Project - Student List

Build and test

Pozos needs a docker image that runs its Python application. To do that, we need to build a Dockerfile with the appropriate instructions :

Dockerfile: ***(commenté, en annexe se trouvera le fichier formater correctement)***

Instruction	Description
FROM python:2.7-stretch	L'image python:2.7 de base que Docker utilise pour notre nouvelle image
MAINTAINER Group1 <group1@pozos.fr>	Le responsable du Dockerfile
ADD student_age.py /	ajout du fichier source à la racine
RUN apt-get update -y && apt-get install python-dev python3-dev libsasl2-dev python-dev libldap2-dev libssl-dev -y &&\ pip install flask==1.1.2 flask_httpauth==4.1.0 flask_simpleldap python-dotenv==0.14.0	Construction de l'image et l'installation de packages requis
VOLUME /data	Le volume attaché pour lire les données (student_age.json)
EXPOSE 5000	Le port 5000 est disponible pour écouter les demandes entrantes
CMD ["python", "./student_age.py"]	la commande à exécuter par défaut par le conteneur lorsqu'on lance notre l'image

```
FROM python:2.7-stretch
MAINTAINER Group1 <group1@pozos.fr>
ADD student_age.py /
RUN apt-get update -y && apt-get install python-dev python3-dev libsasl2-dev python-dev libldap2-dev libssl-dev -y &&\
    pip install flask==1.1.2 flask_httpauth==4.1.0 flask_simpleldap python-dotenv==0.14.0
VOLUME /data
EXPOSE 5000
CMD [ "python", "./student_age.py" ]
```

```
[vagrant@docker simple_api]$ docker images
REPOSITORY    TAG           IMAGE ID       CREATED          SIZE
student_age   latest        218c8e4e51fd   48 seconds ago   1.13GB
python        2.7-stretch   e71fc5c0fcb1   17 months ago    928MB
```

La liste des images disponibles “docker images”. Ici , nous avons construit sur la base de python:2.7.

Il ne nous reste plus qu’à lancer notre conteneur via la cmd docker run

```
[vagrant@docker simple_api]$ docker run --name student_age -p 5000:5000 -v "$(pwd)"/data:/data  student_age
 * Serving Flask app "student_age" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 241-258-015
```

La commande **docker run** créée un container avec l’image **student_age** buildée précédemment.

Les options

- **--name** pour nommer notre conteneur au lieu d’y accéder via son ID
- **-p** mappe le port du conteneur au port 5000 sur l‘@Ip publique de notre machine
- **-v** le volume attaché au conteneur sur lequel il lit et écrit ses données

```
[vagrant@docker simple_api]$ docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS                                       NAMES
344f371ec06e   student_age   "python ./student_ag…"   5 seconds ago   Up 4 seconds   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   student_age
```

On peut lister nos conteneurs avec la commande **docker ps**

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

La commande **curl -u toto:python …** permet de tester notre api en local sur le port 5000 exposé lors de la création de notre conteneur.

Infrastructure As Code

Afin de tester notre API à l'aide d'un serveur web, et pour rendre "reproductible" notre test, nous allons donc maintenant créer un fichier "**docker-compose.yml"** qui va contenir les deux services, **l'api** et le **website**, en temps que conteneur.

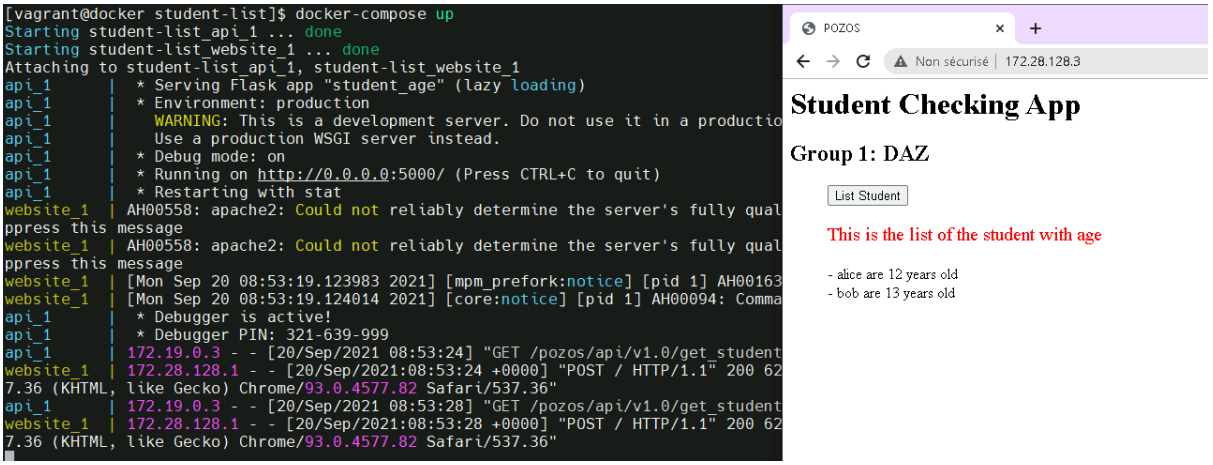
docker-compose.yml: ***(commenté, en annexe se trouvera le fichier formater correctement)***

version: "3.8"	Il faut d'abord spécifier la version de notre fichier, en lien avec la version de docker que nous utilisons. avec docker version "20.10.8", la version "3.8" du format docker-compose.yml
services:	Commençons à déclarer nos services docker-compose:
website:	Nous nommons notre service "**website**"
depends_on: - api	Le site dépend bien évidemment de l'api, nous lui ajoutons donc cette dépendance
image: php:apache	En utilisant l'image "php:apache"
volumes: - ./website:/var/www/html	Nous lui attachons un volume pour monter l'index.php du site qui ne se trouve bien évidemment pas directement dans l'image
environment: USERNAME: toto PASSWORD : python	L'index.php nécessite 2 variables d'environnement, un **USERNAME** et un **PASSWORD**, nous lui ajoutons donc ces variables
ports: - "80:80"	Pour pouvoir accéder au site sur votre navigateur, nous aurons besoin d'exposer le port 80 du conteneur sur l'hôte.
api:	Nous nommons notre service "**website**"
image: student_age:latest	En utilisant l'image "php:apache"
volumes: - ./simple_api/data:/data	Nous lui attachons un volume pour monter le fichier **student_age.json** du site qui ne se trouve bien évidemment pas directement dans l'image, depuis nos sources vers le dossier "/data" du conteneur.
ports: - "5000:5000"	Si nous avons besoin d'accéder à l'api depuis l'hôte, nous exposerons le port "5000" (FLASK) de notre conteneur

Il ne nous reste plus qu'à lancer notre infrastructure via la commande:

```
$docker-compose up
```

Et voilà le résultat, nos conteneurs tournent, et le site retrouve bien les age de Bob et d'Alice:



Docker Registry

Pozos souhaite pouvoir pousser ses images dans leur réseau, et non sur le dockerhub public. On a donc besoin de déployer une registry sur notre machine.

Pour se faire, nous utiliserons les images

- *registry:2* ; la registry en elle même.
- *joxit/docker-registry-ui:1.5-static* ; une interface graphique pour lire sur une page web la registry.

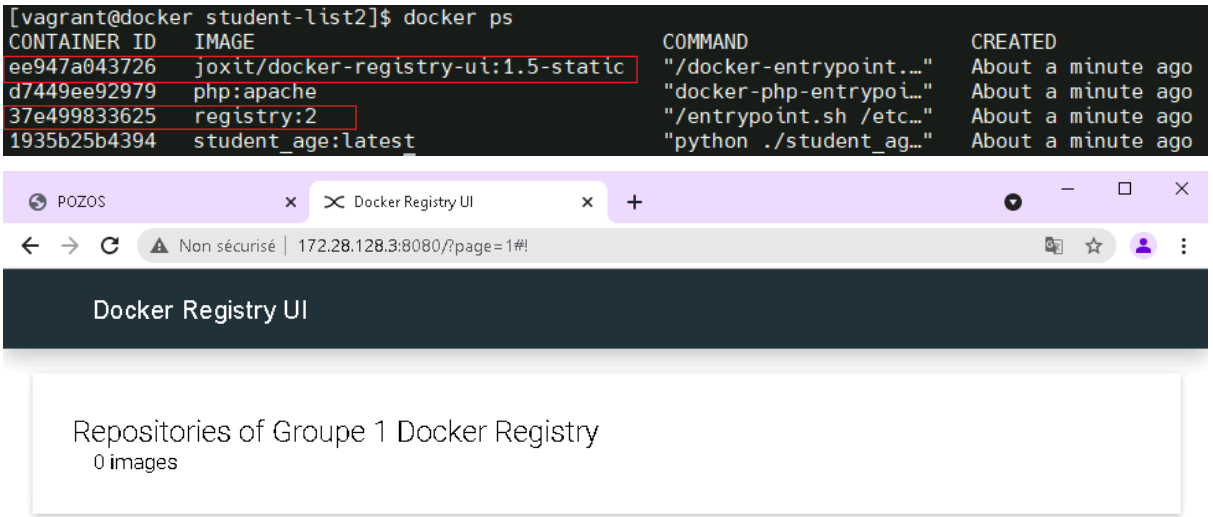
Nous avons donc choisi de la déployer à côté de notre application, dans le même “docker-compose.yml” que nous avons donc complété ainsi :

docker-compose.yml: ***(commenté, en annexe se trouvera le fichier formater correctement)***

version: "3.8"
services:
  website:
  depends_on:
  - api
  image: php:apache
  volumes:
  - ./website:/var/www/html
  environment:
  USERNAME: toto
  PASSWORD : python
  ports:
  - "80:80"
 	
  api:
  image: student_age:latest
  volumes:
  - ./simple_api/data:/data
  ports:
  - "5000:5000"
 	
  registry:
 	
  image: registry:2
  volumes:
  - ./registry-data:/var/lib/registry
 	
  ports:
  - "5050:5000"
 	
  ui:
  image: joxit/docker-registry-ui:1.5-static
	|
	|
	|
	|
	|
	|
	|
	|----- Pareil que pour la partie IaC
	|
	|
	|
	|
	|
	|
	|
	|
	|
	
	Nous nommons notre service “**registry**”
	Utilisant l’image “**registry:2**”
	Nous lui attachons un volume pour assurer la persistance des données de la registry
	Comme le port 5000 est utilisé par FLASK, nous devons attribuer un nouveau pour l’hôte, ici 5050
	
	Nous nommons notre service **“ui”**
	Utilisant l’image

ports:
  - "8080:80"
 	
environment:
  - REGISTRY_TITLE=Groupe 1 Docker Registry
  - REGISTRY_URL=http://registry:5000
 	
depends_on:
  - registry
	“joxit/docker-registry-ui:1.5-static”
	
	Comme le port 80 est utilisé par le site web, nous devons attribuer un nouveau pour l’hôte, ici 8080
	
	Le conteneur a besoin de variable d'environnement comme paramètre, nous lui donnons donc un **titre**
	et l’url de notre registry.
	
	L’ui dépend bien évidemment de la registry, nous lui ajoutons donc cette dépendance

Grâce à ça, nous avons donc notre registry, avec son interface :



Ensuite, il faut préparer notre image à être pousser dans la registry, et donc, la tagger avec “**docker tag**” avant de la pousser dans la registry avec **“docker push”**.

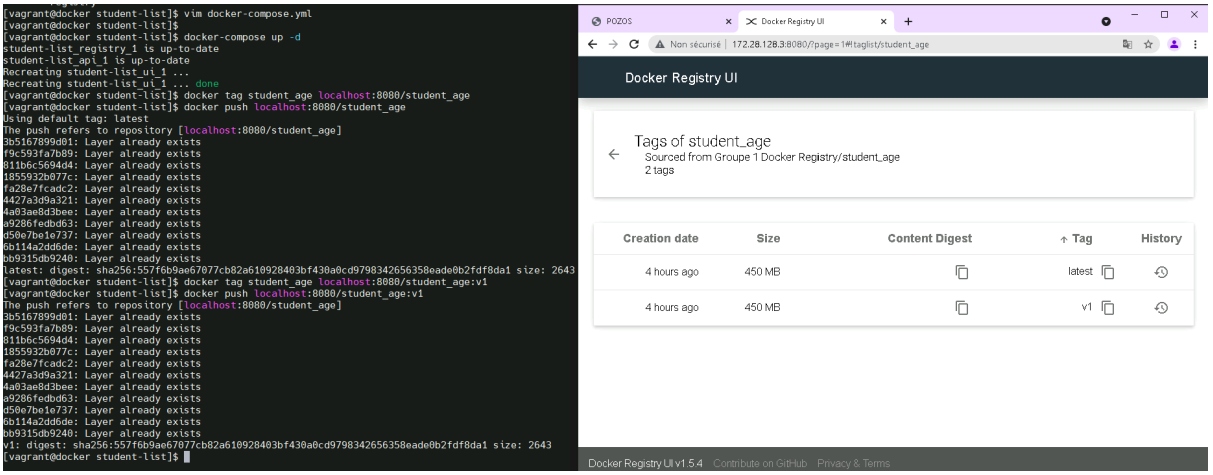
La registry étant en local, exposé sur le port 8080 comme vu précédemment, le tag sera donc composé de *<registry_adress>/<image_name>*, soit **“localhost:8080/student_age”**

```
$docker tag student_age localhost:8080/student_age
$docker push localhost:8080/student_age
```

Dans notre cas, notre image était toujours en “latest” comme nous ne spécifions pas de tag, mais nous pouvons aussi bien leur donner une version, par exemple, v1 :

```
$docker tag student_age localhost:8080/student_age:v1
$docker push localhost:8080/student_age:v1
```

Voilà, notre image est maintenant utilisable via notre registry privé, dans les deux versions, **latest**, et **v1**.



Annexe

Dockerfile:

```
FROM python:2.7-stretch
MAINTAINER Group1 <group1@pozos.fr>
ADD student_age.py /
RUN apt-get update -y && apt-get install python-dev python3-dev libsasl2-dev python-dev
libldap2-dev libssl-dev -y &&\
    pip install flask==1.1.2 flask_httpauth==4.1.0 flask_simpleldap python-dotenv==0.14.0
VOLUME /data
EXPOSE 5000
CMD [ "python", "./student_age.py" ]
```

docker-compose.yaml : partie IaC

```
version: "3.8"
services:
  website:
    depends_on:
      - api
    image: php:apache
    volumes:
      - ./website:/var/www/html
    environment:
      USERNAME: toto
      PASSWORD : python
    ports:
      - "80:80"

  api:
    image: student_age:latest
    volumes:
      - ./simple_api/data:/data
    ports:
      - "5000:5000"
```

docker-compose.yaml : partie Registry

```
version: "3.8"
services:
  website:
    depends_on:
      - api
    image: php:apache
    volumes:
      - ./website:/var/www/html
```

```
  environment:
    USERNAME: toto
    PASSWORD : python
  ports:
    - "80:80"

api:
  image: student_age:latest
  volumes:
    - ./simple_api/data:/data
  ports:
    - "5000:5000"

registry:
  image: registry:2
  volumes:
    - ./registry-data:/var/lib/registry
  ports:
    - "5050:5000"
ui:
  image: joxit/docker-registry-ui:1.5-static
  ports:
    - "8080:80"
  environment:
    - REGISTRY_TITLE=Groupe 1 Docker Registry
    - REGISTRY_URL=http://registry:5000
  depends_on:
    - registry
```