BORDEAUX INP Enseirb-Matmeca

Électronique 3ème année

Spécialité traitement du signal et de l'image

Documentation Docker

Auteur :
M. Romain LOSILLA

Encadrant :
M.Frédéric CHATRIE

Écrit le 09/01/2025

1 Introduction

Cette documentation contient la description de mon apprentissage de l'application docker. On trouvera donc dans cette documentation l'exploration des commandes de base de docker puis la réponse à des questions posées par l'enseignant durant la séance.

2 Compréhension de l'environnement Docker

Docker est une application qui permet de conteneuriser des applications afin de les partager et de les utiliser sur n'importe quel ordinateur. Cette fonctionnalité est très utile, car elle garantit l'exécution du projet que vous avez développé, peu importe votre distribution, la version des outils utilisés ou les autres projets déjà installés sur votre machine. Pour ce faire, l'environnement Docker met à disposition plusieurs outils. Les deux outils principaux que j'ai découverts sont les conteneurs et les images. Leur rôle est simple : l'image est le fichier source qui contient toutes les bibliothèques, dépendances et fichiers dont le conteneur a besoin pour s'exécuter, tandis que le conteneur est une instance de cette image.

3 Commandes

3.1 Création et gestion des images

1. **docker build -t nom_image fichier_Dockerfile** : Permet de créer une image à partir d'un fichier Dockerfile. L'option `-t` permet de nommer l'image que nous créons.
2. **docker images** : Permet de lister les images présentes localement sur la machine.
3. **docker rmi nom_image** : Permet de supprimer une image présente localement sur la machine.

3.2 Création et gestion des conteneurs

1. **docker run [options] nom_image** : Permet de créer un conteneur à partir d'une image. Cette image peut être locale ou distante (présente sur Docker Hub). On peut utiliser de nombreux tags selon l'usage :
   — `-i` : Conteneur interactif.
   — `-t` : Alloue un terminal au conteneur.
   — `-d` : Exécute le conteneur en arrière-plan.
   — `--name` : Donne un nom au conteneur.
   — `-e` : Définit des variables d'environnement pour le conteneur.
   — `-v` : Crée un dossier partagé entre l'hôte et le conteneur.
2. **docker ps** : Permet de lister les conteneurs en cours d'exécution et leurs informations telles que le nom, l'ID et l'image utilisée. L'option `-a` permet d'afficher tous les conteneurs, même ceux arrêtés.
3. **docker exec [options] nom_conteneur commande** : Permet d'exécuter une commande dans un conteneur en cours d'exécution. On peut utiliser les options `-it` (interactif et terminal), comme expliqué précédemment.
4. **docker stop nom_conteneur** : Permet d'arrêter l'exécution d'un conteneur.
5. **docker start nom_conteneur** : Permet de lancer l'exécution d'un conteneur.
6. **docker rm [options] nom_conteneur** : Permet de supprimer un conteneur. L'option `-f` force la suppression des conteneurs encore en cours d'exécution.
7. **docker commit nom_conteneur nom_image** : Permet de sauvegarder les fichiers, dépendances et bibliothèques téléchargés sur le conteneur dans une image.
8. **docker cp fichier nom_conteneur :emplacement** : Permet de copier un fichier présent sur la machine locale vers le conteneur à l'emplacement préciser.

3.3 Résolution des exercices

3.3.1 Exercice n°1

Dans cette première situation, nous devions comprendre comment se comporte un docker run et les subtilités associées à un conteneur. Ainsi, on devait lancer un docker ubuntu, le modifier et quitter celui-ci. Ensuite,

on devait relancer ce docker et s'apercevoir que les modifications effectuées n'étaient plus disponibles sur le conteneur. En effet, lorsque l'on effectue un docker run, on relance une nouvelle instance de l'image ubuntu. Donc, les modifications effectuées dans le conteneur précédent n'y apparaissent pas. Pour que ces modifications soient effectives, nous devons créer une image de notre conteneur modifié. Pour ce faire, on utilise la commande :

$$docker\ commit\ ID\_conteneur\ ubuntu\_v1 \tag{1}$$

Ainsi, on peut relancer l'image ubuntu_v1 qui contiendra les modifications.

3.3.2 Exercice n°2

Dans cet exercice, nous devions copier un fichier présent sur la machine locale (hôte) vers un conteneur. Pour ce faire, on utilise la commande suivante :

$$docker\ cp\ test.txt\ ID\_conteneur : /home/test/ \tag{2}$$

Ainsi, une fois que cette commande est exécutée, on retrouve le fichier test.txt dans le répertoire test du conteneur.

3.3.3 Exercice n°3

Dans cet exercice, nous devions créer un répertoire partagé entre l'hôte et le conteneur. Pour ce faire, nous devons créer un volume qui correspond au dossier que l'on veut partager. On effectue cela grâce au tag -v de la commande docker run :

$$docker\ run\ -it\ -v\ chemin\_hote : chemin\_conteneur nom\_conteneur \tag{3}$$

Ainsi, une fois que cette commande est exécutée, on retrouve le dossier partagé sur le chemin précisé. On peut vérifier que ce dossier est bien partagé en créant un nouveau fichier txt sur le répertoire local et voir s'il apparaît dans le répertoire distant, ce qui est bien le cas.

3.3.4 Exercice n°3

Dans cet exercice, nous devions exporter une GUI d'une application s'exécutant dans le conteneur vers l'hôte. Pour ce faire, on donne en argument, lors de la création du conteneur, la valeur de l'affichage et on partage le dossier qui effectue l'affichage :

$$docker\ run\ -it\ -e\ DISPLAY = \$DISPLAY\ -v\ /tmp/.X11-unix : /tmp/.X11-unix\ \ nom\_conteneur \tag{4}$$

Cependant, lors de l'exécution de cette commande, l'interface graphique (GUI) ne s'affichait pas. Cela était dû au fait que l'affichage ne se faisait pas au bon endroit par défaut. J'ai donc résolu ce problème en forçant l'affichage sur `:0` avec la commande *export DISPLAY= :0*.

4 Dockerfile

Un Dockerfile est le fichier sui décrit la création d'une image Docker. Dans ce fichier, on vient décrire les opérations que l'on souhaite effectué lors de la création de notre image docker.Ces opérations peuvent être variées : mise à jour des paquets, installation de logiciel, installation de librairie, ... Dockerfile possède ces propres commandes, pour le moment j'en utilise deux principalement qui sont :

1. **FROM** Cette commande permet de décrire de quel image, l'image que nous créons va prendre source.
2. **RUN** Cette commande permet d'exécuter la commande spécifiée après.