*Installation et prise en main de DOCKER sur Ubuntu*

|  |  |
| --- | --- |
| Propriétés | Description |
| **Intitulé long** | *Installation du logiciel de conteneurisation Docker dans une VM Linux et prise en main du client Docker* |
| **Formation concernée** | BTS Services Informatiques aux Organisations |
| **Matière** | Bloc 2 SLAM – Conception et développement d’applications |
| **Présentation** | Ce document présente pas à pas l’installation du logiciel de conteneurisation Docker dans un environnement Linux (distribution Ubuntu) et la création d’un premier conteneur avec le client Docker. |
| **Transversalité** | Bloc 1 – Support et mise à disposition de services informatiques  Bloc 2 SISR – Administration des systèmes et des réseaux |
| **Compétences** | Bloc 1 : Déployer un service  Bloc 2 SISR : Installer, tester et déployer une solution d’infrastructure réseau  Bloc 2 SLAM : intégrer en continu les versions d’une solution applicative |
| **Prérequis** | Logiciel de virtualisation, commandes de base Linux et gestion de paquets, conteneurisation (voir fiche-savoir) |
| **Outils** | Docker, VirtualBox, gestionnaire de paquets apt |
| **Mots-clés** | Docker, conteneur |
| **Durée** | 3h |
| **Niveau de difficulté** | Intermédiaire |
| **Auteur(es)** | Zakari BERREMILI avec la relecture de Amal HECKER et Maelle TAURAND |
| **Version** | V7.0 |
| **Date de publication** | 25/09/2023 |
| **Contenu du package** | Machine virtuelle Ubuntu  Ce labo fait partie de l’atelier CI/CD avec Jenkins |

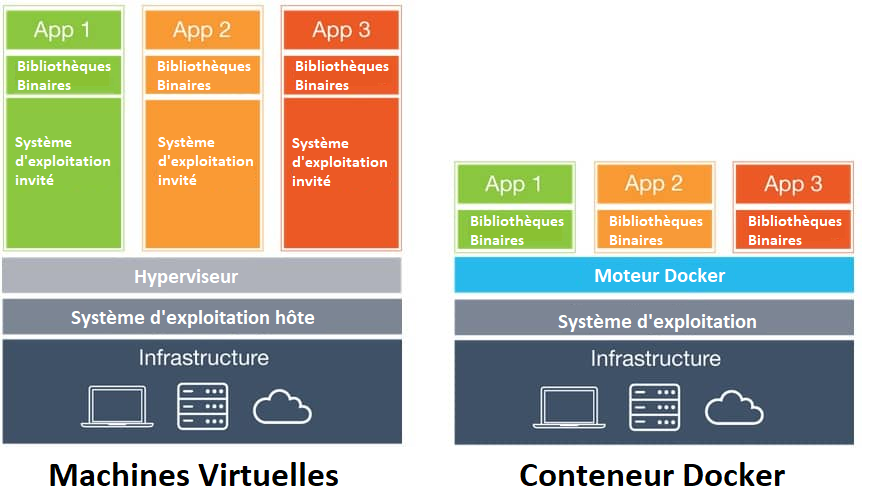


Présentation rapide de Docker

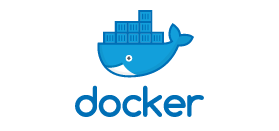
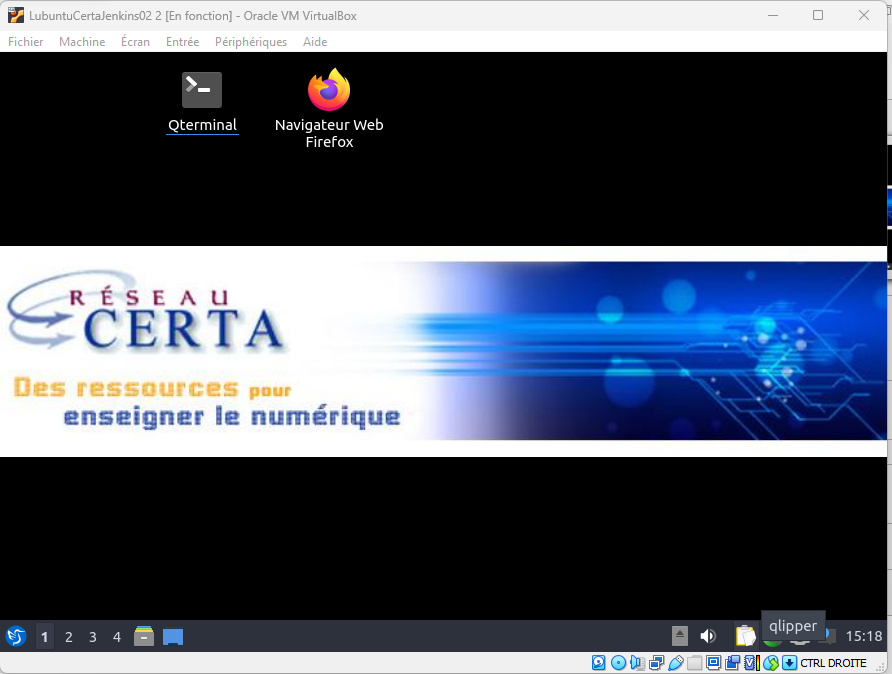
Docker est un logiciel libre Open Source qui permet d’automatiser le déploiement d’applications. Il a été développé par Solomon Hykes de la société dotCloud et a été distribué à partir de mars 2013. C’est une plateforme de virtualisation par conteneur qui va permettre de concevoir, tester et déployer des applications rapidement. Un conteneur est une unité logicielle standard qui empaquette le code et toutes ses dépendances afin que l’application s’exécute rapidement et de manière fiable d’un environnement informatique à un autre. Une image de conteneur Docker est un package logiciel léger, autonome et exécutable qui comprend tout ce qui est nécessaire pour exécuter une application : code, outils système, bibliothèques système et paramètres. Grâce à Docker, il est facile de déployer et de dimensionner les applications dans n’importe quel environnement en s’assurant que le code s’exécutera automatiquement.

C’est un outil fonctionnant en ligne de commandes qui permet de gérer les conteneurs : les démarrer et les arrêter. Les personnes concernées disposent d’un moyen fiable et peu coûteux pour développer, livrer et apporter des modifications rapidement aux applications, tout cela en les distribuant à toutes échelles.

Docker est un système d'exploitation pour conteneurs. De la même manière qu'une [machine virtuelle](https://aws.amazon.com/fr/ec2/) virtualise le matériel serveur (c’est -à-dire qu'il n'est plus nécessaire de le gérer directement), les conteneurs virtualisent le système d'exploitation d'un serveur.



**La machine virtuelle utilisée pour la séance**



Première Partie : Installation de Docker sur la machine virtuelle

Objectifs

Les objectifs de cette séquence sont :

* installer Docker dans un environnement Linux (distribution Ubuntu ou autre avec une architecture 64 bits) ;
* démarrer et arrêter le service Docker ;
* créer une image personnalisée à l’aide du fichier Dockerfile
* gérer des conteneurs : création, démarrage, arrêt et suppression.

Prérequis

Lire la section « Qu’est-ce que Docker ?» de la fiche savoir n°1 sur Docker.

Scénario

**Etape 1 : Installation de Docker**

**Etape 2 : Exécution de la commande Docker sans sudo**

**Etape 3 : Utilisation de la commande Docker**

**Etape 4 : Construction d’une image Docker**

**Etape 5 : Montage d’un fichier dans un conteneur existant**

**Etape 6 : Création d’une page dynamique**

**Etape 7 : Configuration de l'application Web pour utiliser les fichiers du site Web**

**Etape 8 : Création d’un script Bash pour créer et exécuter un conteneur Docker**

Environnement technique

Dans cette séquence, vous utiliserez une machine Ubuntu ou comme nous une machine Lubuntu demandant peu de ressources. (ID : rootsio MDP : Sio1234\*)

Étape 1 :   Installation de Docker

Tâche n°1 : Lire « Qu’est-ce que Docker ? » dans la fiche savoir

Tâche n°2 : Vérifier que Docker n’est pas installé

* Depuis un terminal, lancer la commande : docker

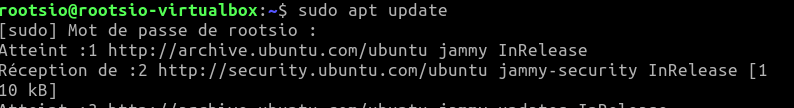
**Question** : Que constatez-vous ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Le package d'installation Docker disponible dans le référentiel officiel Ubuntu peut ne pas être la dernière version. Pour être sûr de disposer de la dernière version, nous allons installer Docker à partir du référentiel officiel Docker. Pour ce faire, nous allons ajouter une nouvelle source de paquets, ajouter la clé GPG de Docker pour nous assurer que les téléchargements sont valables, puis nous installerons le paquet.

Tâche n°3 : Mettre à jour la liste des paquets existante

* su~$ sudo apt update
* N’oubliez pas le mdp Sio1234\*



Tâche n°4 : Installer les paquets nécessaires permettant à apt d'utiliser les paquets sur HTTPS

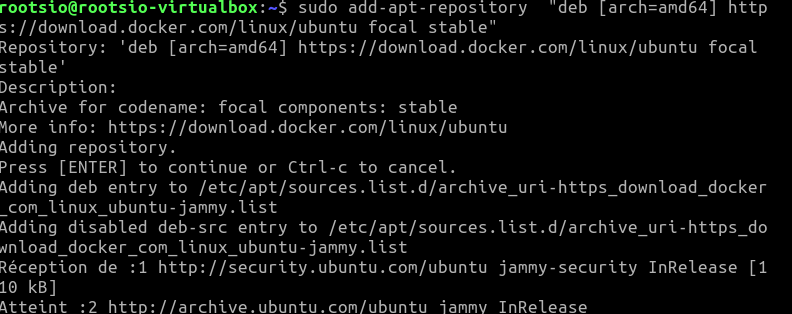
* ~$ sudo apt install apt-transport-https ca-certificates curl software-properties-common

Tâche n°5 : Ajouter la clé GPG du dépôt officiel de Docker au système

* ~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add --
* ~$ sudo cp /etc/apt/trusted.gpg /etc/apt/trusted.gpg.d

Tâche n°6 : Ajouter le référentiel Docker aux sources APT

* ~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"

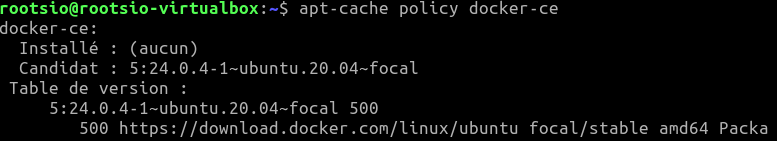


Tâche n°7 : Mettre à jour la base de données des paquets avec les paquets du référentiel Docker qui vient d'être ajouté

* ~$ sudo apt update

Tâche n°8 : Vérifier le dépôt afin de s’assurer que l’installation est réalisée à partir du dépôt Docker (et non du dépôt Ubuntu par défaut)

* ~$ apt-cache policy docker-ce



Notez que le docker-ce n'est pas installé, mais que le candidat à l'installation provient du dépôt Docker pour Ubuntu 20.10 (focal).

Tâche n°9 : Installer Docker

* ~$ sudo apt install docker-ce



Docker est maintenant installé.

Tâche n°10 : Vérifier la bonne installation de Docker

* ~$ docker --version

L'installation de Docker vous donne maintenant non seulement le service Docker (démon) mais aussi l'utilitaire en ligne de commande Docker, ou le client Docker.

Nous allons voir comment utiliser la commande docker.

Étape 2 :   Exécution de la commande Docker sans sudo

Par défaut, la commande Docker ne peut être exécutée que par l'utilisateur root ou par un utilisateur du groupe Docker, qui est automatiquement créé lors du processus d'installation de Docker. Si vous essayez d'exécuter la commande Docker sans la faire précéder de sudo ou sans être dans le groupe Docker, vous obtiendrez un message indiquant que l’utilisateur qui a lancé la commande n’a pas les permissions requises. Par exemple, essayez de lancer avec l’utilisateur etud-sio la commande : docker info, vous obtiendrez un résultat comme celui-ci :



Pour éviter de devoir préfixer toute commande docker par sudo, il faut que l’utilisateur soit dans le groupe docker. Cette commande doit être lancée pour l’administrateur du système. En tant que rootsio, il faut lancer la commande :

* ~$ sudo usermod –aG docker nomUtilisateur

Pour que « rootsio » et « etud-sio » lance la commande sans sudo, il faut taper les commandes suivantes :

* ~$ sudo usermod –aG docker rootsio

Pour vérifier que l’utilisateur a été ajouté au groupe docker, lancer la commande :

* sudo grep docker /etc/group



* ~$ exit

REDEMARRER

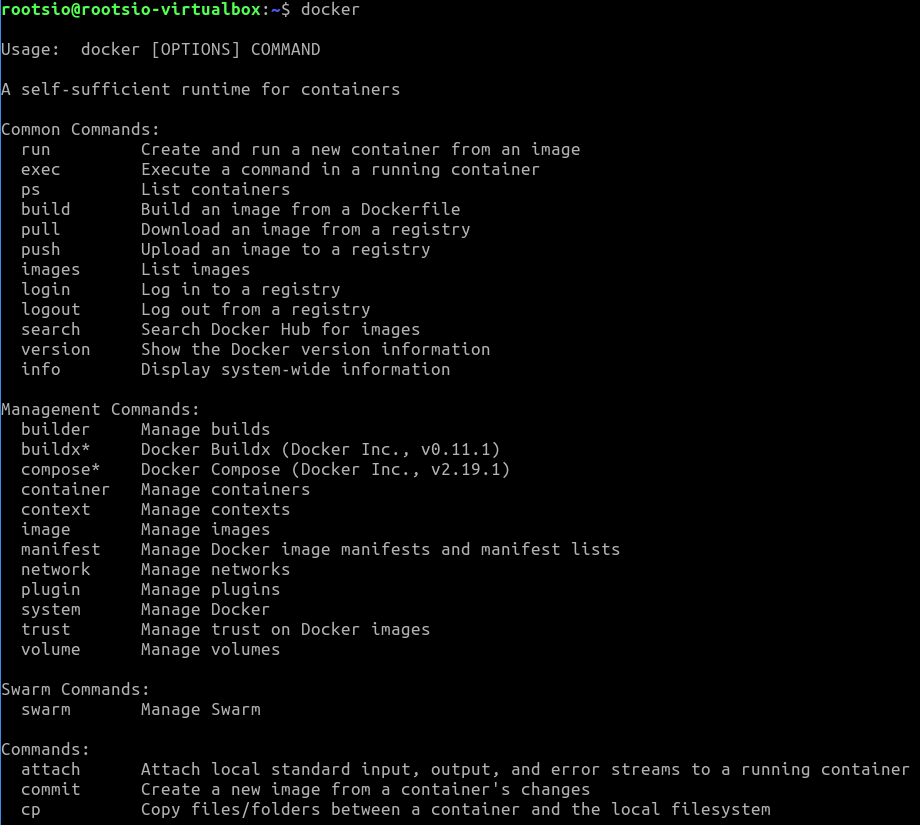
Étape 3 :   Utilisation de la commande Docker

L’utilisation du client Docker en ligne de commande est la méthode principale pour communiquer avec le démon Docker. Cela consiste à lancer la commande « docker ». La syntaxe prend cette forme :

* ~$ docker [option(s)] [command] [arguments]

Pour voir toutes les options et sous commandes disponibles, tapez :

* ~$ docker



Pour voir les informations sur Docker à l'échelle du système, utiliser :

* ~$ docker info

Pour lister les images disponibles sur la machine hôte :

* ~$ docker images

A quoi servent les images sans nom ?

Pour lister tous les conteneurs en cours :

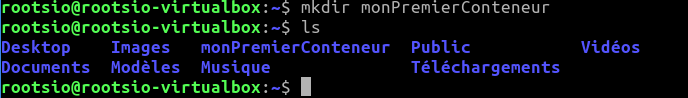
* ~$ docker ps -a

Quelle est la différence entre un conteneur Docker et une image ?

Étape 4 :   Construction du conteneur

Tâche n°1 : Créer un répertoire nommé monPremierConteneur :

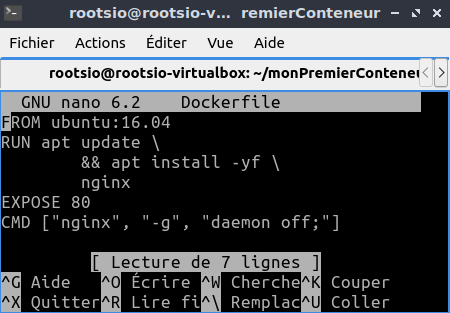
* ~$ mkdir monPremierConteneur ?



Pour ce premier conteneur nous y intégrerons un serveur web : Nginx.

La construction du conteneur nécessite un fichier appelé Dockerfile. Nous allons l’utiliser mais nous verrons plus loin et dans la fiche savoir ses caractéristiques précises. Pour l’instant nous allons créer et manipuler le conteneur.

Tâche n°2 : Créer le fichier Dockerfile dans le nouveau répertoire et insérer le code de la capture d’écran ci-dessous. Sauvegarder. (indice : touch puis nano)

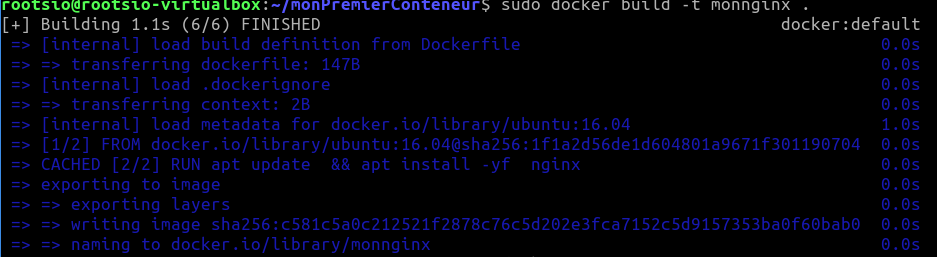


**Question Subsidiaire** : À partir des fiches « savoirs » et de vos recherches, noter le rôle des instructions FROM, RUN, EXPOSE et CMD.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tâche n°3 : Indiquer à Docker de construire le conteneur à partir du fichier Dockerfile (« le builder ») à l’aide de la commande build (Remarque : le point « . » signifie que l’on cherche Dockerfile dans le répertoire courant)

* ~$ docker build -t monnginx .



Tâche n°4 : Exécuter le conteneur docker en appelant la commande run.

Celle-ci va réaliser une curdu port exposé 8000 vers le port 80 de nginx dans le conteneur, le port 80 de votre machine hôte pouvant de ce fait être également utilisé. Pour le conteneur ce sera 80, mais pour y accéder depuis l’hôte ce sera 8000 :

* ~$ docker run -d -p 8000:80 monnginx



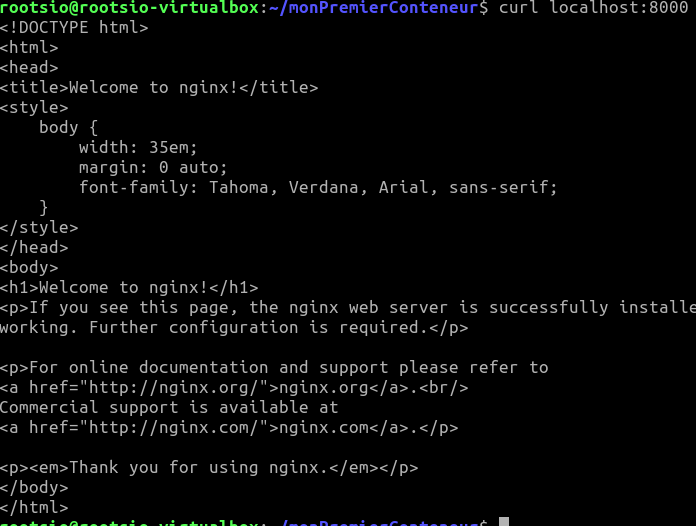
**Question** : Que signifient l’option « -t » et le point « . » dans la commande :

~$ docker build -t monnginx .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tâche n°5 : Tester que le serveur nginx est fonctionnel en utilisant la commande curl.

* ~$ curl localhost:8000



Tâche n°6 : Tester que le serveur nginx est fonctionnel depuis le navigateur, ici Firefox

* interroger l’URL http://localhost:8000



**Question** : Que se passe-t-il si on accède à http://localhost:80 depuis votre navigateur hôte du conteneur ? Pourquoi ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Question** : Où se trouve le fichier .html page d’index associée ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Question** : Peut-on accéder à cette page html avec une autre URL ? Une autre adresse IP ? Un autre port ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Indice : Utilisez la commande ip address et/ou utilisez la commande docker inspect $ID\_court\_du conteneur

Tâche n°7 : A l’aide des commandes appropriées, arrêter et supprimer le conteneur :

* docker ps -a
* docker stop arg1
* docker rm arg1

arg1 est à remplacer par l’identifiant du conteneur (Container ID affiché à l’aide de la commande docker ps –a qui correspond à notre conteneur monnginx).

Étape 5 :   Montage d’un fichier dans un conteneur existant

Le répertoire dans lequel Nginx va, par défaut, chercher les pages html est /var/www/html.

Tâche n°1 : Créer un fichier index.html dans le répertoire monPremierConteneur et insérer le code suivant : (indice : touch puis nano)

<HTML>

<HEAD>

<TITLE> Mon laboratoire docker pour le BTS SIO </TITLE>

</HEAD>

<BODY>

<H1> Le BTS SIO : Un passeport pour mon avenir ! </H1>

</BODY>

<HTML>

Tâche n°2 : Monter le fichier dans votre conteneur :

* docker run -d -p 8000:80 -v $PWD/index.html:/var/www/html/index.html monnginx

**Question** : Quels sont les rôles des options (-d, -p, -v)  ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tâche n°3 : Vérifier le bon fonctionnement. Votre résultat doit être similaire à la capture d’écran ci-dessous.



Tâche n°4 : Afin de modifier la couleur de fond de la page web, introduire une nouvelle ligne dans le fichier index.html de votre hôte :

<HTML>

<HEAD>

<TITLE> Mon laboratoire docker pour le BTS SIO </TITLE>

<LINK rel="stylesheet" href="style.css" />

</HEAD>

<BODY>

<H1> Le BTS SIO : Un passeport pour mon avenir ! H1>

</BODY>

<HTML>

Tâche n°5 : Créer le fichier style.css contenant le script suivant

body {background: lightsteelblue;}

Tâche n°6 : Après avoir arrêté et supprimé les précédents conteneurs, relancer monnginx en montant les deux fichiers précédemment créés.

* docker run -d -p 8000:80 -v $PWD/index.html:/var/www/html/index.html -v $PWD/style.css:/var/www/html/style.css monnginx

Tâche n° 7 : Vérifier le bon fonctionnement. Votre résultat doit être similaire à la capture d’écran ci-dessous.



Étape 6 : Création d’une page dynamique

Dans cette étape, vous utiliserez un script Bash pour automatiser la création d'une application Web à l'intérieur d'un conteneur Docker

Pour les scénarii de tests, lorsqu’il sera nécessaire de tester l’accès à l’application Web, vous pourrez utiliser les méthodes suivantes :

* méthode n°1 : interroger l’URL de l’application depuis le navigateur Web
* méthode n°2 : utiliser la commande curl URL

Avant de pouvoir lancer une application dans un conteneur Docker, nous devons d'abord la créer ou en utiliser une existante. Dans cette étape, vous allez créer un script Python très simple qui affichera l'adresse IP du client lorsqu’un utilisateur visite la page Web. Pour cela, on va installer et se servir du framework Flask

Les développeurs d'applications Web Python utilisent généralement un framework. Un framework est une bibliothèque de code qui facilite la création d'applications Web fiables, évolutives et maintenables aisément pour les développeurs. Flask est un framework d'application Web écrit en Python. D'autres frameworks incluent Tornado et Pyramid.

Vous allez utiliser cette infrastructure pour créer un exemple d'application Web. Flask reçoit des demandes, puis fournit une réponse à l'utilisateur dans l'application Web. Ceci est utile pour les applications Web dynamiques car il permet l'interaction entre les utilisateurs et le serveur. Dans notre exemple ci-dessous, l'application Web affichera dynamiquement l'adresse IP du client.

**Remarque :** La compréhension des fonctions, des méthodes et des bibliothèques de Flask dépasse le cadre de ce cours. Il est utilisé dans ce laboratoire pour montrer à quelle vitesse vous pouvez mettre en service une application Web. Si vous voulez en savoir plus, recherchez sur Internet plus d'informations et didacticiels à propos du framework Flask.

Tâche n°1 : Depuis un terminal, importer flask.

* Eventuellement à ~$ sudo apt install python3-pip
* ~$ pip3 install flask

Tâche n°2 : Créer et éditer le fichier sample\_app.py situé dans un répertoire ~/sample-app à créer également. Ajouter le code présenté ci-dessous.

from flask import Flask

from flask import request

sample = Flask(\_\_name\_\_)

@sample.route("/")

def main():

return "You are calling me from " + request.remote\_addr + "\n"

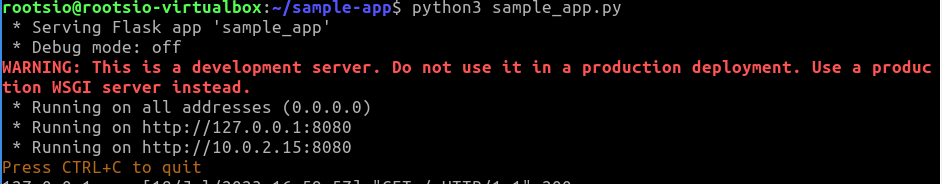
if \_\_name\_\_ == "\_\_main\_\_":

sample.run(host="0.0.0.0", port=8080)

Tâche n°3 : Lancer le script python en ligne de commande pour lancer le serveur. Veiller à bien le laisser en service quitte à changer de fenêtre de terminal si besoin.

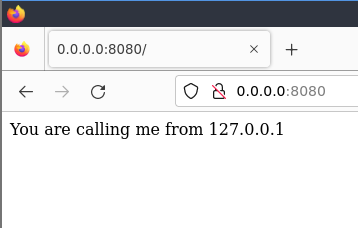
* ~$ python3 sample\_app.py

m



Tâche n°4 : Vérifier que le bon accès au serveur.

* Méthode n°1 : interroger l’URL 0.0.0.0:8080 depuis le navigateur



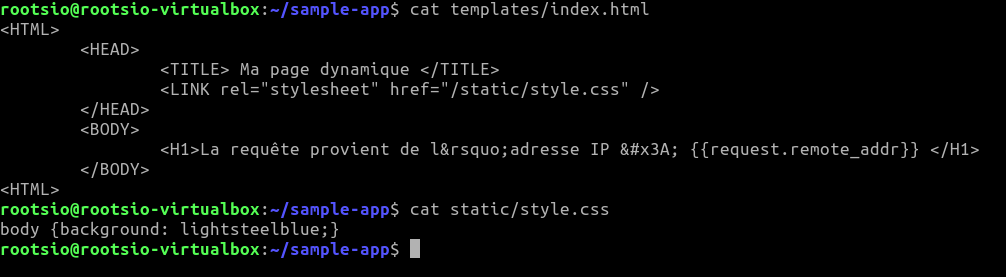
* Méthode n°2 : utiliser la commande curl depuis un autre onglet ou terminal

Tâche n°5 : Arrêter le serveur (CTRL + C depuis le terminal où le serveur est en cours d'exécution)

Étape 7 : Configuration de l'application Web pour utiliser les fichiers du site Web

Dans cette étape, vous allez créer l'exemple d'application Web pour inclure une page index.html et une spécification style.css.

Tâche n°1 : Créer et compléter un fichier index.html dans sample-app/templates et un fichier style.css dans sample-app/static tel que présenté ci-dessous.



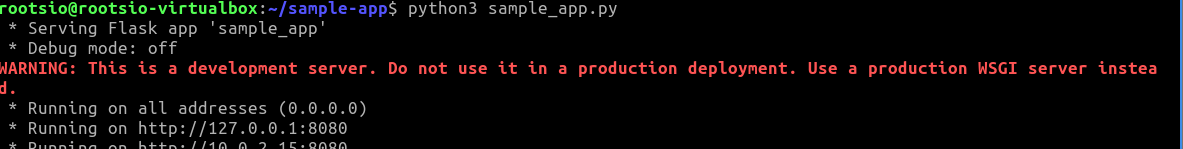
Maintenant que vous avez les fichiers de base du site Web, vous devez mettre à jour le fichier sample\_app.py afin qu'il affiche le fichier index.html au lieu de simplement renvoyer les données.

La génération de contenu HTML à l'aide du code Python peut s'avérer lourde, en particulier lors de l'utilisation d'instructions conditionnelles ou de structures répétitives. Le fichier HTML peut être rendu automatiquement dans Flask à l'aide de la fonction render\_template. Cela nécessite l'importation de la méthode render\_template à partir de la bibliothèque flask et l'édition dans la fonction return.

Tâche n°2 : apporter les modifications mises en surbrillance à votre script.

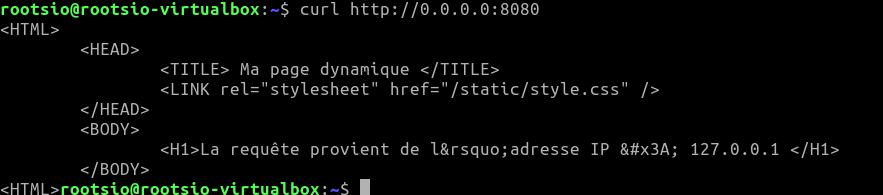
|  |
| --- |
| from flask import Flask  from flask import request  from flask import render\_template  sample = Flask(\_\_name\_\_)  @sample.route("/")  def main():  return render\_template("index.html")    if \_\_name\_\_ == "\_\_main\_\_":  sample.run(host="0.0.0.0", port=8080) |

Tâche n°3 : Enregistrer et exécuter votre script.



Remarque : Si vous avez obtenu la sortie Traceback et une erreur avec un message du type **OserError : [Errno 98]** Adresse déjà utilisée, alors vous n'avez pas arrêté votre serveur précédent. Retournez à la fenêtre du terminal où ce serveur est en cours d'exécution et appuyez sur CTRL+C pour terminer le processus du serveur. Réexécutez votre script.

Tâche n°4: Tester l’accès à l’application web depuis l’URL 0.0.0.0 :8080



Tâche n°5 : arrêter le serveur.



Étape 8 : Création d’un script Bash pour créer et exécuter un conteneur Docker

Dans cette étape, vous allez créer un script Bash qui effectue les actions suivantes afin de créer et d’exécuter un conteneur Docker :

* + - 1. Créer des répertoires temporaires pour stocker les fichiers du site Web.
      2. Copier les répertoires du site Web et sample\_app.py dans le répertoire temporaire.
      3. Créer un Dockerfile.
      4. Construire (build) le conteneur Docker.
      5. Démarrer le conteneur et vérifier qu'il est en cours d'exécution.

Tache n°1 : Créer et éditer le script Bash sample-app.sh présent dans le répertoire ~/sample-app. Ajouter le 'she-bang' et saisir les commandes suivantes :

#!/bin/bash (1)

(1)

mkdir tempdir (1)

mkdir tempdir/templates (1)

mkdir tempdir/static (1)

cp sample\_app.py tempdir/. (2)

cp -r templates/\* tempdir/templates/. (2)

cp -r static/\* tempdir/static/. (2)

echo "FROM python" >> tempdir/Dockerfile (3)

echo "RUN pip install flask" >> tempdir/Dockerfile (4)

echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile (5)

echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile (5)

echo "COPY sample\_app.py /home/myapp/" >> tempdir/Dockerfile (5)

echo "EXPOSE 8080" >> tempdir/Dockerfile (6)

echo "CMD python3 /home/myapp/sample\_app.py" >> tempdir/Dockerfile (7)

cd tempdir (8)

docker build -t sampleapp . (8)

docker run -t -d -p 8080:8080 --name samplerunning sampleapp (9)

docker ps -a (10)

.

* + - 1. Pour créer une structure de répertoire avec tempdir comme dossier parent.
      2. Pour copier le répertoire du site Web et le script dans tempdir.
      3. Pour créer un Dockerfile dans le tempdir. Ce Dockerfile sera utilisé pour construire le conteneur.
      4. Pour installer Flask dans le conteneur à l’aide de la commande Docker RUN.
      5. Pour introduire les commandes Docker COPY afin d’ajouter un répertoire dans le conteneur Docker. Vous allez créer /home/myapp en tant que répertoire parent dans le conteneur Docker. En plus de copier le fichier sample\_app.py dans le fichier Dockerfile, vous copierez également le fichier index.html à partir du répertoire templates et le fichier style.css à partir du répertoire static.
      6. Pour exposer le port 8080 à utiliser par le serveur Web.
      7. Pour exécuter le script Python.
      8. Pour basculer vers le répertoire tempdir et construire le conteneur Docker. La commande **docker build -t** vous permet de spécifier le nom du conteneur et le point (.) indique que vous voulez que le conteneur soit construit dans le répertoire courant.
      9. Pour démarrer le conteneur.
      10. Pour afficher tous les conteneurs Docker en cours d'exécution. Cette commande sera la dernière exécutée par le script Bash.

Tâche n°2 : Lancer le script Bash à partir de la ligne de commande après l’avoir rendu exécutable.



Résultat : Après avoir créé les répertoires tempdir, le script exécute les commandes pour construire le conteneur. Notez que l'étape 7/7 de la sortie exécute le fichier sample\_app.py qui crée le serveur Web. Notez également l'ID du conteneur. Vous verrez cela dans l'invite de commande Docker plus tard dans le laboratoire.

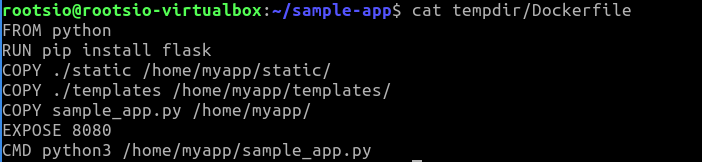


Tâche n°3 : Examiner l’environnement créé pour le conteneur Docker en cours d'exécution et l'application Web.

Remarque : La création des répertoires tempdir n'est pas affichée dans la sortie du script. Vous pouvez ajouter des commandes echo pour imprimer les messages lorsqu'ils sont créés avec succès. Vous pouvez également vérifier qu'ils sont là avec la commande ls. N'oubliez pas que ce répertoire contient les fichiers et dossiers utilisés pour construire le conteneur et lancer l'application Web. Ce n'est pas le conteneur qui a été construit.



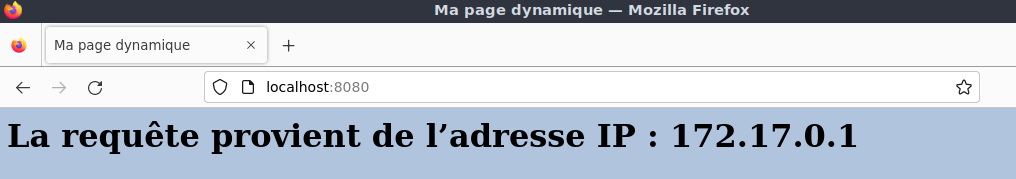
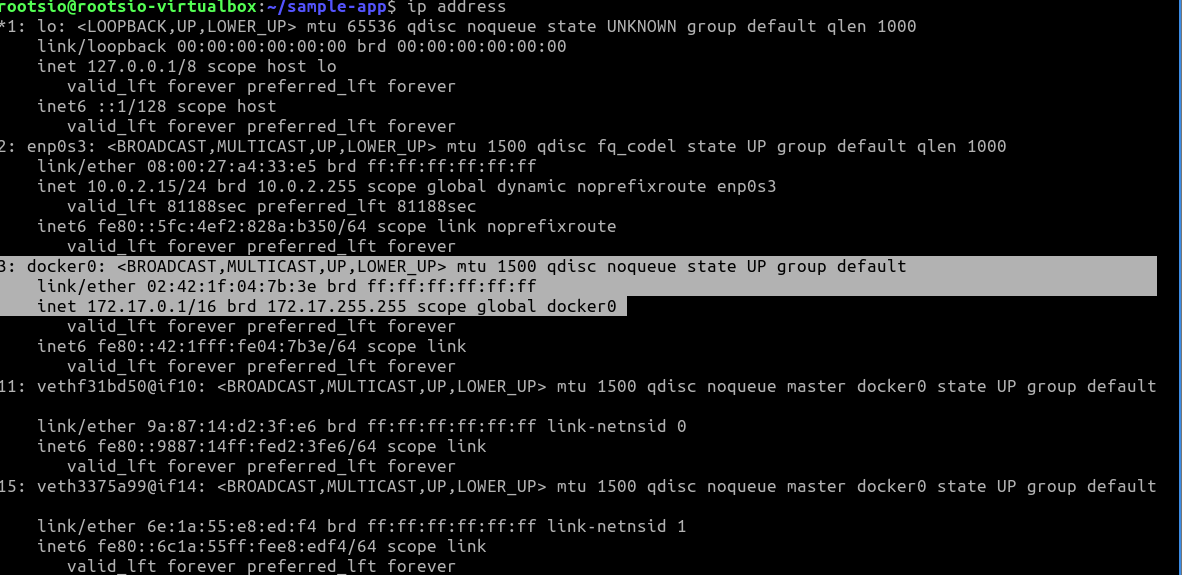
Tâche n°4 : Ouvrez le fichier Dockerfile pour voir à quoi il ressemble dans sa forme finale sans les commandes echo.



Le conteneur Docker crée sa propre adresse IP à partir d'un espace d'adressage réseau privé.

Tâche n°5 : Tester l'accès à l'application Web depuis l'URL localhost:8080

**Question** : le résultat est-il en cohérence avec le résultat de la commande **ip address** ?



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Exercice** : Accéder au conteneur en cours d'exécution et l’explorer.

Tâche n°1 : Accéder au conteneur en cours d'exécution et l’explorer.

Un conteneur Docker est un moyen d'encapsuler tout ce dont vous avez besoin pour exécuter votre application afin qu'elle puisse être facilement déployée dans une variété d'environnements.

Pour accéder au conteneur en cours d'exécution, entrez la commande **docker exec -it** en spécifiant le nom du conteneur en cours d'exécution (samplerunning) et que vous voulez un Shell Bash (/bin/bash). L'option **-i** indique que vous voulez qu'elle soit interactive et l'option **-t** spécifie que vous voulez accéder au terminal. L'invite change et devient **root@containerID**. Votre identifiant de conteneur sera différent de celui indiqué ci-dessous. Notez que l'ID du conteneur correspond à l'ID affiché dans la sortie du **docker ps -a**.

Vous êtes maintenant en accès root pour le conteneur Docker samplerunning.

Tâche n°2 : Explorer la structure des répertoires au niveau racine du conteneur Docker à l’aide de la commande ls

Rappelez-vous que dans votre script Bash, vous avez ajouté des commandes dans le fichier Dockerfile qui ont copié vos répertoires et fichiers d'application dans le répertoire home/myapp.

Tâche n°3 : Entrer à nouveau la commande ls pour ce dossier afin d'afficher votre script sample\_app.py et vos répertoires. Répéter l’opération pour examiner le contenu des sous-répertoires /bin et /etc

Tâche n° 4 : Quitter le conteneur Docker

Tâche n°5 : A l’aide des commandes Docker appropriées, arrêter et supprimer le conteneur Docker.