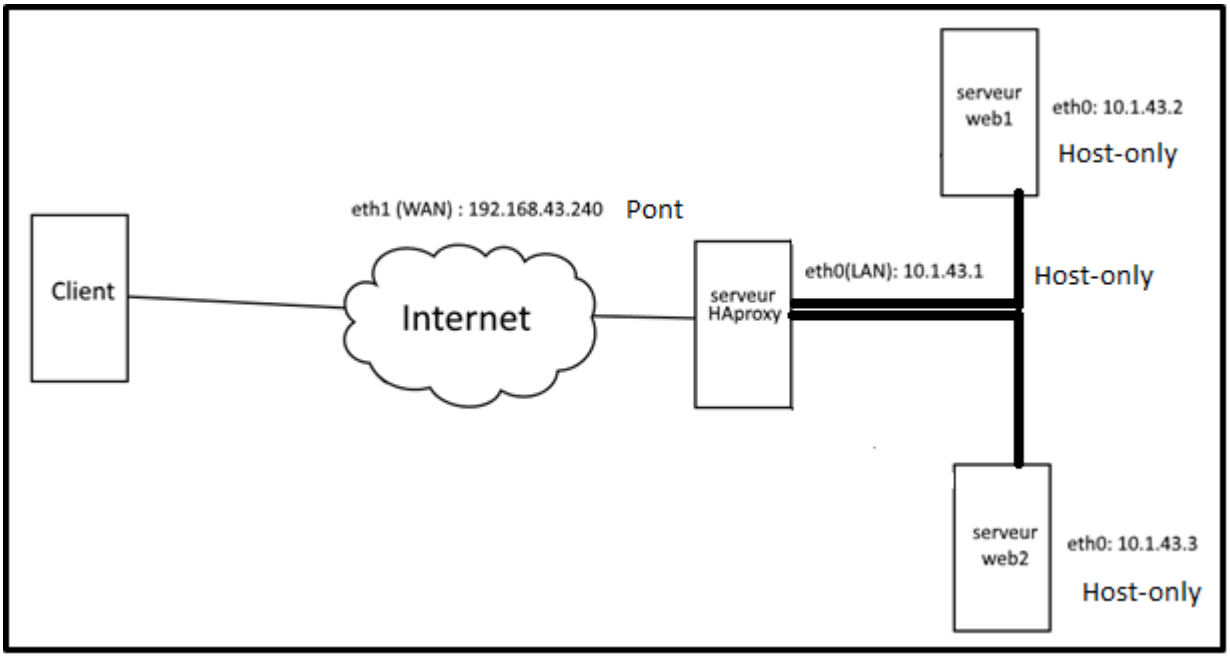
# Haproxy

Haproxy est un paquet permettant la mise en place d'un reverse proxy sur un serveur. Un *reverse proxy* (proxy inverse) est un serveur qui va se placer, par exemple, entre Internet et un cluster de serveurs afin de rediriger, authentifié ou répartir les paquets entre les serveurs. Ce qui nous intéresse ici est la répartition de charge, ou *load balancing*.

**Topologie de l'exercice**




Installation des paquets

Pour HAProxy le paquet s'appelle simplement haproxy et est présent sur le repo de Debian Stretch.

On utilise donc :

```
apt-get install haproxy
```

Ensuite pour les serveurs web1 et web2 il faudra simplement installer apache2, encore une fois présent sur le repo officiel Debian Stratch.

```
apt-get install apache2
```

**IMPORTANT**: N'installez pas Apache2 sur le serveur proxy, cela créerait des conflits par la suite.

**Configuration de HAproxy**

On ouvre le fichier de conf de haproxy

```
nano /etc/haproxy/haproxy.cfg
```

A la fin du fichier on va rajouter les lignes suivantes:

```
listen cluster_web
        bind 192.168.43.240:80

        mode http

        balance roundrobin

        option httpclose
        option forwardfor

        server web1 10.1.43.2:80 check
        server web2 10.1.43.3:80 check

        stats enable
        stats hide-version
        stats refresh 30s
        stats show-node
        stats auth admin:admin
        stats uri /stats
```

Détaillons un peu les paramètres:
Adresse publique du serveur proxy et port (80 pour http 443 pour https)

```
bind 192.168.43.240:80
```

Mode du proxy (http uniquement ou tcp)

```
mode http
```

Mode d'équilibrage, roundrobin correspond à 50/50

```
balance roundrobin
```

Adresse locale des serveurs web

```
server web1 10.1.43.2:80 check
server web2 10.1.43.3:80 check
```

Ensuite les paramètres de l'interface web

```
stats enable
stats hide-version
stats refresh 30s
stats show-node
stats auth admin:admin
stats uri /stats
```

Enregistrez vos modifications et quittez.

Tapez `services --status-all`

Haproxy doit apparaître comme activé (avec un +)

```
 [ + ]  haproxy
```

Si Haproxy ne veut pas démarrer c'est qu'il y a une erreur dans votre fichier de conf.

Quelques exemples:

Vous n'avez pas tapé "*bind*" devant l'adresse du serveur Haproxy
Vous n'avez pas tapé la bonne adresse serveur Haproxy
Vous avez apache2 d'installé sur la machine Haproxy

**Configuration des serveurs web**

Vous avez donc installé apache2 sur les deux serveurs, il va donc falloir configurer les serveurs pour qu'ils utilisent le proxy en tant que passerelle.

On va donc dans la configuration de l'interface

```
nano /etc/network/interfaces
```

```
# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 10.1.43.2
gateway 10.1.43.1
```

On rempli la gateway avec l'adresse locale du serveur Haproxy.

Une fois que c'est fait on va aller modifier le fichier par défault de Apache2 pour différencier nos serveurs.

```
cd /var/www/html
```

On supprime le fichier par défaut et on crée un fichier index.html que l'on rempli de la façon suivante, en fonction des postes:

```
<!DOCTYPE html>
<html>
<body>

<h1>Serveur 1</h1>

<p>Ceci est le serveur web 1. </p>

</body>
</html>
```

Et bien évidemment la même chose pour le serveur 2:

```
<!DOCTYPE html>
<html>
<body>

<h1>Serveur 2</h1>

<p>Ceci est le serveur web2</p>

</body>
</html>
```

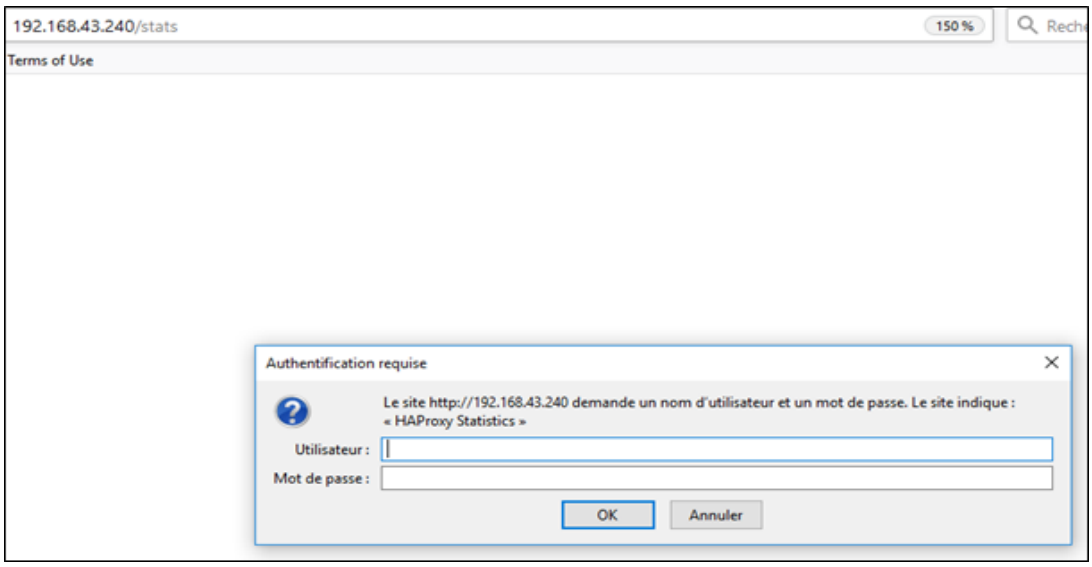
**Test du Haproxy**

A partir d'une machine windows connectée sur le réseau public on va se connecter à l'adresse du Haproxy


On est tombé sur le serveur 2, on actualise la page:


Et on tombe sur le serveur 1.

On va ensuite tester la page de stats en tapant https://adresse.du.serveur.haproxy/stats




Il nous demande un user et mot de passe, pour nous ce sera admin admin.

On arrive ensuite sur la page suivante, avec les statistiques de nos serveurs

## *Statistics Report for pid 629 on Docker1*

### > General process information

**pid =** 629 (process #1, nbproc = 1)
**uptime =** 0d 1h36m57s
**system limits:** memmax = unlimited; ulimit-n = 4033
**maxsock =** 4033; **maxconn =** 2000; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/6; 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
- Disable refresh
- Refresh now
- CSV export

**External resources:**
- Primary site
- Updates (v1
- Online manu

**cluster_web**

| | 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 |
| Frontend | | | | 1 | 8 | - | 1 | 1 | 2 000 | 207 | | | 83 624 | 825 100 | 0 | 0 | 0 | | | | | OPEN | | | | | | | |
| web1 | 0 | 0 | - | 0 | 4 | | 0 | 1 | - | 75 | 75 | 30s | 29 345 | 34 167 | | 0 | | 0 | 0 | 0 | 0 | 1h36m UP | L4OK in 0ms | 1 | Y | - | 0 | 0 | 0s |
| web2 | 0 | 0 | - | 0 | 4 | | 0 | 1 | - | 74 | 74 | 8m37s | 30 832 | 32 198 | | 0 | | 0 | 0 | 0 | 0 | 1h36m UP | L4OK in 0ms | 1 | Y | - | 0 | 0 | 0s |
| Backend | 0 | 0 | | 0 | 8 | | 0 | 1 | 200 | 149 | 149 | 0s | 83 624 | 825 100 | 0 | 0 | | 0 | 0 | 0 | 0 | 1h36m UP | | 2 | 2 | 0 | | 0 | 0s |

Nous allons simuler la panne d'un des serveurs.
Pour ce faire on va tout simplement stopper le service apache sur un des serveurs

```
root@web1:/var/www/html# service apache2 stop
root@web1:/var/www/html#
```

On actualise notre page de stats:

**cluster_web**

| | 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 |
| Frontend | | | | 1 | 8 | - | 1 | 1 | 2 000 | 225 | | | 90 878 | 955 788 | 0 | 0 | 0 | | | | | OPEN | | | | | | | |
| web1 | 0 | 0 | - | 0 | 4 | | 0 | 1 | - | 79 | 79 | 1m | 30 909 | 36 043 | | 0 | | 0 | 0 | 0 | 0 | 44s DOWN | L4CON in 0ms | 1 | Y | - | 3 | 1 | 44s |
| web2 | 0 | 0 | - | 0 | 4 | | 0 | 1 | - | 79 | 79 | 30s | 32 787 | 34 543 | | 0 | | 0 | 0 | 0 | 0 | 1h41m UP | L4OK in 0ms | 1 | Y | - | 0 | 0 | 0s |
| Backend | 0 | 0 | | 0 | 8 | | 0 | 1 | 200 | 158 | 158 | 0s | 90 878 | 955 788 | 0 | 0 | | 0 | 0 | 0 | 0 | 1h41m UP | | 1 | 1 | 0 | | 0 | 0s |

On voit que le serveur web1 apparait bien hors ligne, et maintenant si on va juste sur l'adresse web, on aura que la page du serveur web2.

# SSH et routing via IPTABLES

Actuellement nos serveurs web n'ont pas accès à internet. Pour les mettre à jour, il peut être intéressant de leur donner cet accès en utilisant le serveur Proxy comme routeur. Cela nous permettra aussi d'établir des redirections SSH.

Nous avons donc récupéré un script depuis le site denisrozenkranz.com et l'avons adapté à notre situation. **Tout se passe sur le serveur Haproxy.**

On va aller se placer dans /etc/init.d/

Notre script va se comporter comme un service. Il faut savoir que tout script dans init.d n'a pas d'extension. Appelez donc votre script *firewall* par exemple et non *firewall.sh*.

```
:~# cd /etc/init.d
:/etc/init.d# touch firewall
```

Ensuite donnez les droits d'exécution au fichier

```
chmod +x firewall
```

On vérifie avec ls -l

```
-rwxr-xr-x 1 root root  943 nov.   9 12:13 firewall
```

Les x confirment les droits d'exécution.

Ensuite ouvrez le fichier nouvellement créé avec nano ou vi et remplissez le comme suit

```
start() {
        echo -n "Application des regles IpTables: "

        iptables -F
        iptables -X
        iptables -t nat -F

        #echo 1 > /proc/sys/net/ipv4/ip_forward
        #remplacé par
        sysctl -w net.ipv4.ip_forward=1 #On active le forwarding dans linux

        iptables -P FORWARD DROP #On vérifie que la stratégie de forward par défaut rejette les packets


        iptables -A FORWARD -d 10.1.43.2 -j ACCEPT # On autorise manuellement le forwarding pour nos serveurs
        iptables -A FORWARD -s 10.1.43.2 -j ACCEPT # web. On ne veut pas autoriser le forwarding pour tous les
        iptables -A FORWARD -d 10.1.43.3 -j ACCEPT # hôtes car cela représenterait une vulnérabilité.
        iptables -A FORWARD -s 10.1.43.3 -j ACCEPT #

        #Nating du réseau privé vers extérieur
        iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE

        #configuration accès aux serveurS web par SSH
        iptables -t nat -A PREROUTING -p tcp -d 192.168.43.240 --dport 2221 -j DNAT --to-destination 10.1.43.2:22
        iptables -t nat -A PREROUTING -p tcp -d 192.168.43.240 --dport 2222 -j DNAT --to-destination 10.1.43.3:22

        echo "termine"
}
```

```
case $1 in
        start)
            start
             ;;
        stop)
            stop
             ;;
        restart)
             stop
             start
          ;;
        status)
           /sbin/iptables -L
           /sbin/iptables -t nat -L
          ;;
        *)
           echo "Usage: firewall {start|stop|restart|status)"

esac
exit
```

Sauvegardez votre fichier et quittez

Ensuite tapez

```
systemctl daemon-reload
```

Cela va forcer l'OS a actualiser les scripts des services et donc à prendre en compte votre nouveau script/service.

Ensuite vous pouvez démarrer le script comme un service:

```
service firewall start
```

On vérifie avec les commandes iptables

```
root@Docker1:/etc/init.d# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     all  -- anywhere             10.1.43.2
ACCEPT     all  -- 10.1.43.2            anywhere
ACCEPT     all  --  anywhere            10.1.43.3
ACCEPT     all  --  10.1.43.3           anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

```
root@Docker1:/etc/init.d# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       tcp  --  anywhere             Docker1-1625         tcp dpt:2221 to:10.1.43.2:22
DNAT       tcp  --  anywhere             Docker1-1625         tcp dpt:2222 to:10.1.43.3:22

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere
```

*(Note la destination "Docker1-1625" correspond simplement au hostname de la machine)*

On voit bien nos règles de FORWARDING pour l'accès internet, les règles de PREROUTING pour le SSH ainsi que les règles de POSTROUTING via MASQUERADE pour la translation d'addresses sont bien présente.

**Test du routing et de la connexion SSH**

Depuis les machines web on test la connectivité à internet.
Dans mon cas ces machines n'ont jamais été sur un réseau public, et n'ont donc jamais récupéré de DNS. On va donc l'entrer manuellement, sur les deux machines web.

```
nano resolv.conf
```

```
  GNU nano 2.7.4                    Fichier : resolv.conf

nameserver 8.8.8.8
```

Ensuite on test la connectivité à Internet

```
root@web1:/etc# ping google.com
PING google.com (216.58.204.110) 56(84) bytes of data.
64 bytes from par10s28-in-f110.1e100.net (216.58.204.110): icmp_seq=1 ttl=49 tim
e=40.8 ms
64 bytes from par10s28-in-f110.1e100.net (216.58.204.110): icmp_seq=2 ttl=49 tim
e=77.8 ms
```

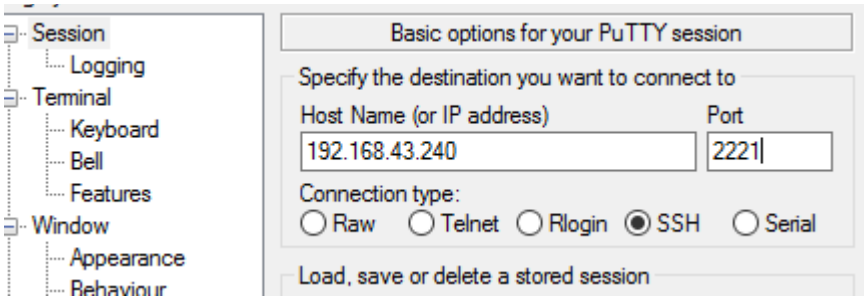
Sur le web1 ça fonctionne, on test donc sur le web2 sur un update

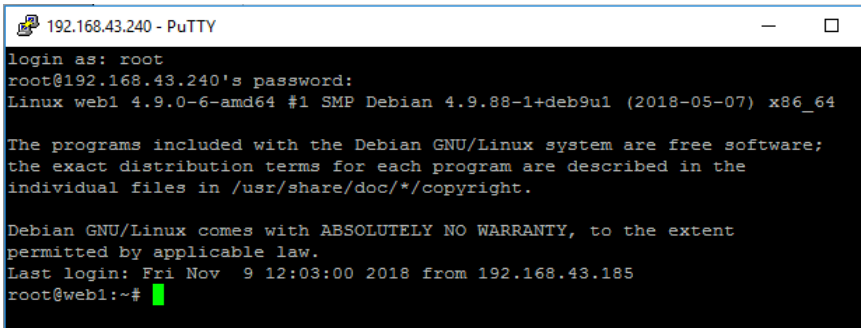
```
root@web2:/etc# apt-get update
Réception de:1 http://security.debian.org/debian-security stretch/updates InRele
ase [94,3 kB]
Ign:2 http://ftp.fr.debian.org/debian stretch InRelease
Réception de:3 http://ftp.fr.debian.org/debian stretch-updates InRelease [91,0 k
B]
Réception de:4 http://security.debian.org/debian-security stretch/updates/main S
ources [185 kB]
Réception de:5 http://ftp.fr.debian.org/debian stretch Release [118 kB]
```

Maintenant on va tester le SSH. (le paquet ssh est normalement proposé à l'installation de Debian, sinon, le paquet est openssh-server)

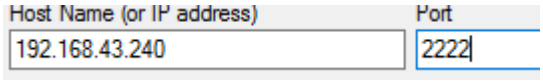
Pour rappel, selon notre script, si on se connecte en SSH à l'adresse  du serveur Haproxy (192.168.43.240) en fonction du port (2221 ou 2222) on arrivera respectivement sur web1 ou web2.

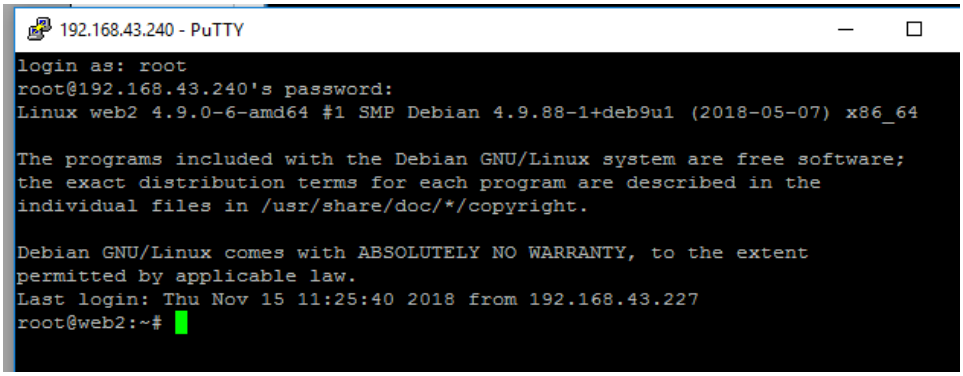
On va donc tester sur notre machine windows en 192.168.43.X avec Putty.




On constate qu'on est bien sur le serveur web1.
On recommence en changeant le port



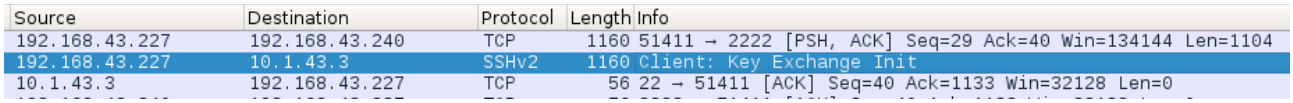




Cela fonctionne.

On peut confirmer nos tests en utilisant wireshark placé sur le serveur Haproxy

Notre machine windows (client) a pour adresse 192.168.43.227

On va donc se connecter au serveur en SSH sur le port 2222 à partir du client et observer les paquets de 192.168.43.227 sur wireshark.



| Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|
| 192.168.43.227 | 192.168.43.240 | TCP | 1160 | 51411 → 2222 [PSH, ACK] Seq=29 Ack=40 Win=134144 Len=1104 |
| 192.168.43.227 | 10.1.43.3 | SSHv2 | 1160 | Client: Key Exchange Init |
| 10.1.43.3 | 192.168.43.227 | TCP | 56 | 22 → 51411 [ACK] Seq=40 Ack=1133 Win=32128 Len=0 |

On constate qu'au départ le client windows (227) passe par le proxy (240) puis établit son SSH directement avec 10.1.43.3