

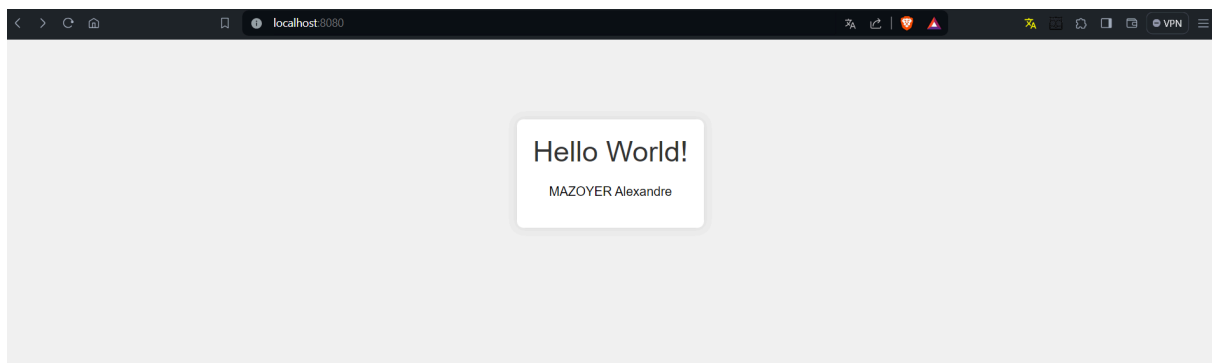
# TP1 Docker - MAZOYER Alexandre

## 3- Installation de l'image Apache HTTP Server

1. **Ouverture de Visual Studio Code :**
    - Ouvrez Visual Studio Code.
    - Accédez à la vue Terminal (Vue > Terminal).
  2. **Pull de l'image Apache HTTP Server depuis Docker Hub :**
    - Utilisez la commande suivante pour télécharger l'image Apache HTTP Server depuis Docker Hub : `docker pull httpd`
- Cette commande récupère la dernière version de l'image Apache HTTP Server depuis le référentiel Docker Hub.

## Exécution du Conteneur Apache HTTP Server

1. **Exécution du Conteneur** J'ai utilisé la commande suivante pour démarrer un conteneur Apache HTTP Server en exposant le port 8080 :
  - `docker run -d -p 8080:80 -v C:\Users\alex\Desktop\COUR\docker\Docker-github\Docker\html:/usr/local/apache2/htdocs --name httpd1 httpd`
2. **Vérification de l'Exécution :**



## Vérification des Images Docker

1. **Vérification des Images :**
  - J'ai utilisé la commande suivante pour afficher la liste des images Docker sur votre système :

`docker images`

```
PS C:\Users\alex\Desktop\COUR\docker\Docker-vs> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
httpd         latest    2776f4da9d55   5 weeks ago    167MB
PS C:\Users\alex\Desktop\COUR\docker\Docker-vs> 
```

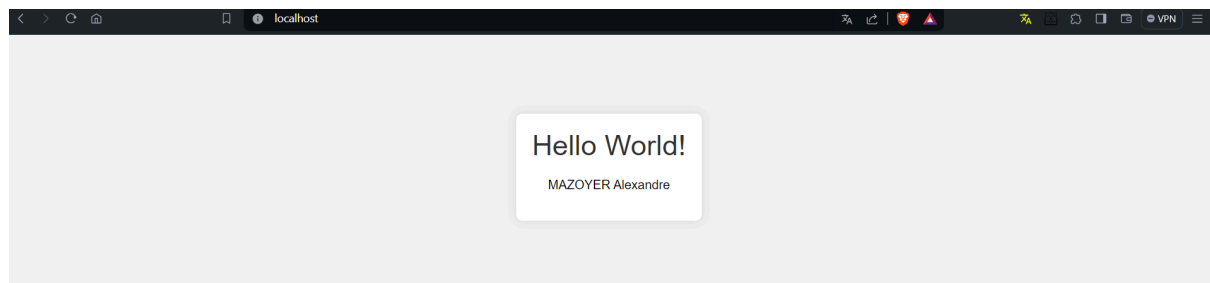
- J'ai utilisé la commande suivante pour afficher la liste des containers Docker sur votre système :

```
C:\Users\alex>docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
23e5202b41dc   httpd         "httpd-foreground"      7 seconds ago Up 6 seconds   0.0.0.0:80->80/tcp       epic_euler
b9def34f0fd2   httpd:latest  "httpd-foreground"      30 minutes ago Up 30 minutes   0.0.0.0:8080->80/tcp      localhost
```

d. Maintenant, on crée le container httpd2 sur le port 80 et on lie le fichier html de httpd1

```
PS C:\Users\alex\Desktop\COUR\docker\Docker-github> docker run -d -p 80:80 -v C:\Users\alex\Desktop\COUR\docker\Docker-github\html:/usr/local/apache2/htdocs --name httpd2 httpd
d52d719d10d5ce197f82be5f0e63bf556a0b96baffda27595601405aaa999001
PS C:\Users\alex\Desktop\COUR\docker\Docker-github> 
```

Voici le résultat :



La page a bien été liée.

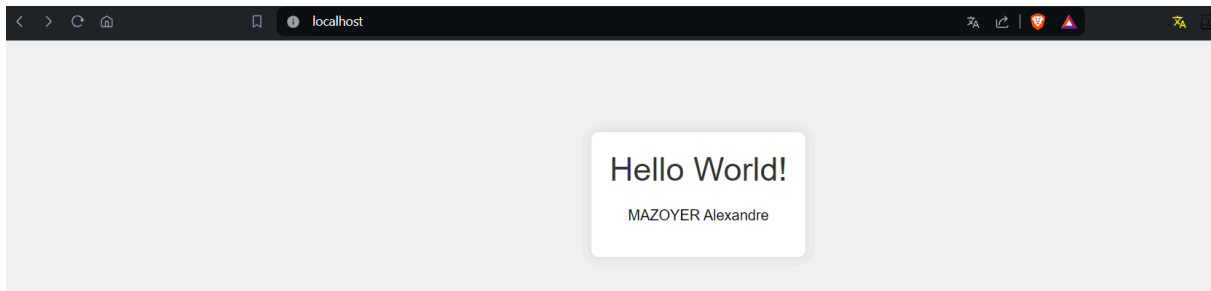
e. supprimer le container :

```
PS C:\Users\alex\Desktop\COUR\docker\Docker-github> docker stop httpd2
httpd2
PS C:\Users\alex\Desktop\COUR\docker\Docker-github> docker rm httpd2
httpd2
```

f. Docker cp : commande :

```
PS C:\Users\alex\Desktop\COUR\docker\Docker-github> docker run -d -p 80:80 --name httpd2 httpd
5671e547cc8263baf59a0d4ca8584117b10b7c54b778b2aa34c3b35d64e8b631
PS C:\Users\alex\Desktop\COUR\docker\Docker-github> docker cp C:\Users\alex\Desktop\COUR\docker\Docker-github\html\index.html httpd2:/usr/local/apache2/htdocs
Successfully copied 2.56kB to httpd2:/usr/local/apache2/htdocs
```

Résultat :



On observe le même résultat que précédemment, il s'agit de 22 manières différentes de procédés, la première est plus efficace est rapide (une commande pour : création du container + copie du fichier html).

4-

- Configuration du fichier Dockerfile:

```

1  # Utilisez l'image officielle d'Apache HTTP Server
2  FROM httpd:latest
3
4  # Copiez votre fichier index.html dans le répertoire des documents d'Apache
5  COPY C:\Users\alex\Desktop\COUR\docker\Docker-github\html/index.html /usr/local/apache2/htdocs/
6
7  # Exposez le port 80 pour permettre l'accès à Apache
8  EXPOSE 80
```

b. Exécuter cette nouvelle image pour servir ./html/index.html:

```

PS C:\Users\alex\Desktop\COUR\docker\Docker-github> docker build -t httpd1 .
2024/02/21 10:46:56 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 0.1s (2/2) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 2B
=> [internal] load .dockerignore
=> => transferring context: 2B
ERROR: failed to solve: failed to read dockerfile: open /var/lib/docker/tmp/buildkit-mount1120860013/Dockerfile: no such file or directory
PS C:\Users\alex\Desktop\COUR\docker\Docker-github> docker run -d -p 8080:80 --name httpd1 httpd
7c8e76e0c8f716ded237bab8950510d826d9b1c132f26eada0e67fcd77ded5eb
PS C:\Users\alex\Desktop\COUR\docker\Docker-github>
```

## Montage de Volumes (option -v) :

Avantages :

- Les modifications apportées dans le répertoire local sont **immédiatement** reflétées dans le conteneur, éliminant le besoin de reconstruire l'image.
- Particulièrement pratique pendant le développement, permettant l'édition du code sans besoin de reconstruction.

Inconvénients :

- Requiert l'accès au répertoire local au moment de l'exécution, ce qui peut poser des problèmes dans certains scénarios de déploiement.

### **Copie (COPY dans le Dockerfile ou docker cp) :**

Avantages :

- Plus portable, car le contenu est directement inclus dans l'image, rendant l'application autonome.
- Utile pour la distribution d'applications autonomes sans dépendance directe sur le système de fichiers local.

Inconvénients :

- Nécessite la reconstruction de l'image pour prendre en compte les modifications dans le contenu, ce qui peut être un processus plus lourd.

Le choix entre le montage de volumes et la copie dépend des besoins et du contexte d'utilisation de notre application.

Si la flexibilité et la réactivité aux changements sont cruciales, notamment pendant le développement, le montage de volumes peut être préférable. En revanche, si on a besoin d'une solution autonome et portable, la copie du contenu dans l'image peut être la meilleure option.

## **5. Utiliser une base de données dans un conteneur Docker**

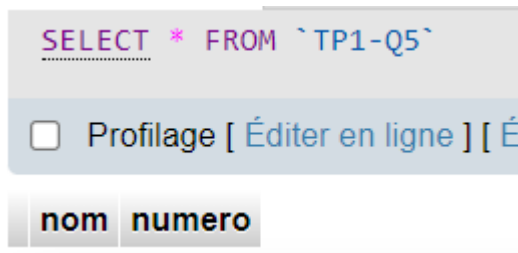
**a. Récupérer les images MySQL (ou MariaDB) et phpMyAdmin depuis le Docker Hub:**

- `docker pull mysql`
- `docker pull phpmyadmin/phpmyadmin`

**b. Exécuter 2 conteneurs à partir des images:**

- **Créez un réseau Docker docker network + exécution des conteneurs**
  - `create mon_network`
  - `docker run -d --name mysql_container --network mon_network -e MYSQL_ROOT_PASSWORD=root mysql`
  - `docker run -d --name phpmyadmin_container --network mon_network -p 8081:80 -e PMA_ARBITRARY=1 -e PMA_HOST=mysql_container phpmyadmin/phpmyadmin`

Création d'une table :



The screenshot shows a database query interface. At the top, a SQL query is entered: `SELECT * FROM `TP1-Q5``. Below the query, there is a checkbox labeled "Profilage" followed by two links: "[ Éditer en ligne ]" and "[ É" (partially visible). At the bottom, there are two columns labeled "nom" and "numero" in a table header format.

## 6. Utilisation de docker-compose.yml

a. À quoi sert la commande **docker-compose** par rapport à **docker run** ?

- La commande **docker-compose** est utilisée pour définir et exécuter des applications multi-conteneurs. Elle est basée sur un fichier de configuration YAML appelé **docker-compose.yml** qui décrit les services, les réseaux et les volumes nécessaires pour l'application.
- **docker-compose** simplifie le processus de gestion des applications complexes avec plusieurs conteneurs en les définissant dans un seul fichier de configuration.

b. Commande pour lancer tous les conteneurs définis dans le fichier YAML et pour les arrêter :

- Pour lancer tous les conteneurs définis dans le fichier **docker-compose.yml** :
  - `docker-compose up -d`
- Pour arrêter tous les conteneurs définis dans le fichier **docker-compose.yml** :
  - `docker-compose down`

c. Exemple de fichier **docker-compose.yml** pour servir une base de données et phpMyAdmin :

```

version: '3'

services:
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
    networks:
      - mon_network

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    environment:
      PMA_HOST: db
      PMA_ARBITRARY: 1
    ports:
      - "8081:80"
    networks:
      - mon_network

networks:
  mon_network:

```

- phpMyAdmin est configuré pour se connecter au service MySQL avec l'alias db.
- Les ports 8081 de l'hôte sont mappés sur le port 80 du conteneur phpMyAdmin.
- Ce fichier définit deux services : *db* pour MySQL (ou MariaDB) et *phpmyadmin* pour phpMyAdmin.

