

☐ Utilisation

Ici on va parler de comment utiliser Docker *en ligne de commandes*, on parlera plus tard de systèmes tel que docker-compose, Kubernetes, Pterodactyl, Portainer ou autre.

▶ Lancer un conteneur

Pour lancer un conteneur on va utiliser la commande docker `run`, voici un exemple :

```
sudo docker run --name mon-super-docker -e HELLO=world -v ./mon-volume:/home -p 8080:80 -it
docker.io/library/debian
```

Ici dans cet exemple, on a plusieurs options à notre commande :

- `--name` va nomer le conteneur
- `-e HELLO=world` va définir la variable d'environnement "HELLO" pour valoir "world"
- `-v ./mon-volume:/home` va dire que le contenu du dossier local "mon-volume" remplacer le dossier du conteneur "/home" (les dossiers/fichiers hôtes dans un volume passe toujours avant)
- `-p 8080:80` va lier le port `8080` de la machine hôte au port `80` du conteneur.
- `-it` va allouer un `tty` à la machine et l'afficher de manière interactive. On peut donc lancer des commandes dans le conteneur.
- `docker.io/library/debian` défini que l'on va lancer l'image officielle de debian depuis les serveurs de Docker Hub

☐ Lister les conteneur et les image (ps, images)

Voici quelques commandes pour afficher des informations sur les conteneurs :

```
# Va afficher la liste des conteneurs en cours
sudo docker ps
```

```
# Va afficher la liste de tous les conteneurs
```



```
sudo docker ps -a
```

```
# Va afficher la liste des images docker
```

```
sudo docker images
```

☐☐ Gérer des images et des conteneurs (start, stop, rm)

```
# Arrêter un conteneur
```

```
sudo docker stop CONTAINER_ID
```

```
# Relancer un conteneur
```

```
sudo docker start CONTAINER_ID
```

```
# Supprimer un conteneur (nécessite qu'il soit préalablement arrêté)
```

```
sudo docker rm CONTAINER_ID
```

```
# Supprimer une image (nécessite qu'aucun conteneur n'en dépende)
```

```
sudo docker image rm IMAGE_ID
```

```
# Note: CONTAINER_ID peut aussi être le nom du conteneur
```

Créer une nouvelle image et la mettre en ligne (build et push)

On va voir en détail cette partie dans la prochaine partie.

☐☐ Exécuter des commandes dans un conteneur en cours (exec)

Si on souhaite *inspecter* un conteneur on peut utiliser la commande `exec`


```
# Va entrer dans une shell bash du conteneur "mon-conteneur"
```

```
sudo docker exec -it mon-conteneur /bin/sh
```

```
# Va exécuter la commande "ls" dans mon-conteneur
```

```
sudo docker exec mon-conteneur ls
```

☐☐ Compiler dans un environnement propre avec Docker (exemple)

Si vous souhaitez compiler votre programme, imaginons en Rust, mais que vous souhaitez le faire dans un environnement neuf pour empêcher des incompatibilité pour les futurs utilisateur·ice·s. Vous pouvez utiliser Docker pour ça !

```
docker run --rm -u "$(id -u)": "$(id -g)" -v "$PWD":/usr/src/myapp -w /usr/src/myapp rust:bullseye cargo build --release
```

Pour décortiquer un peu la commande:

- `--rm` va automatiquement supprimer le conteneur une fois la compilation terminée
- `-u "$(id -u)": "$(id -g)"` définit l'utilisateur à utiliser dans le conteneur à être celui de la machine hôte (pour les permissions de fichiers)
- `-v "$PWD":/usr/src/myapp` va lier le dossier actuel au dossier `/usr/src/myapp` dans le conteneur
- `-w /usr/src/myapp` va définir ce dossier comme étant le dossier actuel
- `rust:bullseye` est l'image utilisée (c'est debian + les outils de développement de Rust)
- `cargo build --release` est la commande de compilation à lancer dans le conteneur

Revision #1

Created 26 April 2023 19:09:33 by SnowCode

Updated 26 April 2023 19:09:33 by SnowCode