FICHE SAVOIRS

RÉPARTITION DE CHARGE SUR UNE PLATE-FORME WEB

Contenu

1. 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 web (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. Mais le même principe peut s'appliquer sur n'importe quel service aux utilisateurs ou service réseau.

2. 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.

3. 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 relai 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. »

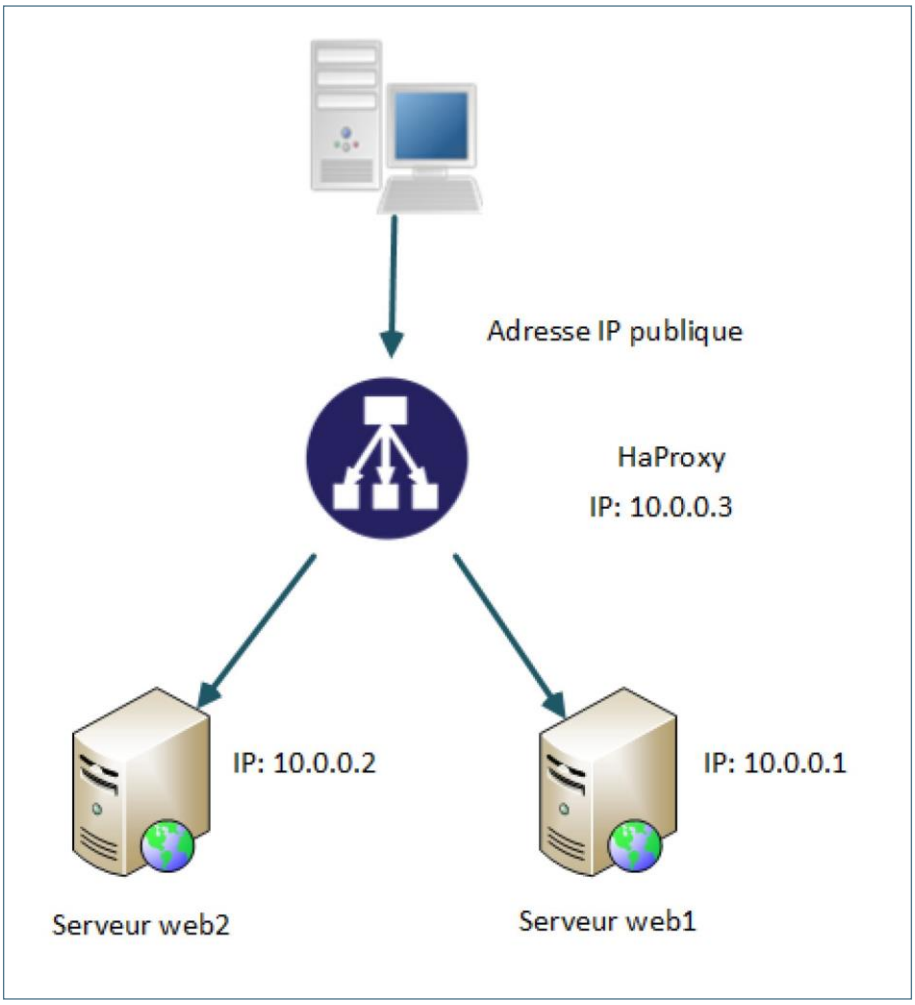
HAProxy peut aider aussi à se prémunir contre les attaques DOS, mais cela ne sera pas abordé dans cette séquence.

4. Architecture

Vous allez utiliser une configuration mettant en œuvre la répartition de charge avec deux serveurs web.

ATTENTION : les adresses IP sont à adapter à votre contexte.

Normalement nous utilisons des serveurs qui sont en cluster au niveau des données avec des technologies comme pacemaker, keepalive, corosync et pour la base de données un cluster galéra pour les bases de données.



Comment cela fonctionne ?

La demande de connexion est adressée au serveur HAProxy sur son adresse publique qui détermine, selon l'algorithme configuré, le serveur auquel il va affecter la connexion, parmi les serveurs disponibles ; ici nous avons les serveurs web : Web1 et 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.

Dans votre cas, vous utiliserez trois machines virtuelles conteneur LXC Debian et les nommer comme suit :

- HaProxy avec comme IP 10.0.0.3 /24 et une adresse IP « publique » ;
- Web1 avec comme IP 10.0.0.1 /24 ;
- Web2 avec comme IP 10.0.0.2 /24.

Pour mettre en place un environnement de test de la nouvelle solution, vous pouvez utiliser un serveur Web avec la page par défaut de Apache et tenir compte de la contrainte suivante :

— du point de vue du client, le site 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.

Il est ensuite nécessaire d'ajouter un serveur sur lequel sera installé HAProxy 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.

5. Mise en place de l’environnement

5A. Installation et première configuration d'HAProxy

Installation et démarrage d'HAProxy

Pour installer HAproxy :

apt update & apt install haproxy.

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

Pour visualiser la version de HAProxy :

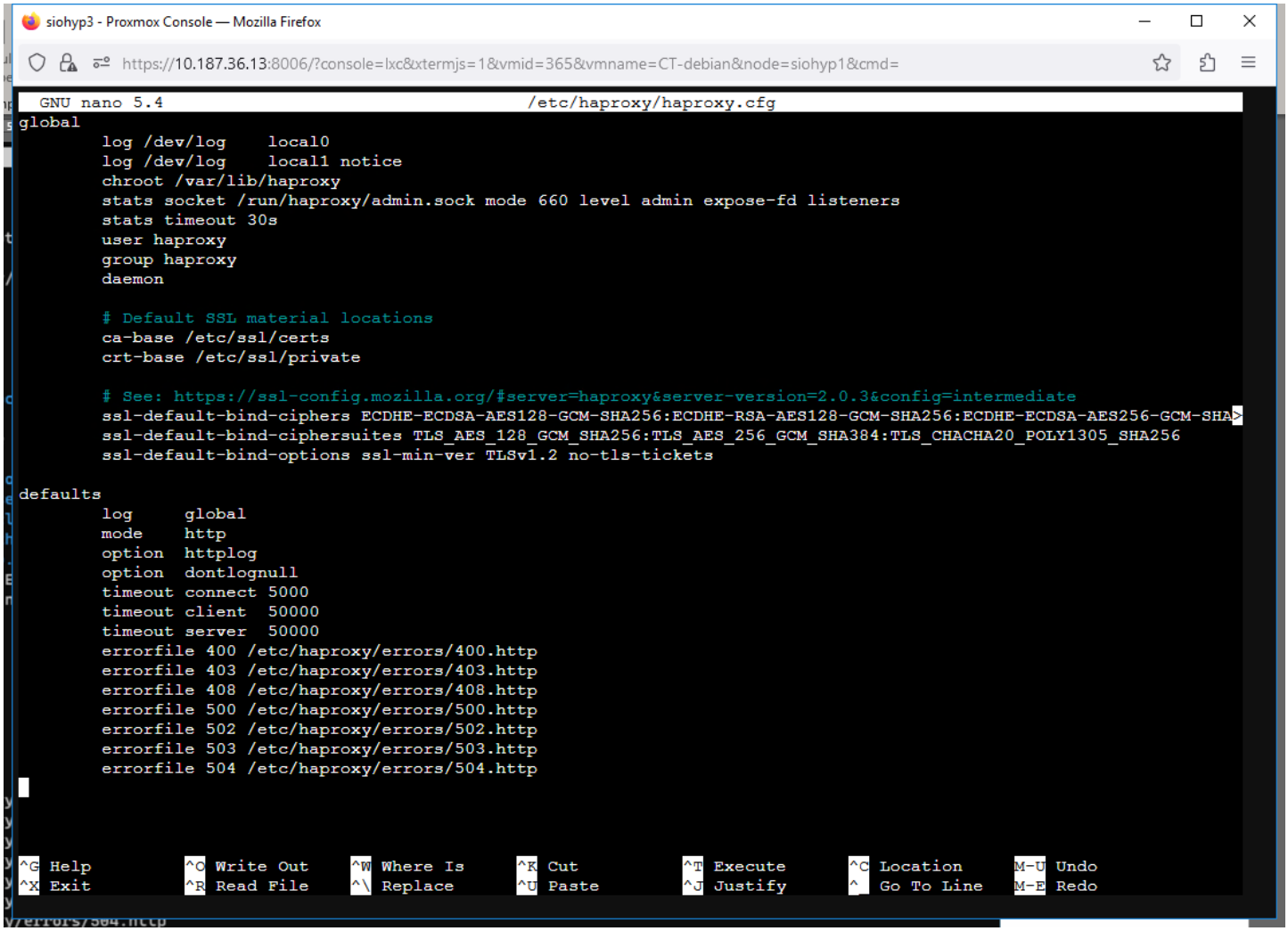
Haproxy -v

```
root@CT-debian:~# haproxy -v
HA-Proxy version 2.2.9-2+deb11u4 2023/02/11 - https://haproxy.org/
Status: long-term supported branch - will stop receiving fixes around Q2 2025.
Known bugs: http://www.haproxy.org/bugs/bugs-2.2.9.html
Running on: Linux 5.15.39-1-pve #1 SMP PVE 5.15.39-1 (Wed, 22 Jun 2022 17:22:00 +0200) x86_64
root@CT-debian:~#
```

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.

Voici le contenu par défaut du fichier /etc/haproxy/haproxy.cfg :



Le fichier de configuration ***/etc/haproxy/haproxy.cfg*** se décompose en plusieurs sections repérées par des mots-clés dont les suivants.

- **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 plateforme 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 @IPpubilqueHaproxy:**80 balance roundrobin option httpclose option httpchk HEAD / HTTP/1.0 server web1 **@IPWeb1:80** check server web2 **@IPWeb2:80** 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.

Important

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, UserAgent, …).
- **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, 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).

Important

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/haproxydconv/2.5/configuration.html#4.2-option%20tcp-chec

---

- 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**).

Important

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 porte la répartition.**

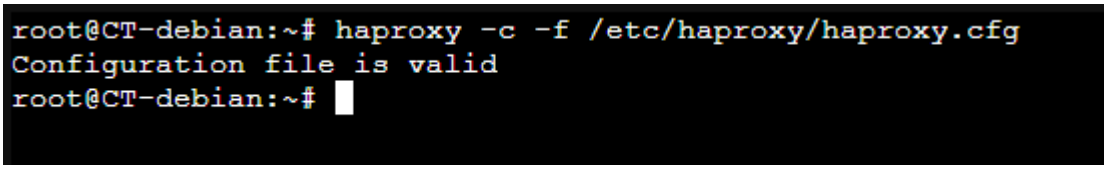
Test d'HAProxy

Pour vérifier que la syntaxe du fichier est correcte :

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

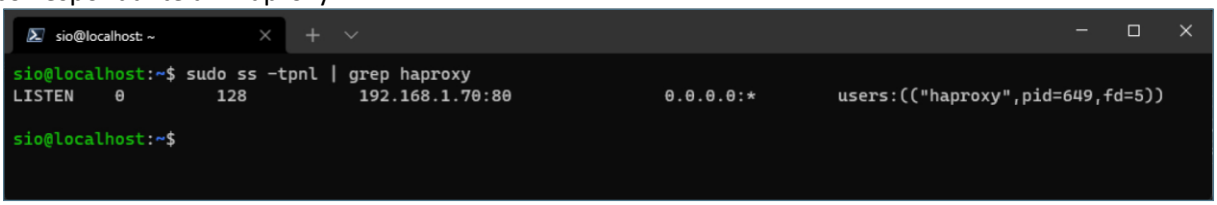
-c pour vérifier (check) le fichier

-f pour spécifier le fichier de configuration



Information

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 ss, en n'affichant que la ligne correspondante à « haproxy » :



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 :

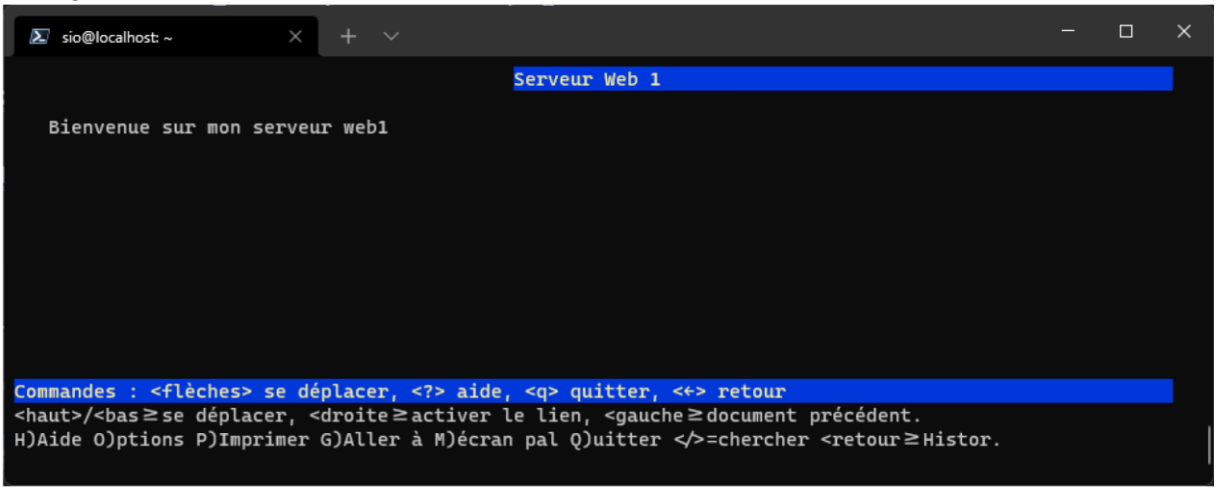
— désactiver momentanément le cache (sur Chrome F12 + F1 + disable cache) ;
— ouvrir des fenêtres de navigation privée ;
— recharger la page sur « Chrome » avec les touches CTRL+F5 ou l'icône correspondant ; — changer alternativement de navigateur, voire de système d'exploitation ; — utiliser le client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

Test de la configuration

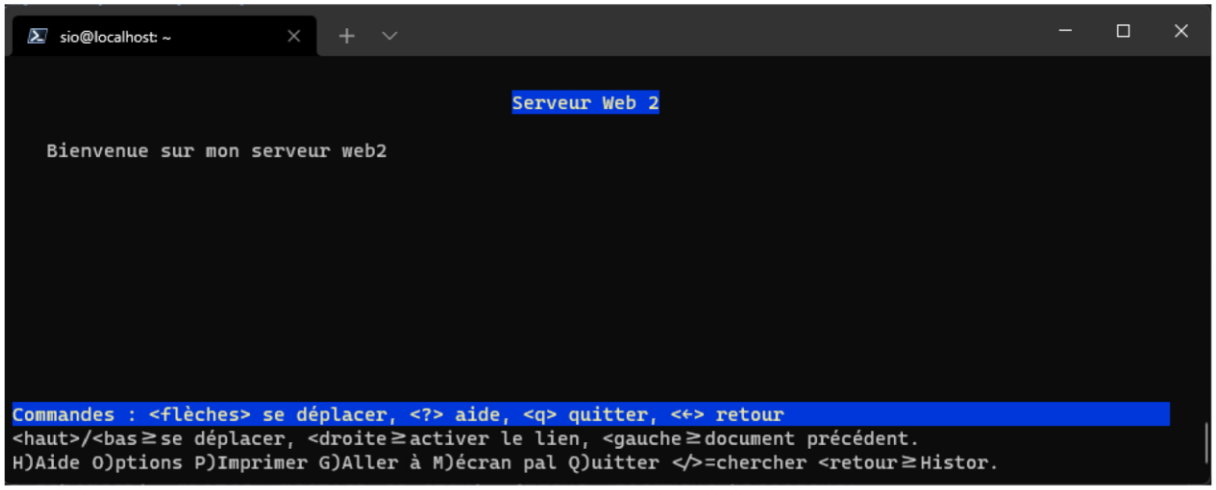
Personnaliser les pages d'accueil des deux serveurs Web afin qu'ils soient différents pour identifier le serveur sur lequel vous êtes.

Depuis votre serveur HAproxy vous pouvez utiliser le navigateur en ligne de commande **Lynx**, il faudra peut-être l'installer. Sinon vous pouvez utiliser un navigateur classique depuis votre navigateur web préféré.

Lynx http://@IPpubliqueHaproxy



Lynx http://@IPpubliqueHaproxy



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 @I**PpubilqueHaproxy:**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 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.

Information

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

5B. 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/2.5/configuration.html.

EXEMPLE

```
frontend proxypublic bind          @IPpubliqueHapproxy:80
default_backend fermeweb backend fermeweb balance
roundrobin option httpclose option httpchk HEAD /
HTTP/1.0
        server web1 @IPWeb1:80 check
        server web2 @IPWeb2:80 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 ***Web1*** est deux fois moins puissant que le serveur ***Web2***.

Il est possible de paramétrer HAProxy pour répartir la charge sur les serveurs, en fonction de leur puissance (algorithme *Weighedleastconnection*) en affectant aux serveurs une pondération différente, reflet de leur puissance : par exemple 100 pour ***Web1*** et 50 pour ***Web2*** (la valeur doit être entre 0 et 255).

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

```
server web1 @IPWeb1:80 weight 100 check server
web2 @IPWeb1:80 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 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 retournée 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 @IPServerWEB1:80 cookie SW1 check
          server web2 @IPServerWEB2:80 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 @IPServerWEB1:80 cookie SW1
check
        server web2 @IPServerWEB2:80 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. Cette persistance est assurée par le biais d'une *stick-table* (qu'il faut déclarer) qui garde en mémoire les adresses IP ayant contacté le serveur.

```
backend fermeweb
        ...
        stick-table type ip size 1m expire 1h
        stick on src    server web1 @IPServerWEB1:80 check
        server web2 @IPServerWEB2:80 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

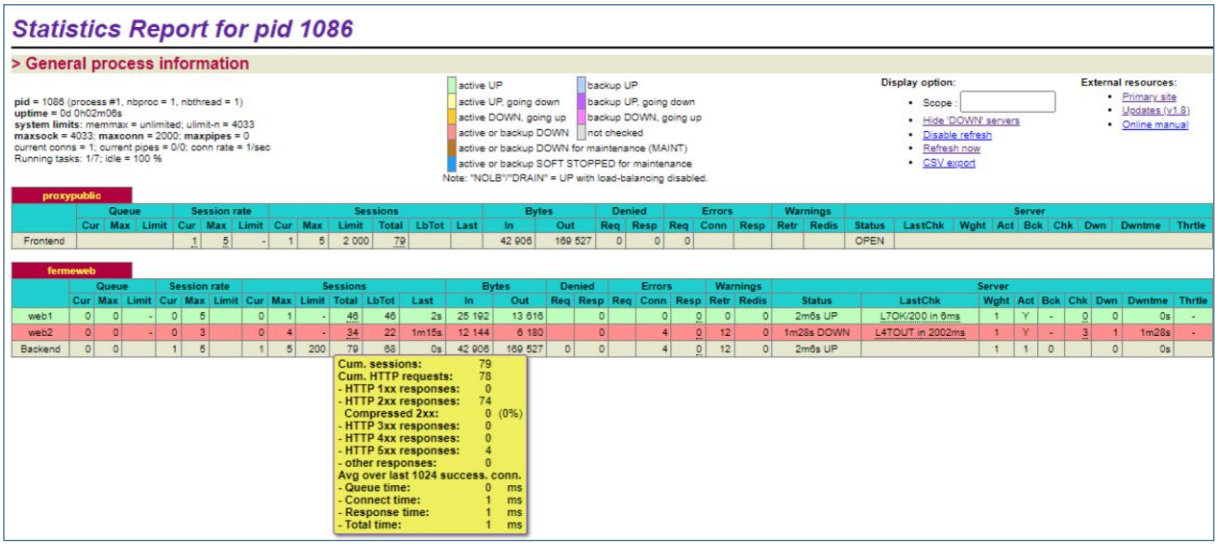
```
frontend proxypublic bind
@IPpubilqueHaproxy:80
        acl acces_interface_admin path_beg /gestadm
use_backend admin if  acces_interface_admin
default_backend fermeweb backend admin
        option httpchk HEAD / HTTP/1.0 server
web3 @IPServerWEB3:80 check
```

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.

5C. Les statistiques



On distingue deux blocs :

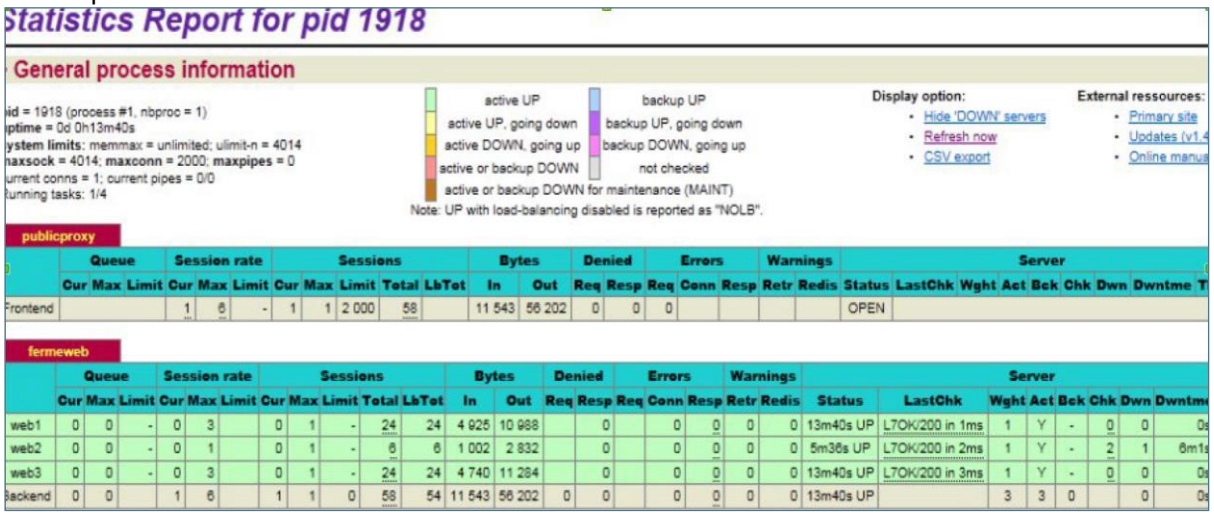
- le bloc frontend (publicproxy) ;
- le bloc backend (fermeweb), disposant de trois serveurs web, deux en « vert » et un en « rouge ».

On constate ici que :

- les serveurs totalisent 79 sessions, dont 48 pour web1, 34 pour web2 ;
- 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 1 minute et 28 s ;
- le serveur frontal web1 est UP depuis 2 min 6 s ;
- 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 1 s et qu'il a redémarré depuis 5 minutes 36 s :



5D. Mise en Pratique

Vous devez disposer de trois machines, ici trois machines linux Debian 10, respecter l'adressage conformément au schéma disponible dans la partie Architecture.

1. Sur la machine HAproxy installer la dernière version de HAproxy.

2 . Sur chacun des serveurs web, créer un fichier HTML il aura comme nom index.html avec le contenu suivant par exemple :

```
< !DOCTYPE html>
<html>
<body>
<h1> Serveur WebX</h1>
<p>bienvenue sur mon serveur webX</p>
</body>
</html>
```

Pensez à modifier le X en fonction du nom de votre serveur web, vous pouvez aussi modifier la couleur du fond de votre page web pour observer les futurs basculements entre vos serveurs web.

3. P our lancer notre serveur web nous allons simplement utiliser **python** avec le module **SimpleHTTPServer** pour avoir notre serveur web opérationnel. Placez-vous dans le répertoire où vous avez créé votre fichier **index.html.**
4. Exécuter la commande suivante :

```
sio@localhost:~/www$ sudo python -m SympleHTTPServer 80
```

Ici nous avons besoin de la commande sudo car nous souhaitons que notre serveur web écoute sur le port 80. Une fois le mot de passe saisie vous devez avoir :

```
sio@localhost:~/www$ sudo python -m SympleHTTPServer 80
[sudo] Mot de passe de sio :
Serving http on 0.0.0.0 port 80 ...
```

5. C onnectez-vous à votre deuxième serveur web et dans un premier temps tentez d'accéder à la page web de votre serveur Web1 avec la commande **curl**.

```
sio@localhost:~/www$ curl http://10.0.0.1
< !DOCTYPE html>
<html>
<body>

<h1> Serveur WebX</h1> <p>bienvenue

sur mon serveur webX</p>

</body>
</html>
sio@localhost:~/www$
```

Cela fonctionne, effectuer la même chose pour votre serveur 2.

6. C onnectez-vous à votre serveur HAproxy et réalisez une configuration de fichier haproxy.cfg pour une répartition égalitaire entre les deux serveurs web.
   a. Ajouter un frontend au fichier haproxy.cfg.

```
frontend proxypublic
        bind @IPpubliqueHAproxy:80          default_backend fermeweb
```

   b. Ajouter un Backend au fichier haproxy.cfg.

```
backend fermeweb
balance roundrobin
option httpclose
       option httpchk HEAD / HTTP/1.0
server web1 10.0.0.1:80 check
server web2 10.0.0.2:80 check
```

c. Redémarrer le service haproxy.service.

d. Connectez-vous à l'aide d'un navigateur à l'interface publique de votre HAproxy, vous devez observer une alternance entre Web1 et Web2.

7. À présent, modifier le poids du serveur web 1 qui a beaucoup plus de ressources. Ajouter les poids suivant 100 pour le serveur Web 1 et 50 pour le serveur Web 2.

```
backend fermeweb
balance roundrobin
option httpclose
       option httpchk HEAD / HTTP/1.0
server web1 10.0.0.1:80 check weight 100
server web2 10.0.0.2:80 check weight 50
```

8. I mplémenter les statistiques, elles doivent être accessibles avec l'URL statsHaproxy avec une authentification basic et elles seront actualisées toutes les 30 secondes.

```
backend fermeweb
balance roundrobin
option httpclose
       option httpchk HEAD / HTTP/1.0
server web1 10.0.0.1:80 check weight 100
server web2 10.0.0.2:80 check weight 50
stats uri /statsHaproxy        stats auth
admin:admin        stats admin if TRUE
stats refresh 30s
```

9. A rrêter le serveur Web1 et tentez d'accéder au site web, que se passe-t-il ? Observez les statistiques.

Seulement le serveur Web 2 répond.

Pour ce qui est des statistiques, le serveur est marqué comme **DOWN**.

HAProxy version 2.5.1-1~bpo10+1, released 2022/01/11

Statistics Report for pid 649

> General process information

pid = 649 (process #1, nbproc = 1, nbthread = 2)
uptime = 0d 0h08m50s
system limits: memmax = unlimited; ulimit-n = 524288
maxsock = 524288; maxconn = 262123; maxpipes = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 704.522 kbps
Running tasks: 0/16; idle = 100 %

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)
active or backup SOFT STOPPED for maintenance
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Disable refresh
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.5)
- Online manual

proxypublic

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	1	1	262 123	3			70 857	1 419 722	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	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
web1	0	0	-	0	4		0	1	-	32	32	3m22s	15 737	9 312		0		0	0	0	0	1m43s DOWN	L4TOUT in 2001ms	100/100	Y	-	3	1	1m43s	-
web2	0	0	-	0	5		0	1	-	39	39	1m8s	19 524	11 349		0		0	0	0	0	8m50s UP	L7OK/200 in 1ms	50/50	Y	-	0	0	0s	-
Backend	0	0		4	6		1	1	26 213	142	71	0s	70 857	1 419 722	0	0		0	0	0	0	8m50s UP		50/50	1	0		0	0s	

Choose the action to perform on the checked servers : Apply

Après le rétablissement de la VM nous avons la possibilité de retrouver la durée pendant laquelle le serveur Web1 n'était plus disponible. Dans ce cas 2 minutes 35 s.

HAProxy version 2.5.1-1~bpo10+1, released 2022/01/11

Statistics Report for pid 649

> General process information

pid = 649 (process #1, nbproc = 1, nbthread = 2)
uptime = 0d 0h10m16s
system limits: memmax = unlimited; ulimit-n = 524288
maxsock = 524288; maxconn = 262123; maxpipes = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 149.551 kbps
Running tasks: 0/16; idle = 100 %

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)
active or backup SOFT STOPPED for maintenance
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Disable refresh
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.5)
- Online manual

proxypublic

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	1	1	262 123	3			75 855	1 637 648	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	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
web1	0	0	-	0	4		0	1	-	32	32	4m48s	15 737	9 312		0		0	0	0	0	34s UP	L7OK/200 in 1ms	100/100	Y	-	3	1	2m35s	-
web2	0	0	-	0	5		0	1	-	39	39	2m34s	19 524	11 349		0		0	0	0	0	10m16s UP	L7OK/200 in 1ms	50/50	Y	-	0	0	0s	-
Backend	0	0		1	6		1	1	26 213	152	71	0s	75 855	1 637 648	0	0		0	0	0	0	10m16s UP		150/150	2	0		0	0s	

Choose the action to perform on the checked servers : Apply

CNED, BP 60200, 86980 Futuroscope Chasseneuil Cedex, France