

Rapport de TD Docker

Exercice 2 : BD

1. Récupérer une image postgres (SGBD)

`docker pull postgres`

2. Lancer un conteneur BD (En suivant les indications disponibles sur le lien Docker Hub de l'image)

`docker run --name Mysql -p 5432:5432 -e POSTGRES_PASSWORD=MyPassword -d postgres`

```
(base) sylvain@sylvain-VirtualBox:~$ sudo docker run --name Mysql -p 5432:5432 -e POSTGRES_PASSWORD=MyPassword -d postgres
df7b471cb061097ffaa7de42beeac49b1ff09408b637e68eb4b577cc8b7cd0f
docker: Error response from daemon: driver failed programming external connectivity on endpoint Mysql (ec08011a16b548857fe9c6ac4c8b218f5b799f0f6b1d4e8666b88b3fd5660ff1): Error starting userland proxy: listen tcp4 0.0.0.0:5432: bind: address already in use.
```

Comme le port 5432 est occupé, il est remplacé par le port 5433

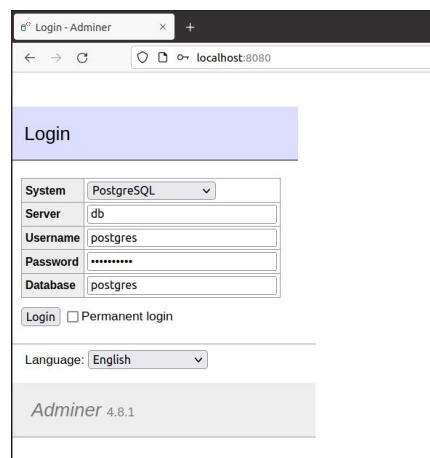
3. Lancer un conteneur adminer (database manager)

`docker pull adminer`

```
(base) sylvain@sylvain-VirtualBox:~$ sudo docker run --name myadminer --link Mysql:db -p 8080:8080 adminer
```

4. Connecter à votre base de données via adminer :

- `http://Localhost :8080` ou `http://adresse-ip-de-la-machine :8080`
- system : postgres ; server : ip de votre machine ; username : postgres ; password : MyPassword; Database : postgres



5. Utiliser adminer pour créer, modifier votre base de données

- Créer une table « user »
- Ajouter des lignes dans cette table
- Afficher le contenu de cette table

Table: user

Select data Show structure Alter table New item

Column	Type	Comment
id	integer	
username	character(64)	

6. Afficher les logs, la consommation des ressources de vos conteneurs a.
Docker logs et docker stats

`docker logs Mysql`

`docker stats Mysql`

7. Donner le fichier docker-compose pour faciliter la configuration de toute cette partie

```
version: '3.1'

services:
  db:
```

```
image: postgres
restart: always
ports :
  -5432:5432
environment:
  POSTGRES_PASSWORD: example

adminer:
  image: adminer
  restart: always
  ports:
    - 8080:8080
```

Copier le code fourni dans le fichier config.yml et exécuter cette ligne dans le terminal:

`docker-compose -f config.yml up`

Exercice 3 : proxy web

1. Proposer une solution de reverse proxy via docker-compose qui permet de
a. créer deux siteweb (simples : web1 et web2) lancés sur deux serveurs web (Nginx ou apache) différents

Tout d'abord, installer l'image nginx:

`docker pull nginx`

Copier le code ci-dessous dans le fichier creation_site.yml

```
Services:

web1:
  image: nginx
  volumes:
    - ./web1: /usr/share/nginx/html
Web2
  image: nginx
  volumes:
    - ./web2: /usr/share/nginx/html
.....
```

Exécuter cette ligne dans le terminal:

```
docker-compose -f creation_site.yml up
```

Après, deux répertoire web1 et web2 ont été crée dans le chemin courant. Créer des fichiers index.html dans ces deux répertoire et enregistrer ce que on veut afficher sur la page de web site

Reexécuter cette ligne dans le terminal pour mettre à jour:

```
docker-compose -f creation_site.yml up
```

2. Créer un serveur de proxy qui va rediriger les requête vers le site 1 ou site 2 selon l'url : site1.localhost ou site2.localhost

Créer un répertoire proxy-conf, créer un fichier default.conf dans le répertoire et stocker les codes fournis

```
server {
    listen 80;
    server_name site1.localhost;

    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_buffering off;
        proxy_request_buffering off;
        proxy_http_version 1.1;
        proxy_intercept_errors on;
        proxy_pass http://web1:80;
    }
}

server {
    listen 80;
    server_name site2.localhost;

    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_buffering off;
        proxy_request_buffering off;
        proxy_http_version 1.1;
        proxy_intercept_errors on;
        proxy_pass http://web2:80;
    }
}
```

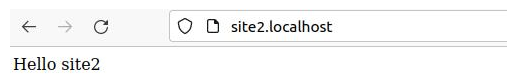
Copier les codes ci-dessous dans un fichier creation_proxy.yml dans le répertoire Ex3 où se trouve le répertoire proxy-conf, web1 et web2.

```
.....
proxy
  image: nginx
  volumes:
    - ./proxy-conf:/etc/nginx/conf.d
  Ports:
```

Exécuter cette ligne au répertoire Ex3 dans un autre terminal:

`docker-compose -f creation_proxy.yml up`

Après visiter `http://site1.localhost` et `http://site2.localhost` pour voir les résultats.



Exercice 4 :

Tester le service suivant :

Créer un fichier `docker-compose.yml`

```
1 version: "3.3"
2
3 services:
4   traefik:
5     image: "traefik:v2.0"
6     container_name: "traefik"
7     command:
8       - "--log.level=DEBUG"
9       - "# Traefik will listen on port 8080 by default for API request."
10      - "--api.insecure=true"
11      - "# Enabling docker provider"
12      - "--providers.docker=true"
13      - "# Do not expose containers unless explicitly told so"
14      - "--providers.docker.exposedbydefault=false"
15      - "# Traefik will listen to incoming request on the port 80 (HTTP)"
16      - "--entrypoints.web.address=:80"
17    ports:
18      - "80:80"
19      - "8080:8080"
20    volumes:
21      - "/var/run/docker.sock:/var/run/docker.sock:ro"
22
23 whoami:
24   image: "traefik/whoami"
25   container_name: "simple-service"
26   labels:
27     - "traefik.enable=true"
28     - "# Explicitly tell Traefik to expose this container"
29     - "# The domain the service will respond to"
30     - "traefik.http.routers.whoami.rule=host(`localhost`)"
31     - "traefik.http.routers.whoami.entrypoints=web"
32     - "traefik.http.routers.whoami.entrypoints=web"
```

et
/hoami