# Labo HTTP Infrastructure

## Partie 1 : Serveur http static avec apache

Objectifs : Installer un serveur apache via une image docker. Ecrire le Dockerfile correspondant et ajouter du contenu basique à une page HTML.

Manipulations : Pour cette partie, nous sommes allés chercher une image officielle apache sur Docker hub. Nous avons choisi l’image php. Dans le Dockerfile, la première ligne (FROM) spécifie le nom de l’image ainsi que sa version. La deuxième ligne permet de copier des fichiers de note système de fichier local dans le système de fichier de l’image. On aurait pu démarrer l’image officielle sans utiliser de Dockerfile, afin d’effectuer quelques tests préliminaires (afin de devoir éviter de toujours reconstruire l’image). Dans ce cas, on va directement chercher l’image sur le site officiel.

Entrer dans le container et accéder aux fichiers de config:

docker exec -it nom\_image /bin/bash

*cd /etc/apache2*

fichier apache2.con*f*

*cd sites-available*

Dans le fichier 000-default.conf, un ligne spécifie le chemin par défaut /var/www/html

Commandes utiles :

*docker build -it res/apache\_php .*

*docker run -p 9090 :80 res/apache\_php*

## Partie 2 : Serveur http dynamique avec javascript

Objectifs : Ecrire une application web dynamique avec javascript. Cette application doit retourner une chaîne de caractères sous format JSON après une requête GET.

Manipulations : Dans le Dockerfile, les deux premières lignes ont le même but qu’au point 1. La dernière ligne représente la commande à effectuer lors du lancement du contnair. Il va donc effectuer la commande node index.js, qui va donc exécuter le script.

Npm init sert à créer le fichier package.json qui contient les informations relatives à l’application (entre-autres, les dépendances).

Notre application dynamique renvoie une nouvelle identité sous format JSON. Pour tous les espions qui doivent changer de couverture, ils y trouveront un nouveau prénom, nom, adresse email ainsi qu’une phrase de 6 mots. Cette phrase est leur phrase de secours en cas de problème.

## Partie 3 : Apache reverse proxy

Objectifs : Développer un reverse proxy qui va être le lien entre nos deux containers et le reste du monde. Toutes les requêtes vont passer par le proxy.

Manipulations : Dans cette configuration, les adresses IP des containers ont dû être hardcodées : c’est une mauvaise pratique puisque rien de garanti que, lorsque l’on va relancer les container docker, les adresses IP attribuées seront les mêmes. Il faut donc vérifier ce point à chaque fois que l’on relance l’application.

Le proxy n’a pas de contenu à proprement parlé, il doit juste rediriger le client suivant la requête effectuée. C’est pourquoi dans cette partie, nous avons dû modifier un fichier de configuration se trouvant dans le dossier /etc/apache2/sites-available. Nous avons créé un nouveau fichier de configuration pour que le proxy sache ce qu’il doit retourner au client selon la requête de ce dernier. Il existe deux mots-clés pour définir une route du proxy à l’hôte virtuel souhaité. Il s’agit de proxypass et de proxypassreverse. Pour que le fichier de configuration soit utilisé, il faut ajouter le module correspondant au Dockerfile avec la commande a2enmod (car, par défaut, les deux mots-clés ci-dessus sont inconnus). Finalement, nous avons dû modifier le fichier hosts dans nos machines afin de lier l’adresse IP de la machine virtuelle à un serveur DNS. Ceci nous permet d’effectuer des requêtes via le navigateur et pas juste telnet.

## Partie 4 : Ajax avec JQuery

Objectif : Utiliser JQuery pour effectuer des requêtes AJAX. Le but est de pouvoir faire automatiquement des requêtes vers le service dynamique.

Manipulations : Pour relancer les containers, on les redémarre dans un ordre particulier. Puisque pour le moment, les adresses sont hardcodées dans le fichier de configuration, on espère qu’ils gardent les mêmes adresses IP.

Manipulation à faire si le paquet node\_modules est absent : npm install.

Nous avons modifié la page html principale de note application pour y mettre un script javascript, qui effectue des requêtes auprès de notre container express, effectué à l’étape 2. Après avoir créé ce petit script, nous avons fait en sorte que la page principale se recharge toutes les deux secondes.

Les navigateurs utilisent ce qu’on appelle *same-origin policy*. Cela signifie que si un script est exécuté avec un certain nom de domaine, alors il ne pourra faire des requêtes que vers ce même nom de domaine, et donc vers la même machine.

Dans le cas de notre labo, on veut faire des requêtes vers la machine express qui fournit du contenu dynamique alors que la requête vient de la machine qui fournit le contenu statique. Il ne serait donc à priori pas possible de faire les choses ainsi.

Nous avons donc mis en place un reverse proxy, pour que la machine qui exécute le script ait l'impression que l'origine du script ainsi que la destination de ses requêtes respectivement vient et va au même point réseau, alors qu'en fait ce sont des machines différentes cachées derrière un intermédiaire (le reverse proxy).

## Partie 5 : configuration dynamique d’un reverse proxy

Objectif : Modifier le fichier de configuration du reverse proxy afin de se débarrasser des adresses ip hardcodées dans le fichier.

Manipulations : Il est possible, depuis la ligne de commande lors du lancement d’un container, de modifier les variables d’environnement de ce dernier. Ceci est très pratique, puisque nous pouvons donc avoir accès, depuis l’extérieur, à du contenu interne. Avec cette pratique, nous allons pouvoir modifier le fichier de configuration contenant les adresses IP des deux machines de manière dynamique. Pour se faire, nous devons exécuter des commandes pendant le lancement du reverse proxy. En allant regarder à la fin du Dockerfile de php (l’image utilisée pour lancer nos containers), on voit que la commande apache2 est lancée en premier plan à la fin du fichier. L’idée est d’exécuter notre propre script avant de lancer cette ultime commande. Nous avons donc repris ce script, apache2-foreground, afin d’ajouter notre code ; nous avons fait en sorte que les IP passées en paramètres de la commande lors du lancement du container modifient les variables d’environnements correspondantes. Finalement, nous avons écrit un script en php pour rediriger les adresses IP vers les variables d’environnement, et non plus vers les adresses hardcodées en dur. Pour le bon fonctionnement de la manipulation, l’étape ultime a été de copier ce fichier de configuration php dans le container sous le chemin /var/apache2/template et d’écraser le fichier de configuration par ledit fichier php dans le document apache2-foreground modifié.