Haute-disponibilité avec HAProxy

1) Présentation

HAProxy est un service, sous linux, permettant la haute-disponibilité. Il intègre une fonction de Load-Balancing (répartition de charge) entre plusieurs serveurs (appelés nœuds ; l'ensemble des nœuds correspond à un cluster), par exemples des serveurs web. Lorsqu'un utilisateur contacte l'adresse IP du serveur HAProxy, il est redirigé de façon transparente vers un des serveurs du cluster, répartissant ainsi la charge.

Les différents algorithmes de répartition de charge sont :

- Roundrobin : les serveurs sont utilisés les uns après les autres (sauf si un poids a été spécifié).
- Leastconn : la requête est envoyée au serveur ayant le moins de connexions.
- First : le premier serveur disponible prend en charge la requête.

HAProxy intègre également une fonction FailOver (tolérance de pannes). Ainsi, lorsqu'un serveur tombe en panne, un autre prend le relais, limitant les interruptions de service et améliorant, par conséquent, la disponibilité.

Ce TP propose de mettre en place un site web sur deux serveurs web distincts et de répartir la charge, entre ces deux serveurs, avec un serveur HAProxy. Sachant que l'on peut faire également de la répartition de charge avec d'autres services (DNS, MySQL…).

2) Matériels nécessaires

💻 **Machine Debian pour le répartiteur de charge HAProxy**

- A partir de vSphere : procédure identique que les serveurs Web pour déployer une nouvelle VM
- Laisser en DHCP
- Installer le paquet HAProxy :
- Vérifier la configuration IP et la connectivité (PING) avec les autres machines
- Démarrer le service HAProxy : (ou stop pour arrêter ; restart pour redémarrer)

💻 **2 Machines Debian pour les serveurs web**

3) Configuration du service HAProxy

- Editer le fichier de configuration de HAProxy :
- Laisser les sections « global » (ajouter juste la « description ») et « defaults » en l'état

```
global
description HAProxy de GSB
    […]
defaults
     [..]
```

- Ajouter à la fin du fichier :

```
#spécifie où HAProxy écoute les connexions entrantes
frontend monSiteWeb
#HAProxy accepte toutes les requêtes venant de toutes ses interfaces réseaux, à destination du port 80
   bind  *:80
#permet de préciser que nous traiterons que des flux HTTP
   mode  http
#permet de spécifier le nom du cluster (ici « mesServeurs »)
   default_backend  mesServeurs

#spécifie où HAProxy renvoie les connexions entrées et acceptées
backend mesServeurs
   #permet de spécifier qu'on enverra des requêtes http (aux serveurs que nous préciserons un peu après)
mode  http
#choix du mode de répartition de charge
   balance  roundrobin
#permet de conserver l'adresse IP des clients (pour les serveurs web) et non l'adresse du répartiteur
   option  forwardfor
http-request  set-header  Forwarded  for=%[src]
#permet de vérifier si le serveur est toujours en fonction ou s'il faut le déclarer inactif (down)
   option  httpchk  HEAD  /
#déclaration des nœuds du cluster
   server  web1  192.168.1.51:80  check
   server  web2  192.168.1.52:80  check

#permet de pouvoir consulter les statistiques de l'équilibrage de la charge et monitorer le service HAProxy
listen stats
#écoute sur le port 1936 pour afficher le tableau de bord
bind  *:1936
#active le tableau de bord (monitoring)
stats  enable
#rafraîchit automatiquement les données toutes les minutes
stats  refresh  1m
#spécifie que l'URL est juste / après le port 1936
stats  uri  /
#masque la version de HAProxy
stats  hide-version
#définit un utilisateur et mot de passe pour afficher le tableau de bord
stats  auth  enc:sio
#affiche une description en haut de page (définit dans global)
stats  show-desc
#affiche les menus pour l'administrateur (mettre en maintenance, etc.)
stats  admin  if  TRUE
#affiche les légendes (adresse IP du serveur…)
stats  show-legends
```

- Redémarrer le service après chaque modification :
- On peut visualiser d'autres informations du service avec :

Pour débugger le fichier de configuration (permet de voir les erreurs et les numéros des lignes correspondantes) :

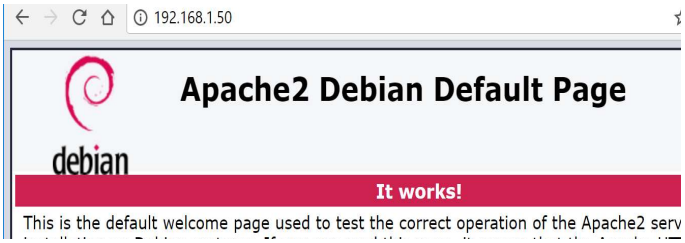
```
haproxy -f /etc/haproxy/haproxy.cfg -c
```

4) Tests

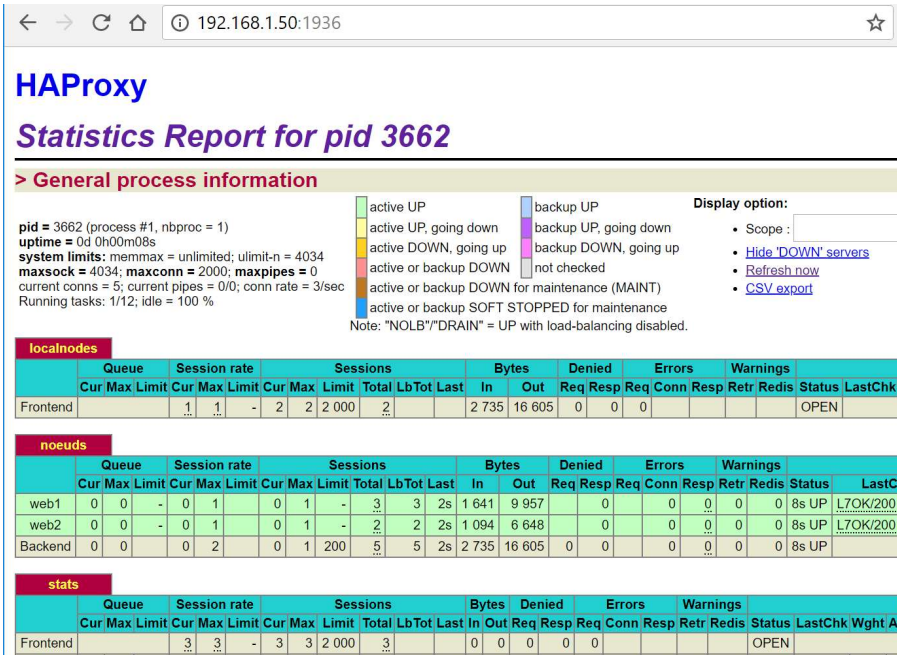
Depuis un poste client, on peut afficher la page du serveur web en contactant directement web1 ou web2 via l'adresse IP.

Mais si on veut bénéficier de la répartition de la charge, on contacte désormais directement HAProxy (et celui-ci affichera une fois la page de web1 (en rouge) et une autre fois (touche F5) la page de web2 qui est restée blanche).

Nous avons donc une répartition de charge égale. Attention, parfois le navigateur garde la page en cache et n'affiche donc pas les modifications. On peut prouver que c'est bien l'autre serveur web qui a envoyé la page en contrôlant le tableau de bord ou en lançant un nouveau navigateur en mode privé.



Visualisation des statistiques (on peut voir également ici la bonne répartition de la charge et l'état des serveurs) :



5) Configuration avancée du service HAProxy

1. On peut modifier le poids de chaque nœud pour ne pas faire une répartition de charge égale (par exemple si on a un nœud plus puissant que les autres).

```
server web1 192.168.1.51:80 weight 4 check
server web2 192.168.1.52:80 weight 1 check
```

Le serveur web1 répondra 4 fois plus que le serveur web2.

2. On peut spécifier qu'un client (internaute) sera toujours traité par le même serveur web.

```
cookie 192.168.1.50 insert indirect
server web1 192.168.1.51:80 cookie web1 check
server web2 192.168.1.52:80 cookie web2 check
```

3. On peut spécifier un serveur (un nœud) qui ne sera actif que lorsqu'un autre serveur sera défaillant.

```
server web1 192.168.1.51:80 check
server web2 192.168.1.52:80 check
server web3 192.168.1.53:80 backup check
```

4. On peut créer plusieurs frontend et plusieurs backend avec des serveurs différents, des écoutes sur des interfaces ou ports différents, des redirections selon des ACL, etc.

```
frontend direction
    bind 172.20.0.1:81
    mode http
    default_backend noeudsDirection
```

Ici, les internautes qui préciseront le port 81 à la fin de l'adresse d'HAProxy seront redirigés sur un backend particulier : noeudsDirection (backend qui pourra par exemple contenir des serveurs spécifiques !).

5. On peut personnaliser les pages d'erreurs en éditant (nano) les fichiers suivants (du fichier de config haproxy.cfg) :

```
errorfile 503 /etc/haproxy/errors/503.http
..etc…
```

6. On peut préciser le nombre de connexions simultanées maximum autorisées.

```
server web1 192.168.1.51:80 maxconn 100 check
server web2 192.168.1.52:80 maxconn 100 check
```

7. On peut écrire une description aux backend (affichée sur le tableau de bord) :

```
backend […]
  description Nœuds pour les serveurs réservés à la direction
[…]
```

6) Redirection sur des serveurs Web selon des règles

On peut définir des ACL (Access List) par exemple pour rediriger un client vers un backend spécifique selon une adresse IP ou réseau, des redirections selon l'URL ou selon le type de client (iPhone…), etc.

Une ACL se définit par un nom ainsi qu'une règle. Si la règle est vérifiée, l'ACL est activée.

Exemples multiples :

- #permet de détecter si l'internaute est sur un mobile (on peut tester sur Chrome avec F12)

- #permet de détecter si l'internaute à une adresse IP spécifique

- #permet de détecter si l'internaute appartient au réseau mentionné

- #permet de détecter si l'internaute accède au serveur web en précisant le chemin « direction » (exemple : 172.20.0.1/direction).

Syntaxe pour rediriger le traffic :

path : vérifie l'exactitude de la chaîne de caractère
path_beg : vérifie si l'url commence par la chaîne de caractère
path_dir : vérifie sur l'url contient le dossier spécifié
path_end : vérifie si l'url termine par la chaîne de caractère spécifiée
path_len : vérifie si l'url contient le nombre de caractères spécifié
path_reg : spécifie une expression régulière
path_sub : vérifie si l'url contient la chaîne de caractères spécifiée
-i : spécifie que le texte ne sera pas sensible à la casse
-m : spécifie que le texte vérifié doit être exactement celui-là

- #permet de détecter si le nombre de serveurs disponibles (statut à « UP ») est inférieur à 2

Il existe des dizaines de règles possibles…

Exemple concret :

```
frontend localnodes
 mode http

 acl cestUnSmartphone req.hdr(User-Agent) -i -m reg (android|iphone)
 acl cestUnDirigeant src 172.20.1.15

 #Précise que si une des deux ACL est activée, on redirige l'internaute sur un backend (donc des serveurs) spécifique
 use_backend noeudsOptimise if cestUnDirigeant or cestUnSmartphone

 #Spécifie le backend à utiliser dans tous les autres cas
 default_backend noeudsNormaux
```

7) Filtrage des internautes selon des règles

On peut filtrer les requêtes des internautes selon des règles de type listes blanches (ou inversement, listes noires). Pour cela, on utilise également des ACL. Ensuite, on spécifie l'action : allow (autorisation) si liste blanche détectée ; deny pour tout le reste. On peut aussi inverser, c'est-à-dire autoriser tout le monde sauf certains en liste noire.

Exemple :

```
frontend[…]
 #on définit les adresses IP ou les réseaux autorisés
 acl listeBlanche src 192.168.1.99
 #on spécifie que le trafic http est autorisé uniquement pour la liste blanche
 http-request allow if listeBlanche
 #tout le reste du trafic http est refusé
 http-request deny
```

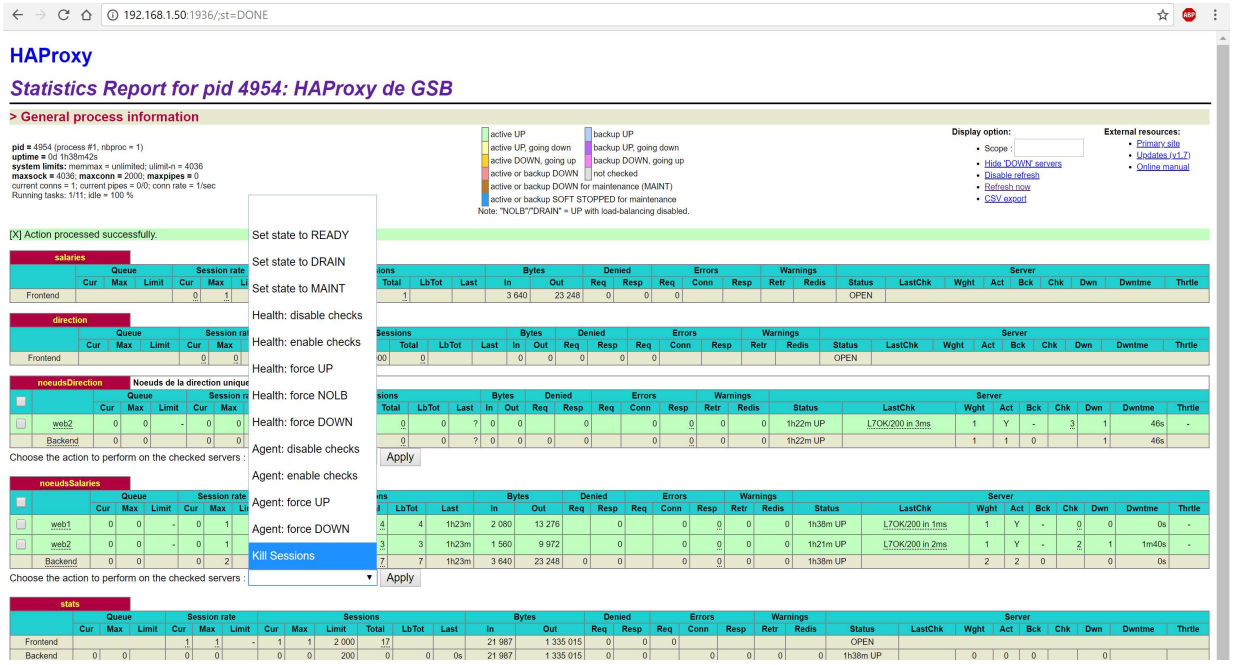
8) Alertes automatiques par email

On peut envoyer des alertes automatiques par email quand un serveur est DOWN.

Pour se faire, voir la documentation du professeur en annexe.

9) Affichage du tableau de bord avec toutes les options

L'interface permet de visualiser en temps réel l'état de nos serveurs (selon les « check » effectués), le nombre de fois où un serveur à répondu aux requêtes des clients, etc.
Elle permet également de forcer l'état des serveurs : pour les mettre en « down », en mode maintenance, etc.



Les états possibles sont :

- Set state to READY : active le serveur (UP).
- Set state to DRAIN : désactive le trafic du serveur (sauf les connexions persistantes) et continue à envoyer des vérifications d'état (check).
- Set state to MAINT : désactive le serveur (DOWN).
- Health disable checks : désactive les vérifications d'état du serveur.
- Health enable checks : active les vérifications d'état du serveur.
- Health force UP : force l'activation du serveur (UP).
- Health force DOWN : force la désactivation du serveur (DOWN).
- Health force NOLB : force le serveur à ne plus accepter de nouvelles connexions non-persistantes.
- Agent : non utilisé ici.
- Kill sessions : termine toutes les sessions en cours.