Laboratoire RES - Infrastructure HTTP

Objectif

L'objectif de ce laboratoire est de créer une infrastructure web permettant gérer des requêtes HTTP afin de servir du contenu statique et dynamique.

Etape 1: Serveur statique HTTP avec apache httpd

La première étape du laboratoire consiste à créer, dans un container docker, un serveur statique HTTP permettant de servir une page web (fabriquée à partir d'un template Bootstrap) lorsqu'on effectue un GET /. Ceci va récupérer le fichier index.html et le servir au client.
L'adresse par défaut du container est 172.17.0.2 sur le port 80. On peut y accéder depuis l'extérieure de la machine Docker en effectuant un port mapping (paramètre -p port_externe:port_interne). Les données HTML, CSS et javascript sont copiées dans le container avec la commande COPY placée dans le Dockerfile, ce qui permet d'y avoir accès dans le dossier /var/www/html/ du container (et donc de les servir au clients).

Etape 2: Serveur dynamique HTTP avec express.js

La deuxième étape consiste à écrire une application web dynamique qui parle HTTP et qui fournit des données JSON, plus précisément un tableau de villes et de hashtags généré aléatoirement, à nos différents clients.

Utilisation du framework express.js afin de répondre à des requêtes qui arrivent et envoyer un payload JSON. Par défaut, express se met sur écoute sur le port 3000. On pourrait aussi se mettre à l'écoute sur le port 80 mais ce n'est pas important car un serveur HTTP peut écouter sur n'importe quel port.

Afin de générer du contenu aléatoire, nous avons utilisé le package chance de npm afin de créer un tableau où chaque ville est associé à un hashtag.

Dans le DockerFile, nous exécutons le script index.js qui s'occupe de répondre à une requête grâce à express.

Malheureusement, nous avons vu au moment de faire le rapport, c'est-à-dire à la fin de toutes les vidéos et de toutes les manipulations, qu'il ne fallait pas envoyer une liste d'étudiants mais quelque chose généré par nos soins. Donc nous avons pris le temps de changer le code sur les différentes branches mais pas de changer tous les noms de fichiers. Cela implique que par exemple, notre .js modifiant la page web toutes les deux secondes

(étape 4) s'appelle toujours students.js mais les données reçues sont tout de même des villes et des hashtags.

Etape 3: Reverse proxy avec configuration statique

Dans cette étape, on configure le serveur apache en mode reverse proxy afin d'avoir un seul point d'entrée. Il permet d'atteindre des containers Docker sans port mapping en passant par un container qui lui est lancé avec un port mapping. Les deux containers des étapes 1 et 2 sont lancés et peuvent être atteints depuis la machine docker (avec un 'docker-machine ssh' puis un telnet sur leur adresses IP).
Pour les atteindre depuis l'extérieur, on crée un serveur apache qui a un fichier de configuration permettant de rediriger des requêtes vers d'autres adresses IP (modules proxy et proxy_http de apache2)
Avec la commande ProxyPass on peut rediriger une requête vers une adresse et un port. Par exemple, on peut rediriger les requêtes "/api/students/" sur l'adresse du container dynamique fournissant des noms de villes et des 'hashtags' aléatoires et la requête "/" sur l'adresse du container statique fournissant une page web.
Le problème avec la configuration de cette étape est que les adresses IP des containers sont codés en dur dans le fichier de configuration, donc si ces containers n'ont pas la bonne adresse on ne pourra pas les atteindre avec le reverse proxy.
Pour se connecter via un navigateur internet, il faut placer dans le fichier "hosts" de sa machine l'IP et le nom de domaine (demo.res.ch) correspondant au reverse proxy pour permettre la résolution DNS par notre navigateur. On pourra ensuite se connecter via l'URL "http://demo.res.ch:8080" (sauf si on map le port 80 du container sur le port 80 de notre machine, celui ci étant le port par défaut HTTP, dans ce cas on peut omettre la spécification du port dans l'URL).

Etape 4: Requêtes AJAX avec JQuery

Dans cette étape, on implémente des requêtes Ajax en arrière plan de demo.res.ch:8080 en utilisant la librairie qui s'appelle JQuery. Elle sont envoyées en arrière plan vers le backend dynamique afin de récupérer les données et mettre à jour une partie de l'interface utilisateur de notre site en mettant à jour le DOM, c'est-à-dire la structure d'une hiérarchie qui décrit une page..

Afin de prouver que les requêtes AJAX sont envoyées via le navigateur, il suffit d'ouvrir le web dev tools du navigateur et d'observer les requêtes être envoyé chaque deux secondes.

Dans cette partie, nous avons aussi mis à jour les images docker afin d'installer vim au moment de la création de l'image.

Etape 5: Configuration dynamique du reverse proxy

C’est avec cette étape qu’on peut recevoir dynamiquement l’adresse des containers avec des variables d'environnement: on créé un script PHP qui récupère les variable STATIC_APP et DYNAMIC_APP qu’on défini au lancement de apache_rp et les place dans la commande ProxyPass. On peut ainsi passer les adresses des containers apache_php et express_students plutôt que les définir en dur.
On peut donc désormais lancer autant de containers statiques et dynamiques que l’on veut avant de lancer le reverse proxy, tant que l’on passe les bonnes adresses lors du lancement du reverse proxy avec le paramètre -e (qui permet de modifier une variable d’environnement).