Haproxy
The Ultimate Load Balancer

Réalisé par :

KEDDAM	Samira	20215669@etud.univ-evry.fr
BOUBERRAGA	Mériem	20213876@etud.univ-evry.fr
Qingsong	YANG	20213338@etud.univ-evry.fr
LAKHEL	Amine	amine.lakhel@universite-paris-saclay.fr

Encadré par :

PETIT Pascal

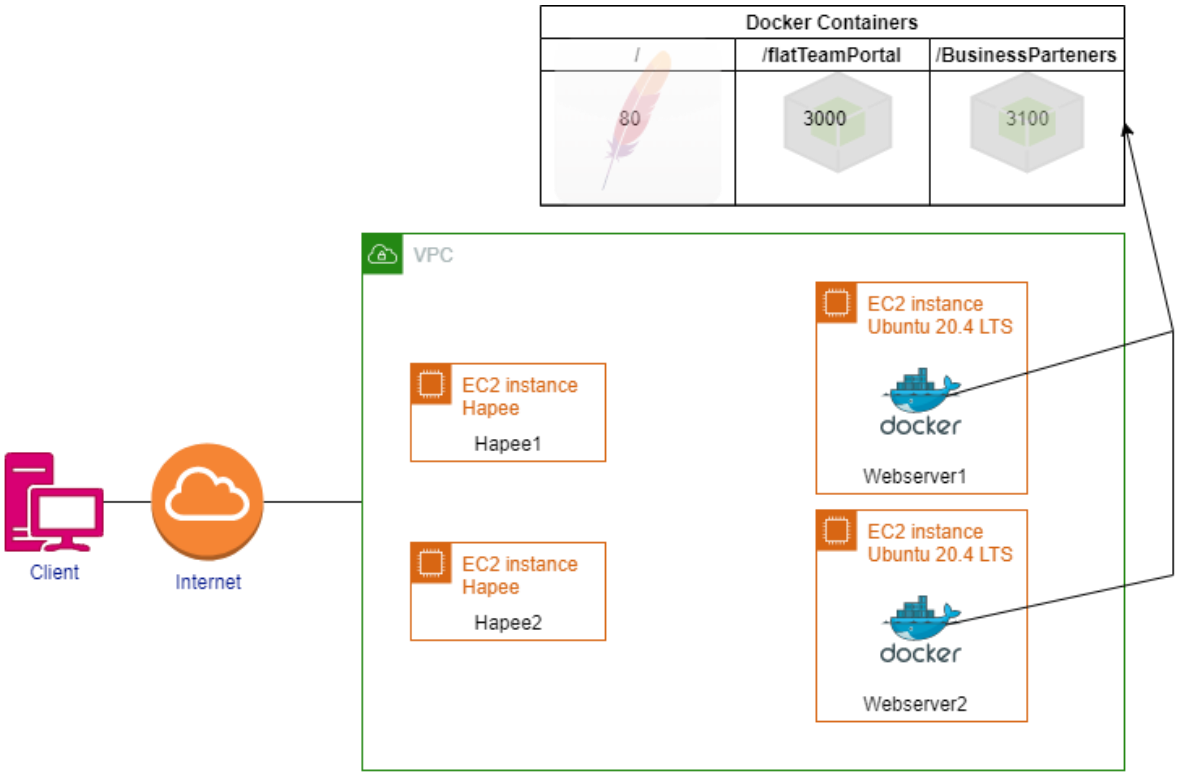
Sommaire :

I. Introduction

Les sites web à fort trafic doivent répondre à des centaines de milliers, voir des millions de demandes simultanées de manière rapide et fiable pour cela la meilleure pratique informatique est de passer par un serveur intermédiaire avec principal objectif load-balancing (répartition des charges) tel que les serveurs Haproxy et Linux virtual server LVS qui ont principalement cette fonctionnalité et bien plus encore, et cela de façon transparente au niveau de l'utilisateur et de manière précise.

Haproxy est une solution libre, fiable et très performante de répartition de charge au niveau de la couche 4 et la couche 7 il est particulièrement adapté aux sites web fortement chargés qui nécessitent de la disponibilité ainsi que la fiabilité
Étant performant, Haproxy est utilisé par des sociétés pour servir des millions de pages chaque jour. HAProxy requiert peu de ressources, et son architecture évènementielle mono-processus lui permet facilement de gérer plusieurs milliers de connexions simultanées sur plusieurs relais sans effondrer le système.

1. Maquette basé sur AWS:



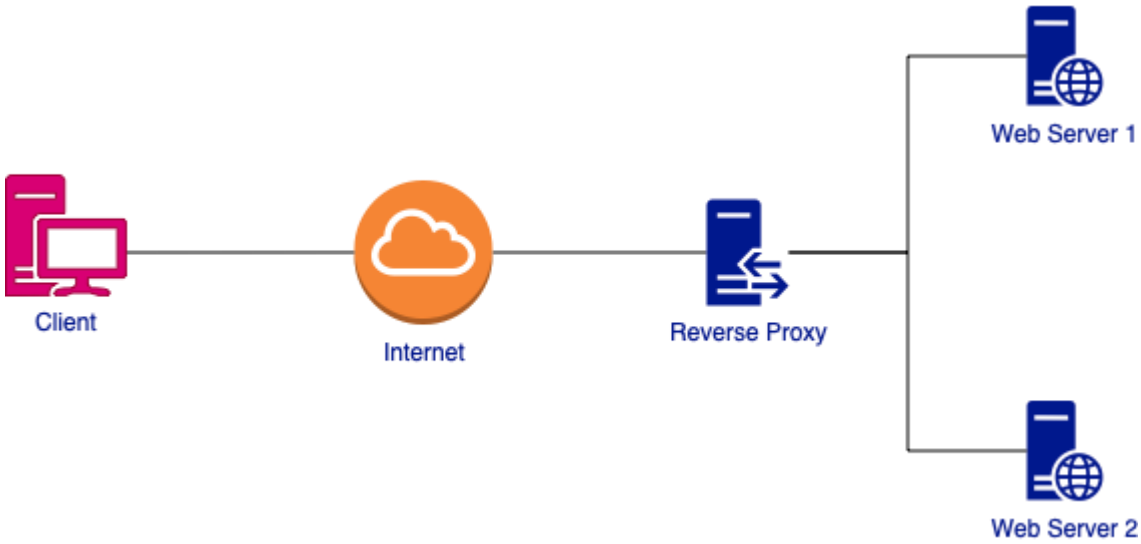
2. Git repository

https://github.com/alakhel/haproxy.git

II. Reverse proxy

1. What's a reverse proxy ?

un intermédiaire entre les serveurs web et ses clients, chaque requête envoyée passe par le reverse proxy, subis des traitements puis elle sera transmise au serveur web.



Il est capable de faire principalement de l'équilibrage de charge, rendre l'infrastructure des serveurs web abstraite et invisible pour le client.
Coder à zéro un reverse proxy n'est pas difficile, plusieurs langages de programmation peuvent être utilisés, l'avantage sera la flexibilité en termes de fonctionnalités et plus ajustable.
Il existe plusieurs outils qui jouent le rôle de reverse proxy notamment NGINX et Haproxy.

2. Load Balancer Layer 4

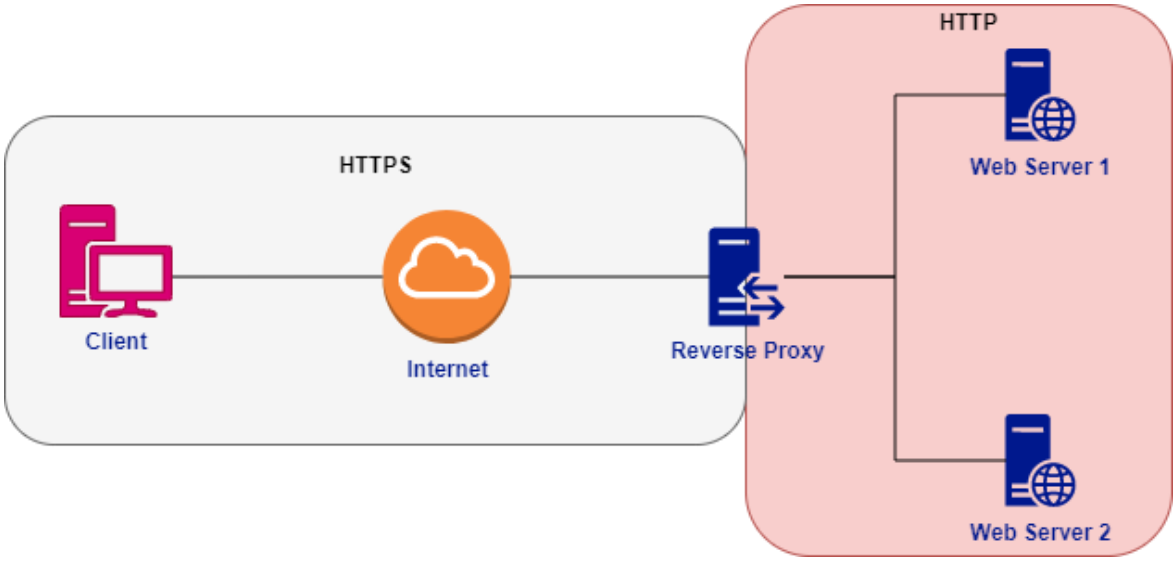
Il distribue les requêtes en se basant sur la couche 4, Il reçoit des requêtes via les clients, il choisit un serveur, ensuite il remplace l'adresse IP dans la paquet par l'adresse IP du serveur, enfin il envoie les requêtes au serveur.

3. Load Balancer Layer 7 and API Gateway

L'équilibrage de charge sur la couche 7 permets d'analyser la requête HTTP puis la transférer en dépendent de ce qui a été analysé, l'ignorer, répondre avec une page spécifique depuis le reverse proxy lui-même.
API Gateway : se baser sur les url dans la requête HTTP et transférer la requête au cluster concerné.

4. SSL Termination (SSL OFFLOAD)

Les requêtes HTTPS entre le client et le reverse proxy, décryptées puis envoyées en HTTP au serveur web. Pour la simple raison, alléger la charge sur les serveurs et réserver ses ressources au traitement des requêtes.



III. Haproxy

Haproxy comprend plusieurs sections qu'on définit sur le fichier de configuration chacune décrit comment le serveur doit fonctionner dans son ensemble, la puissance et les performances d'un serveur haproxy sont obtenus en combinant ces sections :

1. Frontend

On définit le port et l'interface d'écoute, auxquels les clients peuvent se connecter. Un frontend peut écouter sur plusieurs adresses IP et ports. on peut avoir plusieurs frontend.
Chaque frontend contient aussi des règles qui s'appliquent sur les requêtes reçues et laisse une sorte de tag puis les transférer au backend avec use_backend x selon le tag.

```
frontend fe_http
    bind *:80
    acl PATH_flatteam path_beg -i /flatTeamPortal
    use_backend be_front_app if PATH_frontapp
```

2. Backend

Dans cette section on peut déclarer les clusters de serveurs, leurs donner un nom, et définir l'algorithme d'équilibrage de charge.

```
backend be_front_app
    balance roundrobin
    server webserver1 192.168.1.21:80 check
    server webserver2 192.168.1.22:80 check
```

3. Stats

HAProxy est livré avec un tableau de bord appelé HAProxy Stats page qui montre une multitude de mesures qui couvrent la santé des serveurs, les taux de demandes actuelles, les temps de réponse, etc. Ceci pouvant être utile afin de vérifier la répartition de charge ou le bon fonctionnement du service, ce tableau contient l'état des health checkers pour chaque serveur dans chaque cluster, la qualité des requêtes reçues ainsi que le taux des requêtes par client qui est un indicateur hyper important pour détecter les attaques DDOS.

Ces métriques fournissent des données granulaires par frontend, backend et serveur. Il suffit d'ajouter une directive d'activation des statistiques, qui est généralement placée dans sa propre section frontale.

La configuration de cette interface se fait simplement via les options suivantes :

```
backend be_stats
  stats uri /hapee-stats
```

Le mécanisme le plus couramment utilisé est la page de statistiques HTTP. Les statistiques sont consultables sur la page HTTP.

(Voir annexe 1)

4. Healthcheckers

Haproxy propose des HC -L4 et HC-L7

HC-L7 : sont sophistiqués et flexibles, on prend un exemple ou deux ci dessous, premier le health check active puis le healthcheck passive.

Active :
Une requête HTTP de toute type de méthode est envoyée au serveur pour vérifier si un tel serveur est disponible dans tel chemin dans telle méthode HTTP.

```
backend be_myapp
  option httpchk GET /healthz # GET: methode HTTP /healthz: uri
  server srv1 10.0.0.1:80 check # check: pour activer le HC
```

Passive :
à la place d'envoyer régulièrement des requêtes pour vérifier si le serveur est fonctionnel, On surveille également les requêtes clients, avec le mot clé error-limit, dès que y'a x erreur on définit une action à déclencher avec le mot on-error

```
backend servers
   server server1 192.168.0.10:80 check  observe layer7  error-limit 10  on-error mark-down
#on-error actions :
 # mark-down, fastinter, fail-check, sudden-death
```

5. Runtime API and 0 down-time

Haproxy propose le runtime API qui permet de faire des changements comme ajouter des serveurs dans cluster mais l'ajout des backend ou frontend n'est pas possible. Il vous permet de configurer certains aspects de l'équilibreur de charge lors de l'exécution sans avoir à recharger le service.
Encore un autre mais on peut changer la configuration carrément et lancer un reload.

Reload: commence par créer des nouveaux processus et transférer les listeners et donc laisse les anciens processus terminer leur exécution tranquillement.

Dans l'exemple ci-dessous, on exécute la commande pour activer un frontend.

```
$ echo "enable frontend www" | socat /var/run/hapee-2.4/hapee-lb.sock stdio
```

6. ACL

Une ACL est une expression qui renvoie vrai ou faux, que vous pouvez ensuite utiliser pour prendre une décision dans votre configuration. Par exemple, dois-je acheminer cette requête vers le backend A ou le backend B ? ou dois-je rediriger cette demande vers un autre domaine ? ou peut-être devrais-je rejeter la connexion de ce client ?

```
frontend www
  bind :80
  acl image_url path -i -m beg /images/ions dans un bloc,
```

7. API Gateway avec authentification

Dans un environnement de microservice, haproxy permet l'acheminement des requêtes et également aussi protéger les chemins qui demandent que l'utilisateur soit authentifié soit avec un certificat ou mot de passe. Il supporte le SSO aussi avec plusieurs protocoles notamment Kerberos et SAML. L'utilisation d'un 'Identity management server' est possible.

8. Community edition, Enterprise edition, Aloha appliance

- La version Community est open source mais reste limitée.
- La version Entreprise est payante y'a beaucoup plus de fonctionnalités notamment une interface graphique en real time, Pare-feu pour applications Web, Détection de BOT, Route Health Injection.
- Aloha c'est une appliance haproxy, il existe une version virtuelle installable sur n'importe quel serveur physique ou hypervisor et une version Hardware.

Les prix de Hapee :

Détails Produit (Prix HT)

Code du produit	Description du Produit	Description de l'élément de ligne	Quantité	Prix de vente	Prix total
HAP-PRE-1Y	HAProxy Enterprise Edition, Premium Level, 1 Year Subscription (per server)	Souscription annuelle HAPEE avec support 24x7	2,00	EUR 3 650,00	EUR 7 300,00
PSxC	General professional services. Define time/amount in the line item of order form	Prestation d'expertise et de conseil	1,00	EUR 2 000,00	EUR 2 000,00

Sous-total	EUR 9 300,00
Remise	0,00 %
Total	EUR 9 300,00

IV. Haute disponibilité

- Active-Active : toutes les instances de haproxy sont actives, dans ce cas une solution d'équilibrage de charge doit être mise en place devant ces instances (ex : DNS RR, LVS, Haproxy en couche 4).
- Active-Standby : le cas normal où un seul serveur est maître et le deuxième est esclave.

1. Tolérance de panne avec keepalived :

Keepalived n'est pas vraiment nécessaire pour Hapee, mais le principe est pareil, Hapee a sa propre extension de vrrp

```
$ sudo apt-get install hapee-extras-vrrp
```

2. BlueGreen deployment

Give The ability to have a high delivery rate environment with zero down time

On veut dire par là un environnement de mise en prod régulier des updates, qui veut dire changer la configuration de haproxy régulièrement et notamment le redémarrage des serveurs backend.
Haproxy avec outils permet de faire du BlueGreen deployment pour éviter l'interruption du service en utilisant le runtime api, on switch depuis le backend qui a l'ancienne version bleu vers le nouveau qui la version verte.

3. Route Health Injection

Avec l'extension hapee-rhi on peut annoncer l'adresse IP de l'instance avec les protocoles de routage BGP ou OSPF, il utilise le BIRD-Deamon pour y faire. Donc les tant quand l'instance de haproxy n'a plus aucun serveur backend fonctionnel il arrête d'envoyer les annonces. et donc l'entrée dans la table de routage du routeur sera supprimée, donc le routeur n'a plus de connaissance sur l'existence de cette instance.

4. Dynamic-update

The dynamic-update section permet à HAProxy Enterprise d'extraire les fichiers ACL et Map mis à jour à partir d'un serveur distant. Cela permet à un cluster d'instances HAProxy Enterprise de rester synchronisé.

5. Configuration du cluster Haproxy

```
peers mycluster
  # local host, active instance
  peer loadbalancer1 192.168.1.10:10000
  # standby instance
  peer loadbalancer2 192.168.1.11:10000
```

6. Persistance de session

Ajouter un cookie aux requêtes afin que lorsque serveur esclave devient maître sache quel serveur web traitent les requêtes d'un client lambda.

```
cookie SERVER_USED insert indirect nocache
   server webserver1 ip-172-31-1-118.eu-west-3.compute.internal:80 check  cookie webserver1
   server webserver2 ip-172-31-1-119.eu-west-3.compute.internal:80 check  cookie webserver2
```

Dans le navigateur :

Nom	Valeur	Domain	Path	Expires..	Taille
SERVER_USED	webserver1	srv.lakhel.com	/	Session	21

On constate que c'est le serveur 1 qui traite les demandes de ce client.

V. Sécurité

1. Web Application Firewall

-Hapee propose ce qu'on appelle WAF, la documentation n'est pas accessible au public.
-Permet d'atténuer les attaques XSS et SQLi en analysant le trafic HTTP.

2. Bot protection

Fingerprint Module, Javascript Challenge (renvoie des challenges au client pour s'assurer que ce n'est pas un robot) et autre module.
Ces modules et comment les configurer n'est pas accessible, un compte client est nécessaire.

3. Utilisation de stick tables et HTTP Response policies

Utiliser pour construire des compteurs, retenir les sessions avec leurs informations de tables avec le nombre et la durée de session par client.

```
# use a stick table to track request rates
stick-table type ip size 100k expire 2m store http_req_rate(1m)
http-request track-sc0 src
```

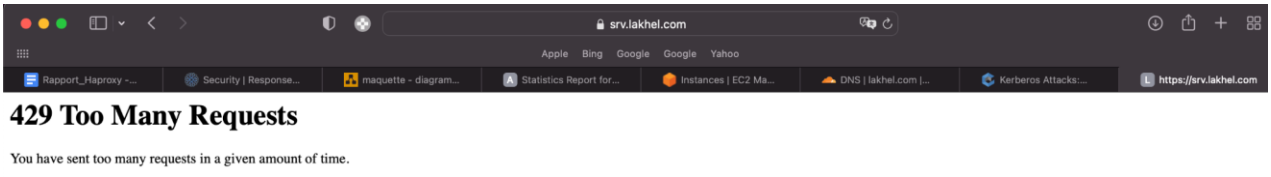
Dans le code ci-dessous on a défini une table qui va retenir le taux de requêtes par client, on utilise les acls, on va aussi ajouter des tags:

```
acl too_many_requests sc_http_req_rate(0) gt 20
```

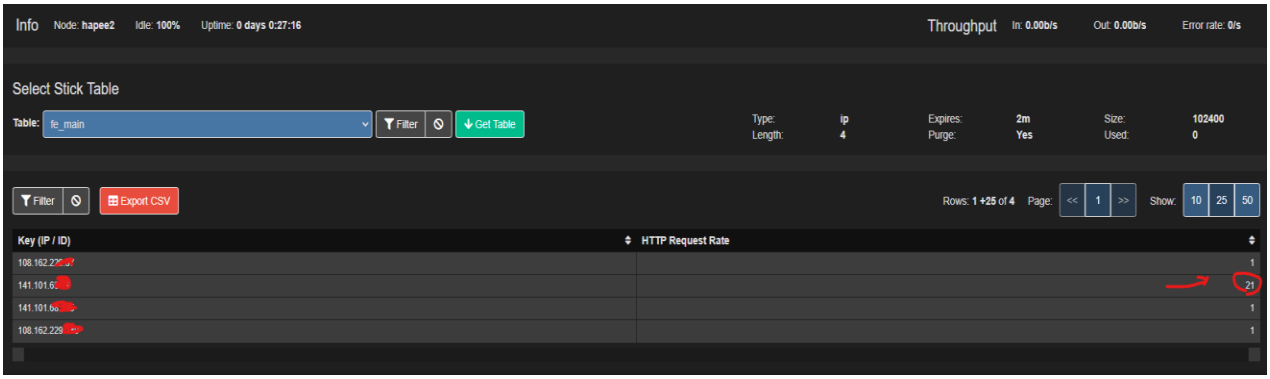
Puis utiliser http-request pour refuser les paquets s'il trouve que le tag est vrai:

```
http-request deny deny_status 429 if too_many_requests
```

Les résultats sur le navigateur après plusieurs tentatives la page ci-dessous est renvoyée. le code 429 renvoie une page par défaut sinon on peut la personnaliser avec http-errors comme exemple ci-dessous :



Et le compteur du stick table nous donne ce résultat :



Personnalisation des pages d'erreurs :

```
http-errors custom_errors
  errorfile 403 /etc/hapee-2.4/errors/403.http
```

Les types de réponses sont :

- **Deny:** refuse la requête HTTP du client ou la réponse du serveur, avec un code d'erreur.
- **reCaptcha :** utilise un module de Hapee, renvoie un captcha pour vérifier si c'est pas un bot.
- **Shadowban** : répondre avec une page fake.

```
acl blocked_ip src -f /etc/hapee-2.4/blocklist.acl
   http-request return content-type text/html file
/srv/www/fake_login.html if { path_beg /login } blocked_ip
```

- **Silent Drop :** fermé la connexion sans notifier le client, le problème : le firewall va garder la session active
- **Tarpit :** refuse la connexion avec un statut 429 mais pas immédiatement, la durée est modifiable. Souvent utilisé contre les bots au lieu de les refuser immédiatement et qu'il réessaye on les bloquent pour un moment.

```
  stick-table  type ip  size 100k  expire 30s  store http_req_rate(10s)
  http-request track-sc0 src
  timeout tarpit 10s
  http-request tarpit deny_status 429 if { sc_http_req_rate(0) gt 20 }
```

- **Reject :** coupe immédiatement la connexion tcp ou refuse la requête HTTP sans réponse. On l'utilise surtout pour les cas où les IP src sont carrément bloquées; le client est notifié que la connexion est fermée.

```
tcp-request connection reject if blocked_ip
```

Les bonnes pratiques :

- Limiter le maximum de connexion par serveur selon sa capacité, ça évite que le serveur soit très alourdi et tombe en panne.
- Refuser le trafic qui n'a pas le blueprint d'un vrai user-agent dans la requête HTTP.
- Vérifier les ports exposés, la version utilisée et si elle contient des failles.
- Ex : #1002188 Potential HTTP Request Smuggling in nodejs (hackerone.com)
  Fails dans la version haproxy 1.5.3 version haproxy.cfg qui permet de bypasser HTTP DENY.

4. Attacks Vectors

- Botnets, Access restrictions bypass.

5. Détection d'attaque en temps réel / Zero Day Attack / Forensics

Surveiller les logs et les stats :

Parmi les techniques utilisées c'est lancer des connexions sur des pages lourdes qui consomment plus de ressources comme page de recherche, donc garder l'œil sur les stats c'est une très bonne pratique.
En cas d'attaque on doit être capable de faire investigation et comprendre pourquoi l'attaque est réussie pour pouvoir pallier le plutôt possible et minimiser les pertes. Donc les logs détaillés doivent être récupérés et gérés par un serveur externe, Haproxy se connecte facilement avec les outils externes soit pour les logs ou stats. Exemple d'outil supporté nativement : prometheus, rsyslog

Zero Day Attack :
Garder l'œil sur les mises à jour critiques de sécurité et mettre en place une stratégie pour garder toujours les serveurs en dernière version stable.

Email Alerts :

Hapee peut utiliser un serveur smtp pour envoyer des alertes en cas de problèmes, ça va servir surtout quand y'a des attaques en cours, en recevant l'alerte on peut réagir et surveiller avec plus d'attention afin d'éviter la catastrophe.

```
# email settings
email-alert mailers smtp_servers
email-alert from alerts@mycompany.com
email-alert to helpdesk@mycompany.com
email-alert level info
```

Notes

The Linux Virtual Server implements three defense strategies against some types of denial of service (DoS) attacks. The Linux Director creates an entry for each connection in order to keep its state, and each entry occupies 128 bytes effective memory. LVS's vulnerability to a DoS attack lies in the potential to increase the number entries as much as possible until the linux director runs out of memory. The three defense strategies against the attack are: Randomly drop some entries in the table. Drop 1/rate packets before forwarding them. And use secure tcp state transition table and short timeouts. The strategies are controlled by sysctl variables and corresponding entries in the /proc filesystem:

/proc/sys/net/ipv4/vs/drop_entry /proc/sys/net/ipv4/vs/drop_packet /proc/sys/net/ipv4/vs/secure_tcp

Valid values for each variable are 0 through to 3. The default value is 0, which disables the respective defense strategy. 1 and 2 are automatic modes - when there is no enough available memory, the respective strategy will be enabled and the variable is automatically set to 2, otherwise the strategy is disabled and the variable is set to 1. A value of 3 denotes that the respective strategy is always enabled. The available memory threshold and secure TCP timeouts can be tuned using the sysctl variables and corresponding entries in the /proc filesystem:

/proc/sys/net/ipv4/vs/amemthresh /proc/sys/net/ipv4/vs/timeout_*

Notes sur la sécurité avec LVS.

VI. La différence entre LVS et Haproxy

LVS se limite à la couche 4, On comprend vite que ce n'est pas juste de le comparer avec Haproxy qui traite aussi la couche 7. La plupart des fonctionnalités qui sont proposées par Haproxy et non supportées par LVS se basent sur la couche 7 puisque c'est là où il y a les données de la requête. La comparaison est faite sans considérer keepalived vue que haproxy se base sur keepalived pour quelques fonctionnalités, par contre Hapee a sa propre extension pour vrrp.

	LVS	Haproxy
Périmètre modèle OSI	Couche 4	Couche 4 et 7
Niveau d'exécution	Kernel	Userspace
Web Application Firewall (WAF)	✗	✓
API Gateway	✗	✓
SSL Termination	✗	✓
Realtime stats GUI	✗	✓
Virtual appliance / Hardware appliance	✗	✓
hitless reload/zeroDownTime	✗	✓
Runtime api	✗	✓
24/7 Support	Peut-être avec RedHat	✓
dedicated on-premise Engineer	Peut-être avec RedHat	✓

use cases

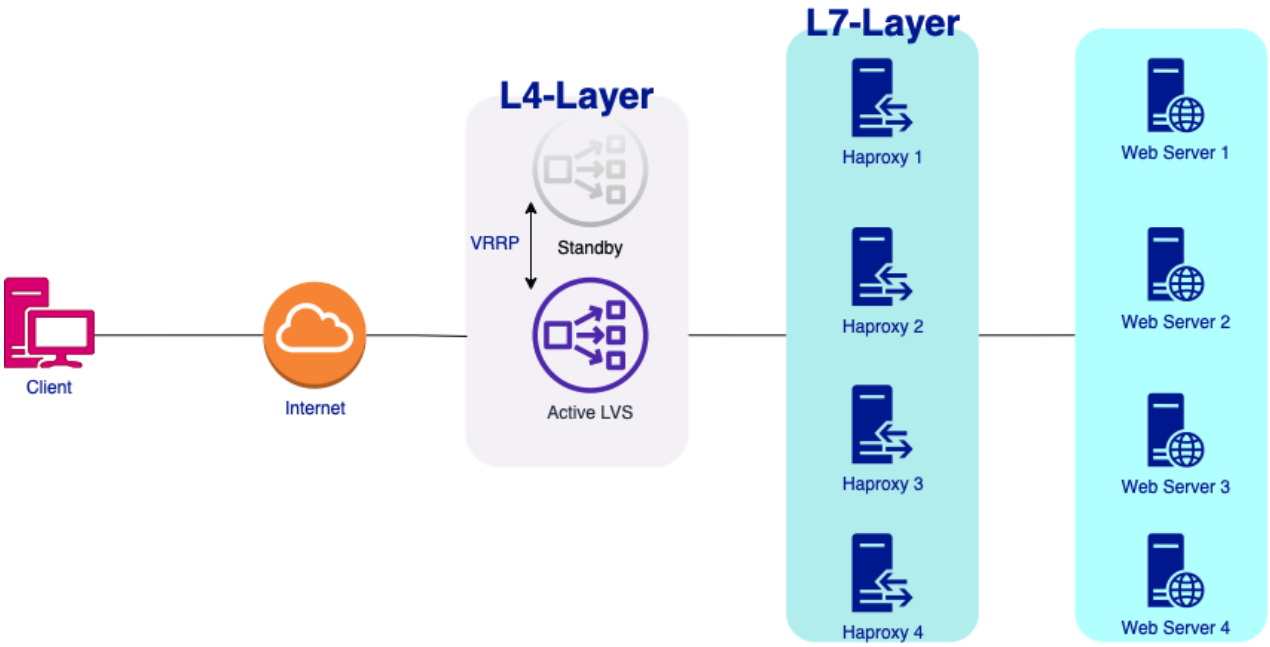
3 cas d'utilisation; (LVS + Haproxy / Haproxy only / LVS only)

Cas 1 : Trafic énorme et une application en microservice

On a une startup qui va lancer plusieurs campagnes publicitaires facebook ads avec un budget de 100K€/jour. D'après les statistiques des campagnes précédentes plus de 30 million visites par jour est à prévoir, 10 million vont faire des achats donc /cart le site e-commerce et les 30 million vont chercher dans la page d'accueil / et les pages de produits /articles/:id.

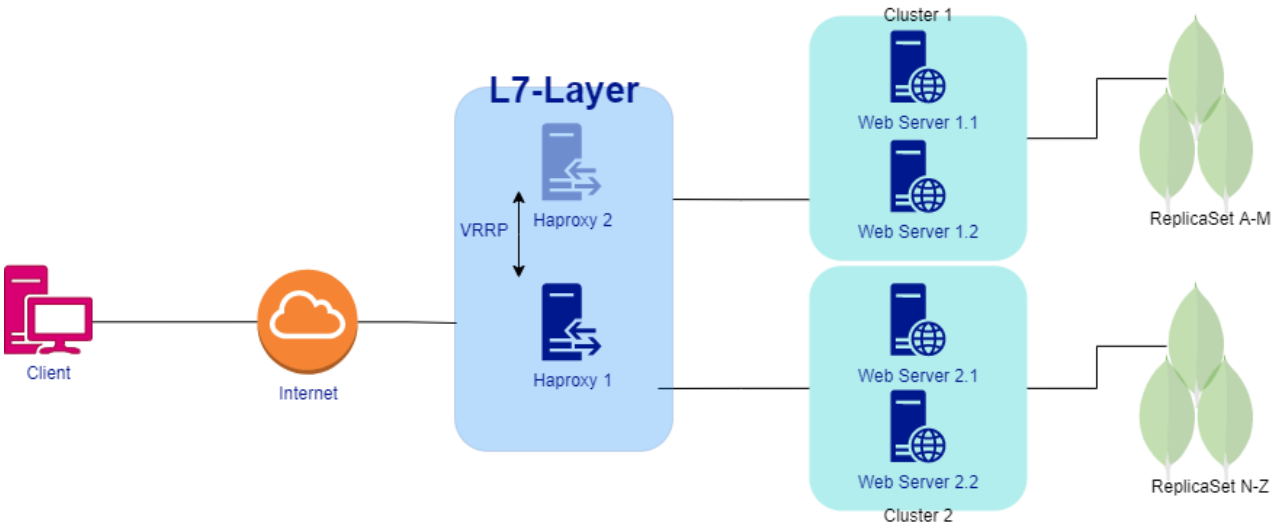
Vu que le trafic est énorme et qu'on a des api.

- on va mettre en place 4 reverse proxys de type Haproxy qui vont servir pour faire principalement un api gateway et équilibrage de charge, il seront tous active
- On va mettre en place un équilibreur de charge LVS pour faire la répartition pour les 4 haproxy et un LVS esclave pour la haute disponibilité.



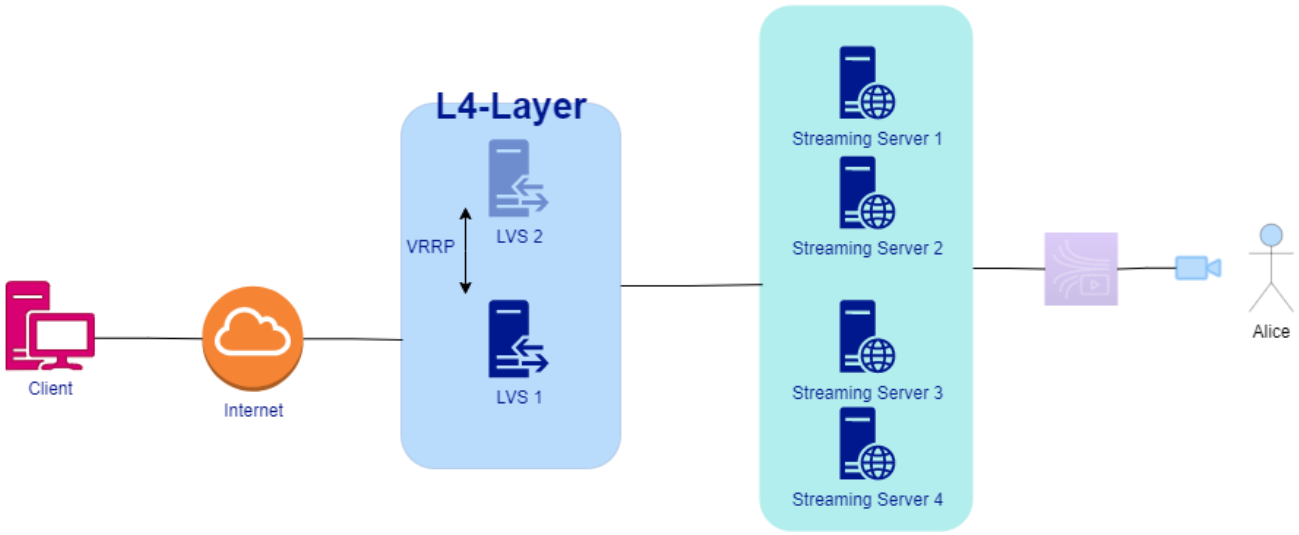
Cas 2 : Trafic très léger et une application qui divisé sur deux partie (exemple du dictionnaire)

L'application est répartie sur deux parties, la première qui va gérer les mots qui commencent par les alphabets de A jusqu'à M, puis la deuxième partie qui va gérer de N à Z.
dans ce cas Haproxy avec sa capacité de parsing des paquets HTTP peut faire l'affaire.



Cas 3 : Des serveur de streaming d'une conférence LIVE

Les backends ils ont les services utilisés pour streaming dans le port 4500.
On utilise LVS pour faire de l'équilibrage de charge entre les deux serveurs, utilisation de haproxy n'est pas du tout une bonne idée, streaming du vidéo demande beaucoup de ressources et utiliser haproxy ne va pas donner les performances vues que LVS est dans le kernel il est beaucoup plus rapide que haproxy qui est dans le user-space.



Conclusion :

Lorsque les professionnels de l'informatique ajoutent des load balancers à leur infrastructure, ils recherchent la possibilité de faire évoluer leurs sites Web et leurs services, d'obtenir une meilleure disponibilité et de passer des nuits plus reposantes en sachant que leurs **services critiques ne sont plus des points de défaillance uniques**. Avant longtemps, cependant, ils se rendent compte qu'avec un équilibreur de charge complet comme **HAProxy Enterprise**, ils peuvent ajouter une intelligence supplémentaire pour **inspecter le trafic** entrant et prendre des décisions à la volée. **restreindre l'accès** à divers points de terminaison, rediriger le trafic non-https vers https, détecter et **bloquer les robots** et les scanners malveillants et définir les conditions d'ajout d'en-têtes HTTP, de modification de l'URL ou de redirection de l'utilisateur.

Haproxy WEB Stats : (annexe 1)

hapee-lb version 2.4.0-1.0.0-268.477, released 2021/12/24

Statistics Report for pid 7145

> General process information

pid = 7145 (process #1, nbproc = 1, nbthread = 1)
uptime = 0d 19h40m46s
system limits: memmax = 645 MB; ulimit-n = 8836
maxsock = 8836; **maxconn** = 4400; **maxpipes** = 0
current conns = 5; current pipes = 0/0; conn rate = 3/sec; bit rate = 49.216 kbps
Running tasks: 0/29; idle = 100 %

- active UP
- active UP, going down
- active DOWN, going up
- active or backup DOWN
- active or backup DOWN for maintenance (MAINT)
- active or backup SOFT STOPPED for maintenance
- backup UP
- backup UP, going down
- backup DOWN, going up
- not checked

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

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

External resources:
- Primary site
- Updates (v2.4r1)
- Online manual

fe_main

	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				1	41	-	5	6	4 400	4 541			94 056	161 521	0	0	4 418					OPEN								

be_front_app

	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
webserver1	0	0	-	0	6		0	1	-	93	31	23m7s	58 655	51 083		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	1	0	0s	-
webserver2	0	0	-	0	9		0	1	-	48	31	1m18s	18 479	50 566		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	1	0	0s	-
Backend	0	0		0	11		0	1	440	141	62	1m18s	77 134	101 649	0	0		0	0	0	0	19h40m UP		2/2	2	0		0	0s	

be_flatteam

	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
webserver1	0	0	-	0	1		0	1	-	2	2	7h17m	1 161	508		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	2	0	0s	-
webserver2	0	0	-	0	1		0	1	-	2	2	6h4m	1 056	508		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	3	0	0s	-
Backend	0	0		0	1		0	1	440	4	4	6h4m	2 217	1 016	0	0		0	0	0	0	19h40m UP		2/2	2	0		0	0s	

be_hap_and_enter

	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
webserver1	0	0	-	0	1		0	1	-	2	2	6h4m	1 225	496		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	2	0	0s	-
webserver2	0	0	-	0	1		0	1	-	1	1	7h17m	518	248		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	3	0	0s	-
Backend	0	0		0	1		0	1	440	3	3	6h4m	1 743	744	0	0		0	0	0	0	19h40m UP		2/2	2	0		0	0s	

be_businessparteners

	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
webserver1	0	0	-	0	1		0	1	-	2	2	6h4m	1 209	480		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	2	0	0s	-
webserver2	0	0	-	0	1		0	1	-	1	1	7h17m	510	240		0		0	0	0	0	19h40m UP	L4OK in 0ms	1/1	Y	-	2	0	0s	-
Backend	0	0		0	1		0	1	440	3	3	6h4m	1 719	720	0	0		0	0	0	0	19h40m UP		2/2	2	0		0	0s	

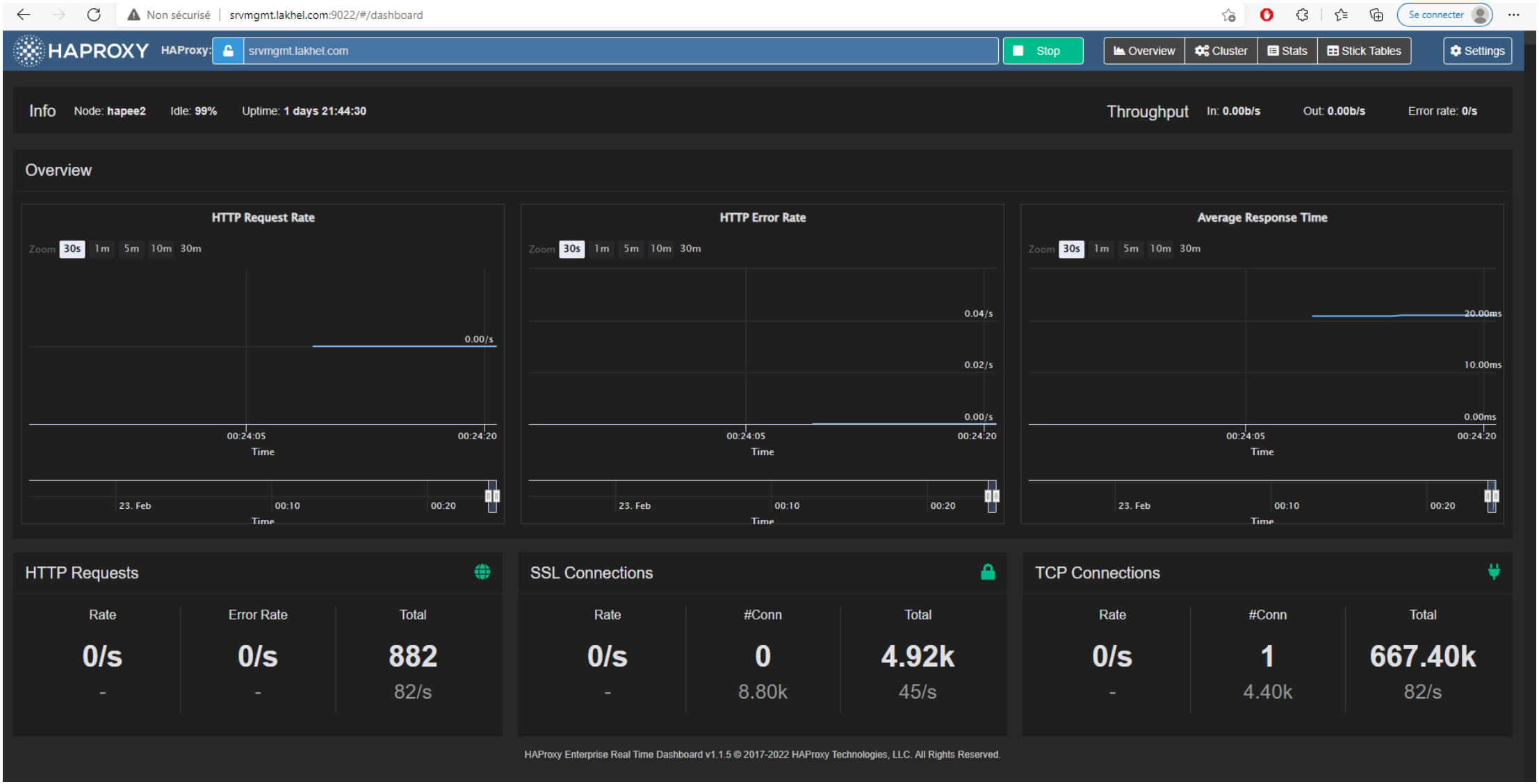
be_stats

	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
Backend	0	0		1	1		1	1	440	2	0	0s	712	51 216	0	0		0	0	0	0	19h40m UP		0/0	0	0		0		

dashboard

	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	6	-	0	6	4 400	26			125 156	8 336 977	0	0	5					OPEN								
dashboard	0	0	-	0	10		0	4	-	31	31	7h16m	16 083	2 588 615		0		0	0	0	0	*no check*		1/1	Y	-				-
Backend	0	0		0	10		0	4	440	31	31	7h15m	125 156	8 336 977	0	0		0	0	0	0	19h40m UP		1/1	1	0		0	0s	

Haproxy WEB GUI : (annexe 2)



Bibliographie :

Documentation officielle de Haproxy
https://www.haproxy.com/

- Version de test disponible sur : https:\\srv.lakhel.com
- L'interface graphique d'administration : https:\\srvmgmt.lakhel.com:9022
- Avec les identifiants : dashboard 1fqan7yi