cette modification en créant un nouveau conteneur avec la nouvelle image.

CMD & Entrypoint

CMD et ENTRYPOINT sont deux mécanismes utilisés pour définir quelle commande ou quel programme sera exécuté par défaut lorsque le container démarre. Bien que leur fonction paraisse similaire, leur comportement diffère légèrement.

———————— **CMD** : Commande par défaut modifiable ————————

L'instruction CMD sert à spécifier une commande par défaut qui sera exécutée lorsque le container démarre. Cependant, si l'utilisateur spécifie une autre commande lors du lancement du container (via la ligne de commande docker run), cette commande remplace celle définie par CMD.

CMD est principalement utilisée pour fournir des valeurs par défaut qui peuvent facilement être remplacées à l'exécution.

———————— **ENTRYPOINT** : Commande fixe, mais extensible ————————

L'instruction ENTRYPOINT, quant à elle, définit une commande ou un programme que le container exécutera toujours, peu importe ce qui est spécifié lors du démarrage. Toutefois, vous pouvez passer des arguments supplémentaires à cette commande. En d'autres termes, ENTRYPOINT fixe la commande de base, tandis que les arguments peuvent être ajustés ou étendus au moment du démarrage.

ENTRYPOINT est souvent utilisé pour s'assurer qu'un programme spécifique est toujours exécuté, tout en permettant à l'utilisateur de modifier ou d'ajouter des paramètres.

———————— **Interaction** entre CMD et ENTRYPOINT ————————

Lorsque CMD et ENTRYPOINT sont utilisés ensemble, CMD fournit des arguments par défaut à la commande définie par ENTRYPOINT. Ces arguments peuvent être modifiés ou remplacés à l'exécution.

Pour savoir quelles commandes ont été définies dans une image Docker :

```
docker image inspect <nom_de_l_image> | jq | grep -i -A 5 "Cmd"
docker image inspect <nom_de_l_image> | jq | grep -i -A 5 "Entrypoint"
```

Vous allez maintenant lancer un container d'une image alpine.

- ➢ Que constatez-vous ? Pourquoi ?

Rajoutez la commande « sleep infinity » à la fin de votre commande 'docker run'.

- ➢ Que constatez-vous ? Pourquoi ?

Les réseaux

Par défaut Docker propose trois types de réseaux :

```
docker network ls
```

- bridge
  - Réseau par défaut pour les containeurs
  - Réseau switché avec adressage spécifique
  - Adressage dynamique
- host
  - Rattache les conteneurs au réseau de la machine hôte
  - Propage le fichier /etc/hosts de la machine hôte dans les conteneurs
- none
  - Réseau sans réseau....
  - Un CT créé dans ce réseau n'aura pas de carte réseau et donc aucune connectivité

Vous pouvez inspecter ces réseaux avec la commande :

```
docker network inspect bridge
```

Remarques :
Chaque type d'objet docker peut être inspecté avec la commande `**docker *type* inspect *name***`. Les objets docker sont : image, container, volume, network, service (swarm), stack (swarm) et secret (swarm).
Pour plus de lisibilité vous pouvez concaténer la commande **jq** à votre première commande, par exemple : **docker network inspect bridge | jq**

Créez un nouveau réseau (type bridge, le plus courant..) :

```
docker network create net1 --subnet 172.31.0.0/16
```

Affectez un nouveau conteneur nginx à ce réseau net1 :

```
docker run -d --net net1 --ip 172.31.0.200 --name ww2 nginx
```

Créez un nouveau réseau net2, toujours de type bridge :

```
docker network create net2
```

Affectez un nouveau conteneur nginx à ce réseau net2.

- ➢ Que constatez-vous sur le réseau affecté à ce conteneur ?

Micro-TP (lemp) : création de deux conteneurs

Consignes : lancez deux conteneurs en mode détaché

- container 1
  - Nom : nginx
  - Image : nginx
  - Réseau : lemp
  - Redirection port depuis le 4208
- container 2
  - Nom: php
  - Image: php:8.2-fpm
  - Réseau : lemp

➢ Quelles commandes permettent de lancer ces deux conteneurs ?

Remarques :
Nginx ne peut pas exécuter du code PHP directement. Pour servir des applications PHP avec Nginx, PHP-FPM (FastCGI Process Manager) est utilisé comme interpréteur, en tant que service séparé.

Les volumes

Les volumes permettent la persistance des données, il en existe trois types :

———————— Type **volume** ————————

Caractéristiques :

- Les plus couramment utilisés.
- Gérés par Docker.
- Persistants, même si les conteneurs sont supprimés. Stockage en local sur l'hôte.
- Peuvent être montés dans un ou plusieurs conteneurs Docker.

Cas pratiques :

- **Stockage de données persistantes** : Les volumes Docker sont souvent utilisés pour stocker des données persistantes, telles que des bases de données, des fichiers de configuration ou des fichiers de logs.
- **Partage de données entre plusieurs conteneurs** : Les volumes Docker facilitent le partage de données entre plusieurs conteneurs, ce qui peut être utile dans une architecture microservices.

Exemple : docker run -d -v my_volume:/path/in/container my_image

———————— Type **bind** ————————

Caractéristiques :

- Montent un répertoire ou un fichier de l'hôte dans le conteneur.
- Non gérés par Docker, simplement un point de montage.
- Partagent des données entre l'hôte et le container en temps réel.

Cas pratiques :

- **Configuration dynamique** : Si vous avez besoin de configurer dynamiquement le contenu d'un container avec des fichiers de l'hôte.

- **Attention** : ils doivent être utilisés avec précaution dans des environnements de production

Exemple : docker run -d -v /path/on/host:/path/in/container my_image

——————————— Type **tmpfs** ———————————

Ces volumes Docker stockent des données dans la mémoire vive de l'hôte plutôt que sur le disque local. Les volumes "tmpfs" sont utiles pour stocker des fichiers temporaires ou des données qui ne doivent pas être persistantes entre les redémarrages du conteneur.

———

Vous allez maintenant supprimer le containeur nginx puis en créer un nouveau avec un volume bind pointant sur un répertoire **nginx-html** dans lequel vous aurez mis le fichier **index.html** ci-dessous vers le dossier **/usr/share/nginx/html** du container :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>TP Docker IM2AG</title>
  </head>
  <body>
    <h1>Vous êtes bien sur le TP Docker de l'IM2AG</h1>
  </body>
</html>
```

- Quelle est la commande ?
- Faites une modification de **index.html** en local puis vérifiez. Pratique... mais est-ce vraiment conseillé d'après vous ??

Supprimez de nouveau ce container puis créez-en un nouveau avec le même volume bind précédemment créé, mais aussi avec un volume (type « volume ») **nginx-logs** pointant vers **/var/log/nginx**.

- Quelle est la commande ?

Commandes intéressantes :

```
docker volume ls
docker volume inspect <volume_name>
```

Micro-TP : volume persistant

Créez un volume "vol-logs-nginx".

- Quel est le contenu de ce volume ?

Créez un conteneur en intégrant ce volume "vol-logs-nginx" se référant à **/var/log/nginx**.

- De nouveau, quel est le contenu du volume ?

Maintenant allez dans le conteneur, ajoutez un fichier dans **/var/log/nginx**, puis supprimez ce conteneur et créez en un nouveau.

- De nouveau, quel est le contenu du volume ? Expliquez le chronologie de "vie" pour ce type de volume ?
- Dans quel contexte peuvent servir ces types de volume ?