Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

hepia
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Projet de semestre
Reverse proxy avec Varnish

INGÉNIERIE DES TECHNOLOGIES DE L'INFORMATION

Auteurs :
Cédric Gerber

Professeur du projet :
Gérald LITZISTORF

17 mars 2016

h e p i a
département I T I
ingénierie des technologies
de l'information

Année académique 2015-2016
Projet de semestre

DISCIPLINE DU PROJET DE SEMESTRE

TITRE DU PROJET

Descriptif :

Les architectures de sécurité basées sur le modèle reverse proxy sont très appréciées.

L'objectif de ce travail consiste à étudier la solution Open Source https://www.varnish-cache.org/ afin de la mettre en œuvre avec un serveur web https://www.nginx.com/

Travail demandé :

L'étude comprend les étapes suivantes :

1) Etude théorique pour comprendre l'architecture, le fonctionnement du cache, les paramètres à ajuster et les règles à définir.

2) Première mise en œuvre sur système RedHat avec administration en ligne de commande.

3) A mi-projet, proposition d'étude spécifique en fonction des résultats précédents.

4) Rapport comprenant :
   - l'étude théorique du point 1),
   - la marche à suivre détaillée du point 2),
   - l'étude spécifique du point 3),
   - les avantages de varnish,
   - les principales difficultés rencontrées,
   - une conclusion.

Sous réserve de modification en cours du projet de semestre

Candidat :
M. GERBER CEDRIC
Département : ITI

Professeur(s) responsable(s) :
Litzistorf Gérald

En collaboration avec :
Projet de semestre soumis à une convention de stage en entreprise : non
Projet de semestre soumis à un contrat de confidentialité : non

Timbre de la direction

Table des matières

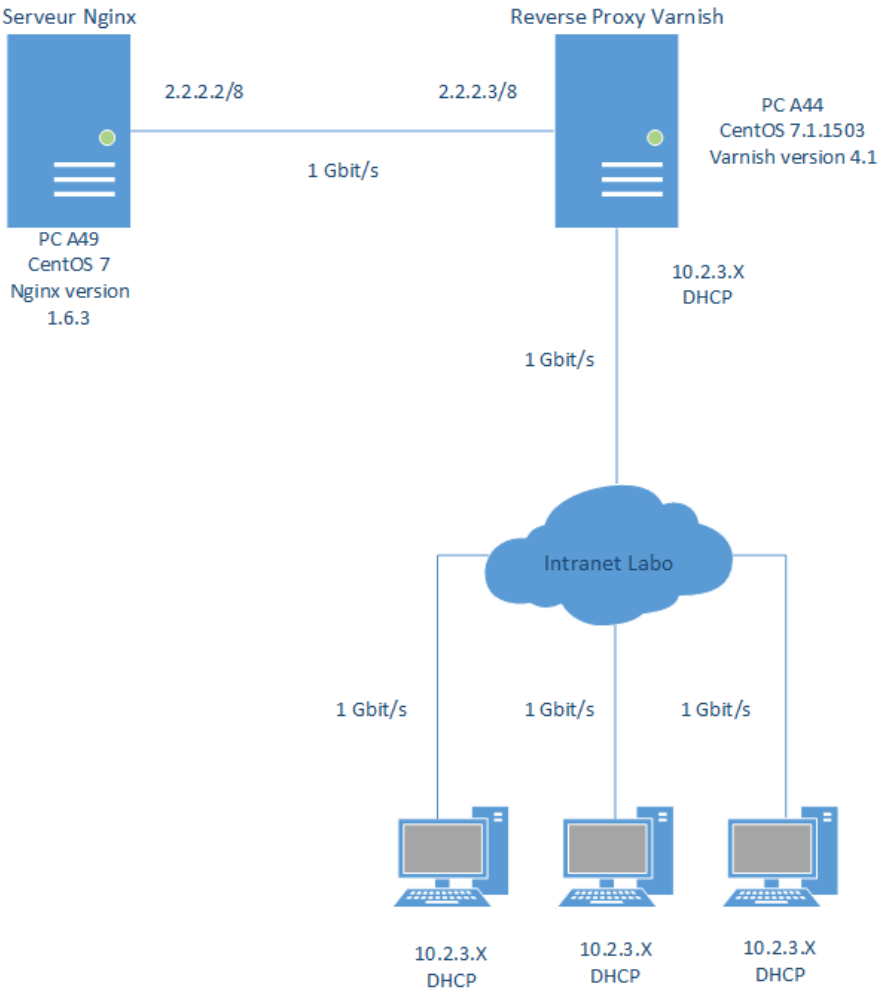
1 Description du projet de semestre

1.1 Introduction

De nos jours, de plus en plus de personnes ont accès à internet et utilisent les services que ce dernier propose. De ce fait, les sites populaires ont de plus en plus de visites et cela peut créer de nombreux problèmes. Les utilisateurs peuvent entre autre avoir des ralentissements lorsqu'il demande une ressource ou même, dans le pire des cas, les services peuvent être hors-service. Pour certaines entreprises, un site hors-service peut engendrer des pertes d'argent et de clients. Les reverses Proxy ont donc été inventé afin d'empêcher que ces problèmes se produisent.
Durant ce projet de semestre, je vais mettre en place un reverse proxy Varnish avec un serveur web Nginx et expliquer comment j'ai procédé. Je vais également détailler toutes les configurations nécessaires pour le déploiement de ces deux systèmes.

1.2 Schéma du réseau

Voici le schéma de principe de mon projet comprenant les débits binaires, les adresses IP utiles, les systèmes d'exploitation ainsi que les versions de Varnish et de Nginx.



1.3 Possibilités de Varnish

1. Mettre en cache des données afin de pouvoir décharger le serveur web et servir plus rapidement les requêtes.
2. Renforcer la sécurité du serveur web en rajoutant une couche d'abstraction supplémentaire entre Internet et le serveur. En rajoutant une machine supplémentaire, on diminue les risques d'hacking du serveur web grâce à un firewall placé dans le reverse proxy. Les hackeurs ont donc une difficulté en plus, car il doivent dans un premier temps, hacker le reverse proxy avant de pouvoir attaquer le serveur web.
3. Permettre la répartition de charges entre plusieurs machines et détecter lorsqu'un serveur est surchargé ou qu'il ne répond plus afin d'aiguiller les requêtes vers un serveur qui fonctionne.
4. Permettre la compression et la décompression de données. Cela permet d'économiser de la bande passante entre le serveur web et Varnish. Mais également, de gagner de la place dans le cache du serveur Varnish et d'économiser de la bande passante entre Varnish et le client.
5. Créer à l'aide de varnishncsa un fichier "log" centralisé dans le serveur Varnish qui nous permet d'avoir le même format de "log" pour tous les serveurs web. Et également de décharger les serveurs d'application de la production de ces "log".

1.4 Serveur web Nginx

Nginx est un serveur asynchrone, c'est-à-dire que le serveur web change d'état pour pouvoir gérer les différentes connexions au lieu de créer un nouveau processus pour chaque client se connectant. De plus, ce serveur est lancé avec un nombre prédéfinit de processus, à l'inverse d'Apache par exemple, qui crée un processus différent lors de chaque connexion. Nginx permet donc de contrôler la montée en charge. Chaque requête est divisée en plusieurs petites tâches afin d'avoir un multiplexage plus efficace que celui des autres serveurs web.
Ce serveur a également la possibilité d'être utilisé comme reverse proxy ou comme proxy pour la messagerie IMAP ou POP3. Je ne vais pas utiliser ces deux fonctionnalités du système Nginx mais simplement la fonction serveur web. Nginx est surtout employé lorsqu'il faut faire face à beaucoup de trafic.

2 Principe de la mise en cache Varnish

De nos jours, il y a deux grands types de contenu dans les serveurs web : premièrement des ressources dites "statiques" et deuxièmement des ressources dites "dynamiques". Les ressources statiques sont simplement envoyées du serveur web au poste client telles quelles, comme une page HTML, une image, un fichier audio, etc. Ces ressources n'ont pas besoin d'être exécutées par un programme du serveur. Au contraire, les ressources dynamiques doivent être créées par un tierce programme avant d'être envoyées au demandeur de la ressource. Par exemple, lorsque le serveur va consulter une base de données ou un annuaire avant d'envoyer la ressource.
Les serveurs web classiques comme Apache arrivent très bien à répondre à des demandes de type statique et gèrent très bien la montée en charge avec ce genre de demandes. Par contre, ces serveurs ont un peu plus de difficultés dès qu'ils doivent régir des données dynamiques.
Dès cet instant, les reverse proxy entrent en action. Le principal but de Varnish est de mettre en cache des contenus dynamiques.
Dans les prochaines sous-sections, j'explique comment Varnish met en cache ces contenus dynamiques ainsi que ses configurations possibles.

2.1 Types de mise en cache

Tout d'abord, Varnish permet de faire deux types de mise en cache :

1. Dans un fichier (file) : il place tous les éléments devant être mis en cache dans un fichier. Il est ensuite possible de configurer la taille et l'emplacement de ce dernier. Le système d'exploitation se charge de le mettre dans la mémoire RAM s'il en a la possibilité. Néanmoins, si on redémarre ou qu'on arrête Varnish cette méthode de mise en cache ne conserve pas les données.
2. Dans la RAM (malloc) : Varnish met tous les éléments devant être mis en cache dans la mémoire RAM du reverse proxy où la taille à allouer de cette mémoire peut être configurée. Pour cela Varnish utilise la fonction malloc() de la librairie standard de C.

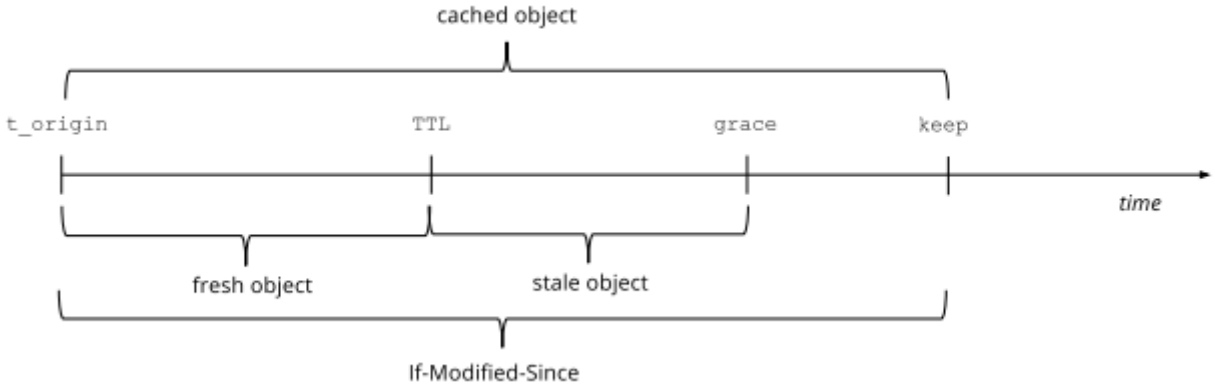
Si le cache doit être contenu entièrement ou principalement dans la mémoire, il faut utiliser malloc. Autrement, on utilise file si le cache dépasse la mémoire physique disponible.
Ces deux choix sont à configurer dans le fichier *varnish.params* dans la variable VARNISH_STORAGE.
Pour finir, chaque objet stocker a besoin d'un "overhead" d'environ 1 kB car Varnish garde une trace du cache.

2.2 Stockage des objets

Varnish voit les ressources à stocker comme des objets. Ces derniers sont conservés grâce à une clé de hashage dans un arbre de hash. Cette clé est créée par défaut grâce à l'entête HTTP et à l'URL. On peut toutefois créer une clé à partir d'autres informations, comme par exemple à l'aide des cookies. De plus, Varnish peut stocker plusieurs éléments derrière une seule clé et choisir l'élément à envoyer suivant les préférences du client.

2.3 Durée de vie d'un objet

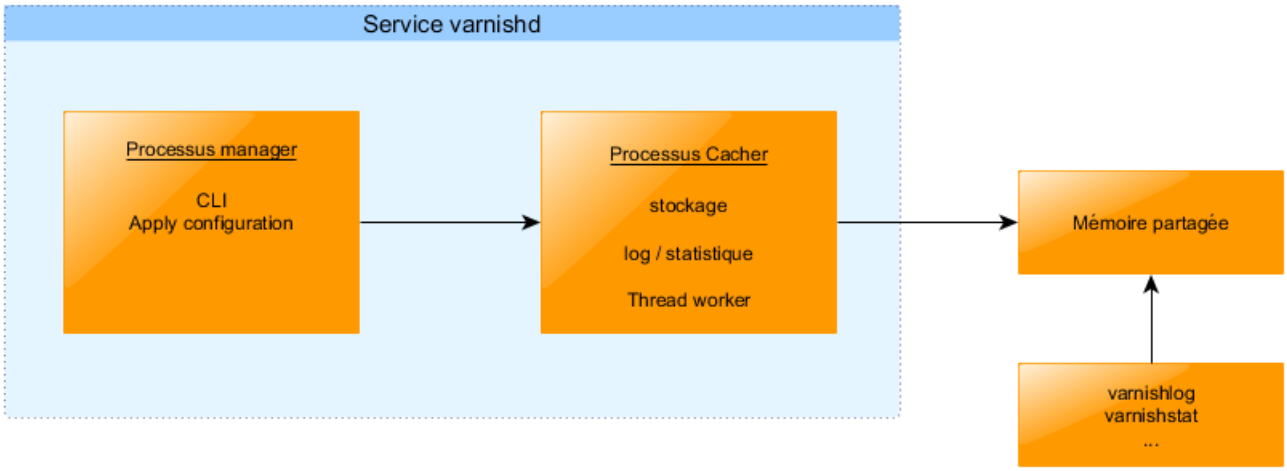
Voici un schéma qui explique la durée de vie d'un objet se situant dans le cache :



Un objet en cache a un attribut t_origin qui correspond au temps auquel il a été mis en cache. Un objet à trois autres attributs : TTL, grace et keep. Lorsque le temps courant d'un objet est dans le TTL, il est considéré comme *Fresh object*. Les objets qui sont considérés comme *Stale object* ont leur temps courant qui se trouve dans la période de temps TTL et grace. Les objets dans t_origin et keep sont utilisés lorsqu'on applique des conditions avec le champ If-Modified-Since de l'entête HTTP.
L'addition des attributs TTL, grace et keep est égale au temps auquel un objet reste dans le cache.

2.4 Architecture du service *varnishd*

Voici un schéma qui représente l'architecture des processus et de la mémoire du service varnishd :



Grâce à la commande *ps* de linux, il est facile de vérifier si le service se comporte vraiment comme sur le schéma. Il y a bien deux processus comme décrit le diagramme ci-dessus. Voici le résultat de cette commande :

```
root      6650  0.0  0.0 108392  2044 pts/2    S+   mar11   0:50 varnishstat
root     20706  0.1  0.0 101308  1016 pts/3    S+   18:09   0:04 varnishlog
varnish  22300  0.0  0.0 124820  5232 ?        Ss   18:51   0:00 /usr/sbin/varnishd -P /
    var/run/varnish.pid -f /etc/varnish/default.vcl -a :80 -T 127.0.0.1:6082 -S  -s
    malloc,256M -p thread_pool_min=10 -p thread_pool_max=500 -p thread_pool_timeout=300 -
    t 5
varnish  22301  0.0  1.5 304108 118172 ?        Sl   18:51   0:00 /usr/sbin/varnishd -P /
    var/run/varnish.pid -f /etc/varnish/default.vcl -a :80 -T 127.0.0.1:6082 -S  -s
    malloc,256M -p thread_pool_min=10 -p thread_pool_max=500 -p thread_pool_timeout=300 -
    t 5
root     22370  0.0  0.0 112664   952 pts/1    S+   18:54   0:00 grep --color=auto
    varnish
```

resultat_ps.txt

2.5 Le processus enfant : Le *Cacher*

Le *Cacheur* a plusieurs fonctions :

1. Écouter les requêtes des clients.
2. Manager les threads "worker".
3. Mettre dans le cache les données à stocker.
4. Mettre dans les "logs" les trafiques.
5. Mettre à jour les compteurs pour les statistiques.

Étant donné que le *Cacheur* peut écouter sur une adresse IP publique et sur des ports connus, le système s'expose à des personnes malveillantes. C'est pourquoi, pour des raisons de sécurité, ce processus appartient à un utilisateur non-privilégié. De plus, ce dernier n'a pas de communication arrière vers le processus parent se nommant le *Manager*.

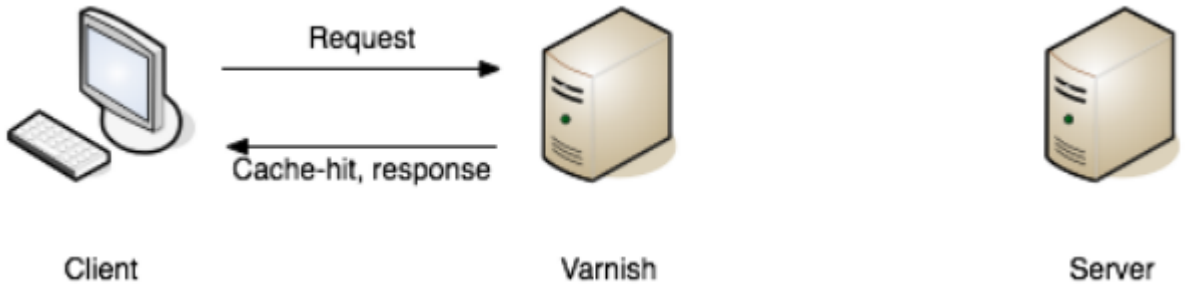
2.6 Champs de l'entête HTTP de la mise en cache

Pour mettre les objets en cache, Varnish utilise les champs de la mise en cache de l'entête HTTP . Les plus utilisés sont :

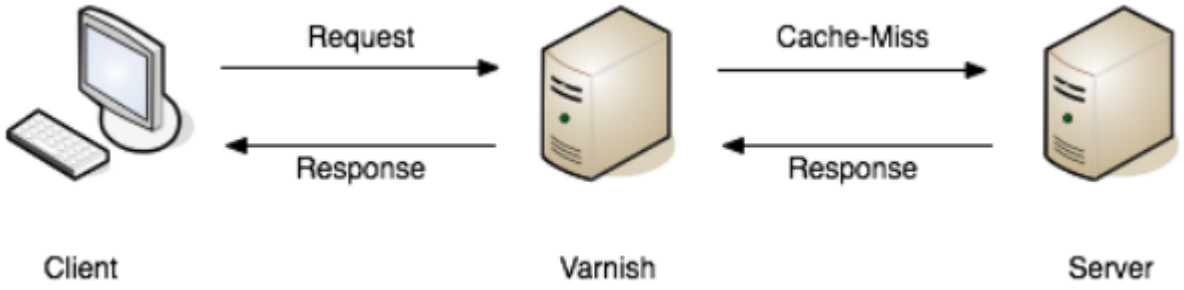
- Expires
- Cache-Control
- Etag
- Last-Modified
- If-Modified-Since
- If-None-Match
- Vary
- Age

2.7 Cache-hit & cache-miss

Lorsqu'une donnée demandée correspond à une donnée dans le cache cela donne un *Cache-hit*. Voici une figure qui illustre ce cas :



En revanche, lorsqu'une donnée demandée ne correspond pas à une donnée dans le cache cela fait un *Cache-miss*. Voici une figure qui explique ce qu'il se passe lors d'un *Cache-miss* :



2.8 Construction des réponses depuis le cache

Il faut trois conditions pour que Varnish créer une réponse depuis le cache pour un client :

1. Qu'un objet étant en cache correspond correctement : *cache-hit*
2. Que la méthode de la requête HTTP soit autorisée par rapport à l'objet demandé.
3. Que la "fraîcheur" de l'objet en cache soit acceptable.

A chaque fois que Varnish trouve une correspondance entre un objet en cache et une requête, il évalue s'il doit faire appel au serveur web ou si la donnée qui se trouve dans le cache est cohérente. La plupart des règles que Varnish utilise sont basées sur les champs de cache de l'entête HTTP.

2.9 VCL - Les sous-routines

Toutes ces sous-routines sont nécessaires pour gérer correctement les objets à mettre en cache ou non.

2.9.1 *vcl_recv*

Cette sous-routine est la première à être utilisée lorsqu'une requête d'un client arrive. Elle décide si Varnish doit répondre à la requête, comment y répondre et quel serveur web il doit utiliser dans le cas où il y en a plusieurs.
Elle a un certain nombre de mot-clés de retour :

- hash : Continue le processus avec l'appel de la routine *vcl_hash*.
- pass : Switch en mode "pass".
- pipe : Switch en mode "pipe".
- synth : Appelle la sous-routine *vcl_synth*.
- purge : Purge l'objet.

2.9.2 *vcl_pipe*

vcl_pipe est appelée quand on se trouve dans le mode "pipe". Dans ce dernier, les données sont transmises du client au serveur web et vice-versa sans qu'elles soient mises en cache, jusqu'à ce que la connexion soit fermée.
Cette sous-routine peut retourner deux mot-clés :

- pipe : Procède avec le mode "pipe".
- synth : Appelle la sous-routine *vcl_synth*.

2.9.3 *vcl_pass*

Cette sous-routine est appelée lorsqu'on se trouve dans le mode "pass". Dans ce mode, les requêtes sont envoyées directement au serveur web puis au client sans que les données soient mises en cache. Les demandes ultérieures sur la même connexion sont toutefois traitées normalement.
vcl_pass a plusieurs mot-clés de retour :

- fetch : Initie une requête au serveur web.
- restart : Redémarre la transaction.
- synth : Appelle la sous-routine *vcl_synth*.

2.9.4 *vcl_hash*

vcl_hash est appelée après *vcl_recv* afin de créer le hash de la requête utilisée ensuite comme la clé de l'objet.
Cette sous-routine a un seul mot-clé de retour :

- lookup : Consulte l'objet en cache. Il appelle la sous-routine *vcl_purge* s'il y a "purge" comme retour de *vcl_recv*. En revanche, si le résultat de la consultation dans le cache est "hit" ou "miss" ou "pass", il appelle respectivement *vcl_hit* ou *vcl_miss* ou *vcl_pass*.

2.9.5 *vcl_hit*

Quand une recherche dans le cache d'un objet est réussie, cette sous-routine est appelée.
Elle peut retourner plusieurs mot-clés :

- deliver : Délivre l'objet au client. Si les données sont dépréciées, une action d'actualisation des données est lancée.
- miss : Rafraichît l'objet en cache à partir du serveur web malgré le succès de la recherche dans le cache.
- pass : Switch en mode "pass".
- restart : Restart la requête.
- synth : Appelle la sous-routine *vcl_synth*.

2.9.6 *vcl_miss*

vcl_miss est appelée lorsque la recherche dans le cache n'a pas aboutie. Son but est de décider s'il faut tenter ou pas, de récupérer les données sur le serveur web.
Cette sous-routine peut terminer avec différent mot-clés :

- fetch : Récupère l'objet demandé sur le serveur web.
- pass : Switch en mode "pass".
- restart : Restart la requête.
- synth : Appelle la sous-routine *vcl_synth*.

2.9.7 *vcl_purge*

Dès que la purge a été effectuée cette sous-routine est appelée. Elle n'a que deux mot-clés de retour :

- restart : Restart la requête.
- synth : Appelle la sous-routine *vcl_synth*.

2.9.8 *vcl_synth*

Cette sous-routine est appelée dans le cas où elle doit délivrer un objet synthétique. Un tel objet est généré en VCL et n'est pas récupéré sur le serveur web.
Elle peut retourner deux mot-clés différents :

- deliver : Délivre directement l'objet au client sans appeler *vcl_deliver*.
- restart.

Voici un exemple d'objet synthétique :
<html><body>< !– Here goes a more friendly error message. –> </body></html>

2.9.9 vcl_backend_fetch

Cette sous-routine est appelée avant d'envoyer une requête au serveur web. Si une requête destinée au serveur web a besoin d'être modifiée, c'est généralement dans cette sous-routine que cela est possible.
Elle peut retourner deux mot-clés :

- fetch : Demande l'objet au serveur web.
- abandon : Abandonne la demande.

2.9.10 vcl_backend_response

vcl_backend_response est appelée après avoir reçue avec succès la réponse du serveur web.
Elle peut retourner plusieurs mot-clés :

- deliver : Crée un objet de cache mis à jour si la réponse est 304. Sinon, le corps de l'objet est demandé au serveur web et on initie l'envoie de celui-ci à tous les clients en attente.
- abandon : Abandonne la requête.
- retry : Recommence la requête vers le serveur web.

2.9.11 vcl_backend_error

Si la requête sur le serveur web n'a pas aboutie ou si le maximum d'essai de demande est dépassé cette sous-routine entre en action. Un objet synthétique est généré en VCL.
Deux mot-clés de retour sont possibles :

- deliver : Délivre et met éventuellement en cache l'objet défini dans la sous-routine comme s'il était extrait du serveur web.
- retry : Recommence la requête vers le serveur web.

3 Mise en place de Varnish sur CentOS

Dans ce chapitre, je vais vous expliquer comment installer Varnish sur la distribution CentOS.

3.1 Installation en ligne de commande

1. Voici les lignes de commande nécessaire pour l'installation de Varnish :
   ```
   [root@localhost cedric]# yum install epel-release

   [root@localhost cedric]# rpm --nosignature -i https://repo.varnish-cache.org/
   redhat/varnish-4.1.el7.rpm

   [root@localhost cedric]# yum install varnish
   ```
2. Ensuite, il faut activer le service et le démarrer :
   ```
   [root@localhost cedric]# systemctl enable varnish

   [root@localhost cedric]# systemctl start varnish
   ```

3.2 Configuration du firewall

Puis, il faut configurer le firewall de la machine afin qu'il autorise les connexions sur HTTP. Pour cela, j'ai écrit un script en bash afin qu'au démarrage de l'ordinateur, ce dernier fasse automatiquement les configurations désirées :

```
#!/bin/sh
iptables -F
iptables -t filter -P INPUT DROP
iptables -t filter -P FORWARD DROP
iptables -t filter -P OUTPUT DROP
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -t filter -A OUTPUT -p tcp --dport 80 -j ACCEPT
iptables -t filter -A INPUT -p tcp --dport 80 -j ACCEPT
echo "firewall configuré pour Varnish"
```

firewall_cedric.sh

Pour que le script soit exécuté au démarrage, quelques manipulations sont nécessaires :

1. Créer son script
2. Faire en sorte que les commandes ou les scripts qui sont dans le fichier */etc/rc.d/rc.local* s'exécutent au démarrage :
   ```
   [root@localhost cedric]# chmod +x /etc/rc.d/rc.local
   ```
3. Placer le script dans le dossier */root*
4. Rendre le script exécutable :
   ```
   [root@localhost cedric]# chmod +x /root/firewall.sh
   ```
5. Ajouter la commande afin de lancer notre script dans le dossier */etc/rc.d/rc.local* :
   ```
   sh /root/firewall.sh
   ```

3.3 Configuration de Varnish

A ce stade, le reverse proxy fonctionne déjà, mais les règles de mise en cache sont celles par défaut.

3.3.1 Fichier de configuration *varnish.params*

"DEAMON_OPTS" est une variable de configuration du service varnishd. Elle se trouve dans */etc/varnish/varnish.params*.
Elle permet de configurer plusieurs paramètres :

1. "-a" : l'adresse IP et le port que le reverse proxy va écouter.
2. "-f" : l'emplacement du fichier de configuration VCL.
3. "-T" : l'adresse IP et le port d'administration.
4. "-t" : le TTL du cache.
5. "-w" : le nombre de threads minimum, le nombre de threads maximum et le temps maximum qu'un thread peut rester bloquer.
6. "-s" : le type de mise en cache effectué par le reverse proxy ainsi que la taille maximum que varnishd peut allouer pour la mise en cache.

Il y a également différentes variables globales dans ce fichier qui sont utilisées lors de l'exécution du service Varnish. On peut observer l'utilisation de ces variables dans le fichier varnish.service qui se situe dans */usr/lib/systemd/system/*. Il n'est pas possible de supprimer des variables globales dans le fichier varnish.params, mais de modifier le fichier varnish.service.
Il existe bien d'autres options, mais elles ne sont pas utilent pour ce projet. Toute ces dernières sont détaillées dans la man page de varnishd.

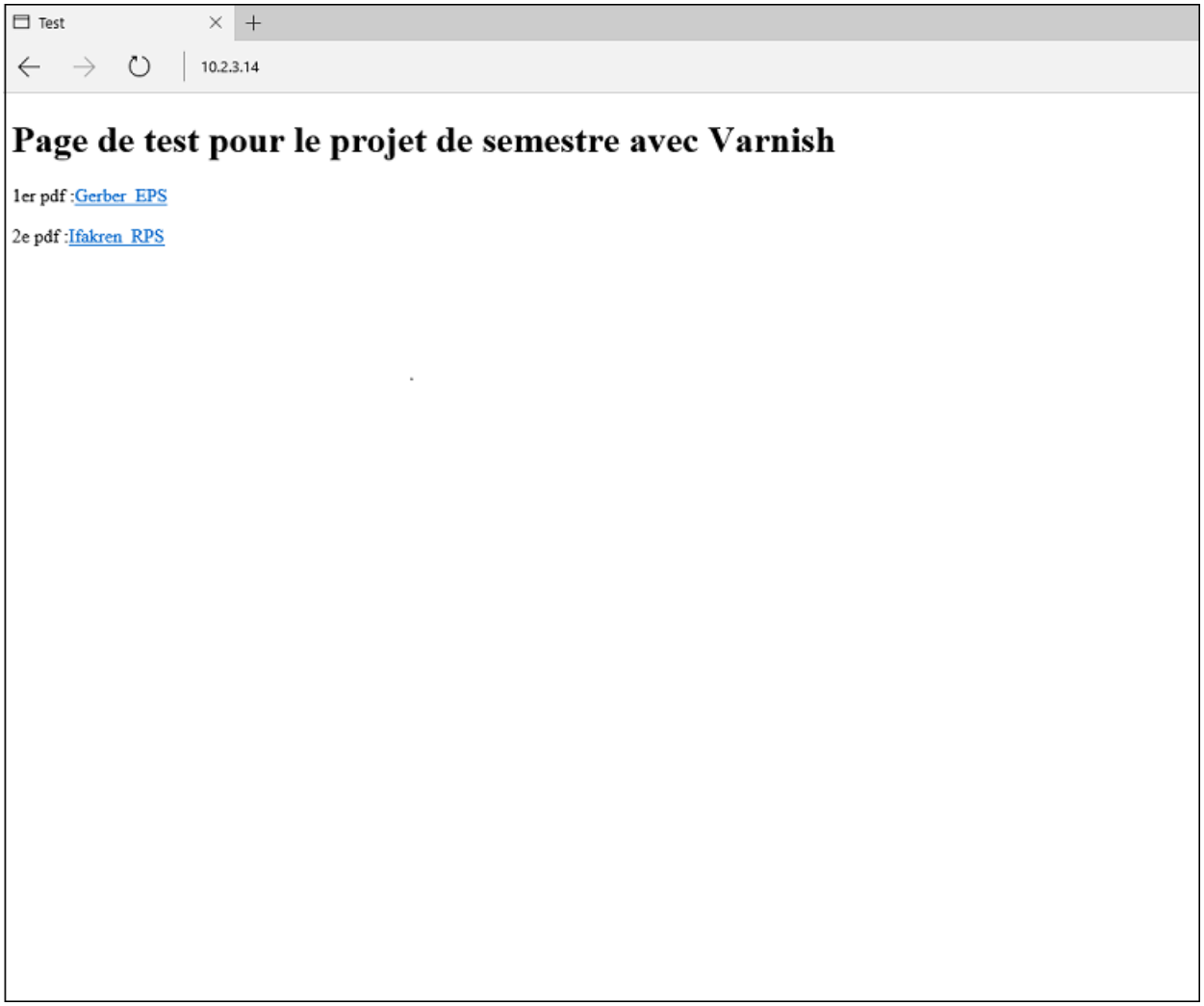
Voici le fichier varnish.params présent sur le reverse proxy :

```
# Varnish environment configuration description. This was derived from
# the old style sysconfig/defaults settings
# Set this to 1 to make systemd reload try to switch VCL without restart.
RELOAD_VCL=1

# Main configuration file. You probably want to change it.
VARNISH_VCL_CONF=/etc/varnish/default.vcl

# Default address and port to bind to. Blank address means all IPv4
# and IPv6 interfaces, otherwise specify a host name, an IPv4 dotted
# quad, or an IPv6 address in brackets.
# VARNISH_LISTEN_ADDRESS=192.168.1.5
VARNISH_LISTEN_PORT=80

# Admin interface listen address and port
VARNISH_ADMIN_LISTEN_ADDRESS=127.0.0.1
VARNISH_ADMIN_LISTEN_PORT=6082

# Shared secret file for admin interface
#VARNISH_SECRET_FILE=/etc/varnish/secret
VARNISH_SECRET_FILE=""
# Backend storage specification, see Storage Types in the varnishd(5)
# man page for details.
VARNISH_STORAGE="malloc,256M"

# User and group for the varnishd worker processes
VARNISH_USER=varnish
VARNISH_GROUP=varnish

# Other options, see the man page varnishd(1)
DAEMON_OPTS="-p thread_pool_min=10 -p thread_pool_max=500 -p thread_pool_timeout=300 -t 1
    "
```

varnish.txt

3.4 Mise en place d'un site web de test

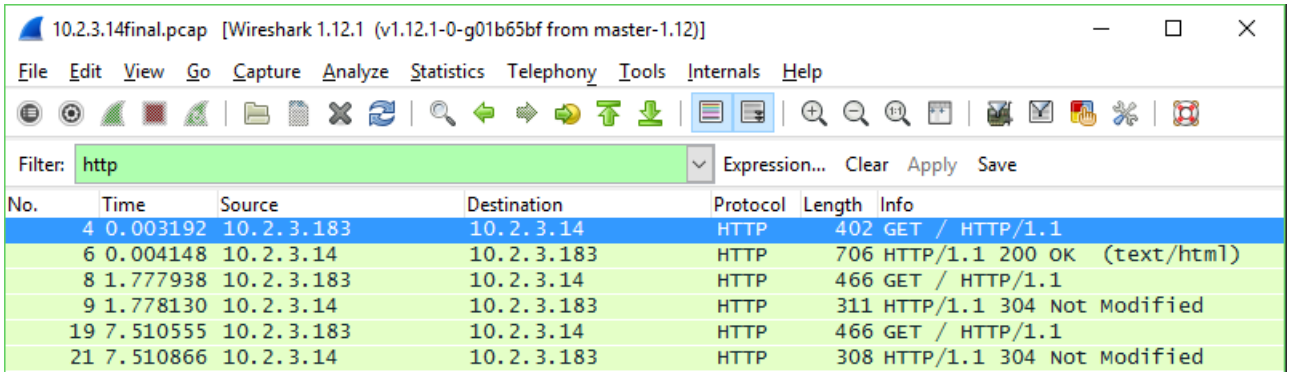
Pour effectuer quelques tests de performance et pouvoir comprendre le comportement de Varnish, j'ai créé une page HTML basique qui est composée de deux fichiers pdf. J'ai ensuite hébergé le tout sur mon serveur web Nginx. Le site web ressemble à ceci :



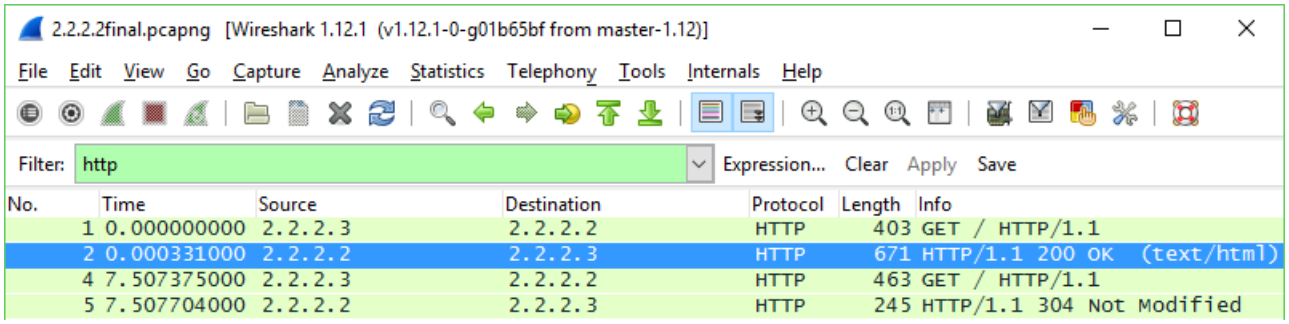
3.5 Analyse Wireshark

Pour mieux comprendre le comportement de Varnish, j'ai effectué deux analyses wireshark. Une du coté "publique" et une du coté serveur web.

Analyse côté "publique" :



Analyse coter serveur web :



Tout d'abord, les paquets numéro quatre et six de l'analyse du côté publique correspondent à la première requête effectuée. Étant la première requête, Varnish doit faire une demande au serveur web car, il n'a pas encore l'objet en cache. Cette demande correspond sur l'image ci-dessus aux paquets un et deux de l'analyse du côté serveur web.
Ensuite, lors de la requête qui correspond aux paquets sept et huit, l'objet demandé est déjà dans le cache. C'est pourquoi, Varnish ne va pas demander l'objet au serveur web, elle n'apparaît donc pas dans la capture côté serveur.
Pour finir, on observe qu'entre les paquets neuf et dix-neuf, il y a un écart de six secondes. Or, les objets en cache sont gardés seulement cinq secondes. Varnish est donc obligé de redemander l'objet une nouvelle fois au serveur web avant de pouvoir l'envoyer au client. C'est ce qu'on peut voir dans l'analyse côté serveur.

3.6 Analyse des logs

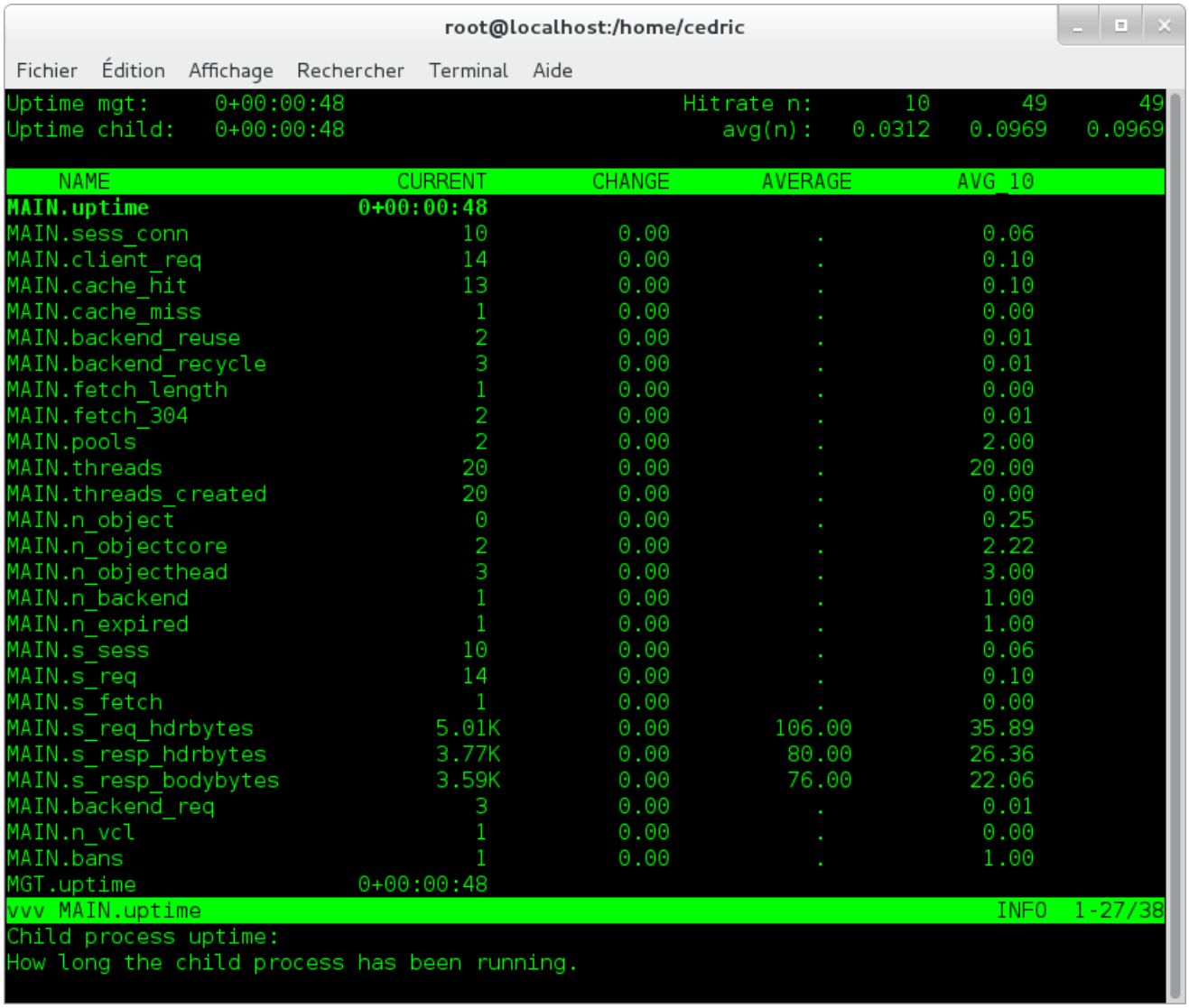
Un utilitaire nommé varnishlog est fourni lors de l'installation de Varnish et permet de récolter les "logs". Grâce à ceux-ci, il est possible de savoir quel est le contenu des champs du protocole HTTP et de connaître quelle sous-routine a été appelée par Varnish. Voici un exemple des "logs" :

```
1  *   << BeReq    >> 31
2  -   Begin          bereq 30 fetch
3  -   Timestamp      Start: 1457802579.871641 0.000000 0.000000
4  -   BereqMethod    GET
5  -   BereqURL       /
6  -   BereqProtocol  HTTP/1.1
7  -   BereqHeader    Host: 10.2.3.14
8  -   BereqHeader    User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko
       /20100101 Firefox/44.0
9  -   BereqHeader    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q
       =0.8
10 -   BereqHeader    Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
11 -   BereqHeader    Pragma: no-cache
12 -   BereqHeader    X-Forwarded-For: 10.2.3.183
13 -   BereqHeader    Accept-Encoding: gzip
14 -   BereqHeader    X-Varnish: 31
15 -   VCL_call       BACKEND_FETCH
16 -   VCL_return     fetch
17 -   BackendOpen    18 boot.default 2.2.2.2 80 2.2.2.3 49433
18 -   Timestamp      Bereq: 1457802579.871690 0.000049 0.000049
19 -   Timestamp      Beresp: 1457802579.872053 0.000412 0.000363
20 -   BerespProtocol HTTP/1.1
21 -   BerespStatus   200
22 -   BerespReason   OK
23 -   BerespHeader   Server: nginx/1.6.3
24 -   BerespHeader   Date: Sat, 12 Mar 2016 17:09:51 GMT
25 -   BerespHeader   Content-Type: text/html
26 -   BerespHeader   Content-Length: 368
27 -   BerespHeader   Last-Modified: Tue, 01 Mar 2016 16:32:38 GMT
28 -   BerespHeader   Connection: keep-alive
29 -   BerespHeader   ETag: "56d5c426-170"
30 -   BerespHeader   Accept-Ranges: bytes
31 -   TTL            RFC 5 10 -1 1457802580 1457802580 1457802591 0 0
32 -   VCL_call       BACKEND_RESPONSE
33 -   VCL_return     deliver
34 -   Storage        malloc s0
35 -   ObjProtocol    HTTP/1.1
36 -   ObjStatus      200
37 -   ObjReason      OK
38 -   ObjHeader      Server: nginx/1.6.3
39 -   ObjHeader      Date: Sat, 12 Mar 2016 17:09:51 GMT
40 -   ObjHeader      Content-Type: text/html
41 -   ObjHeader      Content-Length: 368
42 -   ObjHeader      Last-Modified: Tue, 01 Mar 2016 16:32:38 GMT
43 -   ObjHeader      ETag: "56d5c426-170"
44 -   Fetch_Body     3 length stream
45 -   BackendReuse   18 boot.default
46 -   Timestamp      BerespBody: 1457802579.872127 0.000486 0.000074
47 -   Length         368
48 -   BereqAcct      334 0 334 237 368 605
49 -   End
```

log.txt

3.7 Statistiques - Varnishstat

Lors de l'installation de Varnish, un utilitaire nommé varnishstat s'installe également. Cet outil est très pratique pour visualiser les statistiques du reverse proxy.



Certaines variables sont très utiles afin de savoir si le système a bien été conçu. Par exemple, grâce à "MAIN.cache_hit", on a la possibilité de savoir le nombre de fois qu'un objet a été pris dans le cache et délivré au client. Ou encore, le nombre de fois qu'un objet a dû être demandé au serveur web avant d'être envoyé au client grâce à "MAIN.cache_miss". Cette variable : "MAIN.sess_conn " permet de connaître le nombre total de connexion TCP et celle-ci : "MAIN.client_req" le nombre total de requête. Cette dernière : "MAIN.threads" que je trouve particulièrement intéressante, permet de connaître le nombre de thread créé.

3.8 Quelques commandes utiles

Voici quelques commandes utiles lorsqu'on utilise *Varnish* :

- Pour savoir si le service *Varnish* est lancé :
  ```
  [cedric@localhost ~]$ systemctl status varnish
  ```
  Exemple de ce que nous retourne cette commande :
  ```
  varnish.service - Varnish Cache, a high-performance HTTP accelerator
     Loaded: loaded (/usr/lib/systemd/system/varnish.service; enabled)
     Active: active (running) since ven 2015-12-18 13:44:18 CET; 15min ago
   Main PID: 2767 (varnishd)
     CGroup: /system.slice/varnish.service
             ├─2767 /usr/sbin/varnishd -P /var/run/varnish.pid -f /etc/varnish/...
             └─2770 /usr/sbin/varnishd -P /var/run/varnish.pid -f /etc/varnish/...
  ```
- Pour connaitre la version de Varnish :
  ```
  [cedric@localhost ~]$ varnishd -V
  ```

4 Mise en place du serveur web Nginx sur CentOS

4.1 Installation en ligne de commande

1. Voici les lignes de commande nécessaires pour installer Nginx :
```
[root@localhost cedric]# yum install epel-release

[root@localhost cedric]# yum install nginx
```
2. Ensuite, il faut activer et démarrer le service Nginx :
```
[root@localhost cedric]#  systemctl enable Nginx
[root@localhost cedric]#  systemctl start Nginx
```
3. Et pour finir, configurer le firewall ou le désactiver. J'ai décidé pour l'instant de le désactiver et par la suite de le configurer :
```
[root@localhost cedric]#  systemctl stop firewalld
```

4.2 Quelques commandes utiles

Voici quelques commandes utiles pour *Nginx* :

- Pour savoir si le service *Nginx* est lancé :
```
[cedric@localhost ~]$ systemctl status nginx
```
Voici un exemple de ce que nous retourne la commande ci-dessus :
```
nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled)
   Active: active (running) since ven 2015-12-18 13:44:31 CET; 1h 6min ago
  Process: 1897 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
  Process: 1311 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
  Process: 1302 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
 Main PID: 1903 (nginx)
   CGroup: /system.slice/nginx.service
           ├─1903 nginx: master process /usr/sbin/nginx
           ├─1905 nginx: worker process
           ├─1906 nginx: worker process
           ├─1907 nginx: worker process
           └─1908 nginx: worker process
```

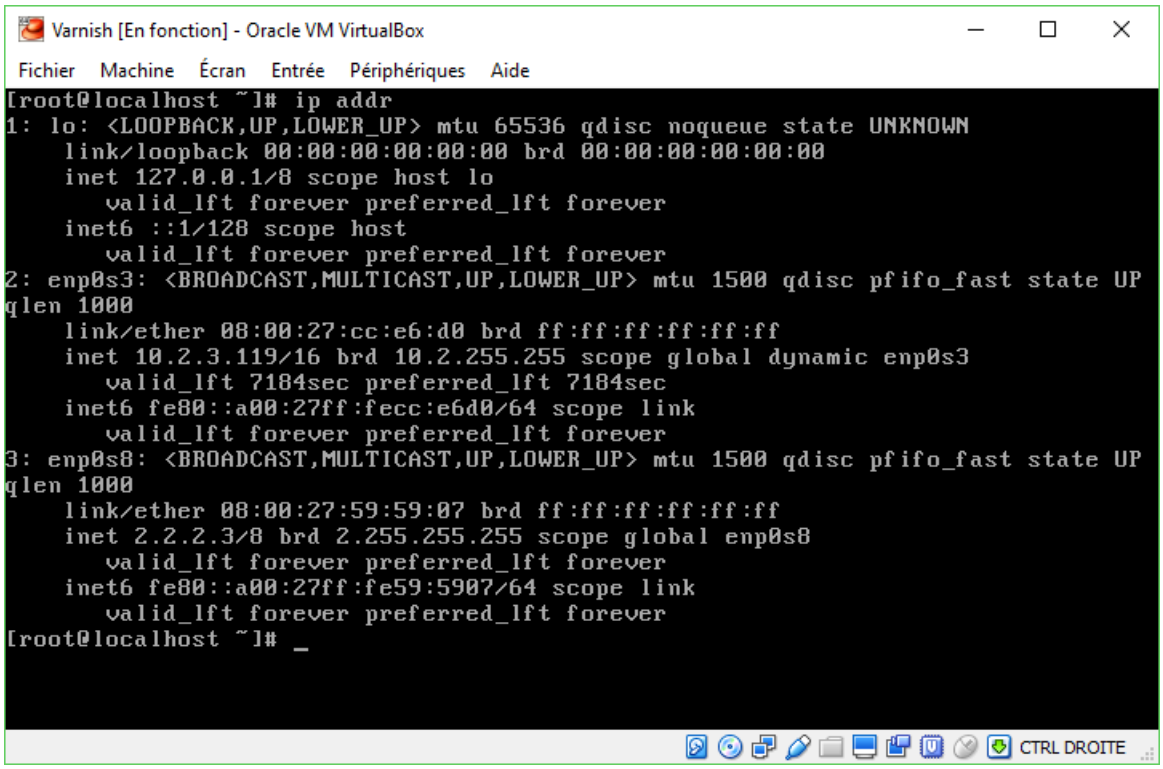
5 Machine virtuelle

Pour ce projet, j'ai dû mettre le reverse proxy Varnish et le serveur web Nginx sur des machines virtuelles. Les buts étant premièrement de tester facilement la fonctionnalité de mon système et deuxièmement que des étudiants puissent réutiliser ce projet durant des laboratoires.

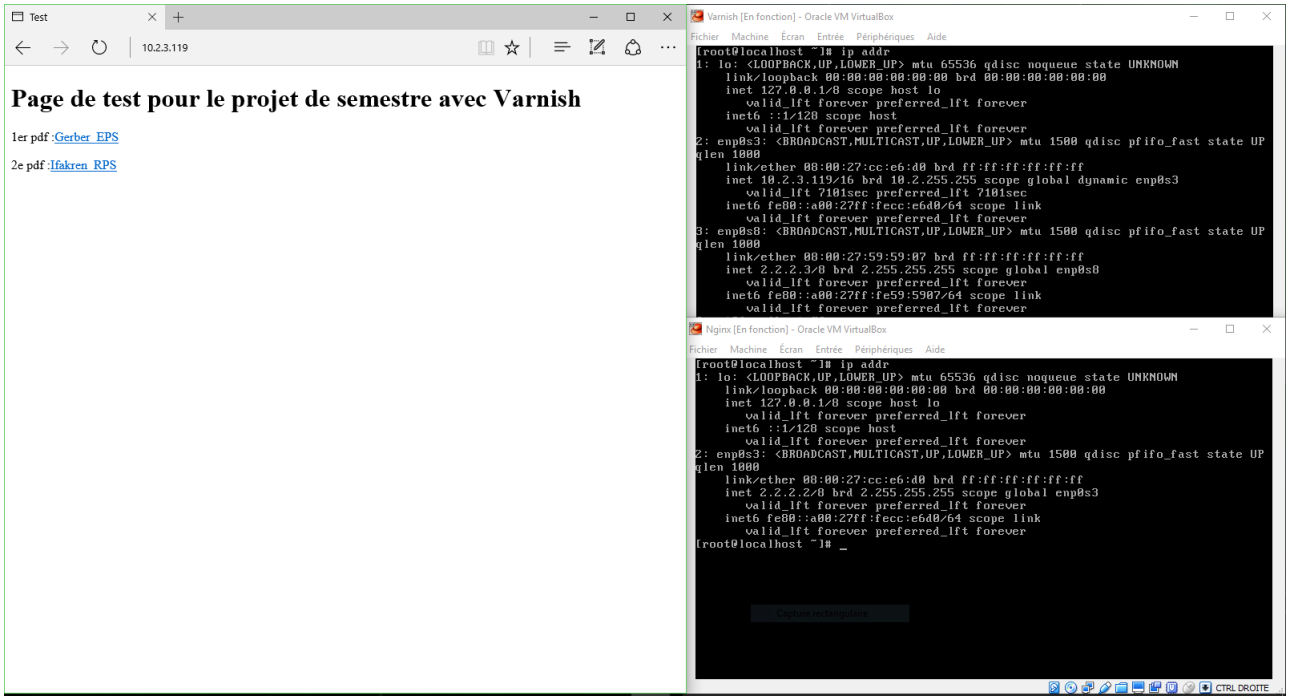
5.1 Manuel d'utilisation

Tout d'abord, il faut se connecter soit au réseaux local du laboratoire soit à un réseau muni d'un serveur DHCP. Ensuite, il faut simplement importer les fichiers *Varnish.ova* et *Nginx.ova*. Pour terminer, il faut démarrer ces machines virtuelles. Dès à présent, les deux systèmes sont lancés et tout est déjà correctement configuré.

Pour effectuer un test, il faut dans un premier temps trouver l'adresse IP de la machine virtuelle Varnish qui se trouve du coté "publique ". Dans notre cas l'adresse IP est de type 10.2.3.X., il faut donc utiliser la commande *ip addr* :



Afin d'accéder au site, il suffit maintenant d'ouvrir un navigateur web et d'entrer l'adresse IP 10.2.3.X :80. Voici le résultat :



6 Problèmes rencontrés

J'ai décidé de travailler avec la distribution GNU/Linux CentOS pour pouvoir me familiariser avec celle-ci car, dans les entreprises les distributions RedHat/CentOS sont très utilisées.

Dans un premier temps, j'ai voulu utiliser l'image CentOS 6 qui est disponible en PXE sur le réseau du laboratoire. Ensuite, pour pouvoir travailler plus aisément sur les machines CentOS, j'ai essayé d'installer l'interface graphique Gnome, mais sans succès. Je me suis donc tourné vers la version 7 de CentOS avec laquelle, j'ai finalement pu installer Gnome. Le GUI m'a été utile pour faire des cherches sur internet, copier des commandes et utiliser Wireshark.

J'ai ensuite rencontré un autre problème lors de l'éxecution du service *varnishd*. Lorsque j'ai voulu modifier certaines valeurs dans le fichier de configuration pour simplifier le service, je me suis rendu compte que mes modifications n'étaient pas prises en compte. Effectivement, après de longues recherches, j'ai observé que le fichier appelé pour exécuter le service faisait lui même des commandes au lieu de venir prendre les configurations dans mon fichier.

Pour finir, le dernier problème que j'ai rencontré concernait les variables globales dans le fichier varnish.params. En essayant l'administration par telnet afin de voir si elle pouvait offrir quelque chose d'intéressant pour mon projet, j'ai constaté qu'il y avait un secret à connaitre pour cette connexion. J'ai voulu enlever cette variable pour mes tests, mais le service ne pouvait plus se lancer à cause d'une erreur. Après des recherches et grâce à un collègue, j'ai découvert un fichier d'exécution pour le service de Varnish. Il fallait donc effectuer la suppression des variables dans le fichier varnish.service qui se situe dans */usr/lib/systemd/system/*. Ne connaissant pas le déroulement d'exécution des services sur Linux, ce déboguage m'a pris un certain temps.

7 Conclusion

Pour conclure ce projet de semestre, je pense que les configurations en langage VCL ne sont pas triviales et qu'être à l'aise avec la programmation est nécessaire. Cependant, les reverse proxy avec Varnish sont efficaces et offrent énormément d'avantages. C'est un programme très complet offrant divers utilitaires qui facilitent l'utilisation et la compréhension. De plus, grâce à la configuration de base, il est possible d'avoir un résultat très rapide.
C'est pourquoi, pour terminer, je conseille aux entreprises ayant un site web à forte affluence d'utiliser Varnish comme reverse proxy.

8 Annexes

8.1 Sources et liens utiles

- varnish-book-5.1-83-g6534a32.pdf
- https://gist.github.com/reifman/4651558
- http://www.eng.ox.ac.uk/Plone/centos/files/vcl-man.txt/view
- http://www.eng.ox.ac.uk/Plone/centos/varnish-setup
- http://gafish.fr/varnish-mise-en-place-parametrage-et-modification-de-code/
- http://ayeba.fr/2011/07/comprendre-et-configurer-un-cache-varnish/
- https://www.mnot.net/cache_docs/index.fr.html
- http://sametmax.com/initiation-a-varnish-accelerer-un-blog-wordpress/