

5DVOP

Student List

Résumé

Rendu du projet de 5DVOP sur la Dockerisation d’un projet existant

DEVIGNE Henri

*Table des matières*

[Contexte du projet 2](#_Toc7660)

[Préparation des images de notre application 3](#_Toc7661)

[PROJET API : 3](#_Toc7662)

[CONTRAINTES 3](#_Toc7663)

[CONSTRUCTION DE L’IMAGE 4](#_Toc7664)

[TEST DE NOTRE IMAGE 6](#_Toc7665)

[Projet Front 7](#_Toc7666)

# Contexte du projet

L’objectif de ce projet est de moderniser l’infrastructure actuelle qui repose sur deux serveurs dont la maintenance devient compliquée (aussi bien pour le matériel, que logiciel).

POZOS souhaites utiliser les technologies Docker afin de rendre « portable », « scallable » et plus facile à déployer ses deux applications qui nécessites beaucoup trop de temps actuellement d’autant plus que les technologies utilisées par cette application, sont marqués comme dépréciées et ne sont plus facilement installable sur un poste client ou un serveur. ( Python 2.7 ne sera plus maintenu dès Janvier 2020 )

Les deux applications que POZOS souhaite déployer son :

* API
  + Le projet API consiste en l’exécution d’un script python (student\_age.py )qui est compatible uniquement Python 2.7 et utilise les librairies FLASK.
* FRONT
  + Le projet FRONT contient un script PHP nommé « index.php » et se basera sur l’image php:apache

Ce document contiendra l’intégralité des démarches afin de construire et rendre disponible notre application.

# Préparation des images de notre application

## PROJET API :

Comme indiqué et demandé par la société, nous utiliserons l’image Docker de python2.7-stretch comme base pour notre image Docker.

### CONTRAINTES

* Le projet utilise python2.7 et fonctionne initialement sur Debian, nous privilégions l’image de base Debian par rapport à Alpine concernant les paquets et les dépendances qui sont actuellement uniquement maitrisées par les développeurs ainsi que pour son grand nombre de paquet dont certains très spécifiques comme libldap2-dev.
* Les paquets suivants doivent être installés :

o python-dev o libsasl2-dev o libldap2-dev o libssl-dev

* Le programme python utilise des dépendances fournies par le gestionnaire de paquet

Python appelé « PIP » o flask o flask\_httpauth o flask\_simpleldap o python-dotenv

### CONSTRUCTION DE L’IMAGE

Commençons par créer notre Dockerfile (simple\_api/Dockerfile)

La première ligne de notre Dockerfile sera l’image sur laquelle elle est basée, à savoir : python2.7-stretch.



Henri Devigne sera responsable du maintien et des évolutions de l’image, nous allons donc le mentionner avec la fonction Docker: « MAINTAINER ».

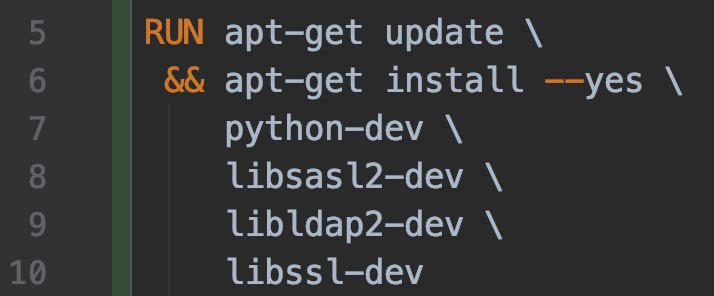
 L’information est bien prise en compte comme nous pouvons le voir en faisant un docker image inspect sur notre image :



Enfin, nous allons procéder à l’installation des différents paquets Debian nécessaires au bon fonctionnement de notre projet, mais tout d’abord, nous allons devoir procéder à la mise à jour des repositories car l’image python2.7-stretch a volontairement supprimé la liste des repositories afin d’alléger l’image de base comme nous pouvons le voir avec la commande : « docker image history python2.7-stretch –no-trunc »

/BIN/SH -C APT-GET UPDATE && APT-GET INSTALL -Y --NO-INSTALL-RECOMMENDS CA-CERTIFICATES CURL NETBASE WGET && RM -RF /VAR/LIB/APT/LISTS/\*

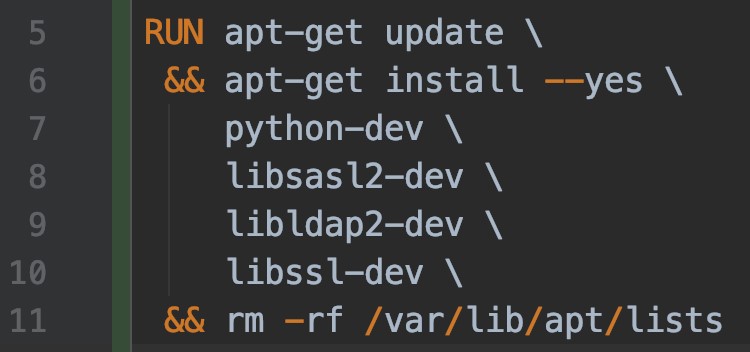
Nous allons lancer donc lancer la commande pour mettre à jours nos paquets, et les installer.



Regardons désormais le poids de notre image après avoir lancer le build ( docker build simple\_api -t simple\_api ) avec docker image history simple\_api



Optimisons désormais le poids de notre image en supprimant le cache APT ( Gestionnaire de paquet sous debian ) en ajoutant à la fin du RUN la suppression du dossier de cache (/var/lib/apt/lists)



Reconstruisons désormais notre image et regardons la différence.



Nous avons gagné approximativement 17MB sur notre image grâce à cette optimisation.

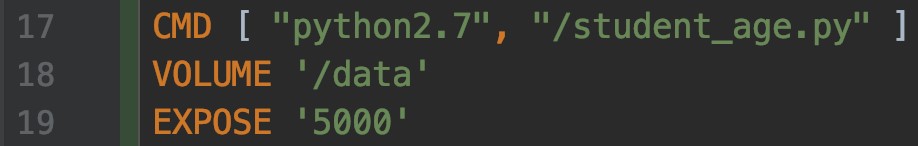
Installons désormais les dépendances Flasks avec l’instruction RUN :



Ajoutons notre fichier python « student\_age.py » dans notre conteneur à la racine (/) comme demandé par les développeurs :

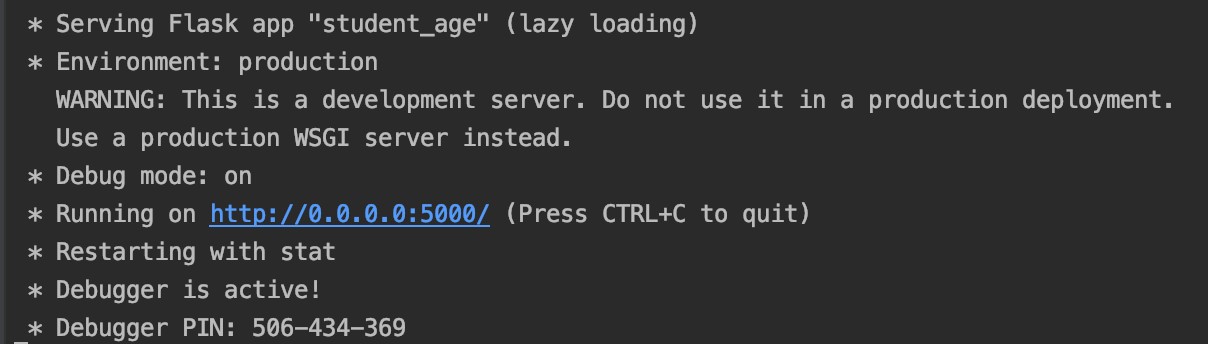


Et enfin, définisons notre entrypoint, le port que peut exposer notre application, ainsi que son volume (/data)

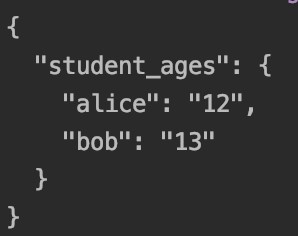


### TEST DE NOTRE IMAGE

Notre application est désormais prête à fonctionner, nous allons tester l’image avec un docker run



Comme nous pouvons le voir l’application semble fonctionner correctement, mais nous devons malgré tout, tester son comportement avec un « curl »



Nous avons donc notre « API » parfaitement fonctionnelle.

## Projet Front

Le projet front se basera sur l’image php:apache.

Contraintes

Voici les variables d’environnement nécessaire au bon fonctionnement du projet :

* USERNAME
* PASSWORD

Et le « website » doit être monté dans « /var/www/html ».

*Orchestration de nos conteneurs*

Il est désormait temps d’orchestrer les conteneurs au préalablement définis.

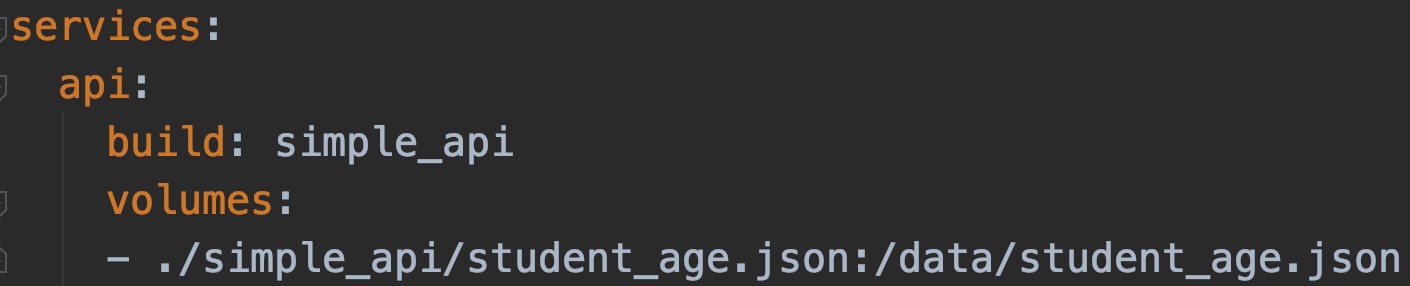
Créeons de ce fait notre docker-compose.yml :

Commençons par indiquer la version utiliser ( ici, la version 3.7 ) :



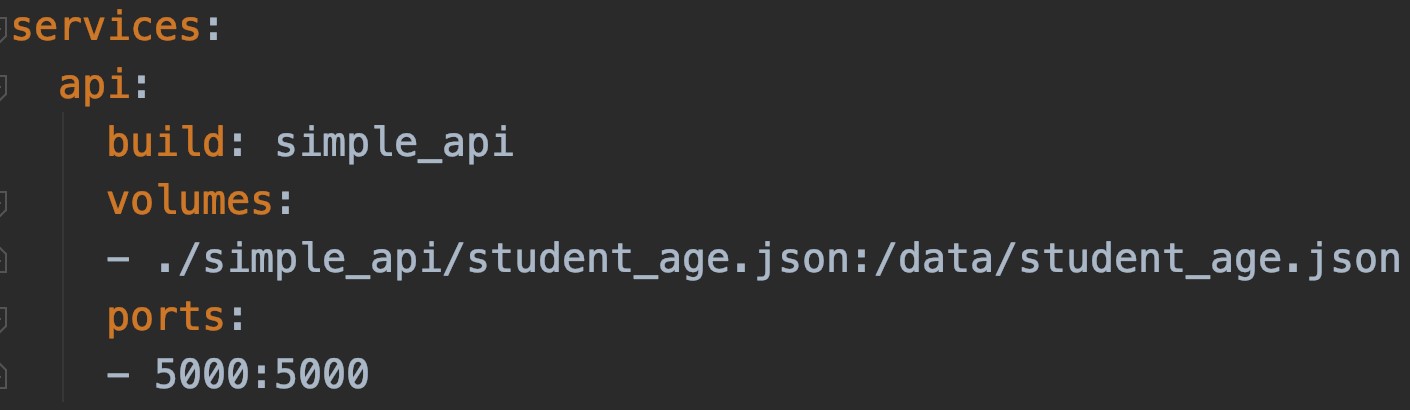
Et définissons ensuite nos services.

Le premier service que nous allons définir est « API »

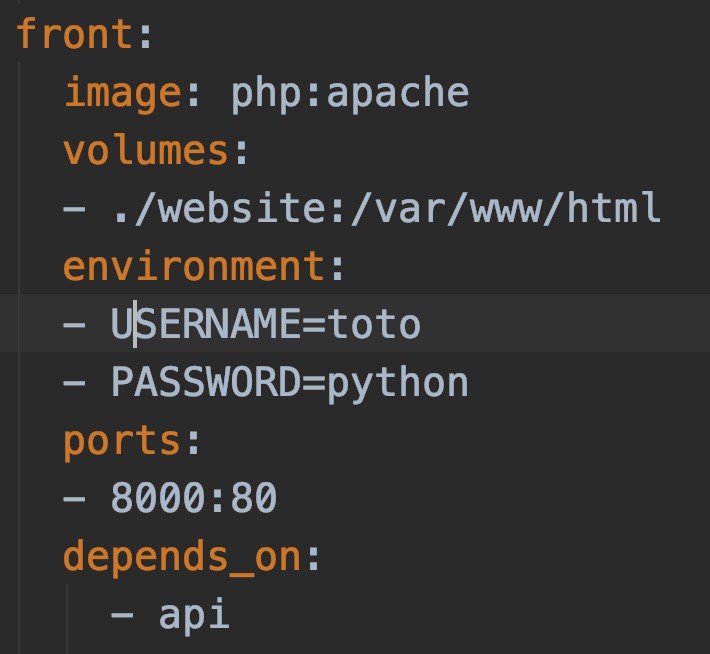


Il n’est pas utilie d’exposer le port de notre API dans la mesure ou SEUL LE FRONT doit pouvoir accéder à l’API.

Toutefois, on l’exposera dans un premier dans pour une phase de test sur le port 5000



Passons maintenant au service FRONT qui contient notre site internet.



Comme expliqué précédemment, notre service est basé sur l’image php mode apache ( php:apache ) et est par défaut, configuré pour servir le dossier /var/www/html, ce pourquoi nous montons le dossier ./website dans le dossier /var/www/html.

Concernant les variables d’environements, nous prenons soins de remplir les variables USERNAME, et PASSWORD comme indiqué dans la documentation des développeurs.

Nous précisons le port binding (à savoir ici : 8000 vers le port 80 du serveur web).

Et nous terminons par gérer la dépendances au service API avec le depends\_on, cela implique, que notre conteneur front démarrera uniquement une fois que le service API sera en ligne.

La dernière action qui nécessite une action, est celle de la configuration de l’adresse du serveur d’API :



Nous utilisons ici : « api :5000 » étant donné qu’au sein d’un même réseau docker-compose, tout les conteneurs peuvent se joindre via leur nom de service déclaré dans le dockercompose.yml

*Automatisation des processus*

Afin de gagner un temps précieux et soulager les équipes INFRA, nous allons automatiser le processus de livraison en automatisant 2 processus :

* Construction des conteneurs
* Déploiement des conteneurs sur la production

Nous utiliserons Jenkins afin d’automatiser ces processus et nous déploierons notre propre Registry docker afin de stocker les images construites.

Nous déployons donc la registry docker conformément aux documentations de Docker

Une image contenant texte

Description générée automatiquement

Je précise que je déploie un redis, pour optimiser les Performances de la registry.