# TRAVAUX PRATIQUES
# DOCKER : MODE CLI

## I. introduction

Dans ce TP, nous allons pratiquer les différentes commandes en ligne de Docker pour créer et gérer des conteneurs.

## II. commandes de bases

à partir d'une image alpine, nous allons tester les commandes suivantes

(1) nommer le conteneur
(2) copie de fichier depuis l'hôte vers le conteneur. vous vérifierez en vous connectant au conteneur pour lister les fichiers.
(3) copie de fichier depuis le conteneur vers l'hôte
(4) changement de répertoire initial
(5) exécution directe d'une commande
(6) suppression automatique du conteneur

## III. travail dans un conteneur et commit

à partir d'une image debian lancé en mode interactif, ajouter un utilisateur nono. Vous allez ensuite créer une nouvelle image à partir de ce conteneur, puis lancer un nouveau conteneur avec cette image. Que constatez vous ? Lancer à nouveau un conteneur avec cette image en précisant l'utilisateur à connecter comme nono

## IV. réseaux

IV.1. **conteneur nécessitant le réseaux.** les conteneurs déployant des services nécessitant le réseaux ont besoin de voir certains de leur port accessible depuis la machine hôte ou depuis d'autre conteneur. Pour cela, il va falloir préciser ces ports. Nous allons utiliser une image de service web nginx.

(1) lancer le conteneur avec un docker run sans option. Que se passe t'il ?
(2) faite un docker inspect du conteneur. Regarder la partie configuration. Qu'est il mentionné au niveau des ports exposés.
(3) utiliser l'option pour exposer le port 80. cherchez sur quel port de l'hôte a été transféré le port 80 du conteneur. Ouvrez un navigateur pour vérifiez que cela fonctionne.
(4) chercher l'adresse IP associée au conteneur et utilisez la dans le navigateur. Qu'obtenez vous ?

Nous allons nous occuper de la couche réseaux. Pour voir les interfaces qui sont disponible, exécutez la commande

```
ip a
```

dans un terminal. Quelles information obtenez vous ? Dans un conteneur actif, exécutez un la même commande. Qu'obtenez vous. Faites la liste des interfaces et leur liens avec Docker.

IV.2. **définition du réseau.** Vous avez pu voir qu'une adresse IP a été affectée au conteneur en 172.17.0.1. Cette adresse dépend d'un réseau crée automatiquement pour les conteneur par le docker engine. la commande suivante permet de lister les réseaux disponible pour docker

```
docker network ls
```

Quels sont les réseaux qui sont disponible ? Quels est le réseaux utilisé par le conteneur nginx ? exécutez la commande

```
docker inspect nom_reseau
```

afin de voir les paramètres de configuration. Si vous lancer un nouveau conteneur, que constatez vous dans le résultat de la commande ?

IV.3. **création d'un sous réseau.** nous allons créer un nouveau sous-réseaux sur la base **176.20.0.1/8**. Pour cela, nous allons utiliser la commande

```
4 docker network create -d bridge --subnet 176.20.0.0/8 --gateway 176.20.0.1 dock1
```

et pour utiliser ce réseau dans un conteneur, il faut ajouter l'option –network #**nom du réseau** afin de mettre le conteneur sur ce réseau.

Essayez dans un conteneur de faire un ping sur un conteneur dans un autre sous réseau. Qu'observez vous ?

## V. les volumes

Dans cette partie nous allons utiliser les volumes pour créer la persistence des données.

V.1. **création d'un volume dans docker run.** Vous allez dans un premier temps créer un volume directement dans la commande docker run en ajoutant l'option –volume. Cette option prend comme valeur le chemin de montage dans le conteneur. Une fois lancé en mode détaché interactif, exécutez la commande suivante dans un shell.

```
5 echo "bonjour le monde" > path/bonjour.txt
```

ou path correspond au chemin d'accès du volume.

Utilisez la commande suivante pour afficher la liste des volumes

```
6 docker volume ls
```

et vous pouvez utiliser cette autre commande pour afficher plus d'information sur le volume

```
7 docker volume inspect idconteneur
```

Par contre si vous détruisez le conteneur ou si vous exécutez à nouveau la commande, vous pouvez voir qu'un nouveau volume est créé. Pour réutilisez le volume, une première méthode consiste à définir le chemin local utilisé. Pour cela, nous allons préciser dans l'option –volume le répertoire local et le point de montage sous la forme

```
8 docker run --tid --name toto --volume /tmp/data:/mydata alpine
```

cette commande va monter le répertoire /tmp/data d' l'hôte dans sous le nom /mydata dans le système de fichier du conteneur. Maintenant utiliser un répertoire de montage défini est une faille de sécurité. Le plus simple au final est de créer un volume nommé au préalable puis de le fournir au moment du démarrage du conteneur. La commande suivante permet de créer un volume nommé que nous allons pouvoir réutiliser dans plusieurs conteneurs.

```
9 docker volume create mydata
```

ensuite il suffira de préciser mydata à la place du répertoire de l'hôte pour créer le point de montage du volume dans le conteneur.