COMPTE RENDU PROJET DOCKER

L'objectif de ce projet est de mettre en place une infrastructure docker pour exécuter une application avec une API Flask et une interface web en HTML+PHP.

Pour réaliser le projet nous allons utiliser une VM centos avec docker installé (qui se trouve sur notre esxi) et on va se connecter par ssh sur notre machine physique via le powershell.

Voici ci-dessous les étapes que nous allons suivre :

Installation de docker :

```
[root@localhost student_list]# yum install -y docker-ce docker-ce-cli containerd.io
```

Vérification de la version :

```
[root@localhost student_list]# docker --version
Docker version 28.0.1, build 068a01e
[root@localhost student_list]#
```

Installation docker compose :

```
[root@localhost astou]# curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -m)" [root@localhost astou]# curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100     9  100     9    0     0      8      0  0:00:01  0:00:01 --:--:--     8
[root@localhost astou]# chmod +x /usr/local/bin/docker-compose
```

Clonage du dépôt git:

Pour faire le clonage nous allons d'abord créer un dossier student_list et se palcer sur ce répertoire pour faire le clonage.

Cette commande exécute un nouveau conteneur Docker (docker run), spécifie l'image docker à utiliser (alpine/git), clone le dépôt dans le répertoire courant (clone https://github.com/guissepm/student-list.git) et /app spécifie le répertoire de destination à l'intérieur du conteneur où le dépôt doit être cloné. Puisque /app est le montage de volume, les fichiers clonés seront sauvegardés dans votre répertoire courant sur la machine hôte.

```
[root@localhost student_list]# docker run --rm -v "$PWD:/app" alpine/git clone https://github.com/guissepm/student-list.git /app
Cloning into '/app'...
[root@localhost student_list]# ls
docker-compose.yml  README.md  simple_api  website
[root@localhost student_list]#
```

Notre clonage a bien réussi et on est parvenu à lister les fichiers après clonage.

1. Construction de l'API Flask
    - Création du fichier dockerfile

      Le fichier docker file permet de construire l'image de l'API.

```
astou@localhost:/home/astou/student_list/simple_api
  GNU nano 5.6.1                                                    Dockerfile
# Utiliser l'image Python 3.8
FROM python:3.8-buster

# Mainteneur
LABEL maintainer="Astou GUEYE"

# Définir le répertoire de travail
WORKDIR /

# Copier les fichiers sources
COPY student_age.py /student_age.py
COPY student_age.json /data/student_age.json
COPY requirements.txt /requirements.txt

# Installer les dépendances
#RUN apt update && apt install -y python3-dev libss1-dev libldap2-dev libldap2-dev libss1-dev
RUN apt update && apt install -y python3-dev libss1-dev libldap2-dev libsasl2-dev

RUN pip3 install -r /requirements.txt

# Déclarer un volume pour stocker les données
VOLUME /data

# Exposer le port 5000
EXPOSE 5000

# Lancer l'application Flask
CMD ["python3", "/student_age.py"]
```

Ce fichier inclut :
Une base python : 3.8-buster.
L'installation des dépendances via requirements.txt c'est-à-dire installer toutes les bibliothèques nécessaires listées dans le fichier requirements.txt.
La copie des fichiers sources pour permettre de les ajouter au conteneur Docker.
L'exposition du port 5000.
Lancement automatique de l'application lors du démarrage du conteneur.

Nous allons construire l'image docker avec la commande suivante :

Avec la commande docker image on peut bien voir que l'image est bien crée :

```
[root@localhost simple_api]# docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
student-list                latest    c0cccb40ac7c   49 seconds ago   1.05GB
```

La commande suivante permet de lancer notre conteneur en arrière-plan en l'exposant sur le port 5000 :

```
[root@localhost simple_api]# docker run -d -p 5000:5000 --name test-student-list student-list
503438d3be3d463aa0e2aa25ec0ee9ed2757d2e632b9aad4e9d8f99987062787
[root@localhost simple_api]#
```

On peut aussi vérifier si le conteneur tourne bien :

```
[root@localhost simple_api]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                       NAMES
58343d3be3d   student-list   "python3 /student_ag…"   31 seconds ago   Up 31 seconds   0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp   test-student-list
```

On va tester l'API avec curl :

```
[root@localhost simple_api]# curl -u toto:python -X GET http://localhost:5000/pozos/api/v1.0/get_student_ages
{
  "student_ages": {
    "alice": "12",
    "bob": "13"
  }
}
[root@localhost simple_api]#
```

- Création du fichier docker-compose.yml

  Le fichier docker-compose permet d'orchestrer les services :

```
astou@localhost:/home/astou/student_list/simple_api
  GNU nano 5.6.1                                                docker-compose.yml
version: '3.8'

services:
  api:
    image: student-list:latest  # Nom de l'image Flask
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./student_age.json:/data/student_age.json
    networks:
      - pozos_network

  web:
    image: php:apache
    ports:
      - "8080:80"
    volumes:
      - ../website:/var/www/html  # Monte le dossier du site web
    environment:
      USERNAME: "toto"
      PASSWORD: "python"
    depends_on:
      - api
    networks:
      - pozos_network

networks:
  pozos_network:
    driver: bridge
```

  Ce fichier contient :

  Deux services : un API Flask sur le port 5000 et un service web PHP avec Apache sur le port 8080.

  monte un répertoire local ../website comme **/var/www/html** pour le serveur Apache.

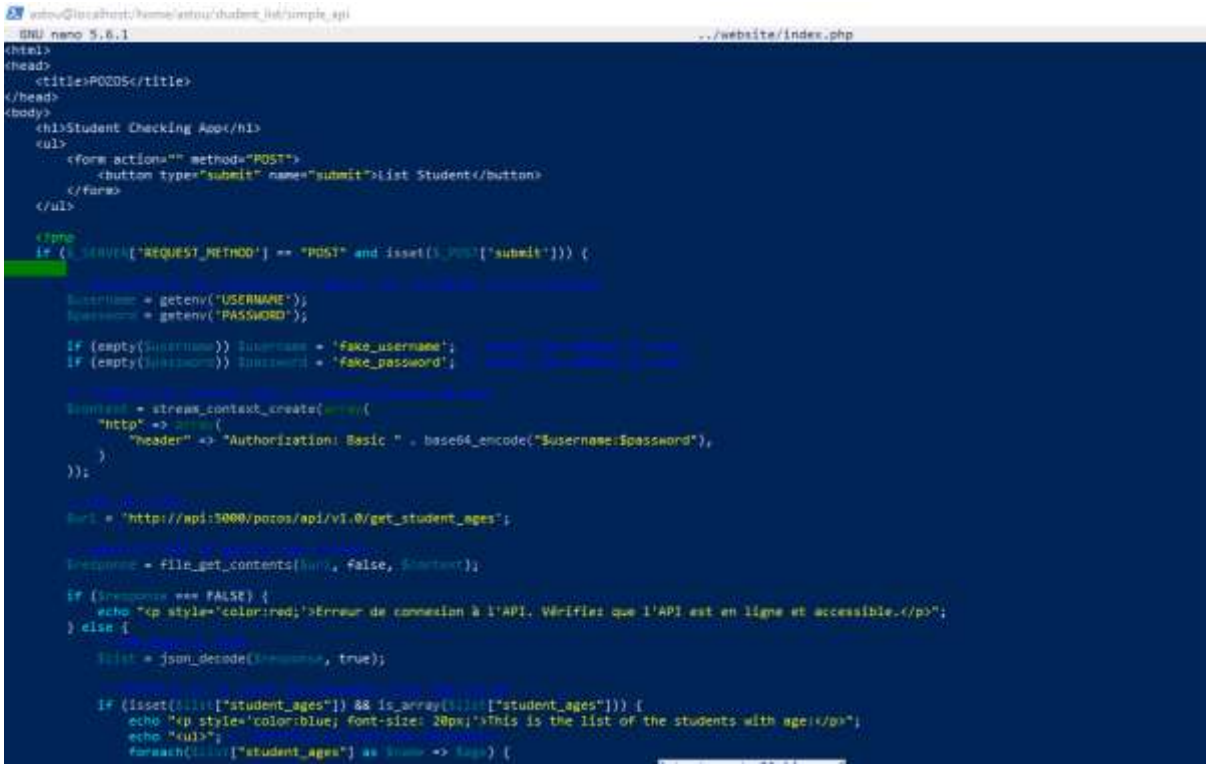
  Définit deux variables d'environnement USERNAME et PASSWORD.

  Une communication entre les services via un réseau Docker pozos_network.

- Pour les autres fichiers nécessaires

  Index.php : page php pour l'interface utilisateur (c'est le fichier php qui se trouve dans website qu'on a édité).

  Ce code PHP suivant permet d'afficher la liste des étudiants et leurs âges en interrogeant une API Flask.

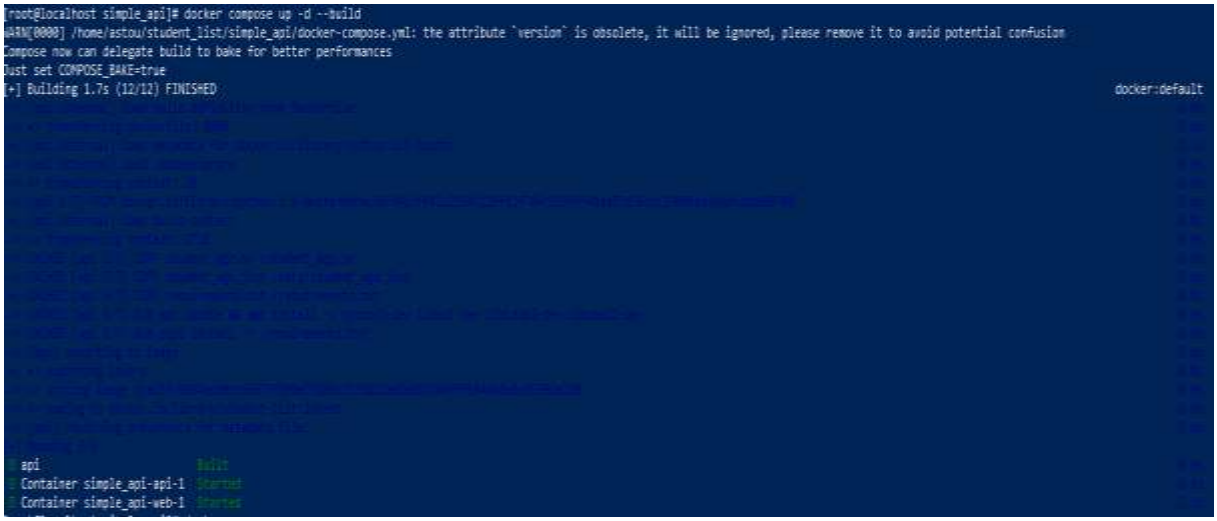


- Lancement et test

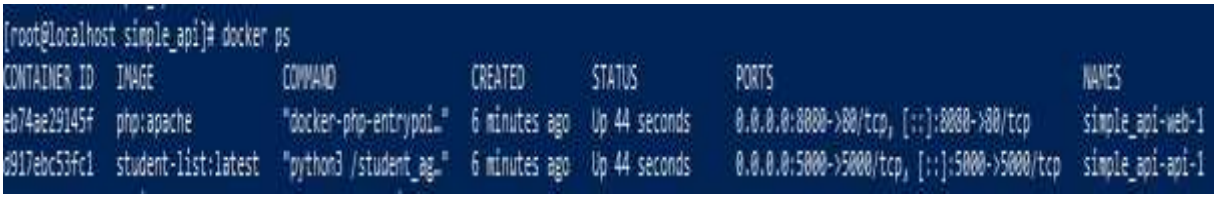
Nous allons d'abord construire et démarrer les services en exécutant cette commande :

docker compose up -d –build

Voici-ci-dessous la sortie de cette commande :



On peut aussi vérifier si les conteneurs tournent bien :



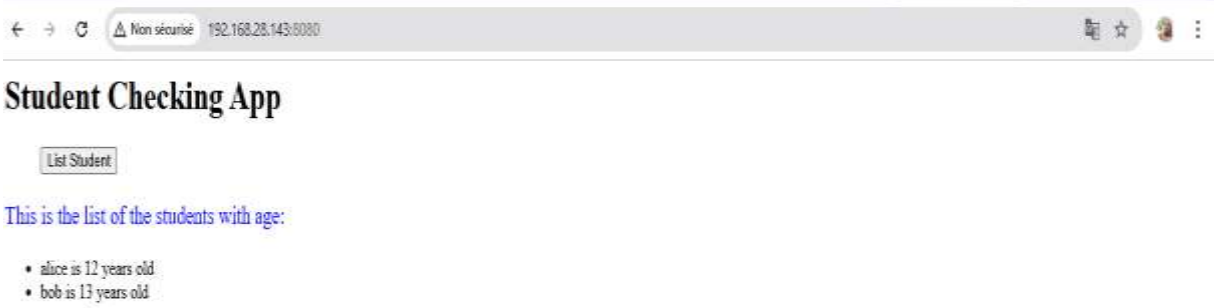
On va tester l'API avec curl :

```
[root@localhost simple_api]# curl -u toto:python -X GET http://localhost:5000/pozos/api/v1.0/get_student_ages
{
  "student_ages": {
    "alice": "12",
    "bob": "13"
  }
}
[root@localhost simple_api]#
```

On peut dire que tout fonctionne bien car l'API renvoie une liste JSON des étudiants.

Maintenant nous allons tester le site web en ouvrant un navigateur pour accéder à http://192.168.28.143:8080 :

Notre site web est bien affiché maintenant on peut appuyer sur List Student pour afficher la liste des étudiants avec leur âge :



2. Déploiement du registre docker

   Pour stocker nos images nous allons utiliser un registre docker privé. Voici les étapes à suivre :

   - Ajouter le service registre dans notre fichier docker_compose.yml

root@192:/student_list

GNU nano 5.6.1 docker-compose.yml

```
version: '3.8'

services:
  api:
    image: student_api:latest  # le nom de ton image Flask créée
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./student_age.json:/data/student_age.json
    networks:
      - pozos_network

  web:
    image: php:apache
    ports:
      - "8080:80"  # Expose le site web sur le port 8080
    volumes:
      - ./website:/var/www/html  # Monte le dossier du site dans le conteneur
    environment:
       USERNAME: "toto"
       PASSWORD: "python"
    depends_on:
      - api
    networks:
      - pozos_network
  registry:
    image: registry:2  # Utilise l'image officielle du registre Docker
    container_name: my_registry
    ports:
      - "5001:5000"  # Expose le registre Docker sur le port 5000
    volumes:
      - ./data:/var/lib/registry  # Persist les données du registre sur le volume ./data
    networks:
      - pozos_network  # Réseau partagé entre les services

networks:
  pozos_network:
    driver: bridge  # Utilisation du réseau bridge pour partager les services

volumes:
  # Définition d'un volume pour persister les données du registre
  data:
```

Il utilise l'image officielle **`registry:2`**, expose le port **`5001`** et monte un volume (**`./data`**) pour persister les images Docker que tu vas pousser dans ce registre.

- Relancer docker-compose.yml en exécutant ces commandes :
  - docker-compose down
  - docker-compose up –d
- Taguer l'image :

```
[root@localhost simple_api]# docker tag student-list localhost:5001/student-list
[root@localhost simple_api]#
```

- Pousser l'image vers le registre :

```
[root@localhost simple_api]# docker push localhost:5001/student-list
Using default tag: latest
The push refers to repository [localhost:5001/student-list]
fd46f2b180f2: Pushed
d43698c7f5da: Pushed
907083c2194a: Pushed
45af59ef749d: Pushed
f475103e0d87: Pushed
6e2f63b0cf05: Pushed
45359261cd7a: Pushed
ad312497d9a5: Pushed
474c7af10697: Pushed
dcc1cfeee1ab: Pushed
eccb9ed74974: Pushed
53d40515380c: Pushed
6af7a54a0a0d: Pushed
latest: digest: sha256:8cdbf1f78fd2508004e2f3b8ff197c41f86ff7c784a0b29f74d45f236b779c58 size: 3051
[root@localhost simple_api]#
```

- Vérification de l'image dans le registre :

```
[root@localhost simple_api]# curl http://localhost:5001/v2/_catalog
{"repositories":["student-list"]}
[root@localhost simple_api]#
```

Le résultat indique que le registre docker fonctionne correctement et que l'image student-list a bien été poussée vers le registre.