

# 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 &&\n    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 libssl2-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@doker 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@doker 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** crée 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@doker 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@doker 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 " <b>website</b> "
depends_on:	Le site dépend bien évidemment de l'api, nous lui ajoutons donc cette dépendance
- api	
image: php:apache	En utilisant l'image "php:apache"
volumes:	Nous lui attachons un volume pour monter l'index.php du site qui ne se trouve bien évidemment pas directement dans l'image
- ./website:/var/www/html	
environment:	L'index.php nécessite 2 variables d'environnement, un <b>USERNAME</b> et un <b>PASSWORD</b> , nous lui ajoutons donc ces variables
USERNAME: toto	
PASSWORD : python	
ports:	Pour pouvoir accéder au site sur votre navigateur, nous aurons besoin d'exposer le port 80 du conteneur sur l'hôte.
- "80:80"	
api:	Nous nommons notre service " <b>website</b> "
image: student_age:latest	En utilisant l'image "php:apache"
volumes:	Nous lui attachons un volume pour monter le fichier <b>student_age.json</b> du site qui ne se trouve bien évidemment pas directement dans l'image, depuis nos sources vers le dossier "/data" du conteneur.
- ./simple_api/data:/data	
ports:	Si nous avons besoin d'accéder à l'api depuis l'hôte, nous exposerons le port "5000" (FLASK) de notre conteneur
- "5000:5000"	

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:

```
[vagrant@doker student-list]$ docker-compose up
Starting student-list_api_1 ... done
Starting student-list_website_1 ... done
Attaching to student-list_api_1, student-list_website_1
api_1 | * Serving Flask app "student_age" (lazy loading)
api_1 | * Environment: production
api_1 |   WARNING: This is a development server. Do not use it in a production
api_1 |   Use a production WSGI server instead.
api_1 | * Debug mode: on
api_1 | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
api_1 | * Restarting with stat
website_1 | AH00558: apache2: Could not reliably determine the server's fully qual
ppress this message
website_1 | AH00558: apache2: Could not reliably determine the server's fully qual
ppress this message
website_1 | [Mon Sep 20 08:53:19.123983 2021] [mpm_prefork:notice] [pid 1] AH00163
website_1 | [Mon Sep 20 08:53:19.124014 2021] [core:notice] [pid 1] AH00094: Comm
api_1 | * Debugger is active!
api_1 | * Debugger PIN: 321-639-999
api_1 | 172.19.0.3 - - [20/Sep/2021 08:53:24] "GET /pozos/api/v1.0/get_student
website_1 | 172.28.128.1 - - [20/Sep/2021:08:53:24 +0000] "POST / HTTP/1.1" 200 62
7.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36"
api_1 | 172.19.0.3 - - [20/Sep/2021 08:53:28] "GET /pozos/api/v1.0/get_student
website_1 | 172.28.128.1 - - [20/Sep/2021:08:53:28 +0000] "POST / HTTP/1.1" 200 62
7.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36"
```

POZOS

Non sécurisé | 172.28.128.3

## Student Checking App

Group 1: DAZ

List Student

This is the list of the student with age

- alice are 12 years old
- bob are 13 years old

# 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)*

<pre>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</pre>	<p>----- Pareil que pour la partie laC</p> <p>Nous nommons notre service <b>"registry"</b> Utilisant l'image <b>"registry:2"</b> 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</p> <p>Nous nommons notre service <b>"ui"</b> Utilisant l'image</p>
--	---

<p>ports:</p> <ul style="list-style-type: none"> <li>- "8080:80"</li> </ul> <p>environment:</p> <ul style="list-style-type: none"> <li>- REGISTRY_TITLE=Groupe 1 Docker Registry</li> <li>- REGISTRY_URL=<a href="http://registry:5000">http://registry:5000</a></li> </ul> <p>depends_on:</p> <ul style="list-style-type: none"> <li>- registry</li> </ul>	<p>"joxit/docker-registry-ui:1.5-static"</p> <p>Comme le port 80 est utilisé par le site web, nous devons attribuer un nouveau pour l'hôte, ici 8080</p> <p>Le conteneur a besoin de variable d'environnement comme paramètre, nous lui donnons donc un <b>titre</b> <b>et l'url de notre registry</b>.</p> <p>L'ui dépend bien évidemment de la registry, nous lui ajoutons donc cette dépendance</p>
---	--

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

```
[vagrant@docker student-list2]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
ee947a043726	joxit/docker-registry-ui:1.5-static	"/docker-entrypoint..."	About a minute ago
d7449ee92979	php:apache	"docker-php-entrypoi..."	About a minute ago
37e499833625	registry:2	"/entrypoint.sh /etc..."	About a minute ago
1935b25b4394	student_age:latest	"python ./student_ag..."	About a minute ago

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**.

The image shows a terminal window on the left and a web browser window on the right. The terminal window displays the execution of Docker commands to build and push an image to a local registry. The browser window shows the Docker Registry UI for the 'student\_age' repository, displaying two tags: 'latest' and 'v1'.

```
[vagrant@docker student-list]$ vim docker-compose.yml
[vagrant@docker student-list]$ docker-compose up -d
student-list_registry_1 is up-to-date
student-list_opt_1 is up-to-date
Recreating student-list_ui_1 ...
Recreating student-list_ui_1 ... done
[vagrant@docker student-list]$ docker tag student_age localhost:8080/student_age
[vagrant@docker student-list]$ docker push localhost:8080/student_age
Using default tag: latest
The push refers to repository [localhost:8080/student_age]
3b5167899db1: Layer already exists
f9c593fa7b89: Layer already exists
811b6c5694d4: Layer already exists
1855932b077c: Layer already exists
f228077eadc2: Layer already exists
4427a3d9a321: Layer already exists
4a83ae8d3bee: Layer already exists
a928bfedbd63: Layer already exists
d58070e1e737: Layer already exists
6b114a2dd6de: Layer already exists
bb9315db9240: Layer already exists
latest: digest: sha256:557f6b9ae67077cb02d610920403bf430a0cd9798342656358eade0b2fdf0da1 size: 2643
[vagrant@docker student-list]$ docker tag student_age localhost:8080/student_age:v1
[vagrant@docker student-list]$ docker push localhost:8080/student_age:v1
The push refers to repository [localhost:8080/student_age]
3b5167899db1: Layer already exists
f9c593fa7b89: Layer already exists
811b6c5694d4: Layer already exists
1855932b077c: Layer already exists
f228077eadc2: Layer already exists
4427a3d9a321: Layer already exists
4a83ae8d3bee: Layer already exists
a928bfedbd63: Layer already exists
d58070e1e737: Layer already exists
6b114a2dd6de: Layer already exists
bb9315db9240: Layer already exists
v1: digest: sha256:557f6b9ae67077cb02d610920403bf430a0cd9798342656358eade0b2fdf0da1 size: 2643
[vagrant@docker student-list]$
```

The browser window shows the Docker Registry UI for the 'student\_age' repository. The page title is 'Tags of student\_age' and it indicates 'Sourced from Groupe 1 Docker Registry/student\_age' and '2 tags'. Below this is a table with columns: 'Creation date', 'Size', 'Content Digest', 'Tag', and 'History'.

Creation date	Size	Content Digest	Tag	History
4 hours ago	450 MB		latest	
4 hours ago	450 MB		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.yml : partie laC

```
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.yml : 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
```