HAProxy

Repartition de charge et Tolerance de panne

Infrastructure mvgroup.local — Proxmox VE

Auteur	Samuel
Version	1.0
Date	Mai 2026
Hyperviseur	Proxmox VE — 10.0.0.0/16
Domaine	mvgroup.local

1. Introduction et objectifs

Ce document decrit la mise en place d'une infrastructure de repartition de charge (load balancing) et de tolerance de panne (failover) basee sur HAProxy au sein du lab mvgroup.local.

1.1 Contexte

Dans un environnement de production, il est essentiel de garantir la disponibilite des services web. Ce projet repond a deux besoins fondamentaux :

- Repartition de charge : distribuer les requetes HTTP entre plusieurs serveurs web pour optimiser les performances
- Tolerance de panne : assurer la continuite du service si l'un des serveurs web tombe en panne

1.2 Solution retenue

La solution retenue est HAProxy (High Availability Proxy), un logiciel open source de reference pour le load balancing. Il est deploye sur une VM Ubuntu 24.04 dediee et distribue le trafic vers deux serveurs web Flask containerises avec Docker.

1.3 Objectifs pedagogiques

- Comprendre le fonctionnement d'un reverse proxy / load balancer
- Mettre en oeuvre la containerisation avec Docker et docker-compose
- Configurer HAProxy en mode round-robin avec health checks
- Valider la tolerance de panne par des tests pratiques
- Analyser les statistiques de trafic via l'interface HAProxy Stats

2. Architecture de la solution

2.1 Schema de l'infrastructure

L'architecture repose sur 3 VMs Ubuntu 24.04 deployees dans Proxmox VE, toutes dans le reseau 10.0.0.0/16 :

2.2 Plan d'adressage

VM / Hostname	Adresse IP	OS	Role
HAPROXY-SAM	10.0.0.30	Ubuntu 24.04	Load balancer HAProxy
WEB1-SAM	10.0.0.31	Ubuntu 24.04	Serveur web Flask (serveur 1)
WEB2-SAM	10.0.0.32	Ubuntu 24.04	Serveur web Flask (serveur 2)

2.3 Flux reseau

Flux	Port	Description
Client → HAPROXY-SAM	TCP 80	Requetes HTTP entrantes
HAPROXY-SAM → WEB1-SAM	TCP 80	Distribution round-robin
HAPROXY-SAM → WEB2-SAM	TCP 80	Distribution round-robin
Admin → HAPROXY-SAM	TCP 8404	Interface statistiques HAProxy

2.4 Stack technique

Composant	Version	Role
HAProxy	2.8.16	Load balancer — reverse proxy
Docker Engine	29.5.2	Containerisation des serveurs web
docker-compose	5.1.4	Orchestration des containers Flask
Flask (Python)	latest	Application web legere
Ubuntu Server	24.04 LTS	OS des 3 VMs

3. Configuration des VMs dans Proxmox

3.1 Creation des VMs

Les 3 VMs ont ete creees dans Proxmox VE avec la configuration suivante :

- OS : Ubuntu Server 24.04 LTS
- CPU : 2 coeurs
- RAM : 2048 Mo
- Disque : 20 Go
- Reseau : vmbr1 — VirtIO

3.2 Configuration reseau (systemd-networkd)

Sur chaque VM, NetworkManager a ete desactive au profit de systemd-networkd pour garantir la persistance de la configuration reseau apres reboot.

Commandes appliquees sur chaque VM :

```
systemctl disable NetworkManager
systemctl stop NetworkManager
systemctl enable systemd-networkd
systemctl enable systemd-resolved
ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

Fichier Netplan — exemple pour HAPROXY-SAM (/etc/netplan/00-config.yaml) :

```
network:
  version: 2
  renderer: networkd
  ethernets:
    ens18:
      dhcp4: no
      addresses:
        - 10.0.0.30/16
      routes:
        - to: default
          via: 10.0.0.1
      nameservers:
        addresses:
          - 10.0.200.2
          - 8.8.8.8
```

4. Deploiement des serveurs web Flask avec Docker

4.1 Installation de Docker

Docker a ete installe sur WEB1-SAM et WEB2-SAM en suivant la methode officielle via le depot Docker pour Ubuntu Noble (24.04).

Ajout du depot officiel Docker :

```
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
chmod a+r /etc/apt/keyrings/docker.asc

echo "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  tee /etc/apt/sources.list.d/docker.list
```

Installation des paquets Docker :

```
apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

4.2 Structure du projet Flask

Le professeur a fourni une archive flask-serveur.zip contenant deux dossiers identiques, avec une seule difference dans src/serveur.py :

- flask/ : application affichant 'Serveur 1' — deployee sur WEB1-SAM
- flask2/ : application affichant 'Serveur 2' — deployee sur WEB2-SAM

Structure de chaque dossier :

```
flask/
  docker-compose.yml    # Orchestration du container
  python/
    Dockerfile          # Image Python + Flask
  src/
    serveur.py          # Application Flask
```

4.3 Deploiement des containers

Transfert de l'archive vers les VMs via SCP avec rebond Tailscale :

```
scp -J root@100.99.224.90 flask-serveur.zip samuel@10.0.0.31:/home/samuel/
```

Lancement sur WEB1-SAM (dossier flask) :

```
cd /root/flask-serveur/flask
ufw allow 80
docker compose up -d
```

Lancement sur WEB2-SAM (dossier flask2) :

```
cd /root/flask-serveur/flask2
```

```
ufw allow 80
docker compose up -d
```

4.4 Verification des serveurs web

Test de reponse depuis chaque VM :

```
curl http://localhost   # Sur WEB1-SAM → repond : Serveur 1
curl http://localhost   # Sur WEB2-SAM → repond : Serveur 2
```

5. Installation et configuration de HAProxy

5.1 Installation

HAProxy a ete installe sur HAPROXY-SAM depuis les depots officiels Ubuntu :

```
apt-get update
apt-get install -y haproxy
haproxy -v   # HAProxy version 2.8.16
```

5.2 Configuration HAProxy

Le fichier de configuration /etc/haproxy/haproxy.cfg a ete edite comme suit :

Section global :

```
global
    log /dev/log local0
    log /dev/log local1 notice
    maxconn 2000
    daemon
```

Section defaults :

```
defaults
    log     global
    mode    http
    option  httplog
    option  dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms
```

Frontend (point d'entree) :

```
frontend web_frontend
    bind *:80
    default_backend web_servers
```

Backend (serveurs cibles) :

```
backend web_servers
    balance roundrobin
    option httpchk GET /
    server WEB1-SAM 10.0.0.31:80 check
    server WEB2-SAM 10.0.0.32:80 check
```

Interface de statistiques :

```
listen stats
    bind *:8404
    stats enable
    stats uri /stats
    stats refresh 10s
    stats admin if TRUE
```

5.3 Explication des parametres cles

Parametre	Description
balance roundrobin	Distribue les requetes a tour de role entre les serveurs
option httpchk GET /	Health check HTTP : verifie que le serveur repond a GET /
check	Active la surveillance automatique de chaque serveur
bind *:80	Ecoute sur toutes les interfaces reseau sur le port 80
timeout connect	Delai max pour etablir la connexion au serveur (5s)
stats refresh 10s	Rafraichissement automatique des stats toutes les 10 secondes

5.4 Demarrage du service

```
haproxy -c -f /etc/haproxy/haproxy.cfg   # Validation de la config
systemctl enable haproxy
systemctl restart haproxy
systemctl status haproxy
```

6. Tests et validation

6.1 Test 1 — Repartition de charge (round-robin)

Ce test valide que HAProxy distribue correctement les requetes entre les deux serveurs en alternance.

Commande executee depuis HAPROXY-SAM :

```
for i in {1..6}; do curl http://10.0.0.30; echo; done
```

Resultat obtenu :

```
Serveur 1
Serveur 2
Serveur 1
Serveur 2
Serveur 1
Serveur 2
```

Le round-robin est confirme : chaque requete est distribuee alternativement entre WEB1-SAM et WEB2-SAM.

6.2 Test 2 — Interface de statistiques HAProxy

L'interface web de statistiques est accessible sur le port 8404 de HAPROXY-SAM :

```
http://10.0.0.30:8404/stats
```

Elle affiche en temps reel :

- Le statut UP/DOWN de chaque serveur backend
- Le nombre de requetes traitees par serveur
- Le taux de session, les bytes entrants/sortants
- Les eventuelles erreurs de connexion

6.3 Test 3 — Tolerance de panne (failover)

Ce test valide que HAProxy detecte automatiquement la defaillance d'un serveur et bascule tout le trafic sur le serveur restant.

Etape 1 — Arreter WEB1-SAM :

```
docker stop flask-python-1   # Sur WEB1-SAM
```

Etape 2 — Verifier la bascule depuis HAPROXY-SAM :

```
for i in {1..4}; do curl http://10.0.0.30; echo; done
```

Resultat :

```
Serveur 2
Serveur 2
Serveur 2
```

```
Serveur 2
```

HAProxy detecte que WEB1-SAM ne repond plus (health check echoue) et redirige 100% du trafic vers WEB2-SAM. Aucune interruption de service pour le client.

6.4 Test 4 — Reprise apres retablissement

Ce test valide que HAProxy reintegre automatiquement un serveur lorsqu'il redevient disponible.

Etape 1 — Redemarrer WEB1-SAM :

```
docker start flask-python-1   # Sur WEB1-SAM
```

Etape 2 — Verifier la reprise du round-robin :

```
for i in {1..6}; do curl http://10.0.0.30; echo; done
```

Resultat :

```
Serveur 2
Serveur 1
Serveur 2
Serveur 1
Serveur 2
Serveur 1
```

HAProxy reintegre WEB1-SAM dans la rotation des que le health check HTTP redevient positif. Le service reprend sa distribution equilibree automatiquement.

7. Synthese des resultats

Test effectue	Resultat	Observation
Round-robin (6 requetes)	Reussi	Alternance parfaite S1/S2
Interface HAProxy Stats	Reussi	WEB1 et WEB2 en statut UP
Failover (arret WEB1-SAM)	Reussi	Bascule automatique sur WEB2
Reprise (redemarrage WEB1)	Reussi	Round-robin reprend automatiquement

L'ensemble des fonctionnalites de repartition de charge et de tolerance de panne ont ete valides avec succes. HAProxy assure la haute disponibilite du service web sans aucune intervention manuelle lors de la defaillance ou du retablissement d'un serveur.

8. Conclusion

Ce projet a permis de mettre en oeuvre une infrastructure de haute disponibilite complete au sein du lab mvgroup.local, en integrant trois technologies complementaires :

- Docker : containerisation des applications web Flask, garantissant leur portabilite et leur isolation
- docker-compose : orchestration simplifiee des containers avec gestion automatique du reseau et des volumes
- HAProxy : load balancer professionnel assurant la repartition de charge et la detection automatique des pannes

Les tests pratiques ont confirme les deux objectifs principaux du projet. D'une part, la repartition de charge en mode round-robin distribue equitablement les requetes entre les serveurs, optimisant ainsi l'utilisation des ressources. D'autre part, la tolerance de panne garantit la continuite du service sans intervention humaine, ce qui est un critere fondamental dans un environnement de production.

Cette solution s'integre naturellement dans l'ecosysteme du lab mvgroup.local (Proxmox, Active Directory, Wazuh, GLPI, WireGuard) et constitue une demonstration concretes des concepts de haute disponibilite enseignes en cours d'infrastructure reseau.

Fin du document — HAProxy Load Balancing & Tolerance de panne