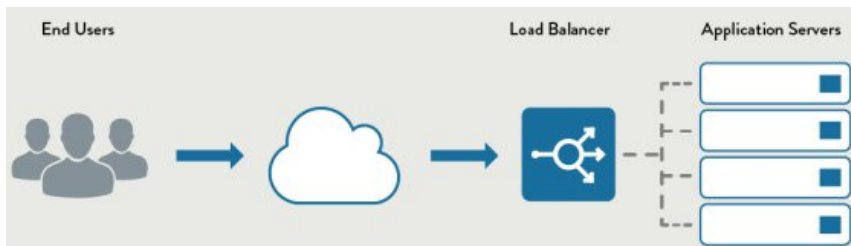


La répartition de charge

Tout serveur a une capacité de traitement limitée. Lors de périodes de pointe, cette capacité peut s'avérer insuffisante. Il est alors nécessaire d'ajouter un ou plusieurs serveurs afin de répartir le travail (la charge) entre eux.

La **répartition de charge (load balancing)** est « un ensemble de techniques permettant de distribuer une charge de travail entre différents ordinateurs d'un groupe. Ces techniques permettent à la fois de **répondre à une charge trop importante d'un service** en la répartissant sur plusieurs serveurs, et de **réduire l'indisponibilité potentielle** de ce service que pourrait provoquer la panne logicielle ou matérielle d'un unique serveur » (source http://fr.wikipedia.org/wiki/R%C3%A9partition_de_charge).

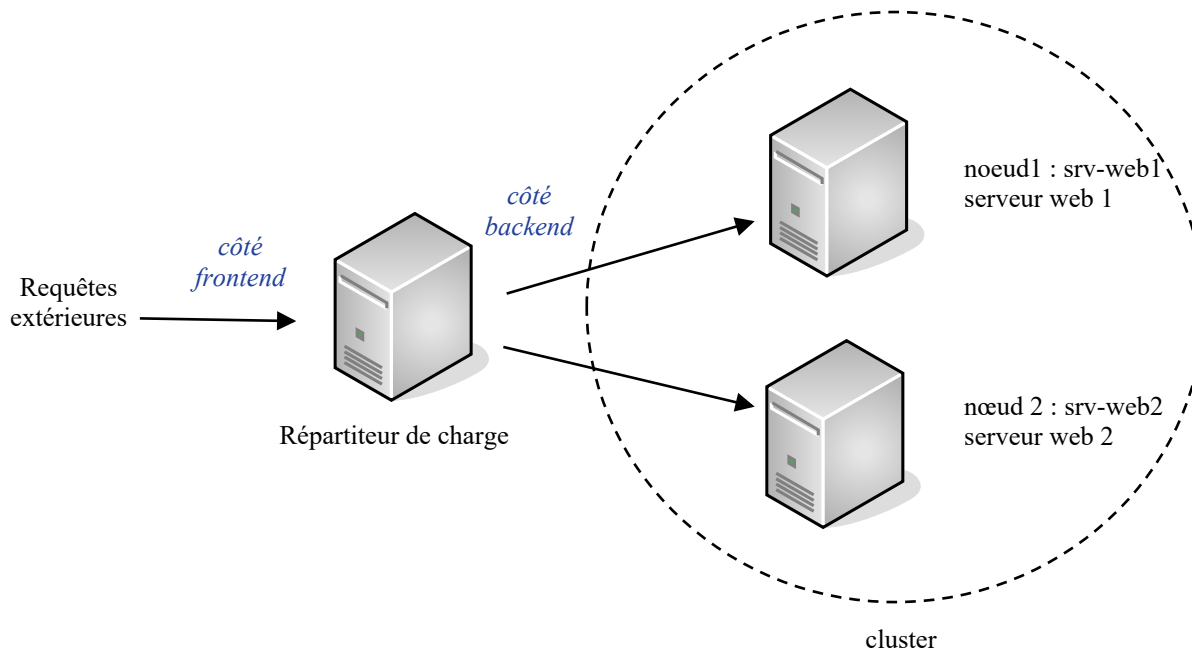


La répartition de charge est donc une des technologies de mise en Cluster réseau qui participe à la haute disponibilité.

Elle s'entend le plus souvent au niveau des serveurs HTTP (par exemple, sites à forte audience devant pouvoir gérer des centaines de milliers de requêtes par secondes), c'est-à-dire en frontal sur une plate-forme web comme nous allons le mettre en œuvre dans cette activité. Mais le même principe peut s'appliquer sur n'importe quel service aux utilisateurs ou service réseau.

Principes de la répartition de charge

Les requêtes (ici HTTP ou HTTPS) arrivent côté **frontend** du **répartiteur de charge**. En fonction d'un algorithme précédemment configuré, le répartiteur envoie ces requêtes côté **backend** sur les différents nœuds du cluster.



Les techniques de répartition de charge les plus utilisées sont :

- **Le DNS Round-Robin (DNS RR)** : lorsqu'un serveur DNS répond à un client, il fournit une liste d'adresses IP, dans un certain ordre, la première adresse étant celle que le client utilisera en priorité (les autres sont des adresses de secours) ; l'ordre sera évidemment différent pour un autre client (permutation circulaire en général). Le Round-Robin peut être mis en œuvre sur n'importe quel serveur DNS.
- **Le niveau TCP/IP ou niveau 4** : le client établit une connexion vers le « répartiteur » (matériel ou outil logiciel) qui redirige ensuite les paquets IP entre les serveurs selon l'algorithme choisi lors de la configuration (RR, aléatoire, en fonction de la capacité des serveurs, etc.).

- **Le niveau « applicatif » ou niveau 7 ou « répartition avec affinité de serveur »** : on analyse ici le contenu de chaque requête pour décider de la redirection. En pratique, deux choses sont recherchées et analysées :
 - les cookies, qui figurent dans l'entête HTTP ;
 - L'URI, c'est-à-dire l'URL et l'ensemble de ses paramètres.

Ce niveau est parfois rendu nécessaire par certaines applications qui exigent que les requêtes d'un même utilisateur soient adressées à un même serveur. Cette technologie de répartition induit bien évidemment des délais supplémentaires car chaque requête HTTP doit être analysée.

Le répartiteur de charge logiciel HAProxy

HAProxy est le logiciel libre de répartition de charge le plus utilisé.

C'est une solution très complète au plan fonctionnel, extrêmement robuste et performante.



Selon la documentation officielle « HAProxy est un relais TCP/HTTP (il fonctionne donc aux niveaux 4 et 7) offrant des facilités d'intégration en environnement hautement disponible. Il est capable :

- d'effectuer un aiguillage statique défini par des cookies ;
- d'effectuer une répartition de charge avec création de cookies pour assurer la persistance de session ;
- de fournir une visibilité externe de son état de santé ;
- de s'arrêter en douceur sans perte brutale de service ;
- de modifier/ajouter/supprimer des en-têtes dans la requête et la réponse ;
- d'interdire des requêtes qui vérifient certaines conditions ;
- d'utiliser des serveurs de secours lorsque les serveurs principaux sont hors d'usage ;
- de maintenir des clients sur le bon serveur d'application en fonction de cookies applicatifs ;
- de fournir des rapports d'état en HTML à des utilisateurs authentifiés.

En outre, il requiert peu de ressources et son architecture événementielle mono-processus lui permet de gérer facilement plusieurs milliers de connexions simultanées sur plusieurs relais sans effondrer le système. »



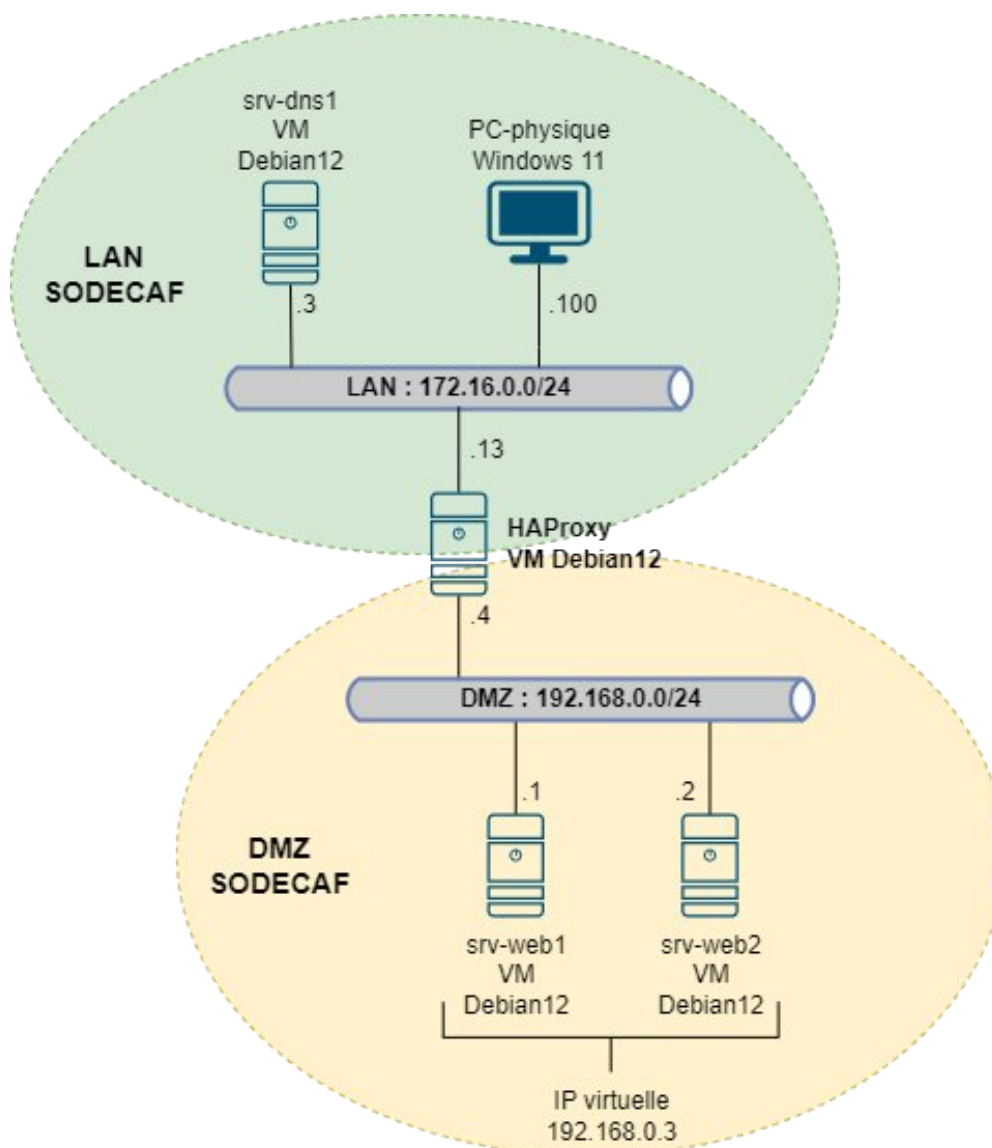
HAProxy peut aider aussi à se prémunir contre les attaques DOS (voir par exemple ici : http://publications.jbfavre.org/web/protegez_votre_serveur_web_avec_haproxy.fr) mais cela ne sera pas abordé dans ce Côté Labo.

Contexte et schéma de l'infrastructure

L'entreprise SODECAF met à disposition de ces clients un site Web sécurisé. La haute disponibilité de cette application est assurée via un cluster de deux serveurs (srv-web1 et srv-web2).

La demande de connexion d'un client est adressée au serveur HAProxy d'adresse IP 172.16.0.13 qui détermine, selon l'algorithme configuré, le serveur auquel il va affecter la connexion, parmi les serveurs disponibles (srv-web1 ou srv-web2).

Une fois la connexion TCP établie, l'équipement de répartition de charge devient pratiquement transparent : dans son rôle de base, il transfère les paquets IP du client vers le serveur sélectionné et vice versa jusqu'à fermeture de la connexion.



L'équipe informatique de la SODECAF souhaite que chacun des deux serveurs web soit utilisé de manière optimale, qu'il n'y en ait pas un surchargé tandis que l'autre est sous-utilisé. Elle décide donc d'utiliser un répartiteur de charge logiciel, HAProxy.

Travail à effectuer

- Démarrez les machines virtuelles précédemment installées : srv-web1, srv-web2, srv-dns1, pfSense.
- Modifiez les adresses IP des serveurs web, ainsi que l'adresse IP du failover. Pensez aussi à modifier la configuration de corosync.
- Le cluster précédemment installé fonctionne en mode actif/passif. Plusieurs modifications sont à réaliser. Par exemple, la ressource serviceWeb doit être configurée selon le mode actif/actif de manière à ce que le service Apache2 soit démarré sur les deux serveurs. Utilisez l'annexe 1 pour réaliser toutes ces modifications.
- L'application web ne sera plus accessible via l'adresse IP virtuelle mais par l'adresse IP d'HAProxy ce qui implique des modifications au niveau du serveur DNS. Procédez à ces modifications.
- Le protocole HTTPS sera utilisé pour notre site web, seulement, **l'encapsulation SSL se terminera au niveau du load-balancer** (HAProxy). Utilisez l'annexe 2 pour réaliser les modifications à apporter sur la configuration d'Apache2 sur les serveurs web.
- Pour pouvoir tester la solution mise en place, la page d'accueil de l'application sera légèrement modifiée et différente sur chacun des serveurs Web. Sur le site web de la Sodecaf, vous afficherez « Accueil1 » pour serv1 et « Accueil2 » pour serv2.
- Importez une nouvelle machine virtuelle Debian 12. Installez HAProxy en vous aidant de l'annexe 3.
- Testez la répartition de charge en utilisant l'annexe 4. Vérifiez également les statistiques à l'adresse <http://172.16.0.13/statsHaproxy>.
- En utilisant le début de l'annexe 5, procédez à une nouvelle configuration d'HAProxy en faisant l'hypothèse que le serveur initialement de secours srv-web2 est trois fois moins puissant que le serveur maître. Proposez et réalisez des tests permettant de vérifier que la solution est opérationnelle.
- Enfin, en utilisant la fin de l'annexe 5, configurez HTTPS sur la machine HAProxy, uniquement côté frontend, afin de rétablir des connexions sécurisées entre les clients et les serveurs web de l'entreprise.

Annexe 1 : Préparation du cluster

Les aménagements à apporter sur le cluster déjà en place sont les suivants :

- la ressource *serviceWeb* doit être configurée selon le mode actif/actif (comme l'est le service MySQL) de manière à ce que le serveur Apache2 soit démarré sur les deux serveurs ;
- le fichier de configuration de MySQL doit être modifié de manière à ce que le service MySQL puisse supporter une écriture simultanée sur les deux serveurs ;

Les modifications à apporter à la plate-forme existante sont les suivants :

La ressource serviceWeb doit être configurée selon le mode actif/actif (comme l'est le service MySQL) de manière à ce que le serveur Apache2 soit démarré sur les deux serveurs :

Pour pouvoir cloner la ressource *serviceWeb*, il faut la sortir du groupe *servweb* qui n'a plus aucune utilité car composé d'une seule ressource *IPFailover* :

```
root@serv1:~# crm resource stop serviceWeb
root@serv1:~# crm resource stop IPFailover
root@serv1:~# crm configure delete servweb
root@serv1:~# crm resource start IPFailover
root@serv1:~# crm configure clone cServiceWeb serviceWeb
root@serv1:~# crm resource start serviceWeb
```

```
crm_mon :
=====
Last updated: Tue Aug 27 23:23:12 2013
...
=====

Online: [ serv1 serv2 ]

Clone Set: cServiceMySQL [serviceMySQL]
  Started: [ serv2 serv1 ]
IPFailover (ocf::heartbeat:IPAddr2): Started serv1
Clone Set: cServiceWeb [serviceWeb]
  Started: [ serv2 serv1 ]
```

Le fichier de configuration de MySQL doit être modifié de manière à ce que le service MySQL puisse supporter une écriture simultanée sur les deux serveurs (voir annexe 3 du Côté Labo « Haute disponibilité d'un service Web dynamique »):

Par rapport à la configuration mise en place dans le premier Côté Labo, deux lignes doivent être ajoutées dans chaque fichier de configuration pour éviter les conflits d'insertion menant à l'échec de la réplication car lors de l'écriture simultanée sur les deux serveurs, il pourrait y avoir des collisions au niveau des clés auto incrémentées.

L'astuce est de configurer chaque serveur pour qu'il génère des clef primaires qui n'entrent pas en conflit. Et cela est possible grâce aux variables `auto_increment_increment` et `auto_increment_offset`.

`auto_increment_increment` contrôle l'incrémement entre les valeurs successives des clés (le pas d'incrémement).
`auto_increment_offset` détermine le point de départ des valeurs des colonnes de type `AUTO_INCREMENT`.

Sur le serveur serv1 (fichier `/etc/mysql/mariadb.conf.d/50-server.cnf`) :

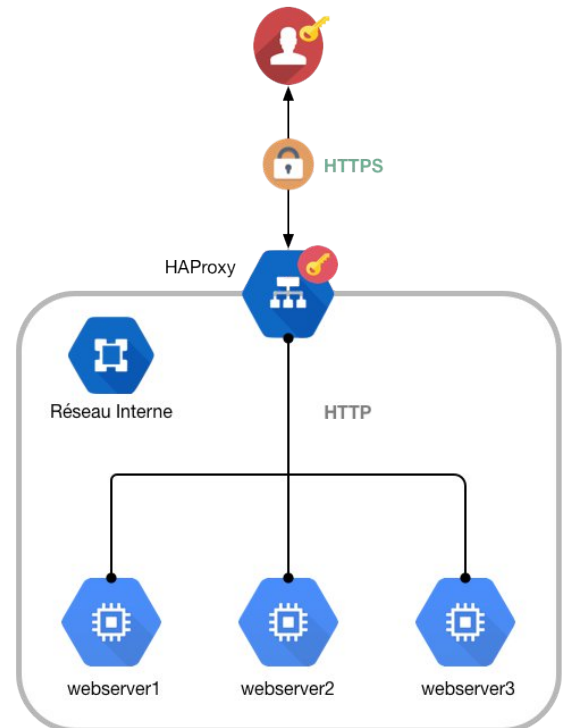
```
auto_increment_offset=1
auto_increment_increment=2
```

Sur le serveur serv2 (fichier `/etc/mysql/mariadb.conf.d/50-server.cnf`) :

```
auto_increment_offset=2
auto_increment_increment=2
```

Annexe 2 : HAProxy et HTTPS

Il est possible d'utiliser le protocole SSL au niveau d'un load-balancer avec HAProxy, afin d'envoyer du trafic normal vers des serveurs web. L'encapsulation SSL se termine au niveau du load-balancer, et non plus sur les serveurs web. Pour chaque requête HTTPS reçue, le load-balancer ouvre une session HTTP avec les serveurs web. Cela permet de déplacer l'utilisation des ressources CPU vers le load-balancer, car HAProxy peut supporter un très grand nombre de requêtes, en restant très économe. Bien sûr, envoyer des données non-chiffrées requiert une sécurisation accrue du réseau privé par lequel elles transitent. L'autre avantage de ce type de configuration est que l'on n'a plus de certificats éparpillés sur chaque serveur/instance, la gestion des certificats est facilitée.



Vous devez donc modifier le fichier de configuration Apache2 de vos serveurs web (par exemple fichier `/etc/apache2/sites-available/sodecaf.conf`), afin de désactiver la partie HTTPS et d'utiliser uniquement le protocole HTTP

```
<VirtualHost *:80>
    ServerAdmin webmaster@sodecaf.fr
    DocumentRoot /var/www/sodecaf
    DirectoryIndex sodecaf.html

    <Directory "/var/www/sodecaf">
        Options -Indexes -FollowSymlinks -MultiViews -ExecCGI
        AllowOverride none
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Annexe 3 : Installation et première configuration d'HAProxy

Installation et démarrage d'HAProxy

```
apt-get update & apt-get install haproxy
```

Il est maintenant nécessaire de procéder à une configuration minimale.

Configuration d'HAProxy

Le fichier de configuration `/etc/haproxy/haproxy.cfg` se décompose en plusieurs sections repérées par des mots clés dont :

- **global** : paramètres agissant sur le processus ou sur l'ensemble des proxies ;
- **defaults** : paramétrages par défaut qui s'appliquent à tous les *frontends* et *backends*. Ces paramètres peuvent être redéfinis dans chacune des autres sections.

Ces deux sections sont déjà alimentées avec des directives et des valeurs acceptables (pour une plate-forme de test) :

Directives et valeurs	Quelques explications
global log /dev/log local0 log /dev/log local1 notice chroot /var/lib/haproxy user haproxy group haproxy daemon	Le paramètre chroot change la racine du processus une fois le programme lancé, de sorte que ni le processus, ni l'un de ses descendants ne puissent remonter de nouveau à la racine.

Les **logs d'HAProxy** sont écrits par défaut dans `/var/log/messages`. Pour une solution en production, il sera nécessaire de gérer les logs plus finement (idéalement intégrés à une solution de journalisation centralisée).

Directives et valeurs	Quelques explications
defaults log global mode http * option httplog option dontlognull timeout connect 5000 timeout client 50000 timeout server 50000 errorfile 400 /etc/haproxy/errors/400.http errorfile 403 /etc/haproxy/errors/403.http errorfile 408 /etc/haproxy/errors/408.http errorfile 500 /etc/haproxy/errors/500.http errorfile 502 /etc/haproxy/errors/502.http errorfile 503 /etc/haproxy/errors/503.http errorfile 504 /etc/haproxy/errors/504.http	mode http : le service relaie les connexions TCP vers un ou plusieurs serveurs, une fois qu'il dispose d'assez d'informations pour en prendre la décision. Les entêtes HTTP sont analysés pour y trouver un éventuel cookie, et certains d'entre-eux peuvent être modifiés par le biais d'expressions régulières. Temps d'expiration des connexions (valeur en millisecondes par défaut mais peut s'exprimer dans une autre unité de temps) Timeout connect 5000 : temps d'attente de l'établissement d'une connexion vers un serveur (on abandonne si la connexion n'est pas établie après 5 secondes) Timeout client 50000 : temps d'attente d'une donnée de la part du client Timeout server 50000 : temps d'attente d'une donnée de la part du serveur Les « errorfile » définissent les messages d'erreurs envoyés aux internautes.

* D'autres modes existent comme « tcp » (connexions TCP génériques)

log global indique que l'on souhaite utiliser les paramètres de journalisation définis dans la section 'global'.

option httplog active la journalisation des requêtes HTTP.

option dontlognull : comme nous le verrons plus loin, HAProxy va se connecter régulièrement à chacun des serveurs afin de s'assurer qu'ils soient toujours vivants. Par défaut, chaque connexion va produire une ligne dans le journal qui sera ainsi pollué. Cette option permet de ne pas enregistrer de telles connexions pour lesquelles aucune donnée n'a été transférée.

Ces options peuvent être désactivées dans chacune des autres sections avec un « no » devant.

Maintenant voyons ce que vous devez ajouter à la fin du fichier `/etc/haproxy/haproxy.cfg`

adresse IP

La directive **listen** permet de créer un service de répartition de charge :

Directives et valeurs	Quelques explications
listen httpProxy bind 172.16.0.13:80 balance roundrobin option httpclose option httpchk HEAD / HTTP/1.0 server serv1 192.168.0.1:80 check server serv2 192.168.0.2:80 check	listen permet de demander à HAProxy d'écouter sur l'IP et le port indiqué. Si on ne précise que « :80 », le HAProxy écoute sur toutes les IPs de la machine, mais ici les requêtes ne pourront venir que de l'interface « externe ». Voir ci-dessous pour les explications des autres directives.



C'est HAProxy qui écoute sur le port 80 : tout service Web écoutant sur ce port ne sera pas possible et devra être arrêté.

balance permet de choisir l'algorithme de la répartition vers les frontaux.

Le **roundrobin** est le plus classique et le plus simple : il consiste à utiliser les serveurs un à un, chacun son tour. Il est possible d'affecter des poids particuliers aux ressources, par exemple pour utiliser deux fois plus souvent le frontal qui dispose d'une très grosse CPU et/ou de beaucoup de RAM.

Les autres modes possibles sont :

- **leastconn** : le serveur sélectionné sera celui ayant précédemment reçu le moins de connexions
- **source** : le serveur est sélectionné en fonction de l'IP source du client
- **uri** : le choix du serveur est fonction du début de l'URI demandée
- **url_param** : le choix du serveur est fonction de paramètres présents dans l'URL demandée
- **hdr** : le choix du serveur est fonction d'un champ présent dans l'en-tête HTTP (Host, User-Agent, ...).

option httpclose force à fermer la connexion HTTP une fois la requête envoyée au client. On évite ainsi de conserver la connexion HTTP ouverte (« keep-alive ») et donc de renvoyer systématiquement cette dernière vers le même frontal tant que la connexion reste ouverte.

Conserver la connexion ouverte pourrait être le comportement recherché, mais, dans le cadre de notre activité, cela permettra de montrer facilement que l'algorithme « roundrobin » fonctionne bien et que l'on tombe sur un frontal différent à chaque rafraîchissement de la page.

option httpchk, suivie d'une requête HTTP, permet de vérifier qu'un frontal web est toujours en vie. HAProxy peut tester la disponibilité des serveurs en adressant une requête HTTP (ici, aucun fichier n'est précisé derrière le « / » présent après le « HEAD », HAProxy va envoyer la requête suivante à chaque serveur : `http://@IP_serveur/`. Si un frontal venait à ne plus répondre à cette requête, il serait considéré comme hors service, et serait sorti du pool des frontaux ; aucun utilisateur ne serait redirigé vers lui.

Cette vérification est en fait activée grâce à l'option **check** de **server** (voir ci-après).

server déclare un serveur frontal, utilisé pour assurer le service. Chaque « server » est nommé (nom libre) et suivi de son IP/port de connexion (port qui pourrait être différent du port d'écoute de HAProxy).

L'option **check** est expliquée ci-dessus (dans **option httpchk**).



Il est courant et parfois obligatoire de scinder la directive *listen* en deux sections *frontend* et *backend* (voir Annexe 5) :

- **frontend** : définit les paramétrages de la partie publique. Les connexions « arriveront » donc ici.
- **backend** : définit la partie privée d'HAProxy. Apparaîtront ici le ou les serveurs Web sur lesquels portent la répartition.

Pensez à redémarrer le service haproxy

```
# systemctl restart haproxy
```


Annexe 4 : Test d'HAProxy

Pour vérifier que la syntaxe du fichier est correcte :

```
haproxy -c -f /etc/haproxy/haproxy.cfg
-c      pour vérifier (check) le fichier
-f      pour spécifier le fichier de configuration
```

```
root@haproxyAR:~# haproxy -c -f /etc/haproxy/haproxy.cfg
Configuration file is valid
```

Pour vérifier que le service a bien le comportement attendu, il suffit de se connecter à deux reprises à partir d'un navigateur.




Attention au cache du navigateur qui renvoie la page en cache au lieu de se connecter au serveur (ce qui ne permet donc pas de tester la répartition de charge) !

Plusieurs solutions sont possibles dont :

- l'activation des touches CTRL+F5, sur « Chrome », pour recharger la page ;
- le changement de navigateur, voire de STA ;
- la possibilité d'utiliser le client « lynx » en ligne de commande :

Avec l'algorithme de répartition de charge round robin, la page web affichée doit alternée entre celle du serveur web 1 et celle du serveur web 2 :

page du serv1

 **SODECAF** Accueil1 SODECAF Constructions Importation

page du serv2

 **SODECAF** Accueil2 SODECAF Constructions Importation

Mise en place des statistiques

Directives et valeurs

listen httpProxy

```
Bind 172.17.0.13:80
balance roundrobin
...
stats uri /statsHaproxy
stats auth root:Btssio2017
stats refresh 30s
```

Quelques explications

stats uri permet d'activer la page de statistiques, en définissant l'endroit où les statistiques pourront être consultées (ici `http://@IP_du_haproxy/statsHaproxy`)

stats auth sécurise l'accès en le protégeant par un nom d'utilisateur et un mot de passe séparés par « : »

stats refresh rafraîchit la page toutes les 30s

Voir en *annexe 6* des exemples de statistiques que l'on peut obtenir.



Pour que les modifications soient prises en compte, il est bien sûr nécessaire de recharger le fichier de configuration : **`systemctl reload haproxy`**

Utilisation des *frontends* et *backends*

Les sections *frontend* et *backend* remplacent la section *listen* et permettent d'affiner la configuration :

- **frontend** définit les paramétrages de la partie publique. Les connexions « arriveront » donc ici.
- **backend** : définit la partie privée de HAProxy. Apparaîtront donc ici le ou les serveurs Web qu'HAProxy sur lesquels portent la répartition.

Vous trouverez, dans la documentation officielle, les directives admissibles dans chacune des sections :

<http://cbonte.github.io/haproxy-dconv/configuration-1.4.html#4.1>

Exemple :

```
frontend proxypublic
  bind 172.16.0.13:80
  default_backend fermeweb
```

```
backend fermeweb
  balance roundrobin
  option httpclose
  option httpchk HEAD / HTTP/1.0
  server web1 192.168.0.1:80 check
  server web2 192.168.0.2:80 check
  stats uri /statsHaproxy
  stats auth root:Btssio2017
  stats refresh 30s
```

Les directives *stats* peuvent être définies dans la section *defaults*, ainsi elles

bind définit le port d'écoute. Il y aura autant de directives *bind* que de ports d'écoute.

Ici, un seul backend est défini (*backend fermeweb*). Aussi, l'intérêt de scinder la directive *listen* en deux sections est limité mis à part une meilleure lisibilité notamment dans la lecture des statistiques.

default_backend définit le backend par défaut. Cette directive est obligatoire même s'il n'y a qu'un seul backend sinon HAProxy considère qu'aucun service n'est défini (erreur « 503 Service Unavailable »).

Répartition inégale sur serveurs hétérogènes

Il est possible que le Cluster soit composé de serveurs disparates, et que l'on ne souhaite pas leur faire porter la même charge. Imaginons par exemple un processeur 2 fois moins puissant, une mémoire moins importante, etc. Nous supposons ici que le serveur *serv2* est deux fois moins puissant que le serveur *serv1*.

Il est possible de paramétrer HAProxy pour répartir la charge sur les serveurs, en fonction de leur puissance (algorithme *Weighed-least-connection*) en affectant aux serveurs une pondération différente, reflet de leur puissance : par exemple 100 pour *serv1* et 50 pour *serv2* (la valeur doit être entre 0 et 255).

Ce paramétrage est possible grâce au mot clé *weight* dans la directive *server* :

```
server web1 192.168.0.1:80 weight 100 check
server web2 192.168.0.2:80 weight 50 check
```



HAProxy et HTTPS

Pour utiliser HTTPS côté frontend d'HAProxy, il faut commencer par créer une **clé privée** et un **certificat** auto-signé sur le serveur HAProxy.

```
apt install openssl-server
mkdir /etc/haproxy/cert
cd /etc/haproxy/cert
openssl genrsa -out privateKey.pem 4096
openssl req -new -x509 -days 365 -key privateKey.pem -out cert.pem
```

On fusionne ensuite le certificat et la clé privée dans un même fichier :

```
cat cert.pem privateKey.pem > sodecaf.pem
```

Il faut maintenant créer, dans le fichier /etc/haproxy/haproxy.cfg, un frontend pour une écoute sur le port 443 (https) :

```
frontend https-in
    bind 172.16.0.13:443 ssl crt /etc/haproxy/cert/sodecaf.pem
    default_backend fermeweb
```

Redémarrez le service haproxy et faites le test sur un navigateur : <https://172.16.0.13>

```
systemctl restart haproxy
```

Enfin, il est possible de rediriger les requêtes du port 80 (http) vers le port 443 (https) :

```
frontend https-in
    mode http
    bind 172.16.0.13:443 ssl crt /etc/haproxy/cert/sodecaf.pem
    http-request redirect scheme https unless { ssl_fc }
    default_backend fermeweb
```

N'oubliez pas de redémarrer le service haproxy

```
systemctl restart haproxy
```

La problématique des sessions (pour information)

Le protocole HTTP utilise de nombreuses connexions TCP pour la session d'un même internaute. Avec la configuration élaborée, les requêtes d'un même internaute sont donc réparties entre les deux serveurs ce qui peut poser des problèmes lors de l'accès à certaines applications qui ont besoin de retrouver en mémoire des informations de *contexte*, relatives aux échanges précédents de l'internaute de la même session.

Pour s'en persuader, il suffit de tenter de s'authentifier. Après la première tentative qui devrait réussir, on obtient de nouveau le formulaire d'authentification car le serveur change et ne retrouve plus le *contexte*. Quand l'authentification finit par réussir, selon le « surf » réalisé, les risques de déconnexion sont très grands.

HAProxy peut effectuer une répartition de charge de manière à ce qu'un même utilisateur, dans le cadre d'une session, soit toujours redirigé vers le même frontal via un **cookie** : cookie existant dans l'application ou cookie créé par HAProxy.



Un **cookie** est une donnée (sous la forme **identifiant+valeur**) conservée sur le navigateur, à la demande du serveur et qui est retourné au serveur avec chacune des requêtes HTTP.

Cookie ajouté par HAProxy

La configuration d'HAProxy permettant l'ajout d'un cookie dans les requêtes HTTP est la suivante :

```
backend fermeweb
    balance roundrobin
    option httpclose
    option httpchk HEAD / HTTP/1.0
    cookie QUELSERVEUR insert indirect
    server web1 192.168.0.1:80 cookie W1 check
    server web2 192.168.0.2:80 cookie W2 check
```

QUELSERVEUR
sera l'identifiant du
cookie
W1 et W2 seront les

Le principe est très simple :

- À la première connexion de l'utilisateur sur le site, HAProxy va déterminer, selon l'algorithme sélectionné dans la configuration (ici roundrobin), vers quel serveur Web le rediriger.
- Quand HAProxy récupère auprès du serveur Web la copie de la page demandée par l'utilisateur, il la renvoie en y insérant un cookie QUELSERVEUR valant W1 ou W2 selon le serveur Web utilisé.
- Ainsi à la prochaine requête de notre utilisateur, son navigateur renverra également ce cookie avec la requête.
- HAProxy saura déterminer, en fonction de la valeur, vers quel frontal renvoyer l'utilisateur.
- Il fait aussi en sorte que des caches ne mémorisent pas ce cookie et de le cacher à l'application.
- À la fin de la session, le cookie est détruit.

Que se passerait-il si un des serveurs avait un problème et venait à planter ?

HAProxy sait déterminer que l'identifiant QUELSERVEUR pointe vers un serveur qui n'est plus actif dans le Cluster. Il détruit le cookie en lui réassignant une nouvelle valeur pour rediriger l'utilisateur vers un nouveau serveur web. Certes la session qui était en cours sera alors perdue (il faudra se ré-authentifier) mais le service sera toujours rendu et le client aura toujours accès à l'application Web.

Cookie existant dans l'application

L'application peut avoir prévu elle-même un cookie **comme c'est le cas pour celle que nous utilisons qui se sert d'un cookie d'identifiant PHPSESSID** (l'initialisation d'une session PHP génère par défaut un cookie dont le nom est PHPSESSID et aucun nom de cookie n'a été spécifié au niveau de l'application). **Dans ce cas, le répartiteur peut être configuré pour « apprendre » ce cookie.** Le principe est simple : quand il reçoit la requête de l'utilisateur, il vérifie si elle contient ce cookie avec une valeur connue. Si tel n'est pas le cas, il dirigera la requête vers n'importe lequel des serveurs en fonction de l'algorithme de répartition appliqué. Il récupérera alors la valeur du cookie à partir de la réponse du serveur et l'ajoutera dans sa table des sessions avec l'identifiant du serveur correspondant. Quand l'utilisateur revient, le répartiteur voit le cookie, vérifie sa table de sessions et trouve le serveur associé vers lequel il redirige la requête.

Selon le développeur de l'application, cette dernière méthode est la moins intrusive.

La directive simplifiée d'HAProxy permettant l'utilisation d'un cookie dans les requêtes HTTP est la suivante :

appsession <cookie> len <length> timeout <holdtime> [request-learn] [prefix] [mode <path-parameters|query-string>]

<cookie> : identifiant du cookie utilisé par l'application qu'HAProxy devra apprendre à chaque nouvelle session.

<length> : nombre maximum de caractères qui sera mémorisé et vérifié à chaque valeur de cookie.

<holdtime> : délai après lequel le cookie sera supprimé s'il n'est pas utilisé. Si aucune unité n'est spécifiée, ce temps est en millisecondes.

[request-learn] : si cette option est spécifiée, haproxy sera capable d'apprendre le cookie trouvé dans la demande au cas où le serveur ne l'envoie pas tout de suite. Il est recommandé de spécifier cette option pour améliorer la fiabilité.

[prefix] : cette option permet à HAProxy d'utiliser le cookie en provenance du serveur et de lui préfixer l'identifiant du serveur (le "prefix" est supprimé à chaque requête vers l'application).

*Exemple : **appsession COOKIEID* len 64 timeout 3h request-learn prefix***

HAProxy utilisera le **cookie COOKIEID de l'application pour définir la répartition de charge**.
Le cookie reste valide pendant trois heures.

*** À adapter**

Répartition de charge de niveau 7 (pour information)

Nous avons maintenant 2 serveurs se répartissant la charge, soit de manière égalitaire, soit en fonction d'un poids attribué à chacun d'eux.

Imaginons maintenant que l'on souhaite dédier un serveur particulier à la gestion d'une mailing liste, ou bien à la partie administrative de notre site, ou bien encore à une application web, le reste du site étant réparti sur les autres serveurs.

Nous pouvons mettre en place un **filtre applicatif via les « acl »** qui redirigera une URL particulière sur ce serveur.

Le principe est le suivant : les ACLs sont déclarées avec, au minimum, un nom, un test et une valeur valide à tester.

Exemple d'utilisation d'ACL :

frontend proxypublic

bind 172.16.0.13:80

acl acces_interface_admin path_beg /gestadm

use_backend admin if acces_interface_admin

default_backend fermeweb

HAProxy vérifiera si le chemin donné par l'URL commence par /gestadm, auquel cas il utilisera le backend « admin ».

backend admin

option httpchk HEAD / HTTP/1.0

server web3 192.168.0.5

:80 check

Il est possible d'ajouter d'autres

D'innombrables autres règles peuvent être ajoutées. Les URL qui n'appartiennent à aucune règle sont envoyées sur le backend par défaut.

Annexe 6 : exemples de statistiques

Statistics Report for pid 1918

General process information

pid = 1918 (process #1, nbproc = 1)
 uptime = 0d 0h02m41s
 system limits: memmax = unlimited; ulimit-n = 4014
 maxsock = 4014; maxconn = 2000; maxpipes = 0
 current conns = 1; current pipes = 0/0
 Running tasks: 1/4

active UP backup UP
 active UP, going down backup UP, going down
 active DOWN, going up backup DOWN, going up
 active or backup DOWN not checked
 active or backup DOWN for maintenance (MAINT)
 Note: UP with load-balancing disabled is reported as "NOLB".

Display option:

• [Hide 'DOWN' servers](#)
 • [Refresh now](#)
 • [CSV export](#)

External resources:

• [Primary site](#)
 • [Updates \(v1\)](#)
 • [Online manual](#)

publicproxy																												
	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings			Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme
Frontend				1	6	-	1	1	2 000	49		9 790	31 965	0	0	0					OPEN							
fermeweb																												
	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings			Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme
web1	0	0	-	0	3		0	1	-	22	22	4 591	10 044		0		0	0	0	0	2m41s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s
web2	0	0	-	0	1		0	1	-	4	4	668	1 888		0		0	0	0	0	38s DOWN	L4CON in 0ms	1	Y	-	2	1	38s
web3	0	0	-	0	3		0	1	-	21	21	4 239	9 668		0		0	0	0	0	2m41s UP	L7OK/200 in 1ms	1	Y	-	0	0	0s
Backend	0	0		1	6		1	1	0	49	47	9 790	31 965	0	0		0	0	0	0	2m41s UP		2	2	0		0	0s

On distingue deux blocs :

- Le bloc frontend (publicproxy)
- Le bloc backend (fermeweb) disposant de trois serveurs web, deux en « vert » et un en « rouge ».

On constate ici que :

- les serveurs totalisent 49 sessions, dont 22 pour web1, 4 pour web2 et 21 pour l'hôte web3.
- le nombre de sessions faible pour web2 s'explique par le fait qu'il est tombé. Après 2 « check » infructueux, il a été déclaré down, et cela depuis 38s ;
- les frontaux web1 et web2 sont UP depuis 2 min 41s ;
- les frontaux web1, web2, et web3 ont eu respectivement 3, 1 et 3 sessions simultanées maximum ;
- on a également une indication sur les quantités de données qui ont transité vers et depuis chaque frontal.

Même une fois web2 redémarré, la trace de son inaccessibilité reste, ce qui est fort instructif pour un administrateur réseau.

Dans le tableau suivant, on voit que le frontal web2 est resté inaccessible pendant 6 minutes, et qu'il a redémarré depuis 5 minutes 36 :

Statistics Report for pid 1918

General process information

pid = 1918 (process #1, nbproc = 1)
 uptime = 0d 0h13m40s
 system limits: memmax = unlimited; ulimit-n = 4014
 maxsock = 4014; maxconn = 2000; maxpipes = 0
 current conns = 1; current pipes = 0/0
 Running tasks: 1/4

active UP backup UP
 active UP, going down backup UP, going down
 active DOWN, going up backup DOWN, going up
 active or backup DOWN not checked
 active or backup DOWN for maintenance (MAINT)
 Note: UP with load-balancing disabled is reported as "NOLB".

Display option:

• [Hide 'DOWN' servers](#)
 • [Refresh now](#)
 • [CSV export](#)

External resources:

• [Primary site](#)
 • [Updates \(v1.4\)](#)
 • [Online manual](#)

publicproxy																													
	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	T
Frontend	1	0	6	-	1	1	2	000	58			11	543	56	202	0	0	0				OPEN							
fermeweb																													
	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	T
web1	0	0	-	0	3		0	1	-	24	24	4	925	10	968		0	0	0	0	0	13m40s UP	L7OK/200 in 1ms	1	Y	-	0	0	0s
web2	0	0	-	0	1		0	1	-	6	6	1	002	2	832		0	0	0	0	0	5m36s UP	L7OK/200 in 2ms	1	Y	-	2	1	6m1s
web3	0	0	-	0	3		0	1	-	24	24	4	740	11	284		0	0	0	0	0	13m40s UP	L7OK/200 in 3ms	1	Y	-	0	0	0s
Backend	0	0	1	6		1	1	1	0	58	54	11	543	56	202		0	0	0	0	0	13m40s UP		3	3	0	0	0	0s