Répartition de charge sur une plate-forme Web

Propriétés	Description
Type de publication	Côté Labo
Intitulé court	**Répartition de charge sur une plate-forme Web**
Intitulé long	**Répartition de charge sur une plate-forme Web avec** réplication des données
Module	BTS SIO2 – **SISR3 – Exploitation des services**
Transversalité	**SI7** : • Stratégies et techniques associées à la continuité de service • Stratégies et techniques de répartition et de réplication **SISR4** : • Contrôler et améliorer les performances d'un système
Présentation	L'objectif de ce Côté Labo (mis en œuvre en module) est d'optimiser l'utilisation des deux serveurs Web en configurant une répartition de charge.
Activités	**D1.3 - Mise en production d'un service** • A1.3.2 Définition des éléments nécessaires à la continuité d'un service **D2.1 - Exploitation des services** • A2.1.2 Évaluation et maintien de la qualité de service **D3.2 - Installation d'une solution d'infrastructure** **D3.3 - Administration et supervision d'une infrastructure** • A3.3.1 Administration sur site ou à distance des éléments d'un réseau, de serveurs, de services et d'équipements terminaux
Pré-requis	Avoir réalisé le Coté Labo « Haute disponibilité du service Web » et éventuellement celui sur la « Haute disponibilité du service FTP ».
Savoir-faire principaux	**En SISR3** : • Caractériser les éléments nécessaires à la qualité, à la continuité et à la sécurité d'un service • Installer et configurer les éléments nécessaires à la qualité et à la continuité du service • Contrôler et améliorer les performances d'un service • Valider et documenter la qualité, la continuité et la sécurité d'un service
Prolongements	**En SI7** : • Réfléchir aux stratégies et techniques associées à la continuité du service rendu • Rédiger, mettre en place et tester un Plan de Continuité D'Activité (PCA) **En SISR3** : • Assurer la haute disponibilité du serveur HaProxy lui-même • Répartir la charge sur d'autres services avec HAProxy • Utiliser d'autres fonctionnalités d'HAProxy • Optimiser le service Web (en dehors de la répartition de charge) • Tester l'efficacité de la répartition de charges et de l'optimisation du serveur Web **En SISR5** : • Répartir la charge sur les services réseau • Superviser le serveur HAProxy

Outils	**SE** : Serveur Linux Debian 9 (Stretch - stable actuelle) **Serveurs/services** : Apache2, PHP7, MariaDB/Mysql-server 5.8 installés et configurés à l'identique sur deux serveurs, Corosync et Pacemaker, HAProxy 1.7.5-2. **Clients** : navigateur sur STA Linux, Windows ou autre système (Lynx pour le navigateur en console) **Contexte** : organisation/GSB-Organisation.doc. **Site officiel de HAProxy:** http://haproxy.1wt.eu/ **Documentation en ligne** : http://haproxy.1wt.eu/#docs **Sources :** • http://1wt.eu/articles/2006_lb/ : article sur la répartition de charges par le développeur d'HAProxy. • Article de Fabien Germain dans Linux Magazine Hors-série n°45
Mots-clés	Disponibilité, HA, HD, Heartbeat, Corosync, Pacemaker, Répartition de charge, HAProxy
Durée	8 heures
Auteur(es)	Apollonie Raffalli et David Duron avec la relecture de Yann Barrot

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 ce Côté Labo. Mais le même principe peut s'appliquer sur n'importe quel service aux utilisateurs ou service réseau.

Principes de la répartition de charge

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 :

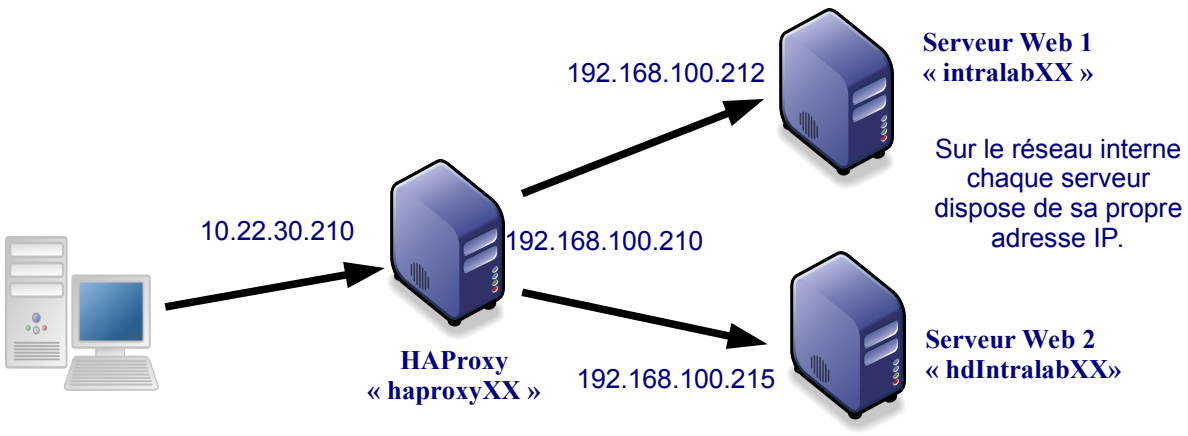
- d'effectuer un aiguillage statique défini par des cookies ;
- d'effectuer une répartition de charge avec création de cookies pour assurer la persistance de session ;
- de fournir une visibilité externe de son état de santé ;
- de s'arrêter en douceur sans perte brutale de service ;
- de modifier/ajouter/supprimer 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.

Architecture et contexte

Une configuration relativement simple va nous permettre de mettre en œuvre la répartition de charge sur les deux serveurs Web préalablement installés avec réplication multi-maître des données (voir Côté Labo « Haute disponibilité du service Web »).



La demande de connexion est adressée au serveur HAProxy (nommé « haproxyXX ») d'adresse IP 10.22.30.210 qui détermine, selon l'algorithme configuré, le serveur auquel il va affecter la connexion, parmi les serveurs disponibles (en l'espèce *intralabXX* ou *hdIntralabXX*).
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.

Le laboratoire pharmaceutique Galaxy-Swiss Bourdin (GSB) met à disposition des visiteurs médicaux une application Web sécurisée de gestion des frais de remboursement (Côté Labo « Service Web sécurisé »). La haute disponibilité de cette application est assurée via un serveur de secours (Côté Labo « Haute disponibilité du service Web »).

GSB dispose ainsi :

- d'un serveur maître *intralabXX* ;
- d'un serveur de secours *hdintralabXX*.

La mise en œuvre de la haute disponibilité a été simulée sur des machines virtuelles présentes dans la ferme des serveurs et le Cluster est accessible par son adresse IP virtuelle correspondant au nom DNS « servIntralabXX.gsb.coop » où XX représente les initiales de vos prénoms et noms.
L'application est installée dans le dossier /var/www/html/appliFrais.

L'équipe informatique de GSB souhaite que chacun de ces serveurs 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'ériger le serveur de secours au même niveau d'activité que le maître.

Pour mettre en place un environnement de test de la nouvelle solution, vous devez « partir » du Cluster installé dans les précédents Côtés Labo et en particulier celui qui traite de la « Haute disponibilité du service Web » en tenant compte des contraintes suivantes :

- le serveur Apache2 doit être démarré sur les deux serveurs ;
- Le service MySQL doit pouvoir supporter une écriture simultanée sur les deux serveurs (voir note ci-dessous) ;
- les trois serveurs communiquent via un réseau privé « interne » ;
- du point de vue du client, l'application sera accessible par l'adresse IP « publique » d'HAProxy ou via le nom d'hôte pleinement qualifié (qui renvoie à l'adresse IP « publique » d'HAProxy).

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.

Les autres services (comme le service FTP) continuent à être accessibles via l'adresse IP virtuelle.

Il est ensuite nécessaire d'ajouter un serveur sur lequel sera installé HAProxy (qui peut être aussi un serveur virtuel) doté de 2 cartes réseaux :

- une considérée comme « publique » permettra l'accès depuis « l'extérieur » ;
- l'autre comme « privée » assurera la communication avec les 2 serveurs Web.

Vous trouverez la documentation suivante :

- **Document 1 :** installation et première configuration d'HAProxy.
- **Document 2** : configuration avancée de HAProxy.
- **document 3 :** exemples de statistiques.

Note

Une base de données devant se mettre à jour en écriture simultanément sur plusieurs serveurs pose des problèmes techniques et il est nécessaire d'utiliser des technologies de clustering comme le fait MySQL Cluster qui est la base de données distribuée de MySQL. Contrairement aux moteurs MyISAM et InnoDB généralement utilisés avec MySQL, l'exploitation effective du cluster nécessite l'utilisation du moteur NDB qui permet de gérer une grappe de serveurs complète. Un protocole implémenté dans chaque nœud s'occupe d'adresser chaque transaction aux différents nœuds concernés dans la grappe.

Mais cette solution professionnelle est, sommes toutes, difficile à mettre en œuvre... Aussi, il est possible de s'en approcher via quelques paramétrages proposés dans le document 1 - Configuration de la réplication MySQL en multi maître du Côté Labo : « Haute disponibilité d'un service Web dynamique » rappelé ci-après.

Dans le cas d'une configuration pour réaliser une répartition de charges, deux autres lignes doivent être ajoutées dans chaque fichier de configuration de MySQL pour éviter les conflits d'insertion menant à l'échec la réplication car lors de l'écriture simultanée sur les deux maîtres, 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'incrémentation entre les valeurs successives des clés (le pas d'incrémentation).
- ***auto_increment_offset*** détermine le point de départ des valeurs des colonnes de type AUTO_INCREMENT.

Ainsi, pour une configuration à N maîtres, mettre :

- auto_increment_increment = N (sur chaque maître) ;
- auto_increment_offset = 1,2, ... , N (chaque maître aura une valeur unique)

Déroulement de la séquence

1. Mise en place de l'environnement de test, installation et première configuration d'HAProxy (aide en annexe 1)

On part ici du principe que l'on utilise la solution mise en place dans les deux Côtés Labo précédents et que l'on doit donc **intégrer le nouveau service HAProxy dans le Cluster existant.**

- Rédigez une note argumentée listant les modifications à apporter à votre environnement afin de pouvoir tester la nouvelle solution.
- Mettez en place cette plate-forme de test.
- Installez HAProxy.
- Procédez à une première configuration d'HAProxy avec mise en place des statistiques et démarrez HAProxy.
- Proposez et réalisez des tests (tests de non régression compris) permettant de vérifier que la solution est opérationnelle.

Attention, nous ne pouvons pas aller très loin à ce stade car nous n'avons pas encore mis en place la persistance des sessions. Le proxy doit renvoyer alternativement sur l'un et l'autre serveur Web sans que l'on ait besoin de s'authentifier.

2. Configuration avancée d'HAProxy (aide en annexe 2)

- Procédez à une nouvelle configuration d'HAProxy en faisant l'hypothèse que le serveur initialement de secours *hdintralab* est trois fois moins puissant que le serveur maître.
- Proposez et réalisez des tests permettant de vérifier que la solution est opérationnelle.
- Revenez à une répartition de charge égalitaire.
- Vérifiez si le changement de serveur ne pose pas de problème lorsque l'utilisateur s'authentifie. Auquel cas, configurez HAProxy pour pouvoir utiliser l'application via le cookie prévu par l'application. *Vous procéderez à tous les tests nécessaires.*

GSB dispose des applications de gestion de parc OCS et GLPI sur un même serveur physique. Ces dernières doivent être accessibles via HAProxy sans toutefois être intégrées dans le mécanisme de répartition de charges.

- Intégrez à HAProxy le(s) serveur(s) Web exploitant les applications de gestion de parc OCS et GLPI.

Document 1 : installation et première configuration d'HAProxy

Installation et démarrage d'HAProxy

La version 1.7.5-2 HAproxy est dans les dépôts Debian 9 (actuellement stable), c'est celle que nous installerons :

```
apt update & apt install haproxy
```

Le démon haproxy est configuré pour se lancer au démarrage de la machine.

Le démon démarre mais le fichier de configuration par défaut ne contient pas toutes les directives nécessaires.
Il est nécessaire de procéder à une configuration minimale.

Configuration basique 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 ;
- **défaults** : 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 puisse remonter de nouveau à la racine.

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

log global indique que l'on souhaite utiliser les paramètres de journalisation définis dans la section 'global'. Les *logs* d'HAProxy sont écrits par défaut dans **/var/log/haproxy.log**.
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.

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

Directives et valeurs	Quelques explications
listen httpProxy bind 10.22.30.210:80 balance roundrobin option httpclose option httpchk HEAD / HTTP/1.0 server web1 192.168.100.212:80 check server web2 192.168.100.212:80 check	**listen** permet de demander à HAProxy d'écouter sur l'IP et le port indiqué par la directive « bind ». Si on indique « *:80 », le HAProxy écoute sur toutes les IP de la machine, mais ici les requêtes ne pourront venir que de l'interface « publique ». 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.

D'autres modes sont possibles :
- **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 Côté Labo, 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 httpchck, 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 pour chaque serveur (voir ci-après).

D'autres paramètres sont utilisables : ils permettent d'adapter la nature du test au service géré par HAProxy. Pour plus de détails, voir la documentation HAProxy :
http://cbonte.github.io/haproxy-dconv/1.7/configuration.html#4.2-option%20tcp-chec
https://www.haproxy.com/documentation/aloha/10-0/traffic-management/lb-layer7/health-checks/ (documentation explicite mais qui ne correspond pas totalement à la version installée).

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 httpchck)**.

Il est courant et parfois obligatoire de scinder la directive *listen* en deux sections *frontend* et *backend* (voir *dans le document 2) :*

- ***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.

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

Pour que les modifications soient prises en compte, il est nécessaire de recharger le fichier de configuration : **root@haproxyXX:~# systemctl restart haproxy**

Pour vérifier que le service fonctionne bien et écoute sur le port 80 avec la commande netstat, en n'affichant que la ligne correspondante à « haproxy » :

```
root@haproxyAR:~# ss -tpnl | grep haproxy
LISTEN    0      128    10.22.30.210:80                 *:*                  users:(("haproxy",pid=1686,fd=6))
```

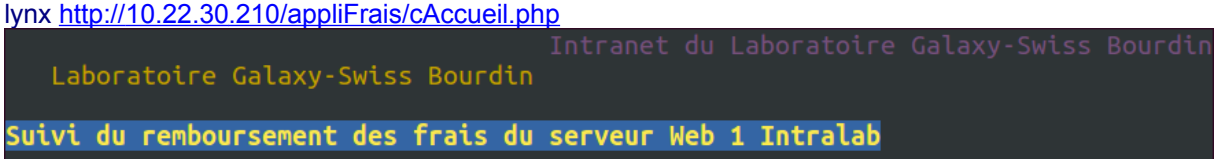
haproxy est bien lancé et attend des requêtes sur le port 80.

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

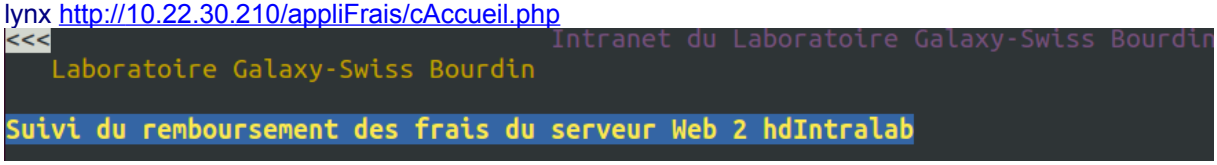
Attention au cache du navigateur qui renvoie la page en cache au lieu de se connecter au serveur (ce qui ne permet donc pas de tester la répartition de charge) !
Plusieurs solutions sont possibles dont :

- désactiver momentanément le cache (sur Chrome F12 + F1 + disable cache) ;
- ouvrir des fenêtres de navigation privée ;
- recharger la page sur « Chrome » avec les touches CTRL+F5 ou l'icône correspond ;
- changer alternativement de navigateur, voire de système d'exploitation ;
- utiliser le client « lynx » en ligne de commande.

lynx http://10.22.30.210/appliFrais/cAccueil.php



lynx http://10.22.30.210/appliFrais/cAccueil.php



HAProxy envoie bien alternativement sur l'un et l'autre serveur Web

Mise en place des statistiques

Directives et valeurs	Quelques explications
listen httpProxy bind 10.22.30.210:80 balance roundrobin … stats uri /statsHaproxy stats auth apo:mdpstats stats refresh 30s	**stats uri** permet d'activer la page de statistiques, en définissant l'endroit où les statistiques pourront êtres 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 dans le document *3* 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.**

Document 2 : configuration avancée de 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 sur lesquels portent la répartition.

Vous trouverez, dans la documentation officielle, les directives admissibles dans chacune des sections : https://cbonte.github.io/haproxy-dconv/1.7/configuration.html

Exemple :

```
frontend proxypublic
        bind 10.22.30.210:80
        default_backend fermeweb

backend fermeweb
        balance roundrobin
        option httpclose
        option httpchk HEAD / HTTP/1.0
        server web1 192.168.100.212:80  check
        server web2 192.168.100.215:80  check
        stats uri /statsHaproxy
        stats auth apo:mdpstats
        stats refresh 30s
```

Les directives *stats* peuvent être définies dans la section *defaults*, ainsi elles s'appliqueront à tous les *backend*.

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 inégalitaire 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 supposerons ici que le serveur *hdIntralabAR* est deux fois moins puissant que le serveur *intralabAR*.

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 *intralabAR* et 50 pour *hdIntralabAR* (la valeur doit être entre 0 et 255).

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

```
server web1 10.22.100.212:80 weight 100 check
server web2 10.22.100.212:80 weight 50 check
```

La problématique des sessions et configuration de leurs persistances

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 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.

La persistance par cookie ajouté 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.

La persistance de chaque session peut se faire via un cookie dédié. HAProxy peut créer lui-même un cookie ou peut utiliser un cookie prévu dans l'application.

Cookie ajouté par HAProxy

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

```
backend fermeweb
        ...
        cookie QUELSERVEUR insert indirect nocache
        server web1 192.168.100.212:80 cookie SW1 check
        server web2 192.168.100.212:80 cookie SW2 check
```

QUELSERVEUR sera l'identifiant du cookie
SW1 et SW2 seront les valeurs possibles

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. L'option ***indirect*** permet d'éviter la génération d'un cookie si un autre cookie valide est déjà présent pour le client
- 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.
- L'option ***nocache*** permet d'éviter la mise en cache du cookie entre le client et HAProxy (le cache ne mémorise pas ce cookie et ce dernier est caché à l'application).
- À la fin de la session, le cookie est détruit.

Que se passerait-il si l'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.

```
backend fermeweb
        ...
        cookie PHPSESSID prefix indirect nocache
        server web1 192.168.100.212:80 cookie SW1 check
        server web2 192.168.100.215:80 cookie SW2 check
```

Exemple de valeur de cookie pour PHPSESSID :
SW1~dgblunnqobg91r7243sidug6u6

La persistance par « source IP »

Il est possible de mettre en place une persistance de la session basée sur l'IP de l'utilisateur. **C**ette persistance est assurée par le biais d'une *stick-table* (qu'il faut déclarer) qui garde en mémoire les adresses IP ayant contactées le serveur.

```
backend fermeweb
        ...
        stick-table type ip size 1m expire 1h
        stick on src
        server web1 192.168.100.212:80 check
        server web2 192.168.100.215:80 check
```

Cette stick-table possède une taille de 1Mo et expire toutes les heures. Lorsqu'un utilisateur est attaché à un serveur il reste sur ce même serveur jusqu'à expiration de la table ou en cas d'erreur du serveur.

Répartition de charge de niveau 7

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 ACL sont déclarées avec, au minimum, un nom, un test et une valeur valide à tester.

Exemple d'utilisation d'ACL :

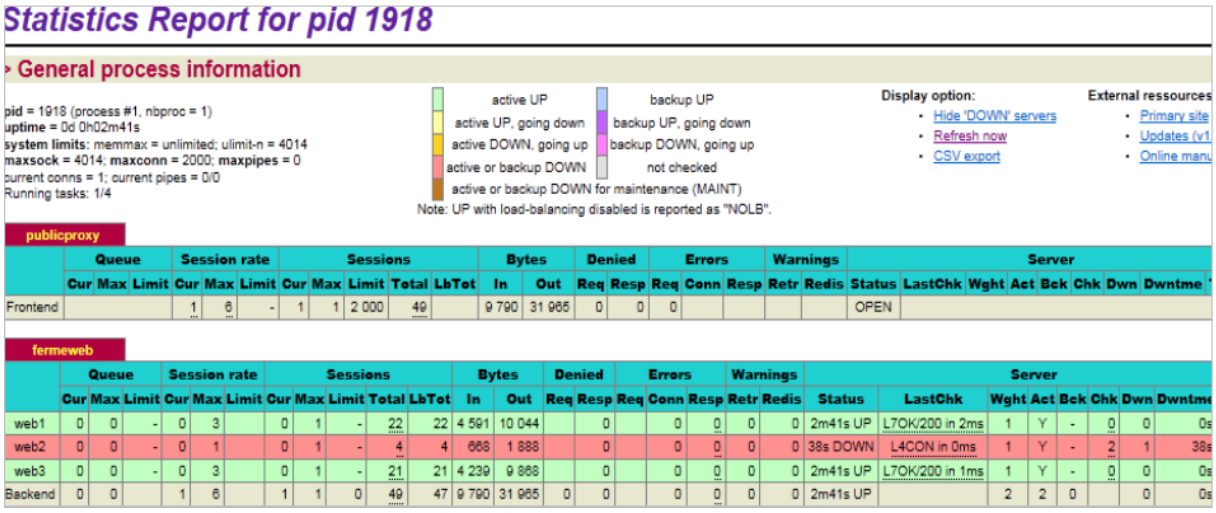
```
frontend proxypublic
        bind 192.168.0.210:80
        acl acces_interface_admin path_beg /gestadm
        use_backend admin if  acces_interface_admin
        default_backend fermeweb

backend admin
      option httpchk HEAD / HTTP/1.0
      server web3 192.168.100.100:80 check
```

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

Il est possible d'ajouter d'autres chemins séparés par un espace sur la ligne de l'ACL.

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

Document 3 : 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 ressources
- Primary site
- Updates (v1
- Online manu

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	9 868		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 :

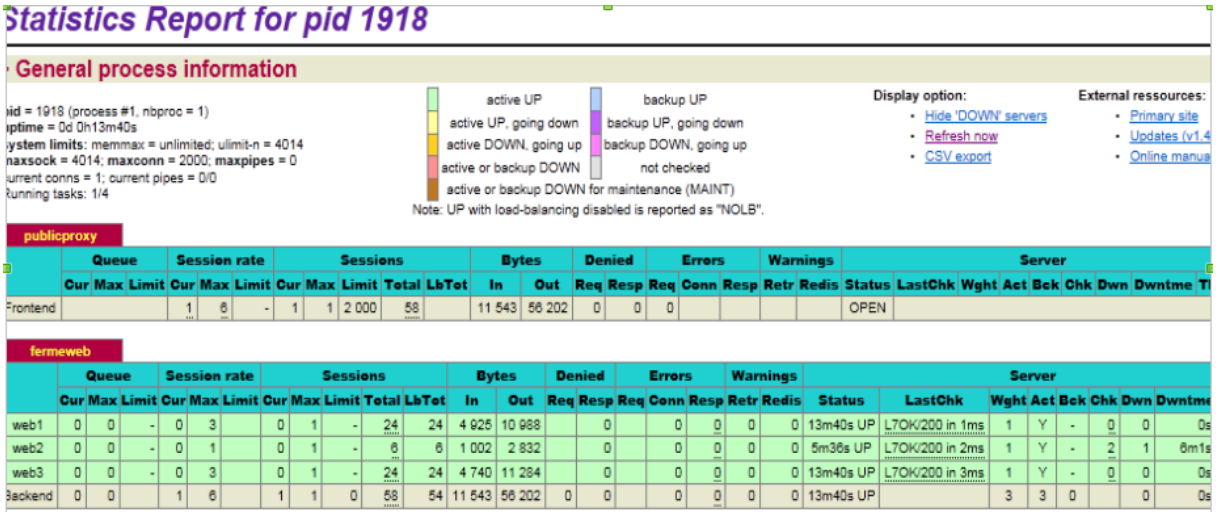
- Le bloc frontend (publicproxy)
- Le bloc backend (fermeweb) disposant de trois serveurs web, deux en « vert » et un en « rouge ».

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 1s, et qu'il a redémarré depuis 5 minutes 36s :



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 ressources:
- Primary site
- Updates (v1.4
- Online manua

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	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
web1	0	0	-	0	3		0	1	-	24	24	4 925	10 988		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	0	58	54	11 543	56 202	0	0		0	0	0	0	13m40s UP		3	3	0		0	0s