# MISE EN PLACE D’UN LOAD BALANCER IPVS AVEC KEEPALIVED (CAS COMPLET)

## 1. CONTEXTE ET ARCHITECTURE

On met en place une infra de load balancing L4 avec IPVS en mode DR (Direct Routing), puis on ajoute Keepalived pour la haute dispo.

### Machines

- FRONTAL : client
- NODE01 : load balancer principal
- NODE02 : load balancer secours
- WWW1 : backend 1
- WWW2 : backend 2

### Adressage

- VIP : **172.20.0.83**
- NODE01 : 172.20.0.81
- NODE02 : 172.20.0.82
- WWW1 : 172.20.0.85
- WWW2 : 172.20.0.86

# 2. PREMIERE ETAPE : IPVS EN MANUEL (SANS KEEPALIVED)

## Objectif

Configurer IPVS directement pour comprendre le fonctionnement.

## 2.1 Vérification initiale

```
ipvsadm
```

Sortie vide → normal, aucune règle.

## 2.2 Création du service virtuel

```
ipvsadm -A -t 172.20.0.83:80 -s rr
```

Explication :

- `-A` : add service
- `-t` : service TCP
- `172.20.0.83:80` : VIP + port
- `-s rr` : round robin

## 2.3 Ajout des serveurs backend

```
ipvsadm -a -t 172.20.0.83:80 -r 172.20.0.85:80 -g
ipvsadm -a -t 172.20.0.83:80 -r 172.20.0.86:80 -g
```

Explication :

- `-a` : add real server
- `-r` : backend
- `-g` : mode DR (Direct Routing)

## 2.4 Vérification

```
ipvsadm -L -n
```

Résultat attendu :

```
TCP 172.20.0.83:80 rr
 -> 172.20.0.85:80 Route
 -> 172.20.0.86:80 Route
```

# 3. MODE DR : COMPRENDRE LE FONCTIONNEMENT

## Ce que fait IPVS

- Le client envoie vers VIP
- Le LB reçoit
- Il change uniquement la MAC destination
- Le backend répond DIRECT au client

Donc :

IMPORTANT → le backend doit pouvoir répondre sans repasser par le LB

# 4. CONFIGURATION DES BACKENDS (WWW1 / WWW2)

## 4.1 Ajout de la VIP sur les backends

Sur WWW1 et WWW2 :

```
ip addr add 172.20.0.83/24 dev ens18 label ens18:0
```

Ou via `/etc/network/interfaces` :

```
auto ens18:0
iface ens18:0 inet static
    address 172.20.0.83/24
```

## 4.2 PROBLEME MAJEUR : CONFLIT ARP

Si on fait ça sans protection :

les backends vont répondre aux requêtes ARP pour la VIP
ça casse tout

## 4.3 NEUTRALISATION ARP (OBLIGATOIRE)

### Méthode utilisée (arptables)

Sur WWW1 et WWW2 :

```
arptables -A INPUT -d 172.20.0.83 -j DROP
```

Explication :

- bloque toute requête ARP vers la VIP
- empêche le backend de répondre

**Persistance**

```
nano /etc/network/if-up.d/arp-block
#!/bin/bash
arptables -F
arptables -A INPUT -d 172.20.0.83 -j DROP
chmod +x /etc/network/if-up.d/arp-block
```

## 4.4 Résultat attendu

- VIP présente sur backend
- MAIS invisible sur le réseau (pas d'ARP)
- seul le LB répond

# 5. TEST IPVS

Depuis le frontal :

```
curl http://172.20.0.83
```

Résultat :

```
www1
www2
www1
www2
```

Round robin OK

# 6. PROBLEME RENCONTRE (CAS REEL)

## Symptôme

- parfois curl ne répond pas
- obligé de faire un ping avant
- ActiveConn augmente mais pas de réponse

## Cause

Problème ARP / cache :

- client ne connaît pas MAC backend
- backend ne connaît pas client
- paquet retour perdu

## Exemple vu dans ton cas

```
ping backend → OK
curl → OK
```

Donc :

👉 le ping réinitialise ARP

# 7. AUTRE PROBLEME : IP FANTOME (CRITIQUE)

Sur WWW1 :

```
inet 172.20.0.83/24
inet 172.20.0.97/24 dynamic
```

dhcp actif → ajoute IP
conflit avec config statique

## Correction

```
systemctl stop dhcpcd
systemctl disable dhcpcd
killall dhcpcd
```

## Nettoyage

```
ip addr flush dev ens18
systemctl restart networking
```

# 8. AJOUT DE KEEPALIVED

## Objectif

- rendre le LB HA
- déplacer la VIP automatiquement

## NODE01 (MASTER)

```
vrrp_instance VI_1 {
    state MASTER
    interface ens18
    virtual_router_id 1
    priority 100
    advert_int 1

    virtual_ipaddress {
        172.20.0.83/24
    }
}
```

## NODE02 (BACKUP)

```
vrrp_instance VI_1 {
    state BACKUP
    interface ens18
    virtual_router_id 1
    priority 90
    advert_int 1

    virtual_ipaddress {
        172.20.0.83/24
    }
}
```

# 9. COUPLAGE KEEPALIVED + IPVS

Ajout dans config :

```
virtual_server 172.20.0.83 80 {
    delay_loop 5
    lb_algo rr
    lb_kind DR
    protocol TCP

    real_server 172.20.0.85 80 {
        TCP_CHECK {
            connect_timeout 3
        }
    }

    real_server 172.20.0.86 80 {
```

```
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
```

## Effet

- IPVS configuré automatiquement
- backend down → supprimé
- plus de requêtes perdues

# 10. RESULTAT FINAL

- VIP portée par un seul LB
- bascule automatique si panne
- IPVS distribue correctement
- backends invisibles au niveau ARP
- plus besoin de ping

# 11. POINTS CRITIQUES A RETENIR

## 1. ARP

Toujours maîtriser ARP en mode DR
Sinon :

- pertes aléatoires
- comportement instable

## 2. VIP

- sur LB → visible
- sur backend → silencieuse

## 3. DHCP

À désactiver sinon :

- IP fantôme
- conflits
- debug impossible

## 4. DEBUG

Commandes utiles :

```
ipvsadm -L -n
tcpdump -ni ens18 arp
ip neigh
```

# 12. WARNING IMPORTANT

Toute IP résiduelle ou dynamique sur une interface utilisée pour IPVS peut provoquer :

- conflit ARP
- perte de trafic
- incohérence de routage

Toujours faire :

```
ip addr flush dev ens18
systemctl restart networking
```

avant mise en production.

# Loadbalancer UDP DNS port 53

Pour de la répartition de charge entre plusisurs serveur DNS mieux vaut utiliser SH au lieu de round robin car SH = source hashing le meme client gardera le meme serveur DNS et à chaque requête ne questionnera pas un autre DNS un à la fois

J’installe deux serveur DNS sur www1 et www2

On voit ici que les deux serveur DNS fonctionnent

```
Address: 172.20.0.85#53
> google.com
Server:         172.20.0.85
Address:        172.20.0.85#53

Non-authoritative answer:
Name:   google.com
Address: 172.217.22.174
Name:   google.com
Address: 2a00:1450:4007:81b::200e
> exit

root@www1:~# nslookup
> server 172.20.0.86
Default server: 172.20.0.86
Address: 172.20.0.86#53
> google.com
Server:         172.20.0.86
Address:        172.20.0.86#53

Non-authoritative answer:
Name:   google.com
Address: 172.217.22.206
Name:   google.com
Address: 2a00:1450:4007:80f::200e
>
```

De manière individuelle

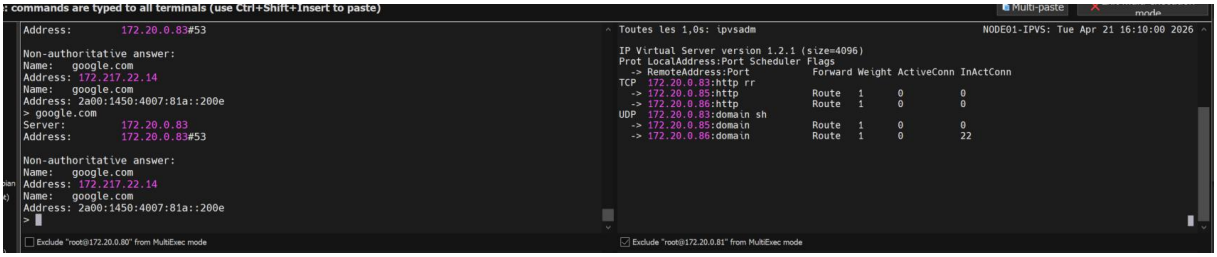
# Conf Keepalived

```
virtual_server 172.20.0.83 53 {
        delay_loop 5
        lb_algo sh
        lb_kind DR
        protocol UDP
        real_server 172.20.0.85 53 {
                UDP_CHECK {
                        connect_timeout 3
}
}

        real_server 172.20.0.86 53 {
                UDP_CHECK {
                        connect_timeout 3
}
}

}
```

Ensuite je restart le service keepalived et je lance ipvsadm pour voir si la conf a bien été prise en compte

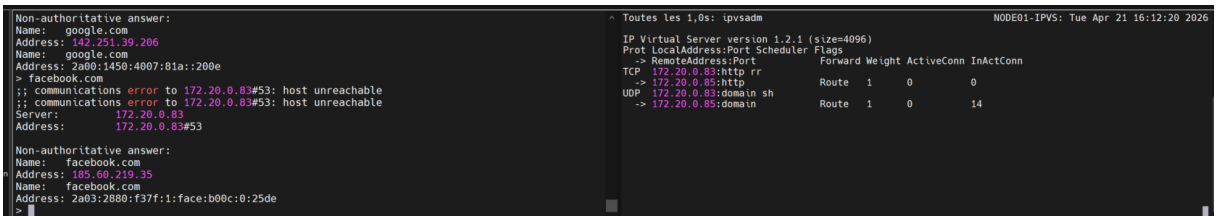
```
root@NODE01-IPVS:~# service keepalived restart
root@NODE01-IPVS:~# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  172.20.0.83:http rr
  -> 172.20.0.85:http             Route   1      0          0
  -> 172.20.0.86:http             Route   1      0          0
UDP  172.20.0.83:domain sh
  -> 172.20.0.85:domain           Route   1      0          0
  -> 172.20.0.86:domain           Route   1      0          0
root@NODE01-IPVS:~#
```

On voit que la machine frontale ne questionne que la 86




Si je coupe le lien de www2 je vais voir si la bascule se fait automatiquement de manière transparente

On voit bien ici la machine 86 sortir du cluster et la 85 prends les requetes




J’ai mis le retry à 1 car c’était trop long d’attendre


# Important en prod serveurs dans un vlan séparer

Pour ne pas avoir des soucis en prod il faut bien séparer les serveurs du réseau client dans un autre VLAN pour éviter d'avoir des conflits d'adresse MAC par exemple la machine frontal était dans le meme vlan que les nœud + backend du coup il pouvait y'avoir des conflits d'adresse mac quand un nœud était down et repasser up il n'arrivait quand meme pas à la joindre en RD routing direct comme configurer dans notre cluster mais si c'est dans un autre vlan vu que les arp broadcast et les broadcaste en général  ne passe pas on préserve nos clients d'une instabilité dans la table d'adresse mac