MISE EN PLACE D'UNE INFRASTRUCTURE WEB REDONDANTE (NGINX // HAPROXY)

Table des matières

I. Contexte 2

II. Objectifs 2

III. Déploiement du service WEB 2

1. Installation des paquets & activation de Nginx : 3
2. Installation des sites ecommerce.fr & ecommerce.uk : 4
   a. Installation & configuration de la base de données (MariaDB) : 4
   b. Configuration de NGINX : 8
   c. Test de connexion aux sites : 9
3. Sécurisation du flux web (HTTPS) : 11
   a. Démonstration via Wireshark : 11
   b. Mise en place du HTTPS : 12
   c. Tests via Nginx et Wireshark : 14

IV. Mise en place de la haute disponibilité des serveurs WEB : 15

1. Déploiement du 2ème Serveur WEB (WEB-02) : 15
2. Installation et configuration de HA-PROXY : 16
   a. Installation : 16
   b. Configuration : 17
3. Tests : 19

V. Haute disponibilité de HAProxy : 20

1. Déploiement du 2ème Serveur HAProxy : 20
2. Installation & Configuration de Pacemaker : 21
   a. Installation & Authentification (Sur les deux serveurs) : 21
   b. Création du cluster (sur le serveur principal) : 22
   c. Création de l'IP Virtuelle (VIP) : 23
   d. Activation de heartbeat pour HAProxy : 24
3. Tests : 24

Compétences mobilisées :

- Réaliser les tests d'intégration et d'acceptation d'un service
- Déployer un service
- Accompagner les utilisateurs dans la mise en place d'un service
- Vérifier les conditions de la continuité d'un service informatique
- Développer la présence en ligne de l'organisation

I. Contexte

La société fictive Ordi Global Corporation souhaite mettre en place une infrastructure web redondante, sécurisée et évolutive.

L'objectif est de disposer d'un service web capable d'héberger plusieurs sites (FR / UK), de sécuriser les échanges en HTTPS et d'assurer la continuité de service en cas de panne.

Pour cela, les technologies suivantes seront utilisées :

- Nginx (serveur web)
- PHP
- MariaDB (base de données)
- HAProxy (répartition de charge)
- Pacemaker (cluster haute disponibilité)
- TLS/SSL (chiffrement des communications)

II. Objectifs

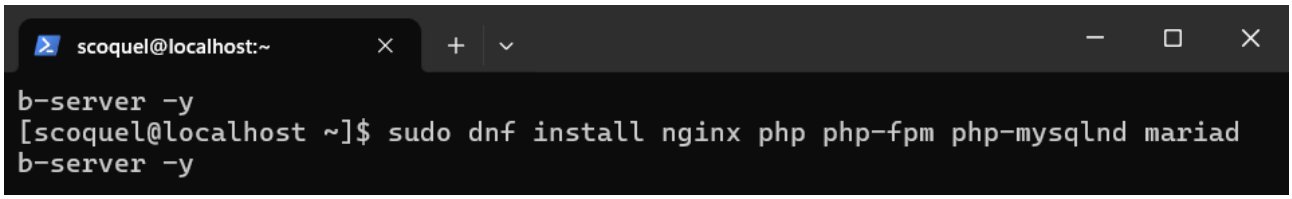
- Déployer un serveur web Linux opérationnel avec 2 sites (FR / UK)
- Configurer un environnement LAMP (Linux, Nginx, MariaDB, PHP)
- Mettre en place le chiffrement HTTPS
- Implémenter une répartition de charge avec HAProxy
- Mettre en place une architecture haute disponibilité (Pacemaker, IP virtuelle)

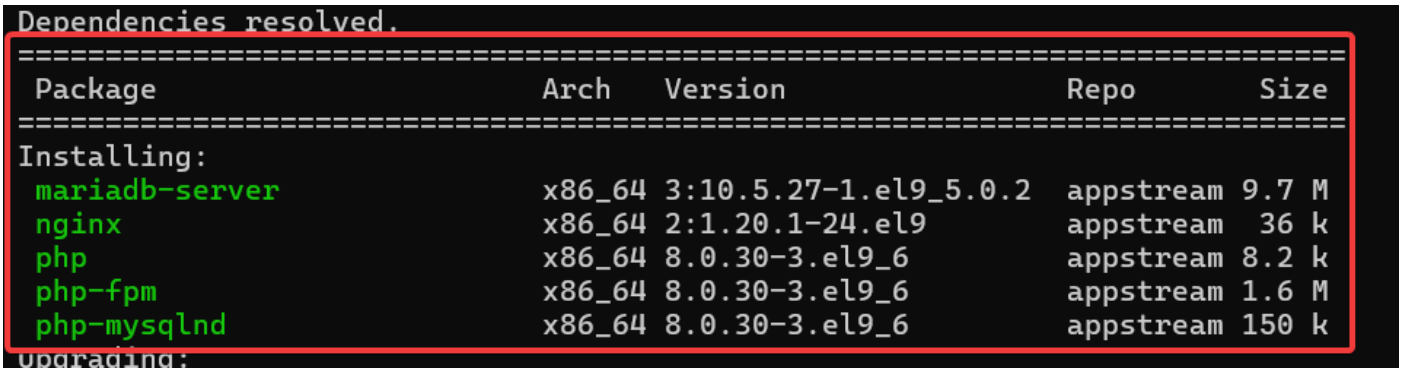
III. Déploiement du service WEB

En premier lieu nous allons déployer un serveur WEB, afin d'héberger deux sites : ecommerce.fr & ecommerce.uk
Le serveur, sous Rocky Linux, sera équipé des technologies suivantes : Nginx avec les extensions PHP et MariaDB

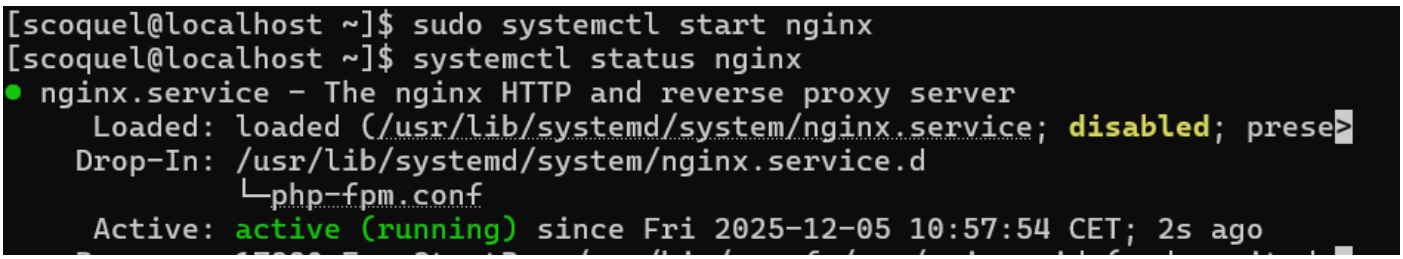
1. Installation des paquets & activation de Nginx :

Une fois notre machine virtuelle installée, nous allons installer les paquets via la commande : **dnf install**

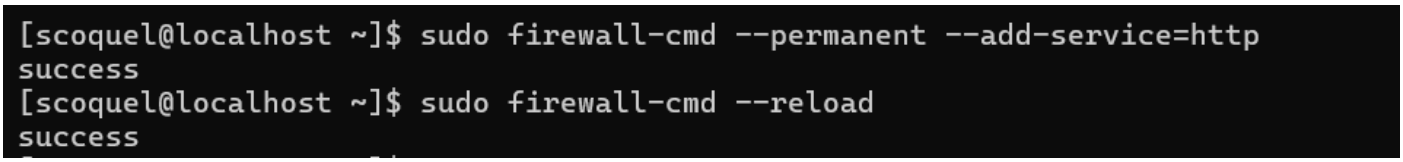




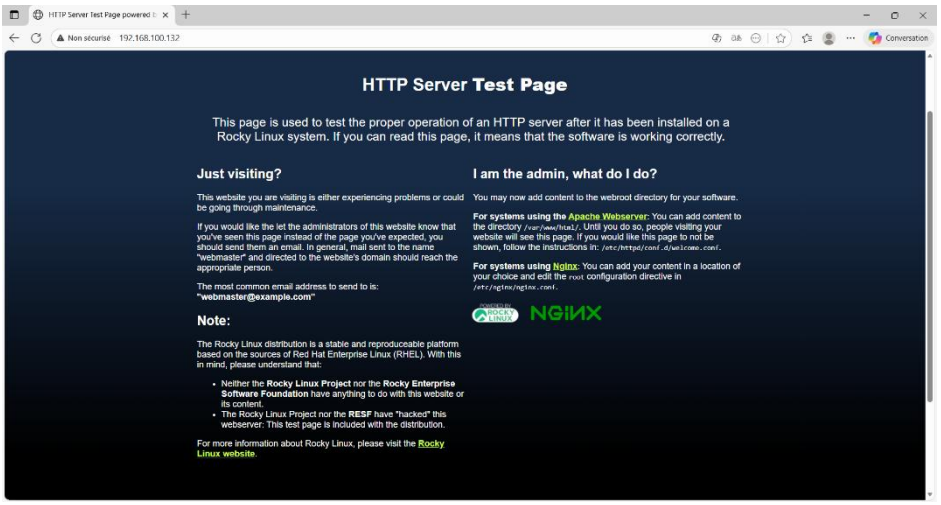
Une fois les logiciels nécessaires installés, nous allons pouvoir activer Nginx, notre solution permettant d'héberger notre site :



Il faut ensuite ouvrir le port http de notre machine pour s'y connecter et faire un premier test :



Essayons maintenant de nous y connecter en rentrant son IP :



La page de test de Nginx apparaît bien, le service est fonctionnel.

2. Installation des sites ecommerce.fr & ecommerce.uk :

Le service web étant fonctionnel, il faut maintenant installer nos sites sur le serveur.

a. Installation & configuration de la base de données (MariaDB) :

Pour initialiser l'installation de la base de données nous allons lancer les commandes suivantes :

- **Sudo systemctl enable mariadb** → pour activer notre base de données au démarrage
- **Sudo systemctl start mariadb** → pour lancer le service de base et pouvoir commencer l'installation
- **Sudo mysql_secure_installation** → pour lancer l'installation interactive de mariadb sur notre serveur

Une fois que nous avons terminé de choisir nos options nous recevons le message suivant :

```
All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
[scoquel@localhost nginx]$
```

Il suffit de s'y connecter via la commande : **Mysql -u root** pour pouvoir ensuite lancer les commandes SQL.

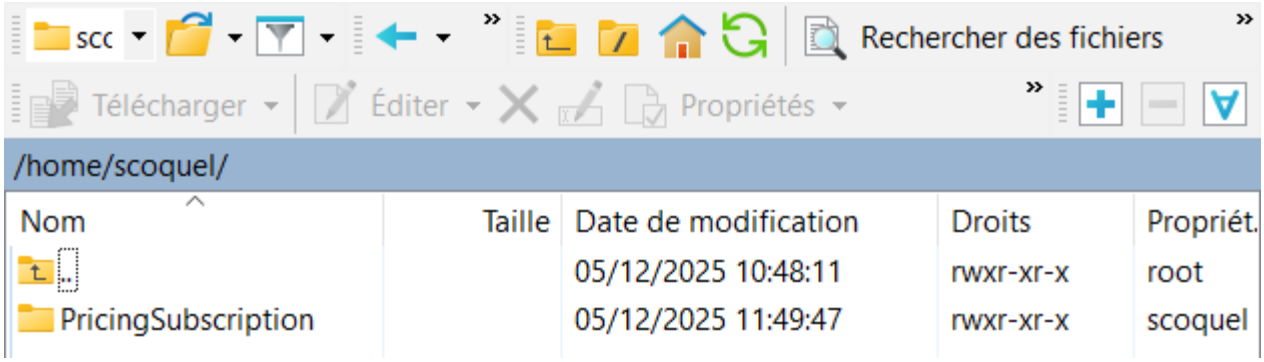
Nous allons :

- Créer une nouvelle base de données
- Créer un utilisateur que notre site va utiliser et qui aura les droits sur cette base

- Alimenter notre base via un script SQL

```
[scoquel@localhost ~]$ sudo mysql -u root -p
[sudo] password for scoquel:
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 12
Server version: 10.5.27-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statemen
t.

MariaDB [(none)]> CREATE USER 'commerce_fr'@'localhost' IDENTIFIED BY 'comme
rce';
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> CREATE DATABASE ecommercefr;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON ecommercefr.* TO 'commerce_fr'@'lo
calhost';
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.000 sec)

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| ecommercefr        |
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.001 sec)
```

Pour alimenter la base via script, il faut au préalable importer le fichier SQL sur notre machine, ici cela à été fait via le protocole SSH en utilisant WinSCP et en important les fichiers du site, qui nous servirons plus tard :



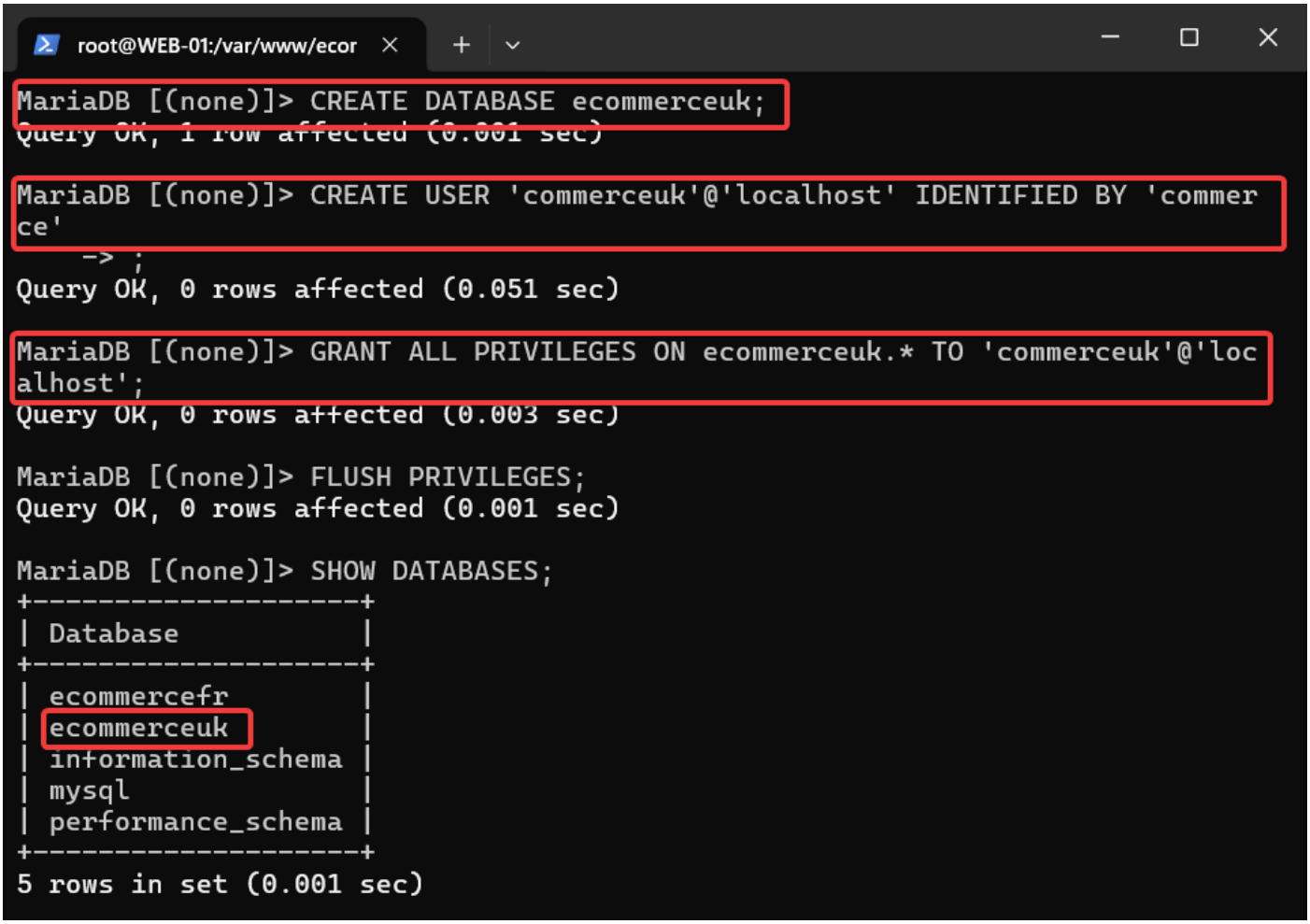

```
[scoquel@localhost ~]$ ls /var/www/ecommerce.fr/
config.php  css  database.sql  images  js  sign-up.php
[scoquel@localhost ~]$
```

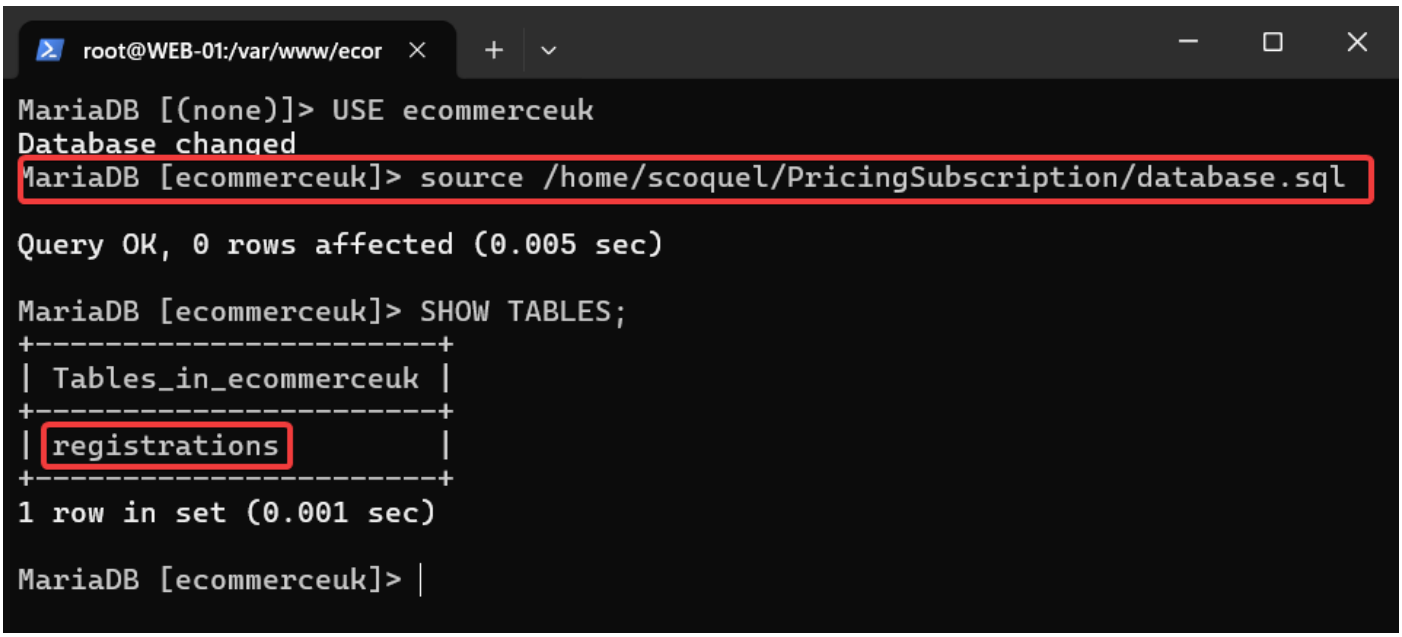
Pour terminer, il faut se reconnecter dans mariadb, aller dans la base via la commande **USE #Nom de la base# et utiliser la commande source #chemin vers le fichier sql#** :

```
MariaDB [(none)]> USE ECOMMERCEFR
ERROR 1049 (42000): Unknown database 'ECOMMERCEFR'
MariaDB [(none)]> USE ecommercefr
Database changed
MariaDB [ecommercefr]> source /var/www/ecommerce.fr/database.sql
Query OK, 0 rows affected (0.003 sec)

MariaDB [ecommercefr]> show tables
    -> ^C


    -> show tables;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual t
hat corresponds to your MariaDB server version for the right syntax to use n
ear 'show tables' at line 3
MariaDB [ecommercefr]> show tables;
+----------------------+
| Tables_in_ecommercefr |
+----------------------+
| registrations        |
+----------------------+
1 row in set (0.001 sec)
```

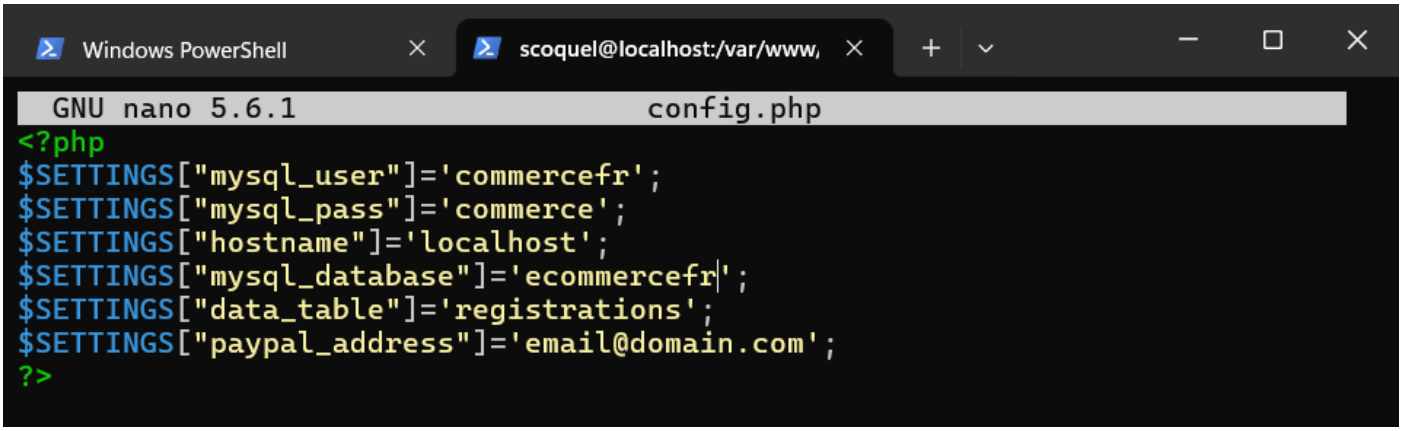
Une table à bien été créée dans la base, notre script à bien fonctionné !

Pour ecommerce.uk, il suffit de répéter le même schéma en créant une autre base de données et un autre utilisateur, le script pour alimenter la base étant le même nous obtenons à la fin :





Pour lier les bases aux différents sites, il faut se rendre dans le code de celui-ci et injecter les informations des différentes bases (utilisateurs, serveur, et base de données) dans le fichier config.php :



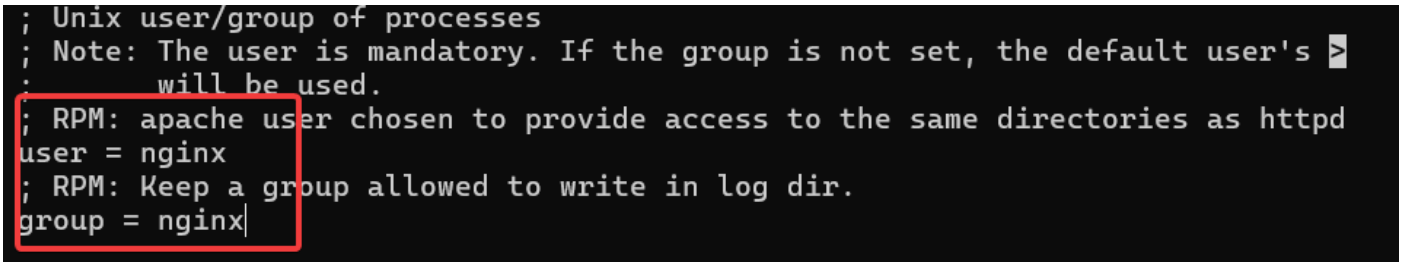
b. Configuration de NGINX :

Passons à la configuration de NGINX qui pour l'instant ne renvoie que la page de test.

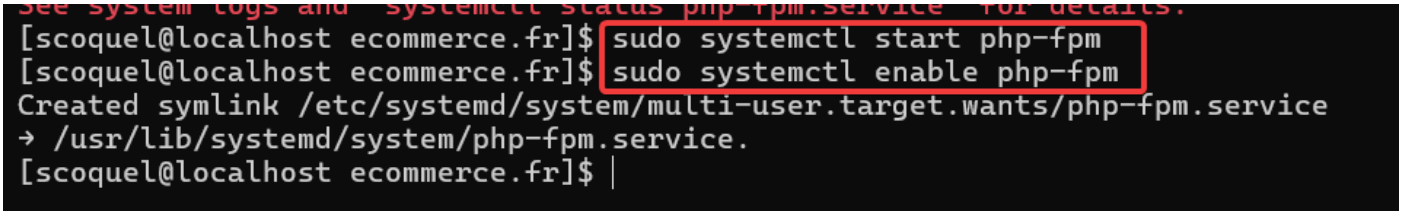
Pour avoir accès à deux sites et qu'ils fonctionnent correctement il faut réaliser deux étapes :

- Configurer php-fpm → Le module qui nous permet de faire fonctionner le php sur nos sites
- Configurer les sites NGINX → Pour que serveur sache quelles pages afficher en fonction de .uk ou .fr

Pour PHP-FPM, il suffit de se rendre et modifier son fichier de configuration pour spécifier l'utilisateur qui utilise le module, à savoir l'utilisateur nginx :

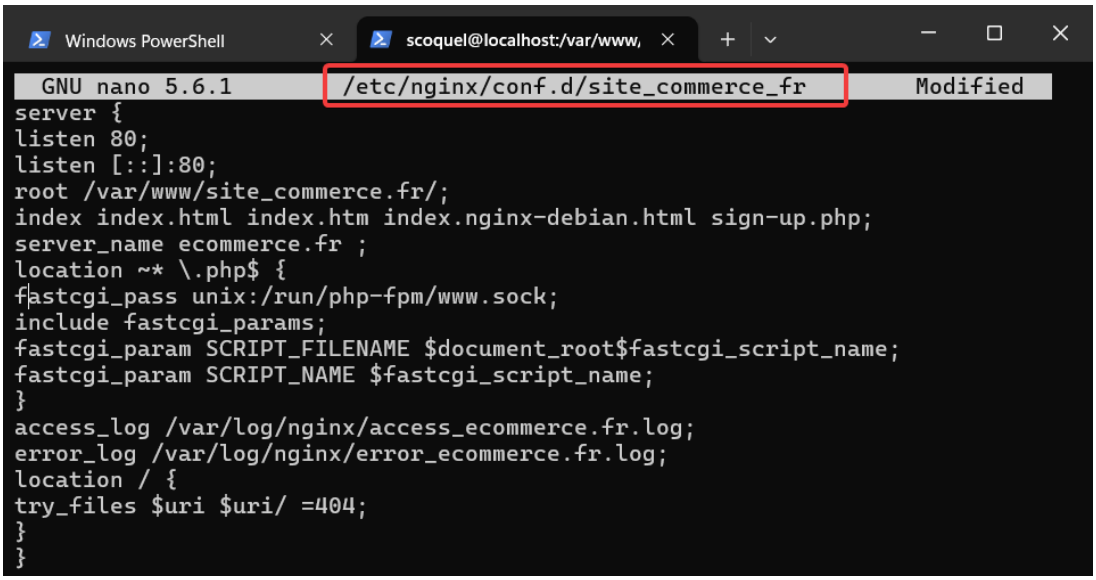


Pour finir il faut lancer et activer le service au démarrage :



Maintenant pour NGINX, nous devons nous rendre dans son répertoire **conf.d (qui se trouve dans /etc/nginx)** pour y ajouter une entrée pour nos sites avec leur nom et quels fichiers le serveur doit utiliser.

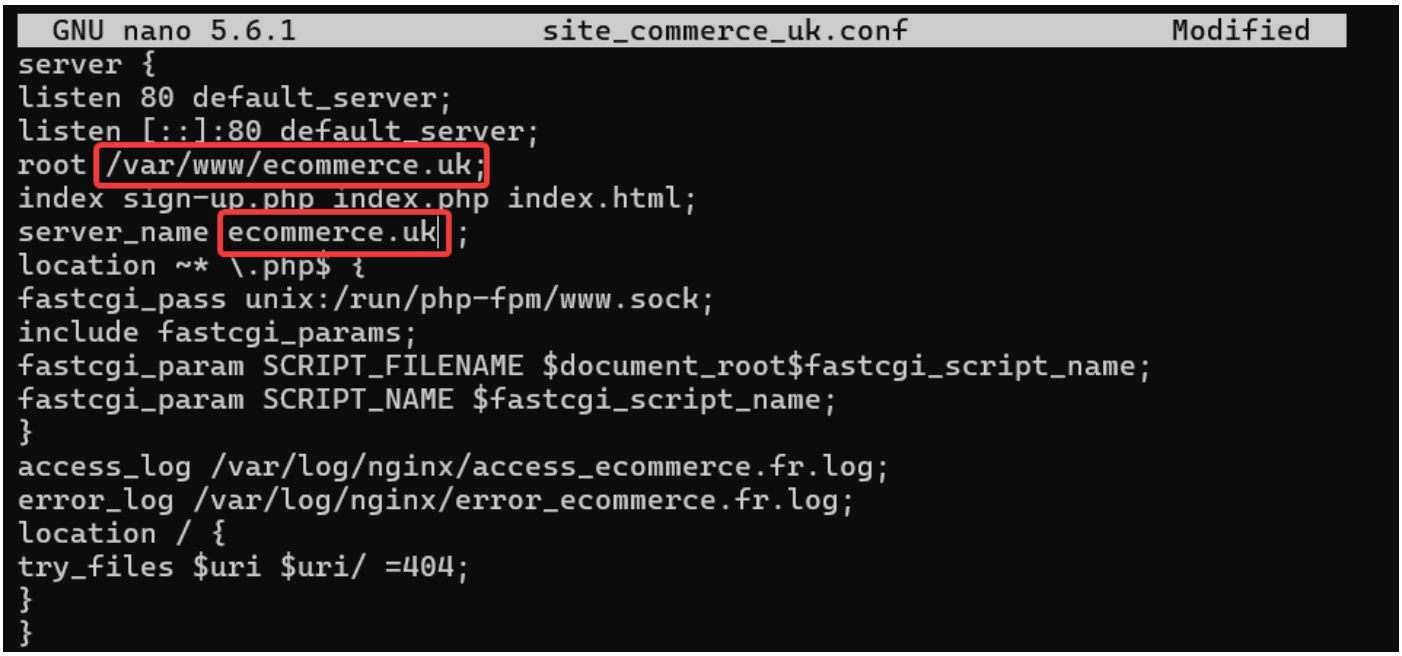
Pour ce faire il faut créer un nouveau fichier et y injecter les paramètres suivants :



Ici les paramètres importants sont **:**

- **Root** → qui va définir la racine du site
- **Server_name** → qui va définir quelle adresse web nous devons taper pour que nginx nous renvoie ce site
- **Listen #port#** → qui indique sur quels ports le serveur doit écouter, ici le port http classique

Nous faisons un autre fichier pour ecommerce.uk :

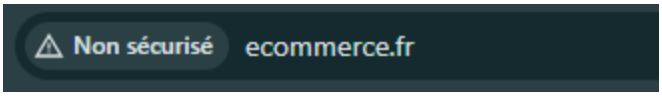


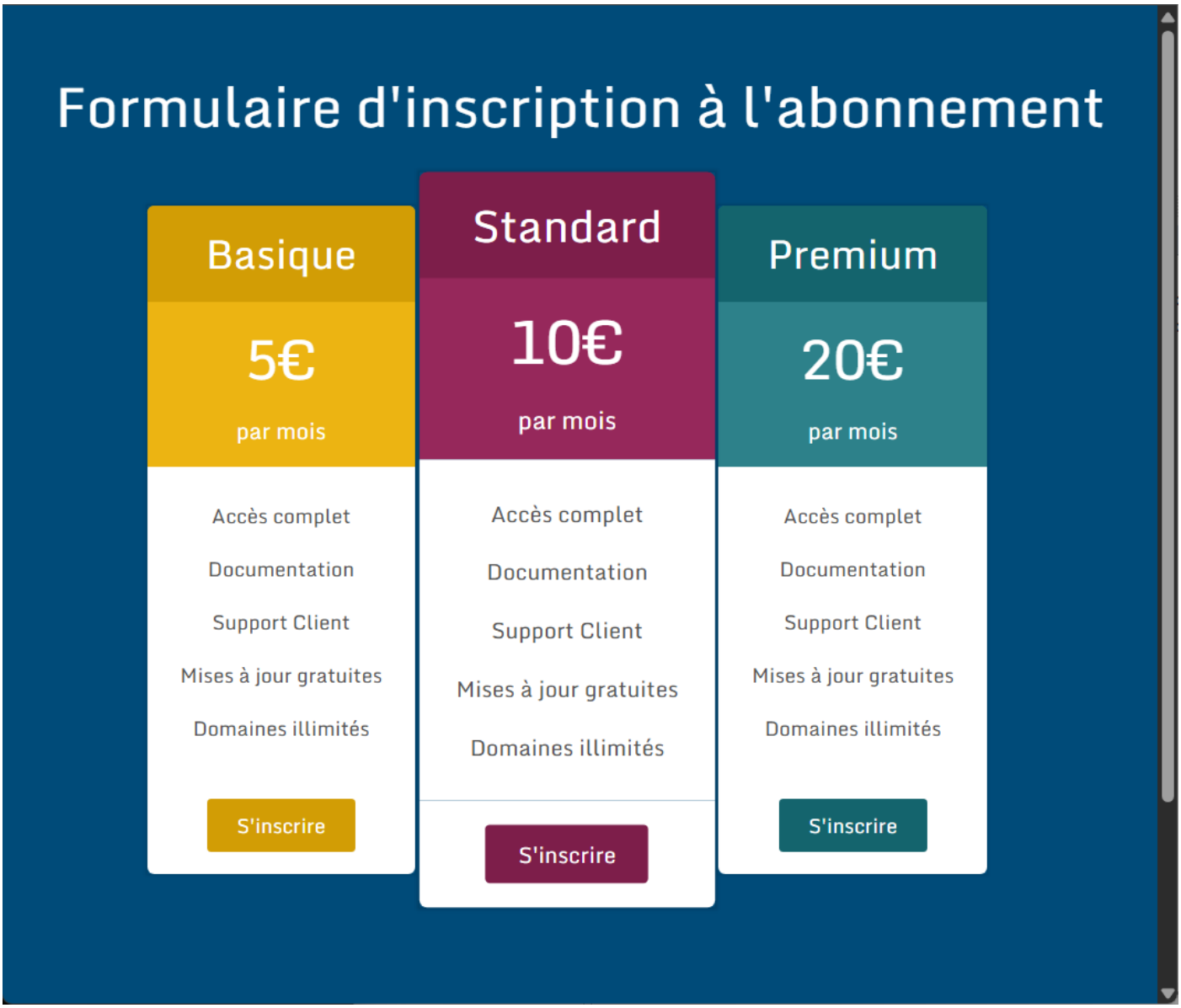
Pour finir il suffit de relancer le service NGINX pour que les nouvelles configurations soient prises en compte.

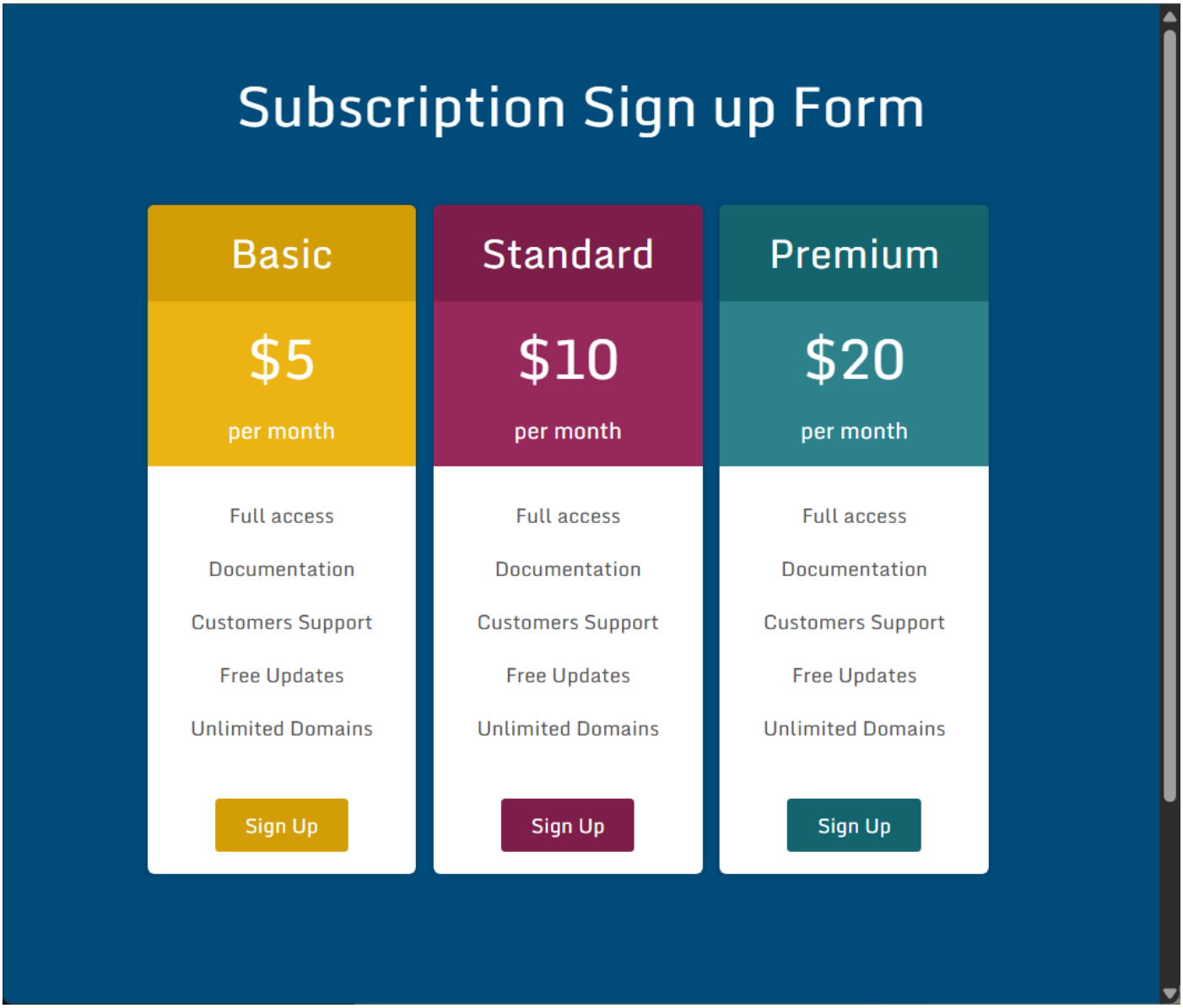
c. Test de connexion aux sites :

Pour tester la connexion il faut d'abord régler les noms de domaines ecommerce.fr & .uk afin qu'ils pointent vers notre serveur local. Ici la solution la plus simple que nous avons utilisée est de modifier le fichier hosts de notre machine pour éviter de devoir créer un serveur DNS.

Une fois le fichier hosts modifié quand nous tapons l'adresse désirée (ici ecommerce.fr) et le site apparait soit en français ou en anglais :





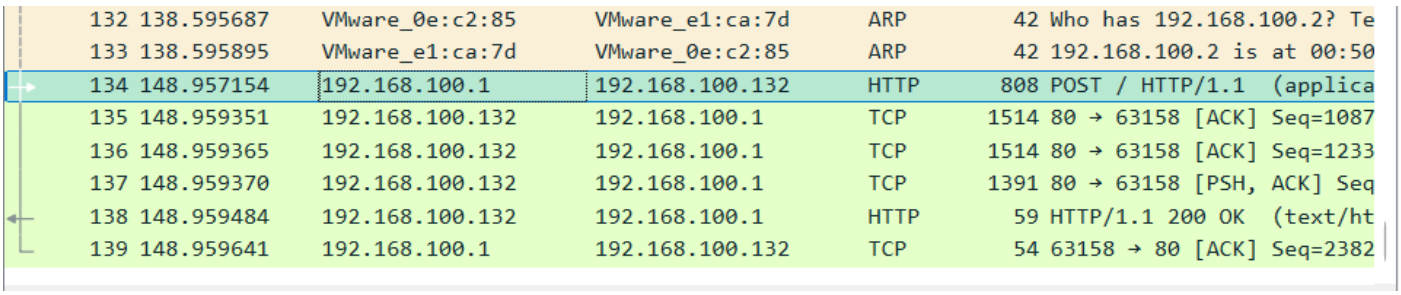


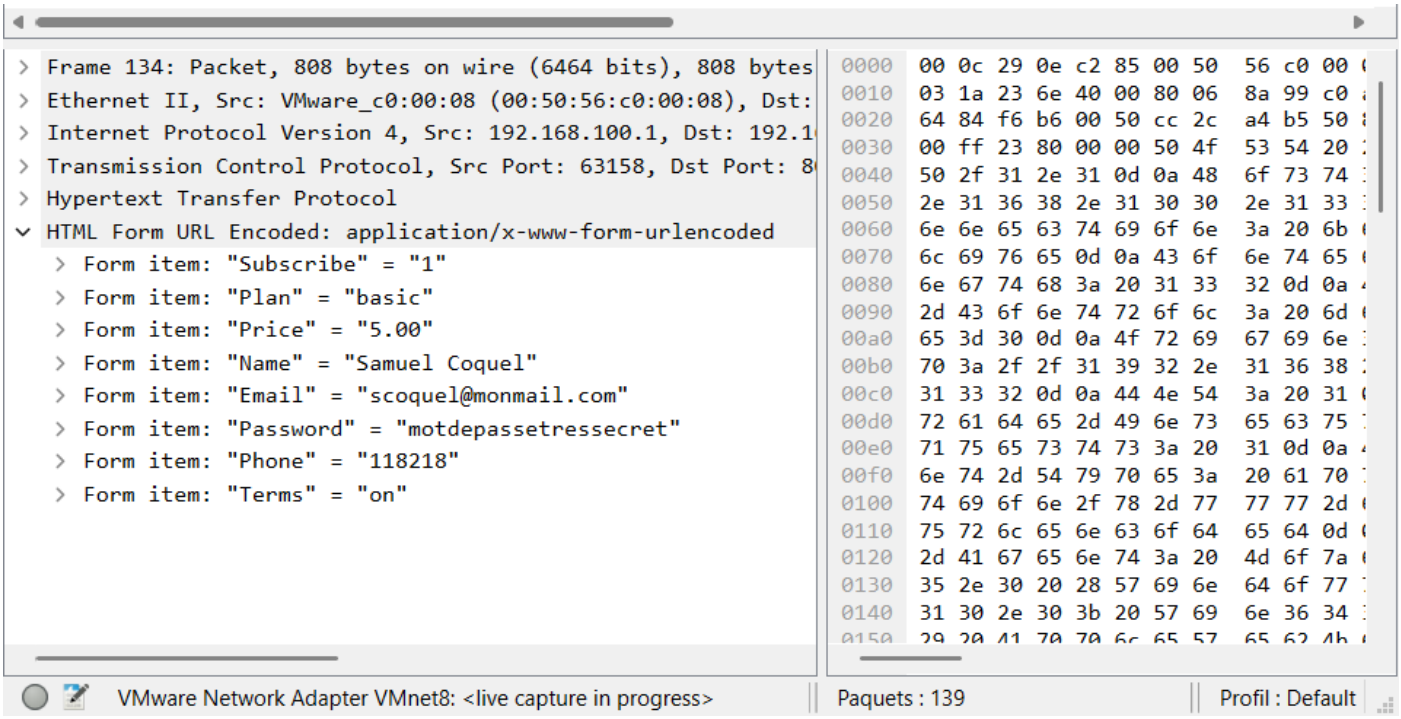
3. Sécurisation du flux web (HTTPS) :

Nos sites sont bien configurés mais un problème subsiste : ils sont en http, qui est un flux non sécurisé. Ici notre site renvoie des informations comme un mot de passe qui peuvent être intercepté en clair.

a. Démonstration via Wireshark :

En utilisant le logiciel Wireshark, nous pouvons avoir un aperçu du traffic que nous envoyons lorsque nous remplissons le formulaire :





Comme nous pouvons voir, le mot de passe est affiché en clair ainsi que les autres informations du formulaire.

b. Mise en place du HTTPS :

Pour mettre en place le HTTPS, il faut tout d'abord disposer d'un certificat, ici nous allons le créer sur notre serveur en autosigné (via openssl)

Dans le répertoire ssl (/etc/ssl) nous allons créer un dossier private avec des droits d'accès root uniquement (700) :

```
[root@WEB-01 ssl]# ls
cert.pem  certs  ct_log_list.cnf  openssl.cnf
[root@WEB-01 ssl]# mkdir private
[root@WEB-01 ssl]# chmod 700 private/
[root@WEB-01 ssl]# ls -la
total 12
drwxr-xr-x.  3 root root   92 Jan 30 09:29 .
drwxr-xr-x. 85 root root 8192 Jan 30 08:52 ..
lrwxrwxrwx.  1 root root   49 Aug 21  2024 cert.pem -> /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem
lrwxrwxrwx.  1 root root   18 Aug 21  2024 certs -> /etc/pki/tls/certs
lrwxrwxrwx.  1 root root   28 Aug 21  2024 ct_log_list.cnf -> /etc/pki/tls/ct_log_list.cnf
lrwxrwxrwx.  1 root root   24 Aug 21  2024 openssl.cnf -> /etc/pki/tls/openssl.cnf
drwx------.  2 root root    6 Jan 30 09:29 private
[root@WEB-01 ssl]#
```

Ensuite nous allons effectuer la commande suivante :

Openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt

Cette commande va nous permettre de créer une nouvelle clé privée qui sera dans private ainsi qu'un nouveau certificat utilisant cette clé.

La demande de certificat va nous demander certaines informations dont le nom du site que nous allons compléter :

```
-----
Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:NORD
Locality Name (eg, city) [Default City]:LILLE
Organization Name (eg, company) [Default Company Ltd]:ecommerceFR
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:ecommerce.fr
Email Address []:admin@ecommerce.fr
[root@WEB-01 private]#
```

Nous allons aussi exécuter la commande suivante :

Openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048

Qui génère un groupe Diffie-Hellman fort qui permet la négociation de PFS (Perfact Forward Secrecy) :

```
root@WEB-01:/etc/ssl/private   root@WEB-02:/var/www/ecom
[root@WEB-01 private]# openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
Generating DH parameters, 2048 bit long safe prime
.....................................................................+.....
.....................................................................+.....
.....+.....
```

Les clés et le certificat étant créés, il faut maintenant les incorporer à notre configuration de site et ouvrir le port HTTPS (443)

Nous allons modifier la configuration des deux sites pour y ajouter les lignes suivantes :

```
try_files $uri $uri/ =404;
}
}
server {
listen 443 http2 ssl;
listen [::]:443 http2 ssl;
ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;
ssl_dhparam /etc/ssl/certs/dhparam.pem;
root /var/www/ecommerce.fr/;
index index.html index.htm index.nginx-debian.html sign-up.php;
server_name ecommerce.fr ;
location ~* \.php$ {
fastcgi_pass unix:/run/php-fpm/www.sock;
include fastcgi_params;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
fastcgi_param SCRIPT_NAME $fastcgi_script_name;
}
access_log /var/log/nginx/access_ecommerce.fr.log;
error_log /var/log/nginx/error_ecommerce.fr.log;
location / {
```

Après avec **nginx -t** nous pouvons vérifier que les syntaxes sont ok :

```
[root@WEB-01 conf.d]# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
[root@WEB-01 conf.d]#
```

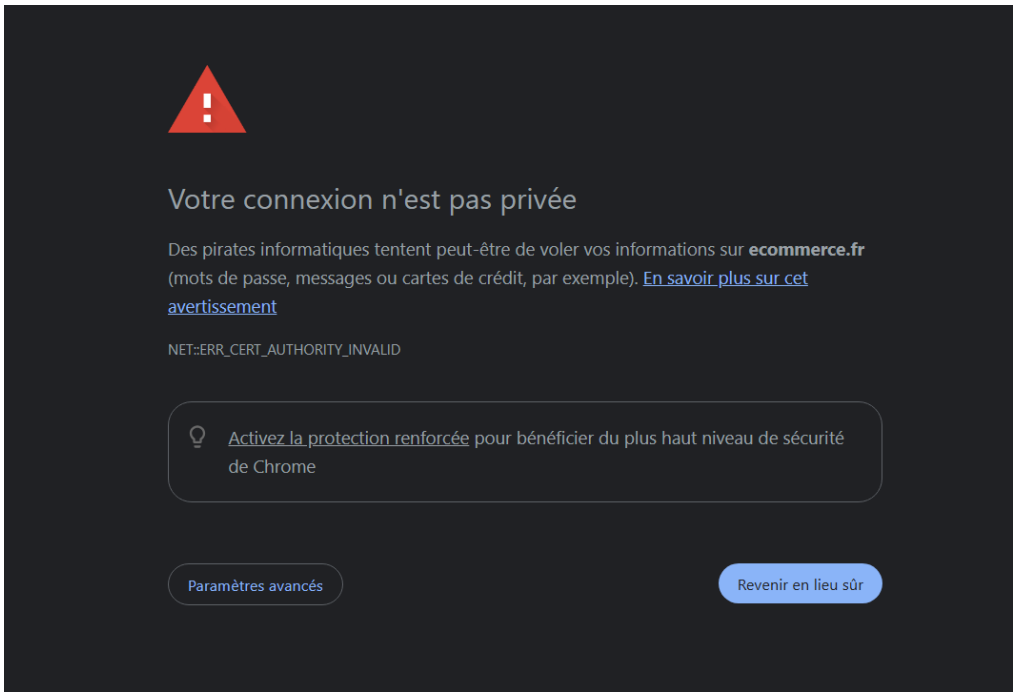
Et ouvrir le port avec firewall-cmd :

```
[root@WEB-01 conf.d]# firewall-cmd --add-service=https --permanent
success
[root@WEB-01 conf.d]# firewall-cmd --reload
success
[root@WEB-01 conf.d]#
```

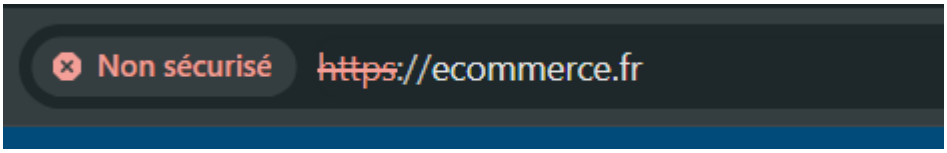
c. Tests via Nginx et Wireshark :

Une fois nginx relancé, il faut se connecter en https sur le site :

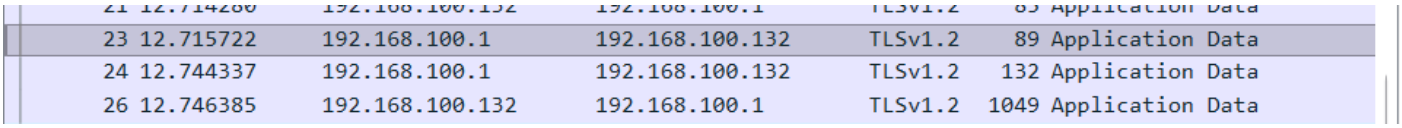
Nous avons bien le message d'erreur indiquant que le certificat est autosigné :

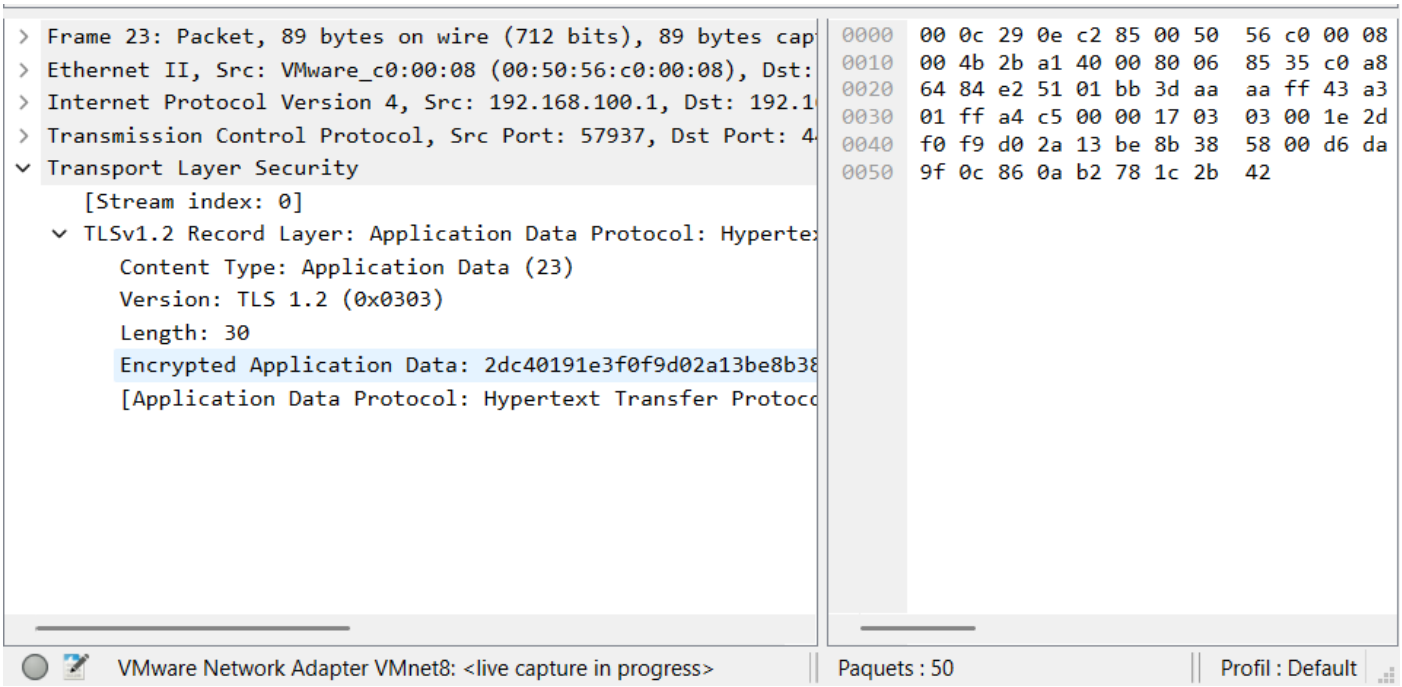


Ainsi que l'indication dans la barre de recherche que nous sommes en https (même s'il y a une barre) :



Si nous allons sur Wireshark nous voyons que là le traffic est bel et bien chiffré et remplacé par des lignes « Application Data » :





IV. Mise en place de la haute disponibilité des serveurs WEB :

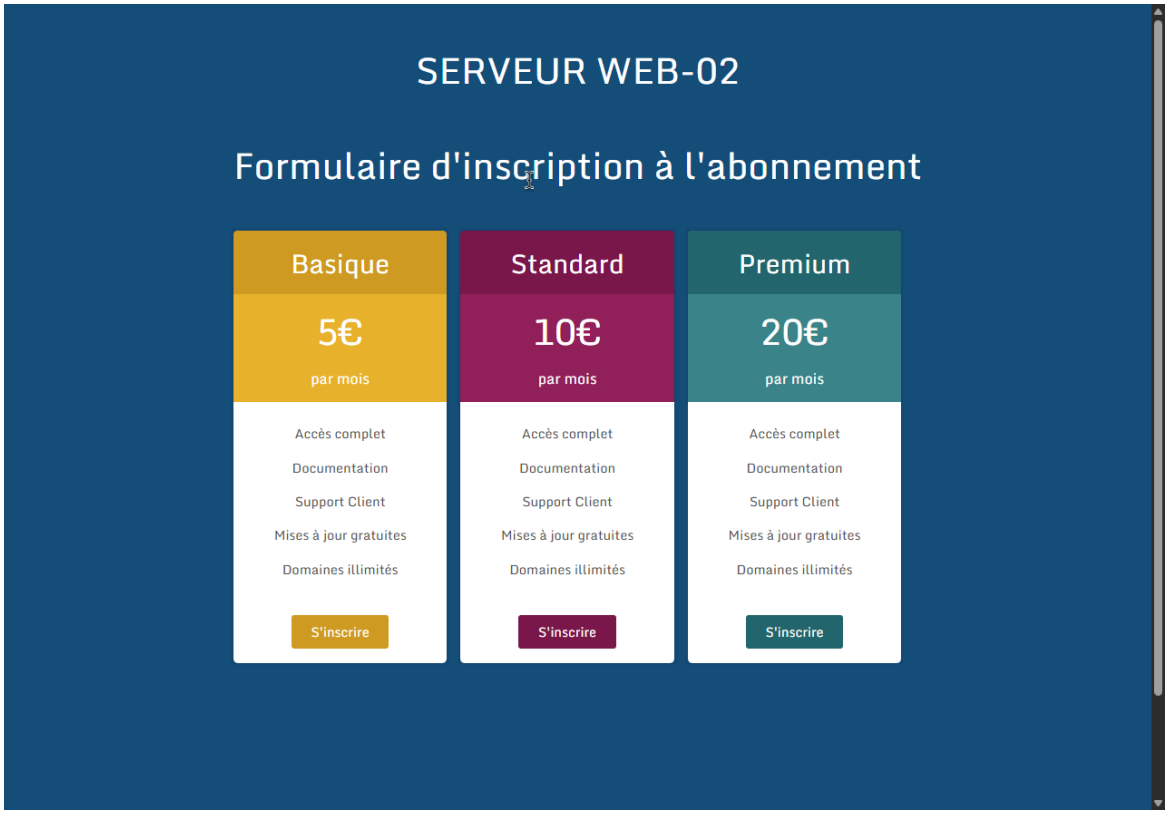
Dans l'architecture actuelle, le serveur web repose sur une seule machine, ce qui constitue un point de défaillance unique (SPOF) En cas de panne, il devient totalement indisponible.

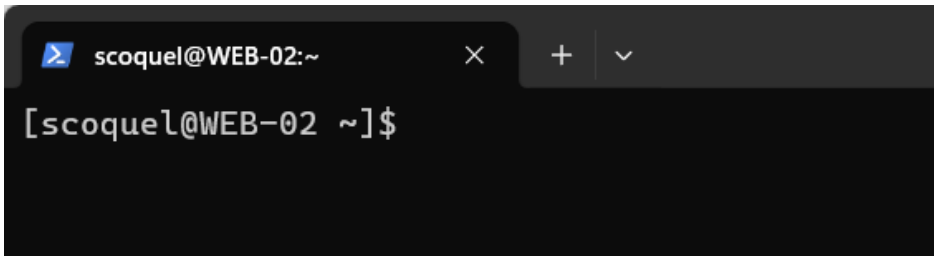
Afin d'assurer la continuité de service, il est nécessaire de déployer un second serveur web. Cette redondance permet de garantir la disponibilité du service en cas de défaillance.

Pour pouvoir accéder de manière transparente aux deux serveurs, nous allons aussi mettre en place une solution de répartition de charge (Load-Balancing) qui va se charger de répartir le traffic entrant vers les deux serveurs. Nous allons utiliser pour cela le logiciel HAPROXY.

1. Déploiement du 2ème Serveur WEB (WEB-02) :

Nous avons déployé un deuxième serveur, nommé WEB-02 avec une configuration identique au premier et fonctionnel, les pages web ont été modifié pour afficher sur quel serveur nous sommes :

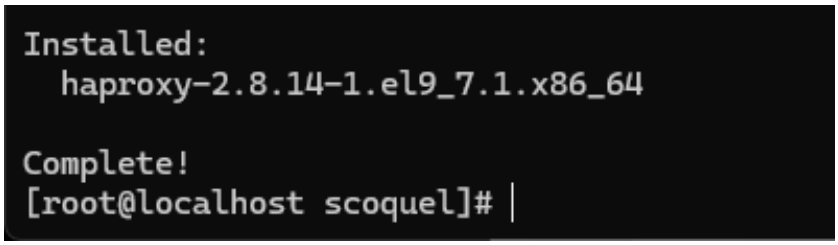




2. Installation et configuration de HA-PROXY :

a. Installation :

Pour mettre en place HA-PROXY, nous avons déployé un 3ème serveur sous Rocky Linux et nous allons installer le logiciel via la commande : **yum install haproxy**



b. Configuration :

Pour le configurer, nous devons nous rendre dans le répertoire suivant : **/etc/haproxy/**

Et éditer le fichier : **haproxy.cfg** (Une copie de backup est faite avant)

```
scoquel@HAPROXY-01:~
[scoquel@HAPROXY-01 ~]$ sudo cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy_bk.cfg
[sudo] password for scoquel:
[scoquel@HAPROXY-01 ~]$ ls /etc/haproxy/
conf.d  haproxy.cfg  haproxy_bk.cfg
[scoquel@HAPROXY-01 ~]$
```

Notre configuration sera la suivante :

```
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000
listen stats
bind *:8080
mode http
stats enable
stats hide-version
stats uri /stats
stats admin if LOCALHOST
stats auth
#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------
frontend main
    bind *:5000
    acl url_static       path_beg       -i /static /images /javascr
    acl url_static       path_end       -i .jpg .gif .png .css .js

    use_backend static          if url_static
    default_backend             app

#---------------------------------------------------------------------
# static backend for serving up images, stylesheets and such
#---------------------------------------------------------------------
backend static
    balance     roundrobin
    server      static 127.0.0.1:4331 check

#---------------------------------------------------------------------
# round robin balancing between the various backends
#---------------------------------------------------------------------
backend app
    balance     roundrobin
    server web01 192.168.137.129:443 ssl verify none check
    server web02 192.168.137.130:443 ssl verify none check
module(load="imudp") # needs to be done just once
input(type="imudp" port="514")
```

Nous avons avec les paramètres suivants :

- Activé l'interface stats d'haproxy, qui nous permettra de suivre en temps réel la répartition de charge

- Paramétré nos deux serveurs de « backend » : c'est-à-dire désigné vers quels serveurs nous allons renvoyer le traffic et avec quel algorithme de répartition, ici Round-Robin (1 fois vers WEB-01, 1 fois vers WEB-02 etc…)

NB : Nous pouvons aussi ajouter le port HTTPS avec un certificat (généré au préalable) avec cette ligne :

```
bind *:443 ssl crt /etc/pki/tls/certs/haproxy.pem
```

Une fois la configuration changée, il faut sur notre machine ouvrir les ports et autoriser HAPROXY dans SElinux :

```
root@HAPROXY-01:/home/sc
[root@HAPROXY-01 scoquel]# setsebool -P haproxy_connect_any 1
[root@HAPROXY-01 scoquel]#
```

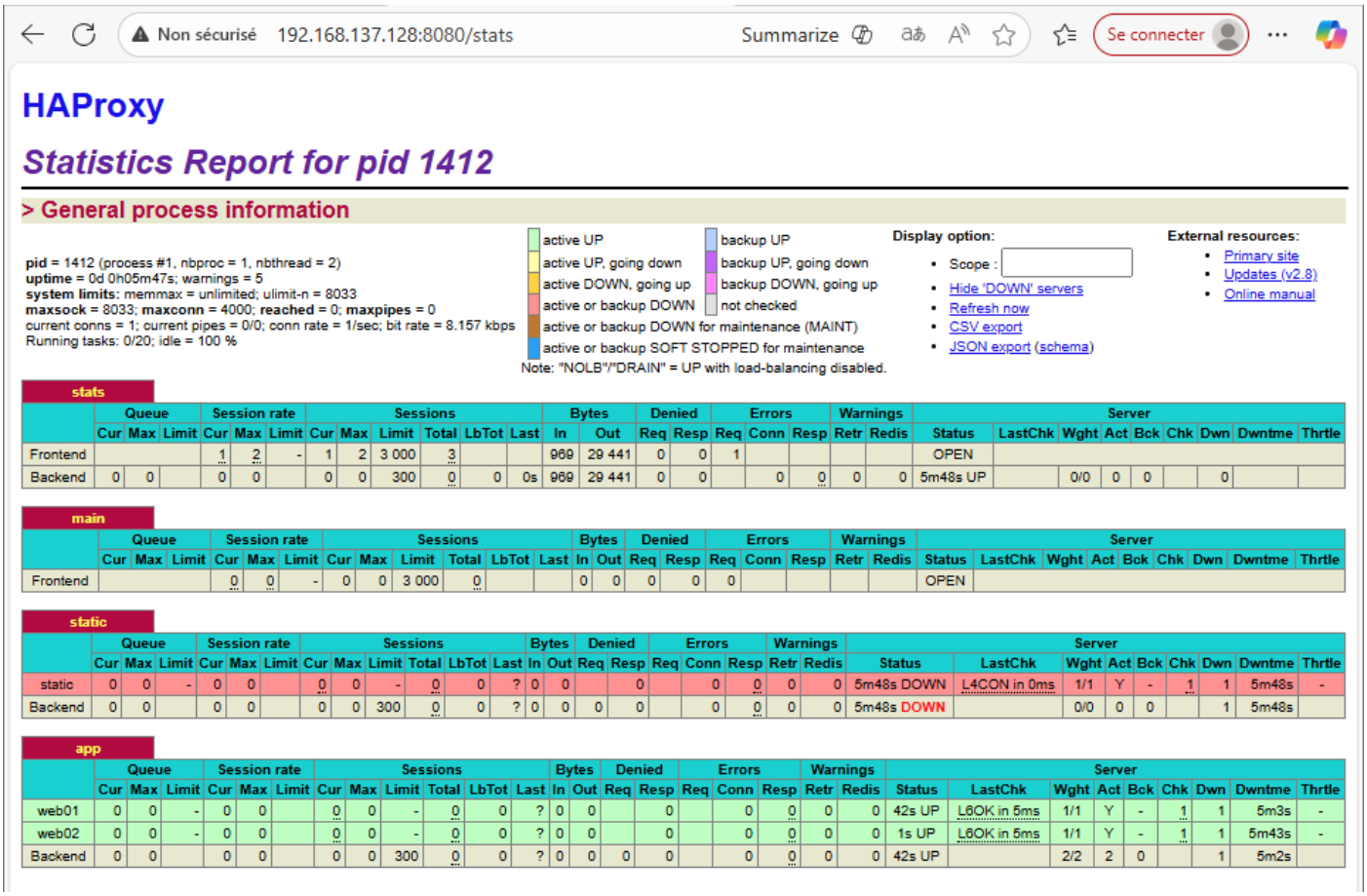
```
root@HAPROXY-01:/home/sc
[root@HAPROXY-01 scoquel]# sudo firewall-cmd --permanent --add-port=8080/tcp
success
[root@HAPROXY-01 scoquel]# firewall-cmd --reload
success
[root@HAPROXY-01 scoquel]#
```

```
root@HAPROXY-01:/home/sc
[root@HAPROXY-01 scoquel]# sudo firewall-cmd --permanent --add-service=http
success
[root@HAPROXY-01 scoquel]# firewall-cmd --reload
^[[Asuccess
```

```
[root@HAPROXY-01 certs]# sudo firewall-cmd --permanent --add-service=https
success
[root@HAPROXY-01 certs]# firewall-cmd --reload
success
```

3. Tests :

Une fois le service redémarré, nous pouvons effectuer des tests en essayant par exemple de nous connecter sur la page stats :



Non sécurisé 192.168.137.128:8080/stats Summarize Se connecter

HAProxy

Statistics Report for pid 1412

> General process information

pid = 1412 (process #1, nbproc = 1, nbthread = 2)
uptime = 0d 0h05m47s; warnings = 5
system limits: memmax = unlimited; ulimit-n = 8033
maxsock = 8033; **maxconn** = 4000; **reached** = 0; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 8.157 kbps
Running tasks: 0/20; idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.8)
- Online manual

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	2	-	1	2	3 000	3			969	29 441	0	0	1					OPEN								
Backend	0	0		0	0		0	0	300	0	0	0s	969	29 441	0	0		0	0	0	0	5m48s UP		0/0	0	0		0		

main

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	3 000	0			0	0	0	0	0					OPEN								

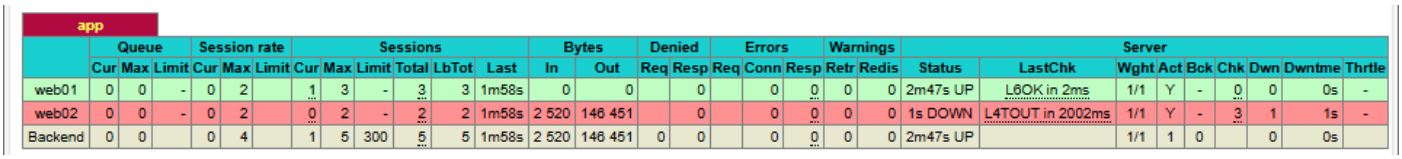
static

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
static	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	5m48s DOWN	L4CON in 0ms	1/1	Y	-	1	1	5m48s	-
Backend	0	0		0	0		0	0	300	0	0	?	0	0	0	0		0	0	0	0	5m48s DOWN		0/0	0	0		1	5m48s	

app

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
web01	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	42s UP	L6OK in 5ms	1/1	Y	-	1	1	5m3s	-
web02	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	1s UP	L6OK in 5ms	1/1	Y	-	1	1	5m43s	-
Backend	0	0		0	0		0	0	300	0	0	?	0	0	0	0		0	0	0	0	42s UP		2/2	2	0		1	5m2s	

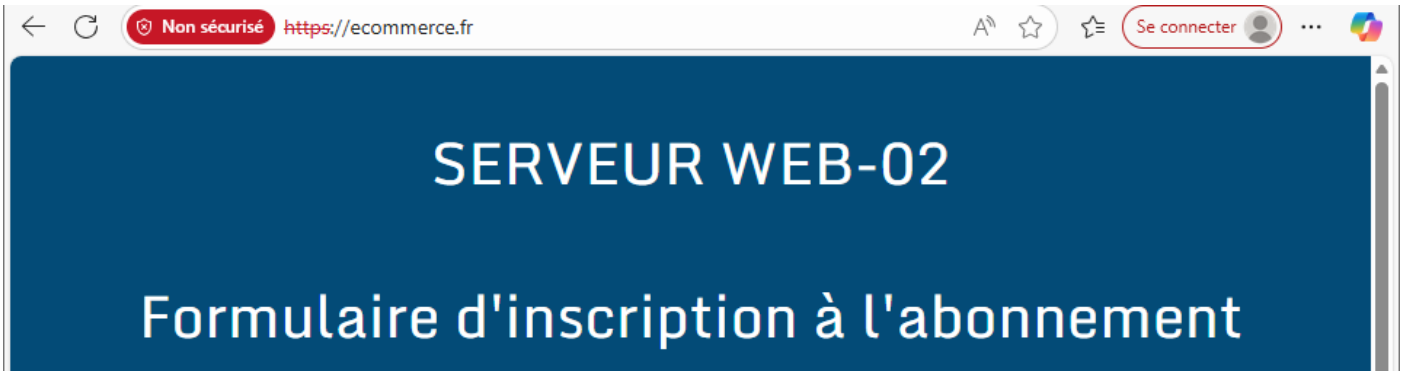
Si l'on désactive un serveur web nous pouvons le voir sur l'interface :

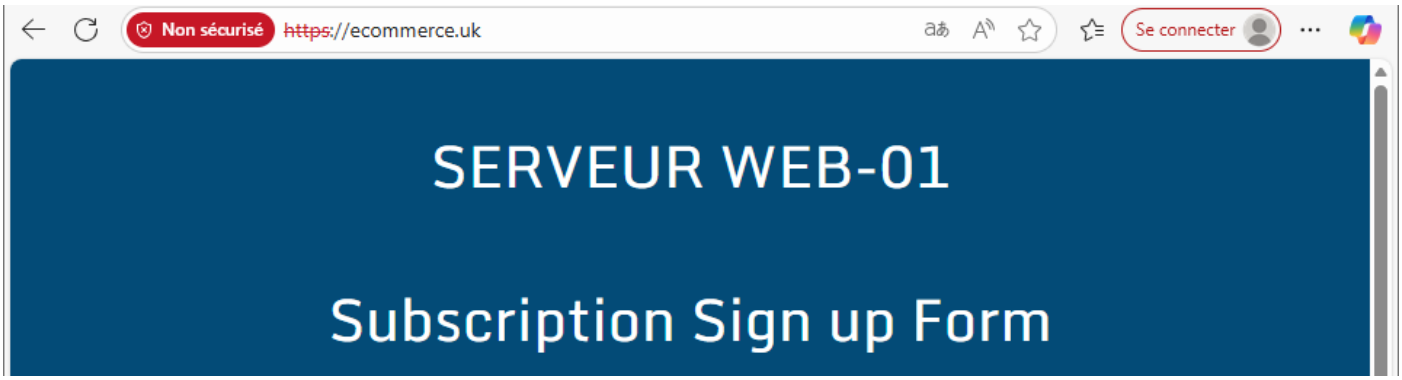


app

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
web01	0	0	-	0	2		1	3	-	3	3	1m58s	0	0		0		0	0	0	0	2m47s UP	L6OK in 2ms	1/1	Y	-	0	0	0s	-
web02	0	0	-	0	2		0	2	-	2	2	1m58s	2 520	146 451		0		0	0	0	0	1s DOWN	L4TOUT in 2002ms	1/1	Y	-	3	1	1s	-
Backend	0	0		0	4		1	5	300	5	5	1m58s	2 520	146 451	0	0		0	0	0	0	2m47s UP		1/1	1	0		0	0s	

Si l'on change notre fichiers host local afin de rediriger ecommerce.fr & .uk sur notre load-balancer le traffic est bien redirigé vers nos deux serveurs web :





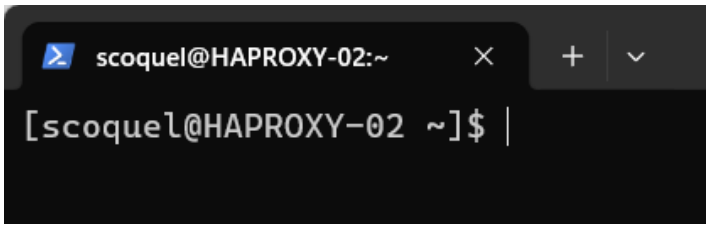
V. Haute disponibilité de HAProxy :

Grâce à la mise en place de HAProxy, nos serveurs WEB bénéficient désormais d'une répartition de charge et d'une première couche de haute disponibilité. Cependant, cette architecture introduit un nouveau point de défaillance unique (SPOF) : le serveur HAProxy lui-même.

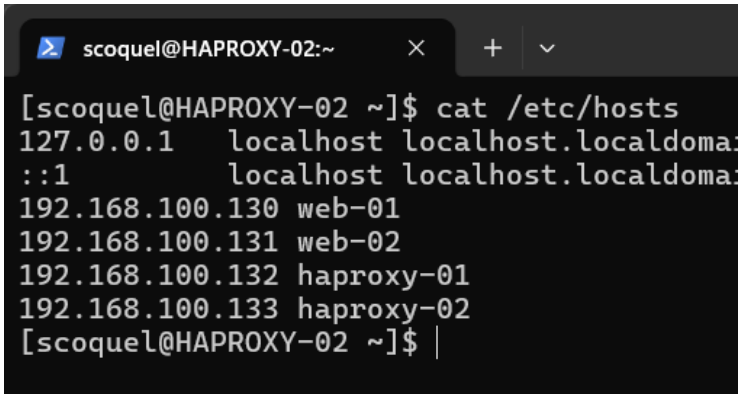
Pour palier à cela, nous allons mettre en place une redondance avec deux serveurs HAProxy et Pacemaker pour assurer la continuité de service.

1. Déploiement du 2ème Serveur HAProxy :

Comme pour les serveurs web, nous avons déployé un deuxième serveur avec une configuration identique au premier :



Nous avons aussi modifié sur tous les serveurs leurs fichier host afin qu'ils puissent communiquer via leur nom d'hôte :



```
[root@HAPROXY-01 scoquel]# nano /etc/hosts
[root@HAPROXY-01 scoquel]# ping haproxy-02
PING haproxy-02 (192.168.100.133) 56(84) bytes of data.
64 bytes from haproxy-02 (192.168.100.133): icmp_seq=1 ttl=64 time=2.00 ms
64 bytes from haproxy-02 (192.168.100.133): icmp_seq=2 ttl=64 time=0.923 ms
^C
--- haproxy-02 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.923/1.461/1.999/0.538 ms
[root@HAPROXY-01 scoquel]# ping web-01
PING web-01 (192.168.100.130) 56(84) bytes of data.
64 bytes from web-01 (192.168.100.130): icmp_seq=1 ttl=64 time=46.8 ms
^C
--- web-01 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 46.755/46.755/46.755/0.000 ms
[root@HAPROXY-01 scoquel]# ping web-02
PING web-02 (192.168.100.131) 56(84) bytes of data.
64 bytes from web-02 (192.168.100.131): icmp_seq=1 ttl=64 time=12.9 ms
64 bytes from web-02 (192.168.100.131): icmp_seq=2 ttl=64 time=13.9 ms
^C
--- web-02 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 12.900/13.397/13.895/0.497 ms
[root@HAPROXY-01 scoquel]#
```

2. Installation & Configuration de Pacemaker :

a. Installation & Authentification (Sur les deux serveurs) :

Pour installer Pacemaker nous avons besoin d'activer la librairie de paquets « highavailability » et ensuite via dnf install, d'installer pacemaker et pcs sur nos serveurs HAProxy :

```
[root@HAPROXY-01 scoquel]# dnf config-manager --set-enabled highavailability
[root@HAPROXY-01 scoquel]# dnf install pacemaker pcs
Rocky Linux 9 - High Availability                                248 kB/s | 143 kB     00:00
Dependencies resolved.
================================================================================================================
 Package                    Architecture   Version                      Repository              Size
================================================================================================================
Installing:
 pacemaker                  x86_64         2.1.10-1.el9                 highavailability        472 k
 pcs                        x86_64         0.11.10-1.el9_7.2            highavailability        4.0 M
Upgrading:
 libsss_certmap             x86_64         2.9.7-4.el9_7.1              baseos                   88 k
 libsss_idmap               x86_64         2.9.7-4.el9_7.1              baseos                   39 k
 libsss_nss_idmap           x86_64         2.9.7-4.el9_7.1              baseos                   43 k
 libsss_sudo                x86_64         2.9.7-4.el9_7.1              baseos                   33 k
 sssd-client                x86_64         2.9.7-4.el9_7.1              baseos                  158 k
 sssd-common                x86_64         2.9.7-4.el9_7.1              baseos                  1.6 M
 sssd-kcm                   x86_64         2.9.7-4.el9_7.1              baseos                  107 k
```

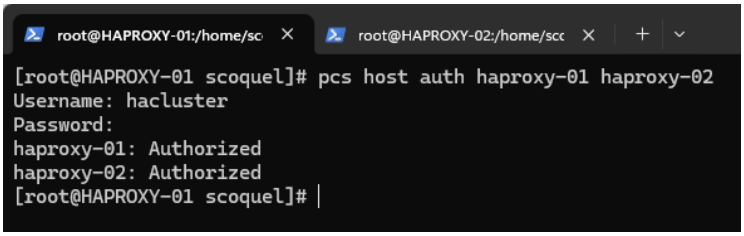
Une fois installé nous allons mettre un mot de passe à notre futur cluster :

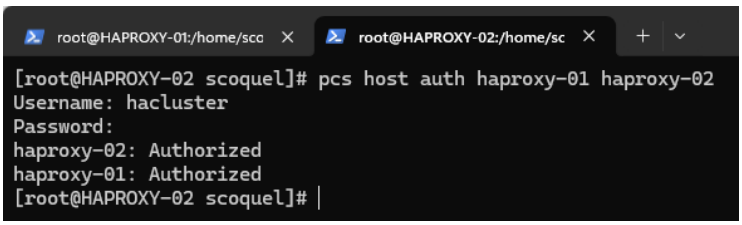
```
[root@HAPROXY-01 scoquel]# passwd hacluster
Changing password for user hacluster.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
```

Nous allons ouvrir les ports nécessaires via le service « High-availability » présent dans firewall-cmd :

```
[root@HAPROXY-01 scoquel]# firewall-cmd --add-service=high-availability --permanent
success
[root@HAPROXY-01 scoquel]# firewall-cmd --reload
success
[root@HAPROXY-01 scoquel]#
```

Et enfin nous allons réaliser l'authentification entre nos deux machines via la commande : **pcs host auth haproxy-01 haproxy-02 (Les noms inscrit dans /etc/hosts)**



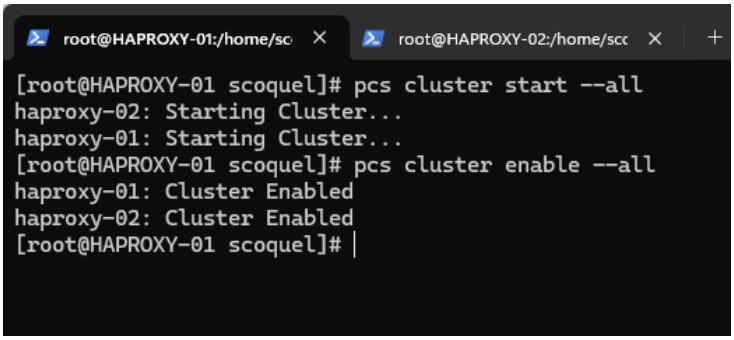


b. Création du cluster (sur le serveur principal) :

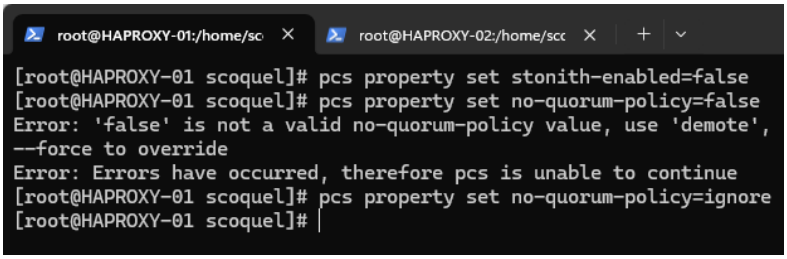
Sur le serveur principal nous allons créer notre cluster à l'aide de la commande : **pcs cluster setup ha_cluster haproxy-01 haproxy-02**

Pour l'activer et le démarrer il faut faire les commandes :

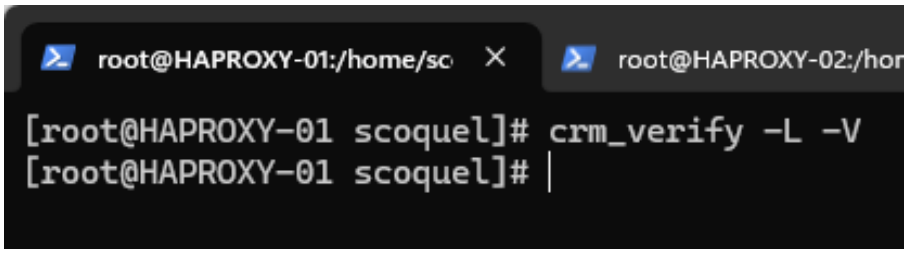
- **Pcs cluster start –all**
- **Pcs cluster enable –all**



Nous allons ensuite appliquer les paramètres suivants :



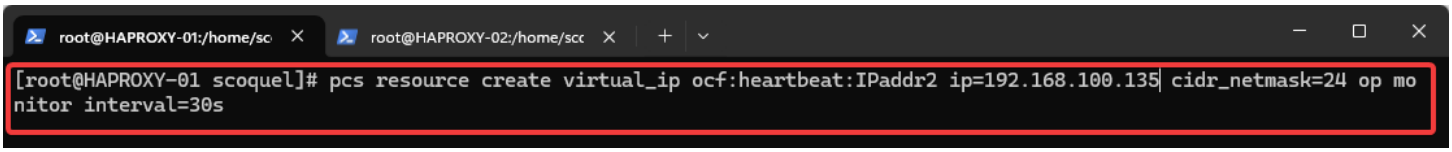
Et enfin vérifier que la configuration est bonne :



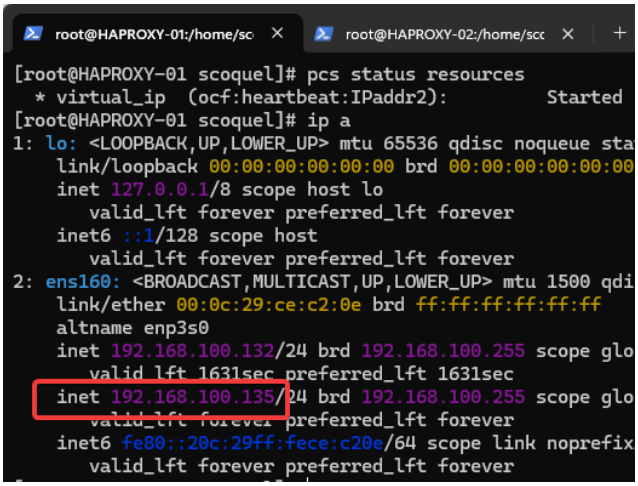
c. Création de l'IP Virtuelle (VIP) :

En tant qu'utilisateur, pour que puissions nous connecter à notre cluster, il faut lui affecter une IP Virtuelle qui sera utilisée par le serveur actif.

Pour l'activer, sur notre serveur principal il faut saisir la commande suivante : **pcs ressource create virtual_ip ocf :hearbeat :IPaddr2 ip=#IP# cidr_netmask=#CIDR# op monitor interval=30s**



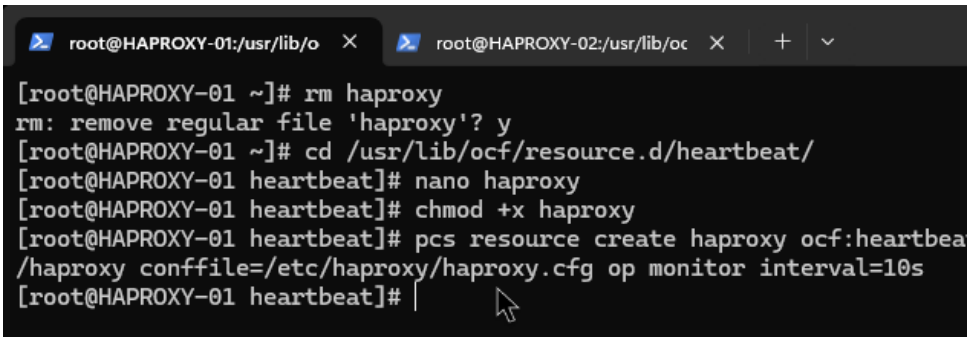
Avec la commande **pcs status ressources** et **ip -a** nous pouvons vérifier que notre ip est bien affectée au cluster et que notre serveur actif l'a bien prise en compte :



d. Activation de heartbeat pour HAProxy :

Pour terminer la mise en place de notre cluster nous allons activer l'agent lié à HAProxy sur les deux serveurs, tout en le téléchargeant au préalable si besoin

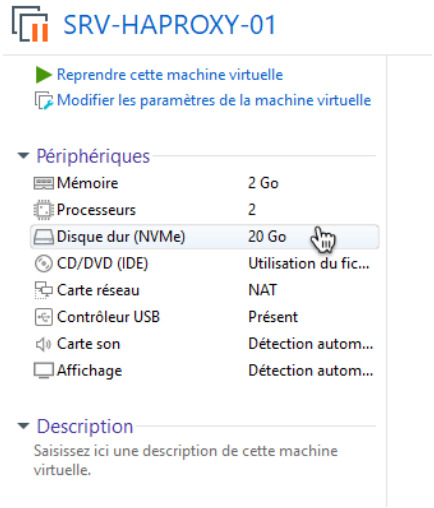
Pour l'activer il faut faire la commande : **pcs ressource create haproxy ocf :heartbeat binpath=#Repertoire# conffile=/etc/haproxy/haproxy.cfg op monitor interval=10s**



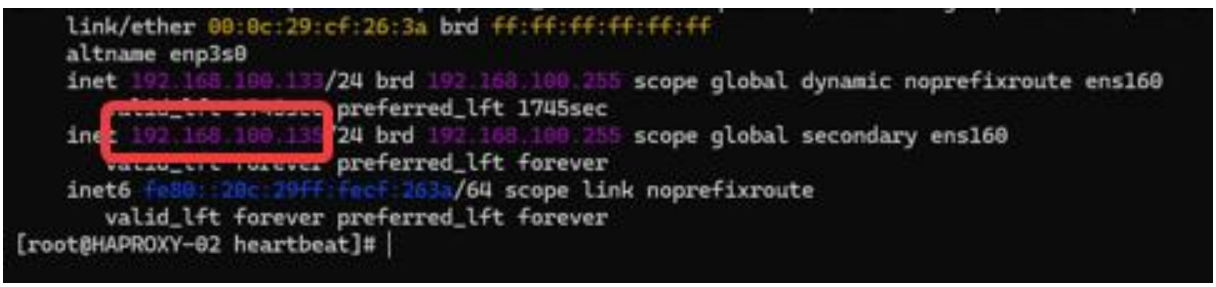
Une fois la commande effectuée sur les deux serveurs notre cluster est fonctionnel, nous pouvons l'essayer.

3. Tests :

Pour effectuer les tests nous allons désactiver notre HAProxy-01 (qui possède actuellement l'ip virtuelle) et voir si notre serveur 02 là récupère bien au bout d'un certain temps :



Au bout d'un court instant, le serveur 02 récupère bien l'ip virtuelle :



Si l’on veut accéder via cette ip nous arrivons bien sur nos serveurs WEB :