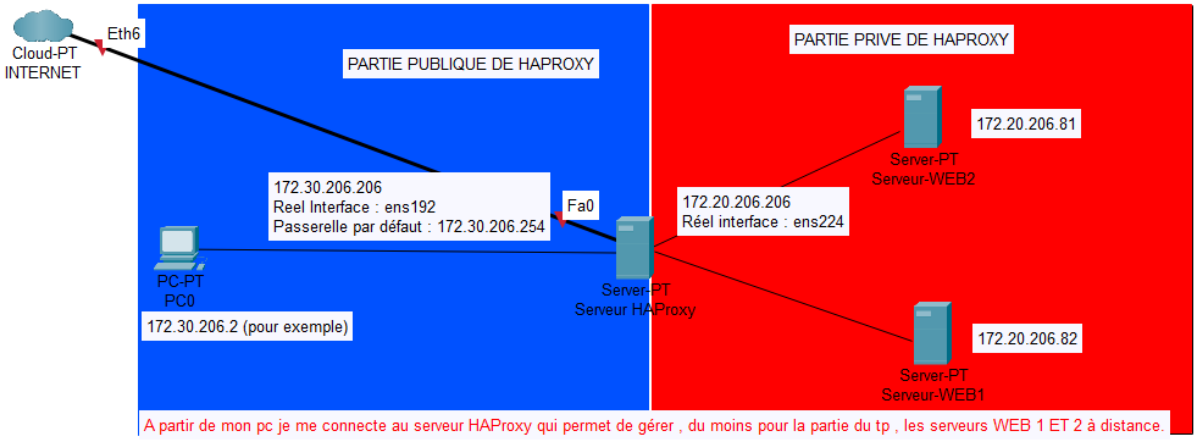
Table des matières


Cloud-PT
INTERNET
Eth6
PARTIE PUBLIQUE DE HAPROXY
PARTIE PRIVE DE HAPROXY
172.20.206.81
Server-PT
Serveur-WEB2
172.30.206.206
Reel Interface : ens192
Passerelle par défaut : 172.30.206.254
Fa0
172.20.206.206
Réel interface : ens224
PC-PT
PC0
172.30.206.2 (pour exemple)
Server-PT
Serveur HAProxy
172.20.206.82
Server-PT
Serveur-WEB1
A partir de mon pc je me connecte au serveur HAProxy qui permet de gérer , du moins pour la partie du tp , les serveurs WEB 1 ET 2 à distance.

Scénario n°1

Démonstration de la répartition de charge sur deux serveurs WEB

```
frontend mon_frontend
    bind 172.30.206.206:80
    default_backend mon_backend



backend mon_backend
    balance roundrobin
    server web1 172.20.206.81:80 weight 2 check
    server web2 172.20.206.82:80 weight 1 check
    option httpchk HEAD / HTTP/1.0
    cookie NOM_COOKIE
```

Je configure HAProxy via la cmd suivante :

nano /etc/haproxy/haproxy.cfg

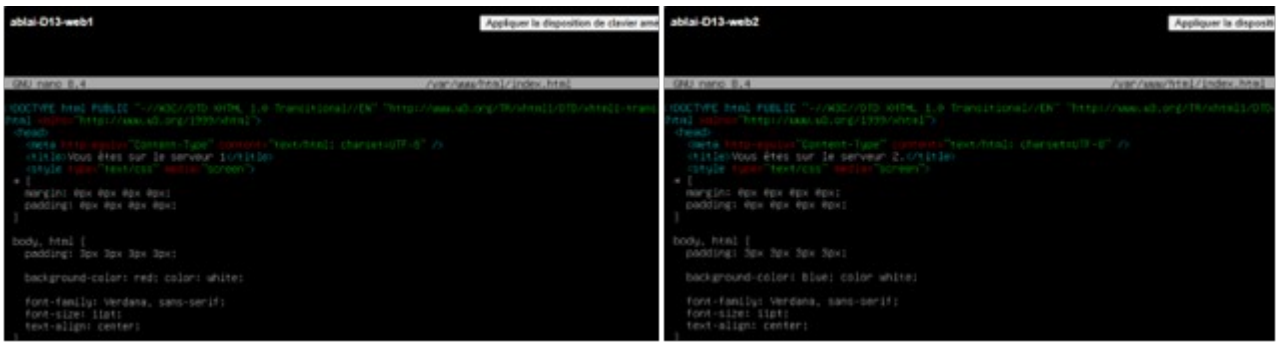
Configuration de la section FRONTEND (Partie publique)

- Avec la commande **« bind »** , **HAProxy** écoute sur l'IP publique 172.30.206.206 port 80 afin de recevoir les requêtes et envoyé 2 fois plus de charges au srv WEB1 que le stv WEB2.
- le **« default backend mon_backend »** définit le backend par défaut (que j'ai laisser de base) pour traiter les requêtes.

Configuration de la section BACKEND (Partie privée)

- La cmd **« balance roundrobin »** permet à ce que la charge soit répartie de façon cyclique entre les 2 srvs WEB.
- Avec la cmd **« server web1 172.20.206.81:80 weight 2 check »**, le srv WEB1 reçoit deux fois plus de requêtes que Web2. La commande « check » vérifie la disponibilité du serveur.
- Avec la cmd **« server web1 172.20.206.82:80 » weight 1 check »,** le srv WEB2 à la moitié de la charge de srv WEB1, la commande « check » vérifiant également la disponibilité du serveur.
- La commande **« option httpchk HEAD / http/1.0 »** permet à HAProxy d'effectuer une requête http HEAD à la racine / pour s'assurer que chaque serveur répond correctement avant d'envoyer du trafic.

Modifications des pages html des deux serveurs pour les différencier.



Test de la répartition de charge

Serveur WEB 1

Serveur WEB 2

Scénario n°2

Comment obtenir des statistiques intéressantes sur la répartition de charge.

Activation de la page statistiques HAProxy (toujours dans le fichier haproxy.cfg)

```
listen stats
    bind *:8404
    stats enable
    stats uri /stats
    stats refresh 10s
    stats auth ablai:Sio2025
```

listen stats : Crée une section d'écoute nommée « stats » dédiée aux statistiques.

bind *:8404 : Indique que l'interface sera accessible sur le port 8404 de toutes les interfaces réseau.

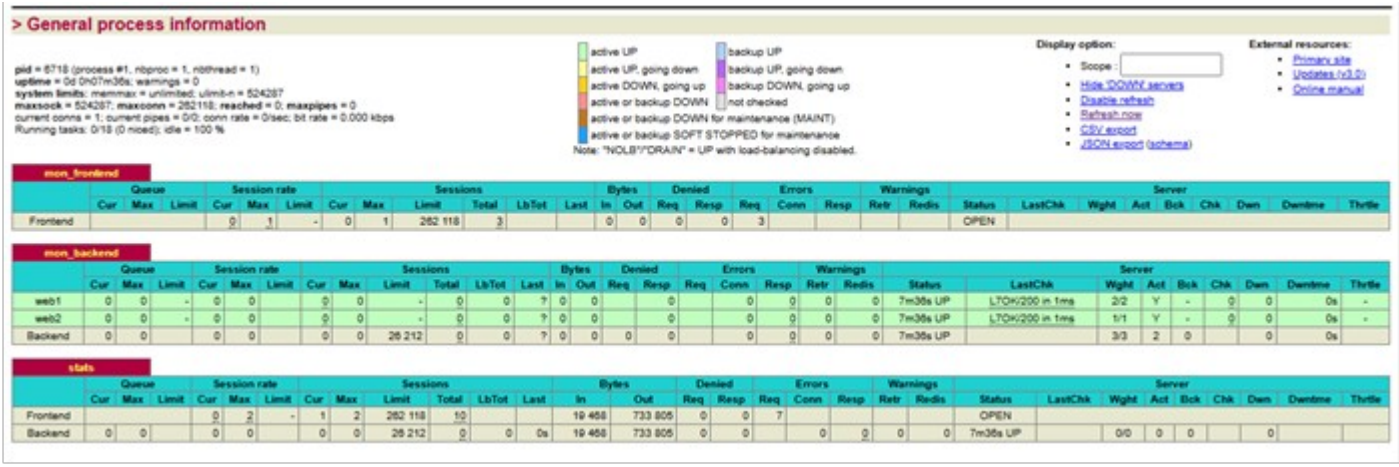
stats enable : Active la page de statistiques HAProxy pour ce listener.

stats uri /stats : Définit l'URL d'accès à la page web des statistiques (http://<ip>:8404/stats).

stats refresh 10s : Rafraîchit la page automatiquement toutes les 10 secondes, pour un monitoring en temps réel.

stats auth ablai:Sio2025 : Protège l'accès par login (authentification HTTP Basic avec ce couple utilisateur/mot de passe).

Page de statistique via l'url 172.20.206.206:8404/stats



Analyse des statistiques HAProxy

L'interface web HAProxy stats permet de surveiller :

- Le nombre actuel de connexions et le taux de sessions pour chaque frontend et backend, donnant une idée précise de la charge et du flux réseau répartis par le load balancer.
- Le trafic échangé (« Bytes In/Out ») par serveur, utile pour vérifier la distribution du trafic entre les serveurs derrière le backend.
- Les erreurs, refus et avertissements permettent d'identifier rapidement des soucis de surcharge, de configuration ou de réseau, impactant la performance ou la disponibilité d'un service.
- Le statut détaillé de chaque serveur backend (UP/DOWN, maintenance, durée, dernier check, etc.) pour suivre la santé et la disponibilité de chaque machine réelle ou virtuelle.

Pour un scénario de répartition de charge, il est donc possible d'y voir si les requêtes sont bien réparties entre les différents serveurs, les temps de réponse, et d'identifier si un serveur sature ou est hors ligne comme nous le verrons dans les prochains scénarios.

Scénario n°3

Observation des états incohérents sur le site-Frais du srv HAProxy (172.20.206.206/siteFrais)

TEST : chaque srv WEB, une connexion d'un utilisateur différent

Dans ce scénario, deux fenêtres affichent la même URL, mais chacune correspond à une session utilisateur différente, « Cottin Vincenne » sur le serveur web1 et « Duduit Frédéric » sur le serveur web2.

Cela est dû au fait qu'il n'y a pas de cookie de session partagé entre les serveurs : chaque backend gère ses propres connexions, donc en l'absence de mécanisme de session centralisé.

L'utilisateur peut changer « d'identité » simplement selon le serveur qui répond à sa requête ou en rechargeant la page.

Informations importante

Cet état incohérent (connexion différente selon le backend) met en évidence l'importance des systèmes et mécanismes de synchronisation de sessions en règle générale.

Cependant, dans ce cas précis, il n'y a aucun cookie permettant au moins l'enregistrement d'une connexion, ce qui explique pourquoi chaque serveur attribue une session indépendante à l'utilisateur, et donc pourquoi on observe des profils différents selon le backend qui répond.

Scénario n°4

Assurer que chaque client reste connecté à un même serveur web.

Config des COOKIES dans la section BACKEND

```
backend mon_backend
    balance roundrobin
    cookie SRVNAME insert indirect nocache
    server web1 172.20.206.81:80 check cookie web1
    server web2 172.20.206.82:80 check cookie web2
```

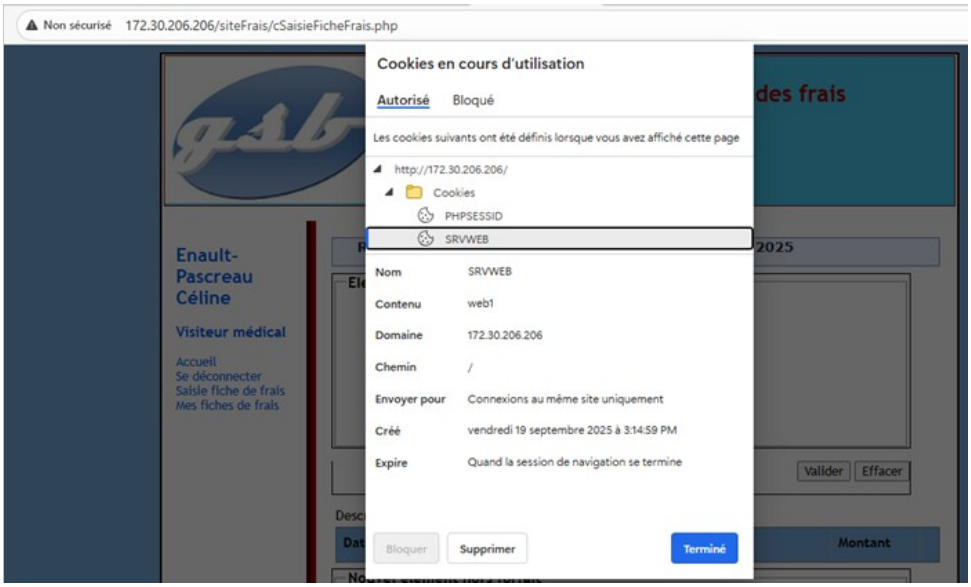
cookie SRVNAME insert indirect nocache

Active la persistance via cookie nommé SRVNAME : HAProxy insère ce cookie dans les réponses envoyées au client.
L'option **indirect** fait que ce cookie n'est pas envoyé aux serveurs backends.
nocache empêche que les réponses contenant ce cookie soient partagées par des proxies intermédiaires ou des caches.

server web1 ... cookie web1 et server web2 ... cookie web2

Pour chaque serveur, la valeur du cookie (web1 ou web2) l'identifie.
Lorsque le client possède ce cookie, toutes ses requêtes suivantes sont directement routées sur ce même serveur.
Cela garantit la constance de session même si plusieurs serveurs participent à la charge.

Déploiement et accès via navigateur, observation des cookies :



Un cookie SRVWEB (nom choisi dans la config HAProxy) apparaît, avec pour valeur web1 :

- Cela signifie que toutes les requêtes suivantes iront systématiquement sur le serveur nommé web1, tant que ce cookie est présenté par le navigateur.
- En cas de rechargement de la page ou de nouvelle navigation, la session reste constante, confirmant que la persistance fonctionne.

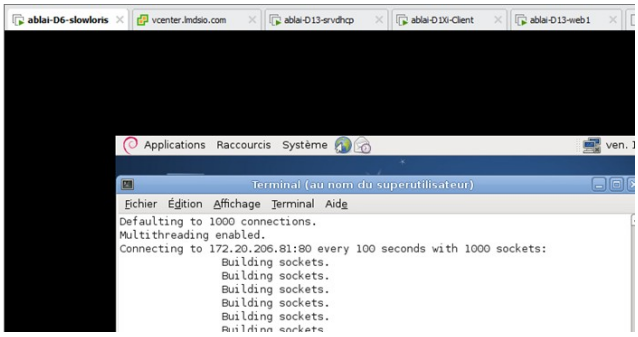
Ce mécanisme est essentiel pour les applications qui nécessitent une "affinité" forte avec un serveur, comme des applications web où la session utilisateur est stockée en mémoire locale du serveur (ex : panier d'achat, authentification).

Scénario n°5

Impact d'une (ou plusieurs) attaques slowloris directement sur le serveur WEB1.

Rappel !

Lancement de deux attaques slowloris

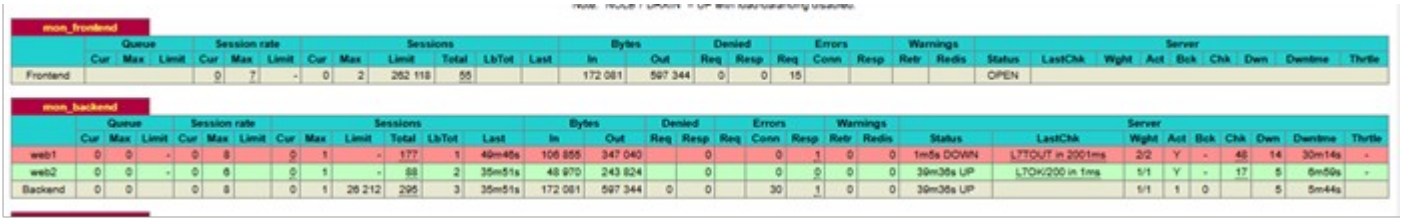


Slowloris est une attaque de type Denial of Service (DoS) particulièrement efficace contre les serveurs web.

Elle consiste à ouvrir de nombreuses connexions HTTP vers le serveur cible (WEB1).
Cela épuise rapidement les ressources du serveur, qui ne peut alors plus répondre aux vraies demandes des utilisateurs.
C'est une attaque "silencieuse" et efficace qui ne consomme pas beaucoup de bande passante mais provoque un gros problème de disponibilité.

"Connecting to 172.20.206.81:80 every 100 seconds with 1000 sockets : Indique que le script essaie d'établir jusqu'à 1000 connexions simultanées sur le serveur web1. Le but est de "garder ouvertes" ces connexions pour épuiser les ressources serveur.

Données sur la page de statistiques HAProxy



mon_frontend

	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	7	-	0	2	262 118	55			172 081	597 344	0	0	15					OPEN							

mon_backend

	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	8		0	1	-	177	1	49m46s	106 855	347 040		0	0	0	1	0	0	1m5s DOWN	L7TOUT in 2001ms	2/2	Y	-	48	14	30m14s	-
web2	0	0	-	0	6		0	1	-	88	2	35m51s	48 970	243 824		0	0	0	0	0	0	30m36s UP	L7OK/200 in 1ms	1/1	Y	-	17	5	6m50s	-
Backend	0	0		0	8		0	1	26 212	295	3	35m51s	172 081	597 344	0	0		30	1	0	0	30m36s UP		1/1	1	0		5	5m44s	

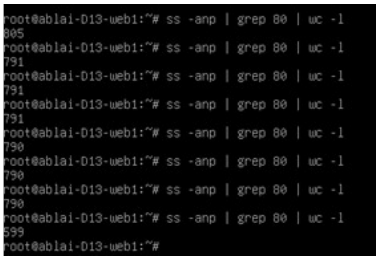
Pour le serveur **web1**, on constate plusieurs indicateurs négatifs :

- La couleur rose indique un état critique ou dégradé.
- La colonne "Status" affiche "DOWN", confirmant que le serveur ne répond plus correctement.
- On observe un nombre élevé de connexions en cours ("Sessions") et des erreurs indiquées dans "Errors" et "Denied".
- Cela signifie que web1 est saturé, submergé par l'attaque Slowloris et incapable de traiter les requêtes normales.

Pour le serveur **web2**, on voit un état normal :

- "Status" est "UP", donc il est toujours disponible et fonctionnel.
- Le nombre de sessions est faible comparé au serveur web1, illustrant qu'il n'est pas ciblé par l'attaque et qu'il absorbe une partie de la charge.

Connexions actives sur le serveur WEB1



```
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
805
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
791
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
791
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
791
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
790
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
790
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
790
root@ablai-D13-web1:~# ss -anp | grep 80 | wc -l
599
root@ablai-D13-web1:~#
```

ss -anp : affiche toutes les connexions réseaux détaillées.

grep 80 : filtre celles qui sont sur le port 80 (HTTP).

wc -l : compte le nombre de lignes, donc le nombre de connexions actives.

- Les résultats affichés (805, 791, 790, 799, 599, etc.) montrent que le serveur a plusieurs centaines de connexions ouvertes en permanence.
- Ce nombre anormalement élevé est typique d'une attaque Slowloris, qui cherche à saturer le serveur en maintenant ouvertes de nombreuses connexions HTTP très lentement.