

Table des matières

[Introduction 3](#_Toc516435334)

[Etape 1 : Static HTTP Server avec apache httpd 3](#_Toc516435335)

[Image Docker 3](#_Toc516435336)

[Arborescence 3](#_Toc516435337)

[Dockerfile 3](#_Toc516435338)

[Page web 3](#_Toc516435339)

[Etape 2 : Serveur http dynamique avec express.js 4](#_Toc516435340)

[JSON retourné par l’application 4](#_Toc516435341)

[Arborescence du répertoire contenant la configuration serveur dynamique 4](#_Toc516435342)

[Dockerfile 4](#_Toc516435343)

[Contenu du fichier index.js 5](#_Toc516435344)

[Etape 3 : Reverse proxy avec apache 5](#_Toc516435345)

[Arborescence 5](#_Toc516435346)

[Fichier de configuration 6](#_Toc516435347)

[Dockerfile 6](#_Toc516435348)

[Etape 4 : requête AJAX avec JQuery 7](#_Toc516435349)

[Emplacement dans le code html du serveur apache\_php 7](#_Toc516435350)

[Création du fichier students.js 7](#_Toc516435351)

[Nécessité du reverse proxy 8](#_Toc516435352)

[Etape 5 : Configuration d’un reverse proxy dynamique 9](#_Toc516435353)

[Arborescence du répertoire contenant la configuration reverse proxy 9](#_Toc516435354)

[Contenu du fichier php 9](#_Toc516435355)

[Execution du script php 9](#_Toc516435356)

[Lancement de l’image du reverse proxy 10](#_Toc516435357)

[Load balancing 10](#_Toc516435358)

[Modification de la configuration du reverse proxy 10](#_Toc516435359)

[Modification du Dockerfile 11](#_Toc516435360)

[Lancement du reverse proxy 11](#_Toc516435361)

[Load balancing avec sticky session 12](#_Toc516435362)

[Configuration du reverse proxy 12](#_Toc516435363)

[Fonctionnement du sticky session 13](#_Toc516435364)

[Balancer Manager 14](#_Toc516435365)

# Introduction

Dans le cadre de notre dernier laboratoire de RES, il nous a été donné pour tâche de construire une infrastructure web complète. Pour cela, il nous est demandé de mettre en place un serveur http apache pour le serveur http ainsi qu’un deuxième serveur pour le reverse proxy et fianlement un serveur nodeJs utilisant le framework express-js afin de créer une application web dynamique.

# Etape 1 : Static HTTP Server avec apache httpd

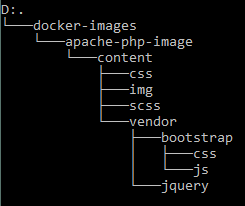
La première partie de ce laboratoire consiste à construire une image docker contenant un serveur apache permettant d’afficher une page web.

## Image Docker

Afin de mettre en place l’image du serveur apache on a décider de suivre les indications de la vidéo et avons opté pour une image de projet php fonctionnant avec apache à l’exception près qu’on a pris la versions 7.0-apache au lieu de la version 5.6-apache.

## Arborescence

Pour cela on a créé un répertoire une arborescence des répertoires suivant.



Le répertoire « docker-images » contient le répertoire de notre première image docker, c’est-à-dire « apache-php-image ». Ce dernier répertoire contient le Dokerfile ainsi que le répertoire « content » qui contient le contenu de la page HTML qui devra être copié dans le dossier « /var/www/html/ » du conteneur.

## Dockerfile

Le Dockerfile de ce serveur indique que la version du conteneur est « php :7.0-apache » et que le contenu du dossier content est copier dans le répertoire « /var/www/html » du conteneur.



## Page web

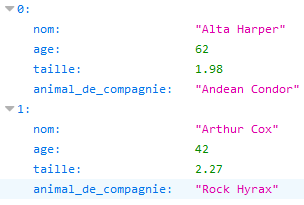
Le template de la page web a été copié sur le site http://startbootstrap.com et modifié à notre convenance. Son contenu se trouve dans le répertoire « content » présenté dans le chapitre précédent.

# Etape 2 : Serveur http dynamique avec express.js

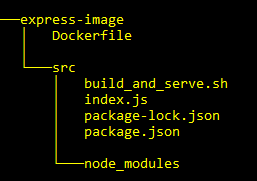
La deuxième étape consiste à créer un serveur générant un contenu aléatoire et retournant un contenu JSON.

## JSON retourné par l’application

L’application retourne des étudiants possédant un nom, un âge, une taille ainsi qu’un animal de compagnie.



## Arborescence du répertoire contenant la configuration serveur dynamique



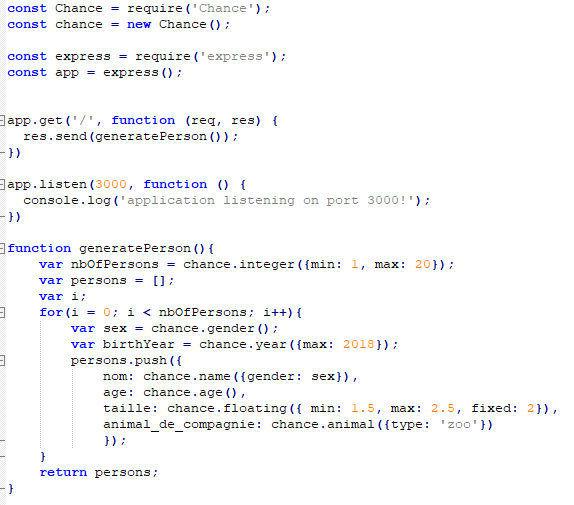
## Dockerfile

Comme l’indique le Dockerfile on a décidé d’utiliser la version 9.11 de nodeJS pour mettre en place cette nouvelle image. De plus, le contenu du répertoire src qui contient le fichier « index.js » est copier dans le répertoire /opt/app du conteneur et se dernier est appelé au lancement du conteneur grâce à la commande « CMD ["node", "/opt/app/index.js"] »



## Contenu du fichier index.js

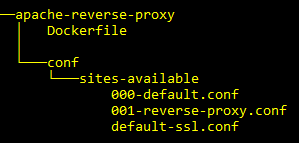
On peut observer dans le fichier « index.js » ci-dessous que le serveur retourne le contenu d’un tableau JSON qui est généré de manière aléatoire grâce à la bibliothéque javascript « chance ». On constate aussi que le contenu est envoyé au client lorsque qu’on accède au serveur avec l’url « http://adresseServeur:3000/ »



# Etape 3 : Reverse proxy avec apache

La troisième étape consiste à créer un reverse proxy dont le rôle de router les requêtes externes vers le bon serveur interne, en l’occurrence le serveur statique ou le dynamique.

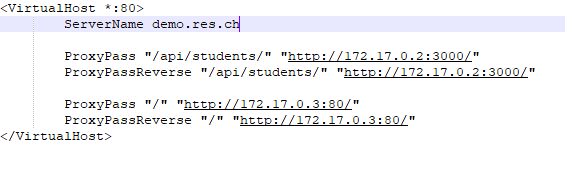
## Arborescence



## Fichier de configuration

Afin de mettre en place se serveur on a créé un nouveau fichier de configuration nommé « 001-reverse-proxy.conf ».

Cette configuration route les requêtes envoyées au nom de serveur « demo.res.ch » vers le serveur dynamique si l’url se termine par « /api/students/ » ou vers le serveur statique si l’url se termine par « / ».



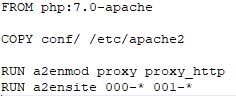
Ce fichier de configuration présente le grand inconvenant de contenir les adresses des serveurs en dur.

## Dockerfile

On voit dans le Dockerfile que la version 7.0 du conteneur php a été utilisé et que les fichiers de configuration son chargé dans le dossier « /etc/apache2 » du conteneur.

Ce Dockerfile contient l’instruction « RUN a2enmod … » qui permet d’activer des modules du serveur apache. Dans ce cas ce sont les module proxy et proxy\_http qui permettent de faire du reverse proxy.

De plus, il y a aussi l’instruction « RUN a2ensite .. » qui permet d’activer des fichiers de configuration du serveur apache.



# Etape 4 : requête AJAX avec JQuery

Cette quatrième partie consiste à mettre à jour de manière dynamique le contenu de la page web avec les données provenant du serveur dynamique.

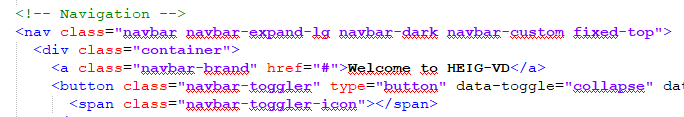
## Emplacement dans le code html du serveur apache\_php

En premier lieu on à décider de placer notre affichage dynamique en haut à gauche de la page web.

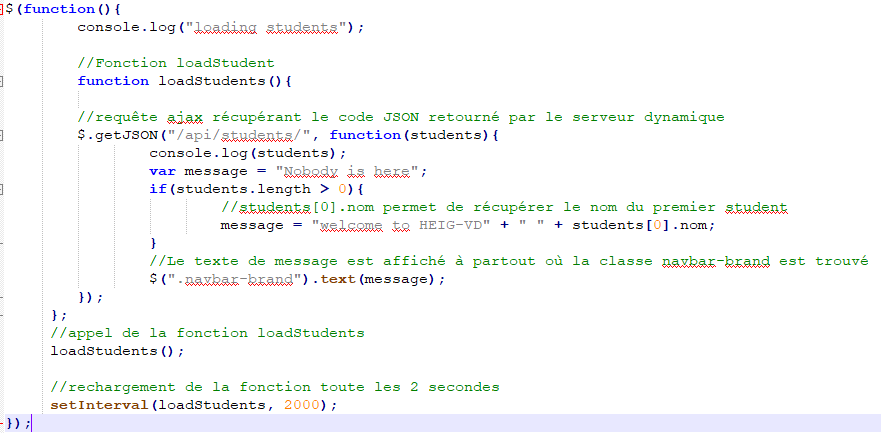


## Création du fichier students.js

Avant de créer le fichier student.js on a tout d’abord identifier le nom de la classe à laquelle nous voulons faire apparaître notre texte dynamique. On a choisi de la placé notre texte à la place de celui de la class « navbar-brand »



Ensuite on à créer le fichier « student.js » que vous trouverai ci-dessous :



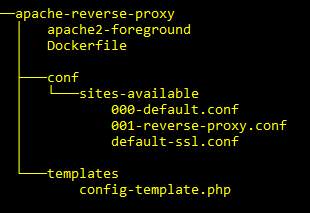
## Nécessité du reverse proxy

Lors de l’execution de cette étape la présence du reverse proxy fut indispensable car c’est lui qui permet au serveur static de faire appel à l’url « /api/students » du serveur dynamique.

# Etape 5 : Configuration d’un reverse proxy dynamique

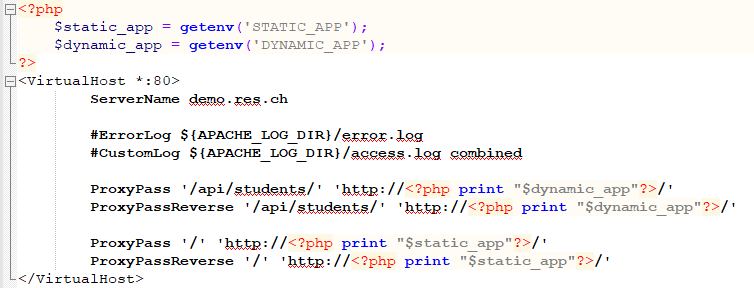
La 5ème étape consiste à remplacer la configuration statique du reverse proxy par une configuration dynamique utilisant des variables d’environnement et du php afin qu’il n’y ait pas besoin de rebuild l’image docker du reverse proxy à chaque changement d’adresse IP.

## Arborescence du répertoire contenant la configuration reverse proxy



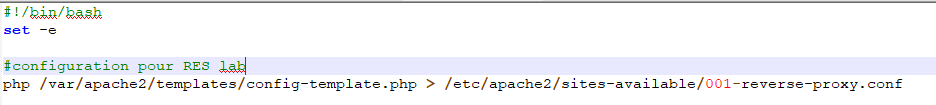
## Contenu du fichier php

L’image ci-dessous montre le contenu du fichier php (« config-template.php ») qui permet de générer de manière dynamique le contenu du fichier « 001-reverse-proxy.conf ».



## Execution du script php

Afin que le script php soit exécuté au lancement de l’image docker, il a été nécessaire d’ajouter une commande au début du fichier « apache2-foreground » (voir image ci-dessous)



Cette ligne de commande execute le script php du fichier « config-template.php » et enregistre le fichier résultant à la place du fichier « 001-reverse-proxy.conf ».

## Lancement de l’image du reverse proxy

Maintenant que le script php est capable d’allé chercher les adresses ip dans les variables d’environnement « STATIC\_APP » et « DYNAMIC\_APP » il ne reste plus qu’à le créer au sein du container. La commande -e « nom\_variable env » passé à la commande « docker run … » permet d’effectuer cela.

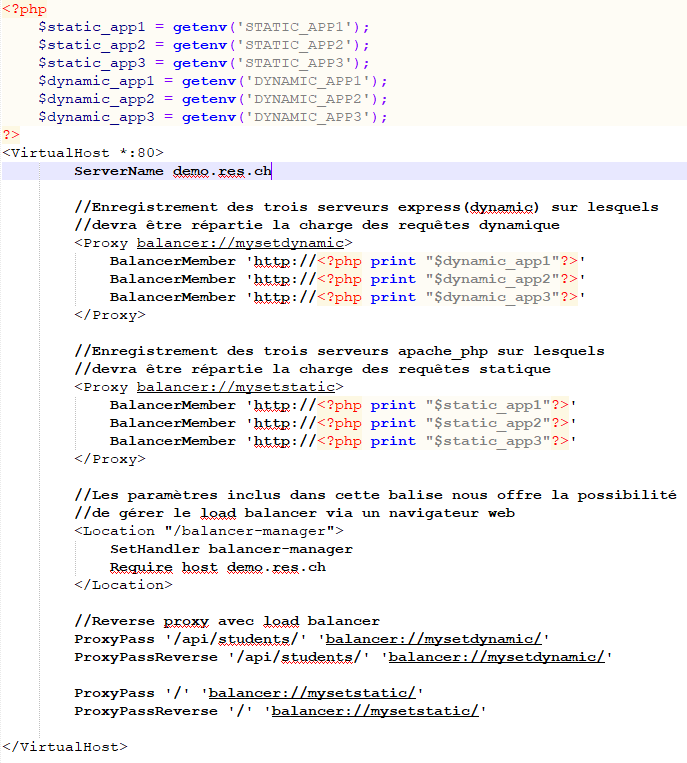
Suite à l’explication précédente, le lancement du container se fait de la manière suivante :

# Load balancing

La partie load balancing consiste à répartir la charge des requêtes entrante sur plusieurs serveurs. Pour cela, on doit démarrer plusieurs serveurs apache\_php en parallèle ainsi que plusieurs serveurs express (NodeJS).  
Par conséquent on à choisi de lancer trois serveurs apache\_php et trois serveurs express.

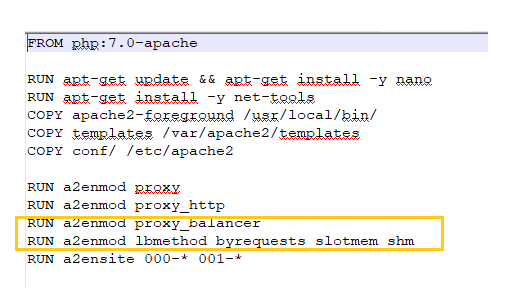
## Modification de la configuration du reverse proxy

La majeure partie des modifications à lieu dans la configuration du reverse proxy. En effet une fois tous nos serveurs lancés se sera à lui de gérer la répartition des charges.  
On a pu trouver toute la documentation nécessaire pour bien réaliser cette étape dans la documentation du module poxy balancer du site internet d’apache.



## Modification du Dockerfile

Il a été nécessaire d’ajouter les modules entourés ci-dessous au serveur du reverse proxy, afin que le load balancing soit fonctionnel.



## Lancement du reverse proxy

A présent il est nécessaire de créer six variables d’environnement afin de pouvoir faire fonctionner le load balancer correctement.

Voici une commande d’exemple :

« docker run -d -e STATIC\_APP1=172.17.0.7:80 -e STATIC\_APP2=172.17.0.6:80 -e STATIC\_APP3=172.17.0.5:80 -e DYNAMIC\_APP1=172.17.0.4:3000 -e DYNAMIC\_APP2=172.17.0.3:3000 -e DYNAMIC\_APP3=172.17.0.2:3000 --name apache\_rp -p 8080:80 res/apache\_rp »

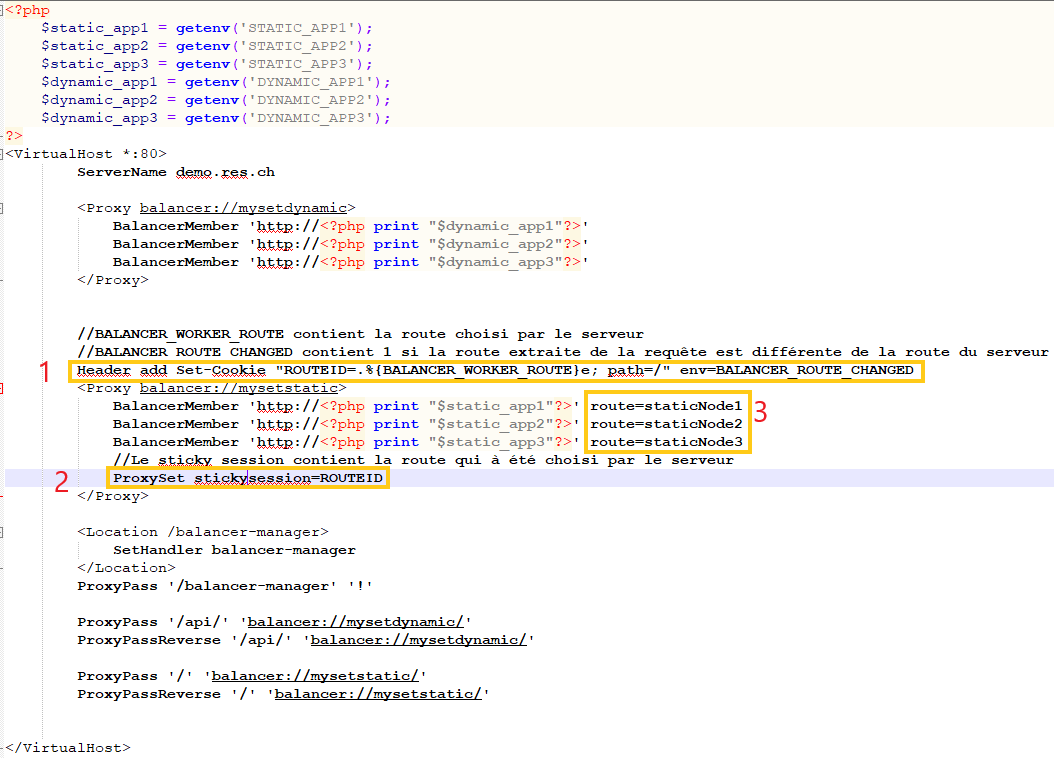
# Load balancing avec sticky session

Le sticky session permet de sauvegarder dans un cookie le chemin du serveur vers lequel reverse proxy (via le load balancing) à dirigé la requête que le client lui a envoyé et de toujours rediriger le client vers ce serveur.

## Configuration du reverse proxy

Afin de pouvoir gérer les requêtes sur les serveur apache\_static avec un sticky session on a commencé par ajouter la ligne 1 voir ci-dessous qui permet de créer un cookie nommé « ROUTEID » contenant comme valeur un nom de route défini dans l’encadré n°3.

La ligne 2 permet d’attacher le « ROUTEID » au sticky session.



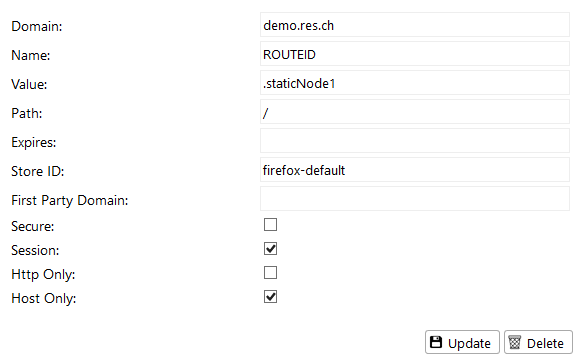
## Fonctionnement du sticky session

Une fois le sticky session mis en place, on constate que l’adresse du serveur static reste toujours fixe tandis que celle du serveur dynamique varie de temps à autre. Cela confirme que les requêtes de la session en cours sont toujours redirigées vers le même serveur statique.



Afin de changer cela il suffit d’éditer le cookie « ROUTEID » et de modifier la valeur de la route vers laquelle les requêtes doivent être dirigés.

Par exemple, il suffit de changer la valeur staticNode1 par staticNode2 pour que les requête soient redirigé vers l’adresse statique 172.17.0.6 (pour la configuration actuel).



## Balancer Manager

Le load manager permet d’observer la différence entre les serveurs dynamique géré en round-robin et les serveurs statique géré avec les sticky sessions.

On constate sur l’image ci-dessous que staticNode1 à été élu un plus grand nombre de fois que les deux autres serveurs statiques car c’est la valeur du cookie qui définissait la route qu’allait prendre les requêtes jusqu’au serveur (encadré 2). Tandis que la répartition de la charge des serveur dynamique (encadré 1) est homogène car elle est répartit en round-robin.

