HTTP Infrastructure

Rapport de laboratoire

Introduction

Ce document présente le rapport du laboratoire de RES HTTP Infrastructure dont le but est de créer une infrastructure HTTP. Ce laboratoire a été réalisé en utilisant la VM Vagrant.

Étape 1

Répertoire GitHub

Le contenu de l'étape 1 et suivantes se trouve sur le répertoire GitHub. Les fichiers de l'étape 1 se trouvent sur la branche fb-apache-static.

https://github.com/mathieumonteverde/Teaching-HEIGVD-RES-2017-Labo-HTTPInfra

qui est un fork du répertoire RES de base.

Template Web

Pour le modèle de site Web Bootstrap, j'ai utilisé le même que tous les autres sites Bootstrap.

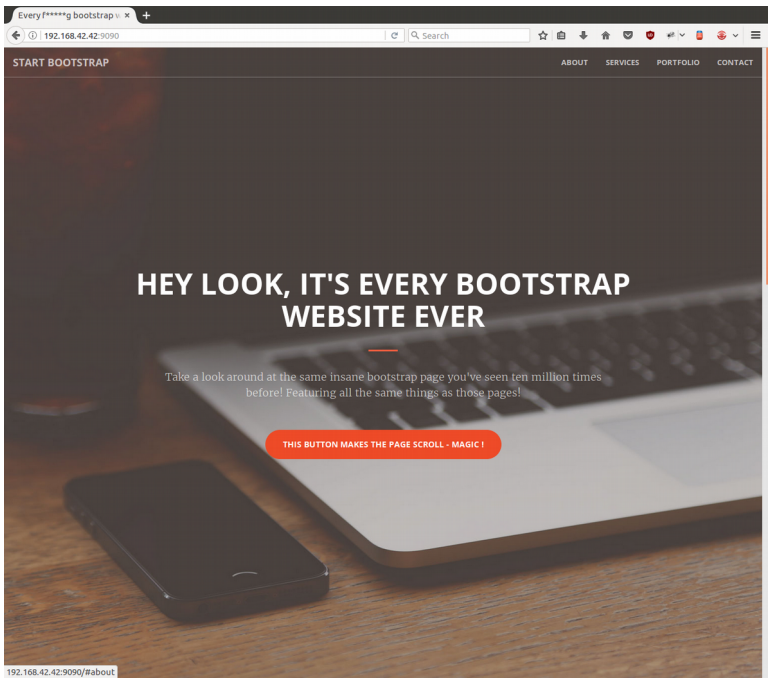

Illustration 1: Every Bootstrap theme ever.

Instructions du Dockerfile

Le Dockerfile contient deux lignes d'instructions :

```
FROM php:7.0-apache

COPY content/ /var/www/html/
```

L'instruction FROM permet de spécifier une image de base pour les instructions suivantes. Ici l'image officielle php/apache version 7.0.

L'instruction COPY ordonne de copier le contenu du dossier contents dans /var/www/html dans le système de fichiers du container Docker qui sera créé avec l'image basée sur ce Dockerfile.

Fichiers de configuration apache

Pour afficher les fichiers de configuration dans un container Docker (issu de l'image PHP/apache utilisée pour ce laboratoire) en exécution, il suffit d'effectuer les étapes suivantes :

- `docker exec -it <nom_container> /bin/bash`
- `cd /etc/apache2/`

Le dossier /etc/apache2/ contient les différents fichiers de configuration du serveur apache

Configuration apache

La configuration apache utilisée pour cette étape est celle proposée par défaut par l'image Docker téléchargée. Le système de fichier pour le serveur apache commence dans le dossier /var/www/html.

Étape 2

Répertoire GitHub

Le répertoire GitHub est le même que l'étape 1. Les fichiers de l'étape 2 se trouvent sur la branche fb-express-dynamic.

Génération dynamique de contenu

La génération dynamique consiste à créer un objet JSON représentant une bataille de deux Pokemons. La bataille est représentée par un nom d'arène, et les noms des deux Pokemons qui se battent. Les noms des arènes et des Pokemons sont stockés dans des fichiers JSON et chargés par une application Node.js dans le container Docker. L'application utilise le framework Express.

Le résultat est envoyé sous la forme :

```
{
        "arena": "Accumula Town",
         "firstFighter": "Beedrill",
        "secondeFighter": "Nidorina"
}
```

Configuration

Étape 3

Répertoire GitHub

Le répertoire GitHub est le même que l'étape 1. Les fichiers de l'étape 3 se trouvent sur la branche fb-apache-reverse-proxy.

Inconvénients de la configuration statique

La configuration statique d'un reverse proxy consiste à écrire manuellement les règles de redirection de requêtes vers les différentes adresses des applications proposant différents types de services. Dans le cas de ce laboratoire, il s'agit d'un serveur apache qui fournit du contenu statique et d'une application Node.js qui fournit des données JSON dynamiques.

L'inconvénient majeur d'une configuration statique est qu'il faut connaître à l'avance les adresses IPs des internes des machines proposant les services. Il est donc très fragile et tout changement dans la distribution des adresses demande de réécrire les règles de redirection (et de reconstruire l'image Docker).

Configuration

La configuration de cette étape est plus complexe que les précédentes. Tout d'abord, la configuration du reverse proxy en utilisant un serveur apache se trouve en local selon cette arborescence :

```
conf
-- sites-available
   -- 000-default.conf
   -- 001-reverse-proxy.conf
```

Configuration 000-default.conf

```
<VirtualHost *:80>
</VirtualHost>
```

Cela permet de refuser l'accès à toute requête sur le site logique par défaut.

Configuration 001-reverse-proxy.conf

```
<VirtualHost *:80>
   ServerName demo.res.ch

   ProxyPass "/api/pokemons/" "http://172.17.0.3:3000/"
   ProxyPassReverse "/api/pokemons/" "http://172.17.0.3:3000/"



   ProxyPass "/" "http://172.17.0.2:80/"
   ProxyPassReverse "/" "http://172.17.0.2:80/"
</VirtualHost>
```

ServerName permet de spécifier l'en-tête Host du protocole HTTP qu'on désire traiter. Les commandes ProxyPass et ProxyReversePass permettent de rediriger les requêtes vers les différentes machines Docker en fonction de l'URL demandé.

Configuration du Dockerfile

```
FROM php:7.0-apache
COPY conf/ /etc/apache2

RUN a2enmod proxy proxy_http
```

```
RUN a2ensite 000-* 001-*
```

On utilise la commande COPY pour copier les fichiers de configuration apache au bon endroit dans le container Docker, et les deux instructions RUN permettent d'activer les modules de proxy, et les configurations de sites.

Étape 4

Répertoire Github

Le répertoire GitHub est le même que l'étape 1. Les fichiers de l'étape 4 se trouvent sur la branche fb-ajax.

Envoi de requête AJAX

Les requêtes AJAX sont envoyées périodiquement toutes les 3 secondes. On peut observer les requêtes et les réponses en ouvrant la console Firebug (dans le navigateur) et en regardant les paquets envoyé dans l'onglet Net.

Utilité du reverse proxy

Sans la mise en place du reverse proxy, les requêtes AJAX devraient être effectuées vers une autre machine (avec une adresse IP différente et donc un nom de domaine différent) que le site statique. Ainsi ces actions seraient interdites par le navigateur car elles briseraient la *Same-origin policy*.

Configuration

La configuration des images Docker et des serveurs apache est la même qu'à l'étape 4. Le code du site statique a simplement été modifié pour effectuer les requêtes AJAX périodiquement. Un fichier de scripts pokemons.js (js/pokemons.js) contient ce code.

Étape 5

Répertoire Github

Le répertoire GitHub est le même que l'étape 1. Les fichiers de l'étape 5 se trouvent sur la branche fb-dynamic-config.

Configuration dynamiques

La configuration du reverse proxy est faite de manière dynamique comme dans les tutoriels, en regardant les adresses IPs des différents containers après les avoir lancé, et en passant ces adresses par paramètres de variables d'environnement quand on crée le container reverse proxy. Cela permet donc de ne pas avoir besoin de reconstruire l'image à chaque changement d'adresse IP.

Scénario de test

Pour tester la configuration, il suffit de :

1. Lancer plusieurs fois des containers res/apache_php en arrière-plan
2. Lancer un container res/apache_php en arrière-plan en lui donnant un nom
3. Lance plusieurs fois des containers res/express-pokemons en arrière-plan
4. Lancer un container res/express-pokemons en arrière- en lui donnant un nom
5. Regarder les adresses IPs des containers spéciaux
6. Lancer le container res/reverse-proxy en lui passant en a paramètre les adresses IPs mentionnées plus haut.
7. Tester avec le navigateur Web le bon fonctionnement du tout

Configuration

La configuration fait appel à trois fichiers principaux. Le *Dockerfile*, qui permet de construire l'image, le script *apache2-foreground*, qui permet d'ajouter des actions avant de construire l'image php:5.6 de Docker, et le fichier *config-template.php* qui permet, lorsqu'il est exécuté, de construire le fichier de config du serveur proxy.

DockerFile

Le Docker file effectue les mêmes instructions qu'à l'étape précédente, en copiant en plus les différents fichiers de configuration nommés ci-dessus dans le système de fichier du container.

apache-foreground

Ce fichier reprend le code qui se trouve dans le fichier du même nom dans l'image php:5.6 sur Github, en y ajoutant simplement l'exécution du script PHP *config-template.php*, et en stockant le résultat dans le fichier de configuration apache du serveur reverse proxy (*001-reverse-proxy.conf*).

config-template.php

Ce script PHP utilise les variables d'environnement fournies en paramètre au lancement du container pour créer le contenu du fichier de configuration *001-reverse-proxy.conf*.