**Chérif Bachir**

Master ILW

# Application Student-list

## Etape 1

```
1 FROM python:2.7-stretch
2
3 # Maintainer
```

- La première ligne du dockerfile “FROM” permet de récupérer une image de base sur laquelle baser notre propre image. Cela évite de repartir sur un os vide et de faire l’installation nous même.

```
3 # Maintainer
4 LABEL org.opencontainers.image.author="CHERIF BACHIR <bachir_c@etna-alternann
  ce.net>"
```

- L’instruction LABEL permet d’ajouter des metadata à une image. Ici par exemple, on renseigne qui est l’auteur de l’image. D’après la documentation DOCKER, author remplace l’instruciton MAINTENER

```
6 # update package
7 RUN apt-get update -y --allow-unauthenticated
8
9 # Install python packages and dependencies
10 RUN apt-get install python-dev libldap2-dev libsasl2-dev python-dev libssl-dd
   ev python3-dev -y --allow-unauthenticated
```

- L’instruction suivante dans le dockerfile est l’instruction RUN apt-get update -y. L’instruction RUN dans un dockerfile permet d'exécuter une commande lors de la mise en place de la machine par docker. Cette première ligne permet donc de mettre à jour l’application et d’installer les packages nécessaires au bon fonctionnement de l’API.

```
11 RUN pip install flask flask_httpauth flask_simpleldap python-dotenv
12
```

- On retrouve une autre instruction RUN celle-ci permet d’installer les dépendances pythons nécessaire au bon fonctionnement de l’API utilisant FLASK engine.

```
13 # Copie the student api at the root of the image OS
14 COPY ./student_age.py /
15
```

- L'instruction suivante est l'instruction COPY qui permet de déplacer un fichier à un endroit précis. Ici on déplace le code source à la racine de l'image docker.

```
16 # Créer un bind mount à /data/
17 VOLUME [ "/data/" ]
18
```

- L'instruction VOLUME permet de créer un dossier partagé entre la machine hôte et l'image docker. Ici on indique que le volume data permettra de partager les fichiers. Cela permet d'avoir une persistence des données.

```
19 # Ouvrir le port d'écoute au lancement du conteneur
20 EXPOSE 5000
21
```

- L'instruction EXPOSE permet elle d'ouvrir un port de l'image. Dans notre cas, on ouvre le port 5000.

```
CMD [ "python", "/student_age.py" ]
```

- La dernière instruction est l'instruction CMD qui permet d'exécuter au lancement du container une commande en particulier. Ici on demande de lancer l'API python.

Notre dockerfile est à présent terminé. Nous allons à présent tester le test de l'image.

```
docker build -t simple-api:latest .
```

- Cette commande nous permet de créer l'image. on donne un nom ainsi qu'un tag avec le paramètre -t puis . correspond à l'emplacement du dockerfile à build.

```
[node2] (local) root@10.0.4.4 ~/student-list/simple api
docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS         PORTS                    NAMES
b2319ff5bf45   simple-api:latest  "python student_age.…"   2 minutes ago   Up 2 minutes   0.0.0.0:5000->5000/tcp   simple-api-app
[node2] (local) root@10.0.4.4 ~/student-list/simple api
```

```
[node2] (local) root@10.0.4.4 ~/student-list/simple api
$ docker run --name simple-api-app -d -v $PWD/student_age.json:/data/student_age.json -p 5000:5000 simple-api:latest
b2319ff5bf45db8bf4bfa609fc00e33344ffc89cb4f7bd55b42eb3754e570504
[node2] (local) root@10.0.4.4 ~/student-list/simple api
$ docker images
REPOSITORY    TAG           IMAGE ID       CREATED          SIZE
simple-api    latest        721f09cf691e   37 seconds ago   1.13GB
python        2.7-stretch   e71fc5c0fcb1   17 months ago    928MB
[node2] (local) root@10.0.4.4 ~/student-list/simple api
```

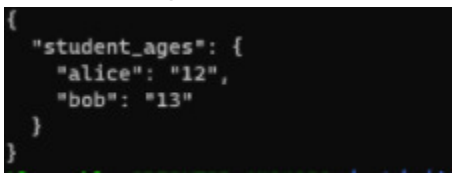
```
Removing intermediate container 4e288e92ba3f
 ---> e99b82d75f1b
Step 6/9 : COPY ./student_age.py /
 ---> 1d69d54bbe45
Step 7/9 : VOLUME [ "/data/" ]
 ---> Running in d95123399fd6
Removing intermediate container d95123399fd6
 ---> 9f2e12d22d24
Step 8/9 : EXPOSE 5000
 ---> Running in 49d4bcfbb591
Removing intermediate container 49d4bcfbb591
 ---> 79a1e8e621da
Step 9/9 : CMD [ "python", "student_age.py" ]
 ---> Running in 8b0ea8edadf9
Removing intermediate container 8b0ea8edadf9
 ---> 721f09cf691e
Successfully built 721f09cf691e
Successfully tagged simple-api:latest
[node2] (local) root@10.0.4.4 ~/student-list/simple api
$ docker run --name simple-api-app -d -v $PWD/student_age.json:/data/student_age.json -p 5000:5000 simple-api:latest
b2319ff5bf45db8bf4bfa609fc00e33344ffc89cb4f7bd55b42eb3754e570504
[node2] (local) root@10.0.4.4 ~/student-list/simple api
$ docker images
REPOSITORY   TAG           IMAGE ID       CREATED          SIZE
simple-api   latest        721f09cf691e   37 seconds ago   1.13GB
python       2.7-stretch   e71fc5c0fcb1   17 months ago    928MB
[node2] (local) root@10.0.4.4 ~/student-list/simple api
$
```

```
docker run --name simple-api-app -d -v $PWD/student_age.json:/data/student_age.json -p 5000:5000 simple-api:latest
```

- la commande docker run permet de mettre en place un container. on précise avec --name le nom du container, -d permet de dire que l'on souhaite lancer le contianer en mode détacher, -v permet de préciser un volume de partage, -p permet de rediriger un port, et à la fin on précise quelle image on souhaite utiliser.

```
[node2] (local) root@10.0.4.4 ~/student-list/simple api
$ curl -u toto:python -X GET http://127.0.0.1:5000/pozos/api/v1.0/get_student_ages
```

- Lorsqu'on lance la commande curl voici le résultat obtenu:



# Infrastructure As Code

```yaml
version: "3.3"
services:
  website:
    image: php:apache
    environment:
      - USERNAME=toto
      - PASSWORD=python
    volumes:
      - ./website:/var/www/html
    ports:
      - 80:80
    networks:
      - pozos_network
    depends_on:
      - api
```

- dans le docker-compose.yml on retrouve des services. Notre premier service est le service website qui permet de mettre en ligne le site web de l'application.

  La première instruction est "image", elle permet de construire un container à partir d'une image en particulier. Ici on choisit une image de php avec apache.

  La deuxième instruction est " environment" permet de préciser des variables d'environnement. On précise ici le username et le password d'accès à l'API.

  La troisième instruction est "volumes" permet de définir quels dossiers sont partagés entre l'hôte et le container. Ici on partage le contenu de website dans /var/www/html.

  La quatrième instruction est "ports" nous permet de définir quel port du container est redirigé vers quel port de l'hôte.

  La cinquième instruction est "networks" nous permet de définir sur quel réseau le service va communiquer.

  La sixième instruction est "depends_on" qui permet de s'assurer d'une hiérarchie de lancement dans les services.

```
16  api:
17    build: "./simple_api"
18    volumes:
19      - ./simple_api/student_age.json:/data/student_age.json
20    ports:
21      - 5000:5000
22    networks:
23      - pozos_network
```

- Le deuxième service est le service api. Il va permettre de lancer les instructions de la commande docker run de tout à l'heure.

  En premier lieu on retrouve l'instruction "build" qui permet de build une image à partir d'un dockerfile.

  Ensuite on retrouve de nouveau l'instruction "volumes" qui permet de partager notre fichier json de l'hôte à l'api.

  On a ensuite de nouveau l'instruction "ports" qui nous permet de rediriger les ports du container.

  En dernière instruction on retrouve de nouveau l'instruction "networks".

```
44  networks:
45    pozos_network:
```

On retrouve ensuite la partie networks qui nous permet de lié les différents services. Ici nous n'en avons qu'un seul que nous avons nommé pozos_network sur lequel nos deux container communiquent.

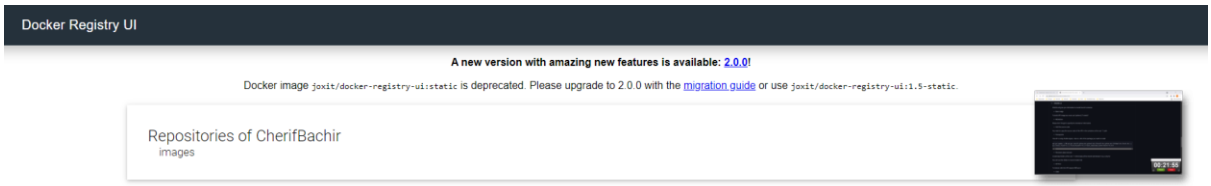
Une fois notre fichier terminé, nous pouvons le lancer avec la commande ci-dessus. Il lancera alors deux containers représentés par les services dans le fichier. Et en nous rendant à l'adresse localhost on a le résultat suivant :


# Docker Registry

```
  p_registry:
    image : registry:2
    ports:
      - 4999:5000
    environment:
      - REGISTRY_STORAGE_DELETE_ENABLED=true
    networks:
      - registry_network
  registry_ui:
    image: joxit/docker-registry-ui:static
    depends_on:
      - p_registry
    networks:
      - registry_network
    ports:
      - 8090:80
    environment:
      - REGISTRY8URL=http://private_registry:5000
      - DELETE_IMAGE=true
      - REGISTRY_TITLE=CherifBachir
networks:
  pozos_network:
  registry_network
```

- Pour mettre en place le private registry, j'ai ajouté dans le docker-compose.yml les éléments nécessaires à la mise en place comme présenté ci-dessus. L'ui accessible sur le port 8090 et le registry sur le port 4999 depuis l'hôte.

  J'ai ensuite relancer la commande docker-compose up -d qui permet de mettre ne place notre private registry.




- Afin de mettre mon image sur le private repository, j'ai réalisé les commandes que voici:

```
docker tag student-list_api:latest localhost:4999/studentlistapi
docker push localhost:4999/studentlistapi
```