HAProxy : de la découverte à l'expertise

Guide Complet & Support de Cours

Maîtrisez HAProxy, le load balancer de référence. Apprenez à configurer la haute disponibilité, la terminaison SSL, les ACL et l'optimisation des performances.

Catégorie : Proxy

Document généré le : 27/06/2026

Formateurs

Martin LEKPA
Poste : Tech Lead et formateur Observabilité
Site : https://martin.lekpa.fr
Email : martin@lekpa.fr

Pour aller plus loin : Nos formations Proxy

Dynatrace

Voir la formation en ligne

Elasticsearch

Voir la formation en ligne

Logstash

Voir la formation en ligne

Prometheus

Voir la formation en ligne

Opentelemetry

Voir la formation en ligne

Grafana

Voir la formation en ligne

Beats

Voir la formation en ligne

Alloy

Voir la formation en ligne

Questions Fréquentes

Pourquoi choisir HAProxy plutôt qu'un autre proxy ?
HAProxy est réputé pour sa rapidité extrême et sa fiabilité. Contrairement à Nginx qui est un serveur web polyvalent, HAProxy est un pur spécialiste du load balancing avec des fonctions de santé et de persistance beaucoup plus fines.

HAProxy gère-t-il le trafic HTTPS ?
Oui, parfaitement. Il peut soit faire du 'SSL Pass-through' (laisser le backend déchiffrer) soit de la 'Terminaison SSL' (HAProxy déchiffre et soulage les serveurs web).

Qu'est-ce qu'une ACL dans HAProxy ?
Une ACL (Access Control List) permet de prendre des décisions de routage intelligentes basées sur des critères comme l'URL, l'adresse IP source ou des en-têtes HTTP.

L'écosystème HAProxy : De l'Open Source à HAProxy One

Comprendre les différences, avantages et interconnexions entre HAProxy OSS, Enterprise, Fusion, Edge et la plateforme unifiée HAProxy One.

1. HAProxy OSS vs Enterprise : Les moteurs

Les briques de base pour le traitement du trafic (Data Plane).

HAProxy OSS (Community)

C'est le moteur open-source de référence. Il est idéal pour les architectures où la performance pure est recherchée et où les équipes disposent d'une forte expertise pour gérer la configuration et la sécurité manuellement.

Avantages :

- Gratuité totale.
- Performance brute identique à la version Enterprise.
- Transparence du code source.

Inconvénients :

- Pas de support commercial.
- Fonctionnalités avancées (WAF, protection bot) absentes ou complexes à intégrer.

Compatibilité :

C'est le socle technologique. Tout ce qui fonctionne en OSS fonctionne en Enterprise.

HAProxy Enterprise

Version stabilisée et optimisée pour la production critique. Elle inclut des modules propriétaires et un support 24/7.

Avantages :

- Modules exclusifs (WAF avancé, Dashboard global, protection contre les bots).
- Support technique expert.
- Cycles de vie Long Term Support (LTS).

Inconvénients :

- Coût de licence commerciale.

Compatibilité OSS :

Ascendante totale. Vous pouvez copier votre fichier haproxy.cfg d'une version OSS vers Enterprise sans modification. L'inverse n'est vrai que si vous n'utilisez pas de modules propriétaires (WAF, etc.).

2. HAProxy Fusion : Le Control Plane

Centraliser la gestion de plusieurs instances HAProxy.

Gestion centralisée

HAProxy Fusion est une interface de gestion moderne qui permet de piloter un parc entier d'instances HAProxy Enterprise. Elle résout le problème de la gestion silotée de la configuration.

Fonctionnalités clés :

- **Configuration unifiée :** Appliquer des changements sur plusieurs clusters en un clic.
- **Observabilité globale :** Visualiser l'état de santé de toute l'infrastructure sur un seul dashboard.
- **Automatisation :** API riche pour l'intégration CI/CD.

Compatibilité OSS :

Limitée. HAProxy Fusion est conçu pour piloter les fonctionnalités avancées de la version Enterprise. Il ne permet pas de gérer nativement les fichiers de configuration des instances OSS communautaires.

3. HAProxy Edge : La couche ADN

Sécurité et performance à l'échelle mondiale.

Application Delivery Network (ADN)

HAProxy Edge est un réseau global distribué (similaire à un CDN) qui place la protection et l'accélération au plus proche de l'utilisateur final.

Avantages :

- **Protection DDoS :** Absorption des attaques massives en bordure de réseau.
- **Chiffrement SSL global :** Terminaison SSL distribuée.
- **Performance :** Réduction de la latence via des points de présence (PoP) mondiaux.

Compatibilité OSS :

Totale en tant qu'origine. HAProxy Edge peut utiliser n'importe quel serveur HAProxy OSS comme backend ('Origin'). Il n'y a aucune dépendance logicielle entre votre instance locale et la couche Edge.

4. HAProxy One : La plateforme unifiée

L'écosystème complet réuni sous une seule interface.

La vision de plateforme

HAProxy One n'est pas un produit isolé, mais la **plateforme SaaS** qui unifie tous les services HAProxy : Enterprise, Fusion, Edge et même les appliances ALOHA.

Interconnexions :

- **OSS / Enterprise :** Le moteur de traitement (Data Plane).
- **Fusion :** Gère les moteurs locaux ou cloud.
- **Edge :** Fournit la porte d'entrée globale sécurisée.
- **ALOHA :** L'appliance matérielle ou virtuelle pour les centres de données.

En résumé : HAProxy One permet de passer d'un load balancing traditionnel à une stratégie de distribution d'application globale, sécurisée et managée.

Compatibilité OSS :

HAProxy One est l'ombrelle commerciale. La version OSS n'y est pas intégrée en tant que service managé, mais elle reste le composant interne que vous pouvez faire évoluer vers Enterprise pour rejoindre cet écosystème.

5. Comparaison et Verdict

Comment choisir selon votre besoin.

Matrice de choix

Produit	Cible	Usage principal	Lien OSS
OSS	DevOps / Experts	LB pur, Test.	-
Enterprise	Entreprises	Prod critique, WAF.	Config 100% compatible
Fusion	Ops / SRE	Gestion multi-clusters.	Non supporté
Edge	Global Apps	CDN, Anti-DDoS.	Agnostique (Backend)
HAProxy One	DSI	Plateforme unifiée.	Passerelle d'évolution

Le Load Balancing : Principes, Algorithmes et Cas d'usage

Découvrez le fonctionnement de la répartition de charge (Load Balancing), son rôle dans la haute disponibilité et le détail des algorithmes comme le Round Robin ou le Least Connections.

1. Qu'est-ce que le Load Balancing ?

Définition et rôle fondamental dans les architectures web.

Le répartiteur de charge

Le **Load Balancing** (ou répartition de charge) est un dispositif qui distribue le trafic réseau ou applicatif sur un ensemble de serveurs (le *pool de backends*). Son but est d'optimiser l'utilisation des ressources, de maximiser le débit, de réduire le temps de réponse et surtout d'éviter la surcharge d'un seul serveur.

Pourquoi est-ce indispensable ?

Sans répartition de charge, votre infrastructure possède un **SPOF** (Single Point of Failure). Si votre unique serveur tombe, le service s'arrête. Le load balancing permet la **Haute Disponibilité** (redondance) et la **Scalabilité horizontale** (ajout de serveurs pour absorber la croissance du trafic).

2. Les Algorithmes de répartition

Comment le répartiteur choisit-il le serveur de destination ?

Round Robin (Tourniquet)

C'est l'algorithme le plus simple : les requêtes sont envoyées aux serveurs de manière séquentielle. Il est idéal quand les serveurs ont des capacités identiques.

Info : C'est l'algorithme par défaut jusqu'à la version 3.2 de HAProxy.

- **Points positifs :** Simplicité extrême, aucun overhead, équité parfaite sur des serveurs identiques.
- **Points négatifs :** Inefficace si les serveurs ont des puissances différentes ou si certaines requêtes sont plus lourdes que d'autres.

Exemple de configuration HAProxy :

```
backend app_servers
    balance roundrobin
    server s1 192.168.1.10:80 check
    server s2 192.168.1.11:80 check
```

Weighted Round Robin

Chaque serveur possède un **poids** proportionnel à sa capacité de traitement. Un serveur plus puissant traitera une proportion plus importante de requêtes.

- **Points positifs :** Adapté aux parcs serveurs hétérogènes, comportement prévisible.
- **Points négatifs :** Nécessite une configuration manuelle des poids et ne s'adapte pas à la charge réelle instantanée.

Exemple de configuration HAProxy :

```
backend app_servers
    balance roundrobin
    server s1 192.168.1.10:80 check weight 100
    server s2 192.168.1.11:80 check weight 200
```

Least Connections (Moins de connexions)

Cet algorithme envoie la nouvelle requête au serveur qui a le **moins de connexions actives**. Très efficace pour les sessions à durées variables.

- **Points positifs :** Équilibre dynamique efficace, idéal pour les connexions persistantes (Bases de données, WebSockets).
- **Points négatifs :** Overhead léger pour le suivi des connexions, peut être trompeur si les requêtes n'ont pas la même complexité.

Exemple de configuration HAProxy :

```
backend app_servers
    balance leastconn
    server s1 192.168.1.10:80 check
    server s2 192.168.1.11:80 check
```

Weighted Least Connections

Combine l'algorithme des moins de connexions avec un système de **poids** pour favoriser les serveurs les plus robustes ayant peu de charge active.

- **Points positifs :** Le plus précis pour des infrastructures mixtes avec des sessions longues.
- **Points négatifs :** Plus gourmand en ressources de calcul pour l'équilibreur (overhead).

Exemple de configuration HAProxy :

```
backend app_servers
    balance wlc
    server s1 192.168.1.10:80 check weight 100
    server s2 192.168.1.11:80 check weight 200
```

Least Response Time

Dirige les requêtes vers le serveur ayant le **temps de réponse le plus court** et le moins de connexions actives. Idéal pour une réactivité maximale.

- **Points positifs :** Optimise directement l'expérience utilisateur, évite les serveurs ralentis.
- **Points négatifs :** Calcul intensif, risque d'instabilité si les temps de réponse varient brutalement.

Exemple de configuration HAProxy :

```
backend app_servers
    balance rtime # Requiert HAProxy 2.x+
    server s1 192.168.1.10:80 check
    server s2 192.168.1.11:80 check
```

IP Hash / Source Hash

L'adresse IP de l'utilisateur détermine le serveur de destination, garantissant la **persistance de session** (stickiness).

- **Points positifs :** Persistance sans gestion de cookies côté serveur.
- **Points négatifs :** Déséquilibre possible si beaucoup d'utilisateurs partagent la même IP (Proxy/NAT), redistribution totale si le nombre de serveurs change.

Exemple de configuration HAProxy :

```
backend app_servers
    balance source
    server s1 192.168.1.10:80 check
    server s2 192.168.1.11:80 check
```

Random (Aléatoire)

L'équilibreur de charge choisit un serveur au hasard dans le pool. Simple et statistiquement efficace sur des volumes de trafic massifs.

Info : C'est l'algorithme par défaut depuis la version 3.3 de HAProxy.

- **Points positifs :** Overhead nul, très performant pour les clusters à très grande échelle.
- **Points négatifs :** Aucune garantie d'équité on de petits volumes de requêtes.

Exemple de configuration HAProxy :

```
backend app_servers
    balance random
    server s1 192.168.1.10:80 check
    server s2 192.168.1.11:80 check
```

3. Outils et Exemples d'applications

Les solutions logicielles et cloud courantes.

Logiciels spécialisés

- **HAProxy :** Le standard de l'industrie pour la haute performance et la fiabilité. Découvrir la formation HAProxy.
- **Traefik :** Conçu pour le cloud-native, idéal pour Docker et Kubernetes. Découvrir la formation Traefik.
- **Nginx :** Serveur web polyvalent avec des fonctions de répartition robustes.

Cloud et Matériel

- **Cloud Load Balancers :** AWS ELB, Azure Load Balancer ou Google Cloud LB sont des services managés très populaires.
- **Hardware :** Des équipements physiques comme F5 BIG-IP, bien que moins fréquents aujourd'hui au profit du logiciel et du cloud.

Conclusion

Un élément critique pour la résilience.

L'évolution vers le Service Mesh

Le load balancing ne se limite plus à l'entrée du réseau. Dans les architectures microservices (Kubernetes), on utilise des *Service Meshes* (comme Istio ou Linkerd) pour gérer la répartition de charge directement entre les composants internes de l'application.

Anatomie de la configuration HAProxy

Comprendre la structure d'un fichier haproxy.cfg : sections Global, Defaults, Frontend et Backend, ainsi que la gestion des inclusions et du rechargement à chaud.

1. Les blocs de configuration

Le fichier haproxy.cfg est structuré en plusieurs sections (Global, Defaults, Frontend, Backend, Listen, Program) qui définissent le comportement global et le flux du trafic.

Section Global

La section **global** définit des paramètres au niveau du processus système (sécurité, performance, logs). Elle s'applique à l'ensemble de l'instance HAProxy.

```
global
    log /dev/log local0
    maxconn 4096
    user haproxy
    group haproxy
    daemon
```

Section Defaults

La section **defaults** permet d'éviter les répétitions. Les paramètres définis ici (mode, timeouts, logs) sont hérités par toutes les sections *frontend* et *backend* qui suivent.

```
defaults
    log global
    mode http
    timeout connect 5s
    timeout client 50s
    timeout server 50s
```

Section Frontend

Le **frontend** définit comment HAProxy reçoit les requêtes. On y précise l'adresse IP et le port d'écoute (bind), ainsi que les règles de routage (ACL) vers les backends.

```
frontend http-in
    bind *:80
    acl is_static path_beg /static
    use_backend static_servers if is_static
    default_backend app_servers
```

L'importance du default_backend : Cette directive définit le backend par défaut vers lequel HAProxy envoie les requêtes qui ne correspondent à aucune règle d'ACL (use_backend). Il est fortement recommandé d'en définir un systématiquement pour éviter que HAProxy ne rejette des requêtes avec une erreur 503 si aucune condition de routage n'est satisfaite.

Section Backend

Le **backend** définit le pool de serveurs de destination. La syntaxe commence par le mot-clé backend suivi d'un nom unique. Pour déclarer plusieurs backends (par exemple pour séparer le trafic API du trafic Web), il suffit de définir plusieurs blocs avec des noms distincts.

```
backend app_servers
    balance roundrobin
    server web1 10.0.0.1:80 check

backend api_servers
    balance leastconn
    server api1 10.0.0.10:80 check
```

Section Listen

La section **listen** est un raccourci qui combine les fonctionnalités d'un *frontend* et d'un *backend* dans un seul bloc. C'est idéal pour les services simples comme l'exposition des statistiques ou le proxying TCP pur.

> **Note :** Bien que pratique, ce rôle combiné n'est pas idéal pour les applications complexes composées de multiples domaines web et pools de serveurs. Dans ces cas, fusionner les fonctions peut complexifier inutilement la configuration. Nous recommandons alors de définir des sections *frontend* et *backend* séparées.

```
listen stats
    bind *:8404
    stats enable
    stats uri /monitor
    stats refresh 5s
```

Section Program

La section **program** permet à HAProxy de lancer et de surveiller des processus externes (disponible depuis la v2.2).

> **Note de dépréciation :** La fonctionnalité *program* est dépréciée depuis HAProxy 3.1 et sera supprimée en version 3.3. D'ici là, son comportement change lors des rechargements : le processus maître démarre le programme, mais c'est un processus travailleur (worker) qui l'exécute. Il est désormais recommandé d'utiliser des gestionnaires de processus dédiés comme *Systemd*, *SysVinit*, *Supervisord* ou *s6-overlays*.

```
program my-agent
    command /usr/local/bin/agent-script.sh
    user haproxy
    group haproxy
```

2. Simplifier avec les inclusions

Comment organiser une configuration complexe en plusieurs fichiers.

Utilisation de dossiers de configuration

Plutôt que d'avoir un fichier unique de plusieurs milliers de lignes, HAProxy permet de charger un répertoire entier. Cela est particulièrement utile dans les environnements automatisés (Ansible, Puppet) où chaque application peut avoir son propre fichier de configuration.

Lors du lancement du service, on utilise l'argument **-f** pointant vers un dossier. HAProxy concatène les fichiers trouvés par ordre alphabétique.

```
haproxy -f /etc/haproxy/haproxy.cfg -f /etc/haproxy/conf.d/
```

Exemple d'organisation en production

Dans une production avec de nombreux services, on adopte souvent une nomenclature numérique pour contrôler l'ordre de chargement :

```
/etc/haproxy/
??? haproxy.cfg       # Global & Defaults
??? conf.d/
    ??? 10-frontend-http.cfg
    ??? 11-frontend-https.cfg
    ??? 20-backend-ecommerce.cfg
    ??? 21-backend-blog.cfg
    ??? 99-stats.cfg
```

Cette structure permet d'ajouter ou supprimer une application simplement en gérant un fichier dans conf.d/, facilitant ainsi l'automatisation via CI/CD.

3. Mise à jour et Rechargement à chaud

Modifier la configuration sans interrompre le trafic.

Zéro Down-time Reload

HAProxy est conçu pour la haute disponibilité. Lorsque vous modifiez la configuration, vous pouvez recharger le service sans perdre de connexions actives. Le nouveau processus prend le relais sur les nouveaux sockets tandis que l'ancien termine de traiter les requêtes en cours.

```
# Vérifier la syntaxe avant de recharger
haproxy -c -f /etc/haproxy/haproxy.cfg

# Rechargement via systemd
systemctl reload haproxy
```

Rechargement sous Docker (SIGHUP)

Dans un environnement Docker, utiliser docker restart provoque une coupure nette car il tue le processus. Pour effectuer un rechargement 'hitless' (sans coupure) après avoir modifié votre fichier de configuration, envoyez le signal **SIGHUP** au conteneur. Le processus maître d'HAProxy créera un nouveau processus avec la nouvelle configuration tout en laissant l'ancien terminer les requêtes en cours.

```
# Recharger la configuration sans redémarrer le conteneur
docker kill -s HUP my-running-haproxy
```

La Runtime API

Pour des modifications ultra-dynamiques (comme changer le **poids d'un backend** ou passer un serveur en **maintenance**) sans même recharger le fichier, HAProxy propose une **Runtime API** accessible via une socket Unix. Des outils comme socat ou l'utilitaire haproxy-cli permettent d'interagir avec cette API.

```
# Exemple : désactiver un serveur via la socket
echo "disable server app_servers/web1" | socat stdio /var/run/haproxy.stat
```

```
# Exemple : ajouter un nouveau serveur à chaud avec haproxy-cli
haproxy-cli exec "add server app_servers/web3 10.0.0.3:80 check"
```

4. Sécurité : DoS, Filtrage IP et WAF

HAProxy est souvent la première ligne de défense de votre infrastructure. Il permet de filtrer le trafic et d'analyser sa dangerosité avant qu'il n'atteigne vos serveurs applicatifs.

Protection contre les attaques DoS

Grâce aux **stick-tables**, HAProxy peut suivre le comportement des IPs en temps réel pour bloquer les abus (trop de connexions simultanées).

```
frontend http-in
    bind *:80
    # Table de suivi : 100k entrées, expire après 30s d'inactivité
    stick-table type ip size 100k expire 30s store conn_cur
    # On suit l'IP source
    tcp-request connection track-sc0 src
    # On rejette si l'IP dépasse 15 connexions simultanées
    tcp-request connection reject if { sc0_conn_cur gt 15 }
```

Sécurité IP : Whitelisting et Blacklisting

Le filtrage IP permet de restreindre l'accès à des ressources sensibles ou de bloquer des attaquants connus.

- **Whitelisting (Liste blanche) :** On bloque tout sauf les IPs autorisées (stratégie *Default Deny*).
- **Blacklisting (Liste noire) :** On autorise tout sauf les IPs malveillantes.

```
# Exemple de Whitelisting pour les stats
acl network_allowed src 127.0.0.1 172.20.0.5
http-request deny if !network_allowed

# Exemple de Blacklisting via un fichier externe
acl bad_guys src -f /etc/haproxy/blacklist.lst
http-request deny if bad_guys
```

Web Application Firewall (WAF)

Le WAF inspecte le contenu des requêtes (headers, body, paramètres) pour bloquer les attaques de type Injection SQL ou XSS (OWASP Top 10). HAProxy peut intégrer un WAF nativement ou déléguer l'analyse via le mécanisme **SPOE** (Stream Processing Offload Engine) à un moteur comme ModSecurity.

```
frontend http-in
    # On délègue l'analyse à l'agent ModSecurity via SPOE
    filter spoe engine modsecurity config /etc/haproxy/modsec.conf

    # On bloque la requête (403) si l'agent identifie une menace
    http-request deny deny_status 403 if { var(txn.modsec.code) -m int gt 0 }
```

> **Le fichier modsec.conf :** Attention, ce fichier n'est pas celui qui contient vos règles de sécurité (WAF). C'est la configuration du *SPOE agent*. Il définit **comment** HAProxy communique avec le service externe : quels messages envoyer (headers, IP, etc.), les timeouts de réponse, et quel backend utiliser pour joindre l'agent ModSecurity.

5. Les ACL (Access Control Lists)

Les ACL sont le cœur de l'intelligence de HAProxy. Elles permettent de tester des conditions (URL, headers, IP) pour prendre des décisions de routage.

Syntaxe et Routage

Une ACL se définit par un nom, une méthode de test (fetch) et une valeur.

```
frontend http-in
    # Teste si le chemin commence par /api
    acl is_api path_beg /api
    # Teste si le header Host contient 'admin'
    acl is_admin hdr_dom(host) admin.mondomaine.com

    # Actions basées sur les ACLs
    use_backend api-servers if is_api
    http-request deny if is_admin !network_allowed
```

6. Observabilité et Supervision

HAProxy offre une visibilité native sur l'état de santé du trafic et des serveurs.

Dashboard de Statistiques

L'interface web intégrée permet de visualiser l'état des backends et le débit en temps réel.

```
listen stats
    bind *:8404
    stats enable
    stats uri /stats
    stats refresh 5s
    stats auth admin:password # Optionnel : authentification
```

Exposition Prometheus

HAProxy peut exposer ses métriques nativement pour être collectées par un serveur Prometheus.

```
frontend stats_and_metrics
    bind *:8404
    # On expose les métriques sur l'URI /metrics
    http-request use-service prometheus-exporter if { path /metrics }
```

Support OpenTelemetry (OTel)

Depuis la version 3.4, HAProxy peut envoyer des traces directement vers un collecteur OpenTelemetry pour un tracing distribué complet.

```
global
    otel-exporter my-collector
        endpoint otel-collector:4317

frontend http-in
    # Envoi de la trace au collecteur
    http-request otel-send-trace
```

7. resolvers (DNS dynamique)

La directive resolvers permet à HAProxy de résoudre dynamiquement les noms DNS et de suivre leurs changements (scaling, Kubernetes, Docker, service discovery).

Structure de base

La section **resolvers** définit un serveur DNS utilisé par HAProxy pour résoudre les noms d'hôtes dynamiques.

```
resolvers dns
    nameserver dns1 127.0.0.11:53
    resolve_retries 3
    timeout resolve 1s
    timeout retry 1s
    hold valid 10s
```

Utilisation avec un backend

Les resolvers permettent à HAProxy de résoudre dynamiquement des noms DNS et de les combiner avec **server-template** pour gérer automatiquement le scaling.

Pourquoi server-template ? Contrairement à une déclaration statique, server-template génère automatiquement plusieurs instances backend (web-1, web-2, web-3) sans configuration manuelle, indispensable en Docker/Kubernetes.

```
resolvers docker_dns
    nameserver docker 127.0.0.11:53
    hold valid 5s

backend app
    server-template app 1-5 app.local:80 check resolvers docker_dns resolve-prefer ipv4
```

Docker / Service Discovery

Dans Docker, le DNS interne (127.0.0.11) permet de résoudre dynamiquement les conteneurs créés par scaling.

Pourquoi server-template ? Docker ne garantit ni IP fixe ni nom stable. server-template permet à HAProxy de suivre automatiquement les conteneurs sans modifier la configuration.

```
resolvers docker
    nameserver docker 127.0.0.11:53
    hold valid 10s

backend app
    server-template web 1-10 web.local:80 check resolvers docker resolve-prefer ipv4
```

Kubernetes / Microservices

Dans Kubernetes, les pods changent dynamiquement selon le scaling du cluster.

Pourquoi server-template ? Les endpoints évoluent en permanence. server-template permet d'adapter automatiquement HAProxy sans reload ni configuration manuelle.

```
resolvers kube_dns
```

```
    nameserver kube 10.96.0.10:53
    hold valid 10s

backend api
    server-template api 1-20 api.default.svc.cluster.local:80 check resolvers kube_dns resolve-
prefer ipv4
```

Comportement de cache DNS

Les directives hold permettent de contrôler la durée de mise en cache des réponses DNS.

- **hold valid** : durée de validité d'une résolution réussie
- **hold nx** : durée pour les réponses NXDOMAIN
- **hold refused** : durée pour les refus DNS

```
resolvers dns
    nameserver dns1 8.8.8.8:53
    hold valid 30s
    hold nx 5s
    hold refused 5s
```

Conclusion

La puissance par la modularité.

Une structure robuste

Maîtriser l'anatomie de la configuration HAProxy est la première étape pour construire des infrastructures résilientes. La séparation nette entre l'écoute (frontend) et le traitement (backend) offre une flexibilité que peu d'autres solutions égalent.

Bonnes Pratiques HAProxy : Optimisation, Sécurité et Haute Disponibilité

Découvrez les meilleures pratiques pour configurer HAProxy afin d'assurer performance, sécurité et résilience de vos applications.

1. Optimisation des Performances

Assurer que HAProxy utilise au mieux les ressources disponibles pour traiter le trafic.

Paramètres globaux et réutilisation

Pour tirer le meilleur parti de HAProxy, certains paramètres globaux sont indispensables pour limiter la charge sur les backends et assurer la résilience réseau.

- **Maxconn :** Définit la limite globale de connexions (ex: 10 000).
  Avantage : Protège le système contre l'épuisement de la RAM et des descripteurs de fichiers.
  Risque : Si la valeur est trop basse, HAProxy rejettera des clients légitimes lors des pics de trafic.
- **Keep-alive :** Activez option http-keep-alive.
  Avantage : Réduit la latence en réutilisant les connexions TCP existantes (évite le 3-way handshake répétitif).
  Risque : Peut maintenir trop de connexions ouvertes sur des serveurs backend limités si le timeout est trop long.
- **Retries & Redispatch :**
  Avantage : Améliore la disponibilité perçue par l'utilisateur. Si un serveur tombe pile au moment de la requête, HAProxy réessaie sur un autre.
  Risque : Peut aggraver la surcharge d'un cluster si les serveurs tombent à cause d'une saturation (effet cascade).

```
global
    maxconn 10000
    log stdout format raw local0

defaults
    option http-keep-alive
    retries 3
    option redispatch
```

Optimisation des Timeouts

Une erreur fréquente est de conserver les valeurs par défaut. Il faut définir des timeouts explicites.

- **Standards :** 5s pour la connexion, 30s pour le client et le serveur.
  Avantage : Libère rapidement les ressources bloquées par des clients ou des serveurs léthargiques.
  Risque : Un timeout serveur trop court coupera des transferts de fichiers volumineux ou des exports longs.
- **Protection Slowloris :** Gardez un timeout http-request court (5s).
  Avantage : Empêche un attaquant de saturer les connexions en envoyant les headers très lentement.

```
defaults
    timeout connect 5s
    timeout client 30s
    timeout server 30s
```

```
backend slow_api
    timeout server 2m
```

Choisir le bon algorithme

Le choix de l'algorithme impacte directement l'utilisation de vos ressources et la qualité de l'expérience utilisateur. Pour approfondir les mécanismes théoriques, consultez notre article : Le Load Balancing : Principes, Algorithmes et Cas d'usage.

- **Round Robin :**
  Avantage : Simplicité, équité parfaite si les requêtes sont homogènes.
  Risque : Inefficace si certains serveurs sont plus puissants que d'autres ou si certaines requêtes sont beaucoup plus lourdes.
- **Least Connections :**
  Avantage : Algorithme le plus intelligent pour l'APM ; il évite d'envoyer du trafic vers un serveur déjà en train de peiner.
  Risque : Moins performant sur des flux très courts car le calcul du nombre de connexions ajoute un léger overhead.
- **Source Hash :**
  Avantage : Persistance sans injecter de cookies.
  Risque : Déséquilibre si de nombreux utilisateurs sortent derrière un même proxy/NAT d'entreprise (une seule IP pour des milliers de clients).

2. Sécurité et Durcissement (Hardening)

Protéger vos applications contre les menaces courantes via HAProxy.

Gestion du TLS/SSL

Le chiffrement TLS est fondamental pour la sécurité des communications. HAProxy permet de centraliser la terminaison SSL.

- **HTTP/2 via ALPN :**
  Avantage : Accélère le chargement des pages via le multiplexage.
  Risque : Consomme un peu plus de CPU sur HAProxy pour la gestion des flux.
- **HSTS :**
  Avantage : Empêche les attaques de downgrade SSL (Man-in-the-Middle).
  Risque : Si vous perdez vos certificats ou souhaitez repasser en HTTP, les navigateurs bloqueront l'accès jusqu'à expiration du max-age (souvent 1 an).

Désactivation des protocoles obsolètes

Pour protéger vos flux contre les vulnérabilités protocolaires (POODLE, BEAST), il est impératif de limiter les protocoles acceptés.

- **TLS 1.2 / 1.3 uniquement :**
  Avantage : Niveau de sécurité maximal conforme aux standards PCI-DSS / RGPD.
  Risque : Incompatibilité avec de très vieux clients (vieux terminaux de paiement, vieux navigateurs Android/IE).

```
global
    ssl-default-bind-options no-sslv3 no-tlsv10 no-tlsv11
```

```
```

Protection contre les Attaques (DDoS, Slowloris)

HAProxy peut servir de première ligne de défense contre les attaques de type DDoS ou Slowloris via des stick-tables.

- **Rate Limiting :**
  Avantage : Bloque automatiquement les IP qui saturent le service avant qu'elles n'atteignent vos serveurs.
  Risque : Risque de faux positifs (ex: une école ou une entreprise sortant avec une seule IP publique sera bloquée si trop d'utilisateurs naviguent simultanément).

Bonnes pratiques WAF (Web Application Firewall)

HAProxy peut inspecter le contenu des requêtes pour bloquer les tentatives d'injection SQL ou XSS.

- **Filtrage de méthodes :**
  Avantage : Réduit la surface d'attaque en interdisant des méthodes comme PUT ou DELETE si elles ne sont pas utilisées.
  Risque : Peut casser certaines fonctionnalités d'API modernes si la configuration n'est pas mise à jour.
- **Inspection de payload :**
  Avantage : Bloque les attaques connues au niveau de la couche 7.
  Risque : Impact significatif sur le CPU si les règles d'inspection (Regex) sont trop complexes ou nombreuses.

Bonnes pratiques ACL

Les ACL sont le moteur de décision. Une gestion rigoureuse évite les erreurs de routage.

- **Nomenclature :**
  Avantage : Facilite la lecture des logs et le troubleshooting.
  Risque : Des noms d'ACL flous (ex: acl test) rendent la configuration illisible et dangereuse.
- **Externalisation (Maps) :**
  Avantage : Permet de mettre à jour des listes (ex: domaines autorisés) sans modifier le fichier principal et parfois sans rechargement via la Runtime API.
  Risque : Nécessite une gestion rigoureuse des fichiers externes pour éviter les fichiers manquants au démarrage.

3. Haute Disponibilité et Résilience

Assurer la continuité de service et la robustesse de votre infrastructure.

Configuration des Health Checks

Les health checks évitent d'envoyer des clients vers un serveur 'zombie' (port ouvert mais application crashée).

- **Check Applicatif (HTTP) :**
  Avantage : Garantit que l'application répond réellement (ex: base de données connectée).
  Risque : Si l'endpoint de santé est trop complexe, le check lui-même peut ralentir le backend (évitez les calculs lourds dans /health).

```
backend app_servers
```

```
    option httpchk GET /health
    http-check expect status 200
    server s1 10.0.0.1:80 check inter 2s fall 3 rise 2
```

Gestion des Serveurs de Backup et Maintenance

Prévoyez toujours une issue de secours.

- **Serveur de Backup :**
  Avantage : Évite la page d'erreur 503 totale. Peut servir à afficher un site statique 'dégradé'.
  Risque : Si le backup n'est jamais testé, il peut échouer au moment où on en a besoin.
- **Page de Maintenance :**
  Avantage : Communication propre vers les utilisateurs lors d'incidents majeurs.
  Risque : Si elle est mal configurée, elle peut empêcher les robots d'indexation (SEO) de comprendre que le site reviendra.

4. Organisation et Modularité

Structurer sa configuration pour faciliter l'automatisation et la maintenance.

Séparation des blocs et fichiers multiples

Dans une infrastructure complexe, avoir un seul fichier de 3000 lignes est une mauvaise pratique majeure.

- **Séparation logique :** Les sections **Global** et **Defaults** doivent être isolées du routage métier.
- **Inclusions :** Utilisez des dossiers (ex: conf.d/) pour charger vos backends par application.

Avantage : Permet de déployer une nouvelle application (nouveau fichier) via CI/CD sans risquer de corrompre les ACL d'un autre service.
Risque : HAProxy charge les fichiers par ordre alphabétique. L'ordre Global > Defaults > Frontend > Backend doit être respecté.

```
# Lancer HAProxy en chargeant un répertoire
haproxy -f /etc/haproxy/haproxy.cfg -f /etc/haproxy/conf.d/
```

5. Maintenance Opérationnelle et Haute Disponibilité

Gérer les mises à jour et éliminer les points de défaillance uniques (SPOF).

Reload à chaud (Zero Downtime)

Pour appliquer une modification sans déconnecter les utilisateurs, il ne faut jamais faire un restart du service. Vous devez faire un reload:

```
systemctl reload haproxy
```

ou dans le monde Docker par exemple :

```
sudo docker compose kill -s HUP haproxy
```

- **Master-Worker Mode :** Utilisez le mode moderne (standard sous Systemd) qui permet de transférer les

sockets d'écoute aux nouveaux processus.
- **Avantage :** Les sessions longues (WebSockets, transferts) se terminent normalement sur l'ancien processus pendant que le nouveau prend les nouveaux appels.
  Risque : En cas d'erreur de syntaxe, le reload échoue. **Toujours** tester avant : haproxy -c -f /etc/haproxy/haproxy.cfg.

Haute Disponibilité Réseau (Keepalived)

Si HAProxy tombe, votre application est injoignable. La solution est de mettre en place un cluster HAProxy Redondant.

- **VIP (Virtual IP) :** Une adresse IP flottante est gérée par **Keepalived** via le protocole VRRP.
- **Fonctionnement :** Si le HAProxy 'Master' ne répond plus, la VIP bascule instantanément sur le 'Backup'.

Avantage : Élimine le SPOF matériel ou logiciel.
Risque : Le 'Split-Brain' (les deux nœuds croient être Master) peut arriver si le lien réseau entre les deux est instable, provoquant des conflits d'IP.

Logging et Monitoring

Utilisez des logs détaillés pour le debugging applicatif.

```
frontend http_front
    # Capture le Host et le User-Agent pour l'analyse
    capture request header Host len 64
    capture request header User-Agent len 128
```

HAProxy : Ajout et suppression dynamique de backends

Découvrez comment utiliser la Runtime API de HAProxy pour créer, configurer et supprimer des backends et des serveurs à chaud, sans aucun redémarrage.

1. Introduction à la gestion dynamique

Traditionnellement, modifier la structure des backends nécessitait un reload du fichier de configuration. La Runtime API permet désormais d'effectuer ces changements en mémoire.

Attention : L'ajout et la suppression dynamique de backends et serveurs via la Runtime API nécessite **HAProxy 2.8** ou une version supérieure.

Configuration de base

Pour utiliser ces commandes, votre instance HAProxy doit exposer une socket d'administration avec un niveau **admin** dans la section **global** de votre configuration :

En classique :

```
stats socket /run/haproxy/admin.sock user haproxy group haproxy mode 660 level admin
```

et/ou via une IP (accès distant mais attention aux risques de sécurité si vous mettez une IP autre que localhost) :

```
stats socket ipv4@127.0.0.1:9999  level admin  expose-fd listeners
```

2. Lexique des commandes clés

Pour manipuler l'infrastructure à chaud, il est essentiel de comprendre le rôle de chaque verbe de la Runtime API.

help

La commande help permet d'afficher l'ensemble des commandes possibles. La syntaxe est donc la suivante, seul le endpoint change en fonction du type d'activation que vous avez fait:

Le chemin à utiliser avec socat peut changer en fonction de votre configuration

- Classique :

```
echo 'help' | socat stdio /var/run/haproxy.stat
```

- IP/PORT :

```
echo 'help' | socat stdio tcp4-connect:127.0.0.1:9999
```

add (backend / server)

La commande add permet de créer un nouvel objet (backend ou serveur) en mémoire. Un backend peut être créé

ex nihilo ou en héritant des paramètres d'une section defaults existante (indispensable pour les timeouts).

> **Attention :** Il est possible d'avoir des problèmes de résolution quand vous ajoutez un serveur. Cela peut se voir en lançant cette commande dans laquelle vous verez qu'un serveur n'a pas d'IP :
>
> ```
> $ echo "show servers conn default" | socat stdio /run/haproxy/admin.sock
> # bkname/svname bkid/svid addr port - purge_delay served used_cur used_max need_est idle
> _sess unsafe_nb safe_nb idle_lim idle_cur idle_per_thr[10]
> default/web1 4/1 192.168.48.3 80 - 5000 0 0 1 1 0 0 0 -1 0 0 0 0 0 0 0 0 0 0
> default/web2 4/2 192.168.48.2 80 - 5000 0 0 1 1 0 0 0 -1 0 0 0 0 0 0 0 0 0 0
> default/web3 4/3 - 80 - 5000 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0
> ```
>
> Et vous pouvez éviter cela en précalculant l'IP :
>
> ```
> IP=$(getent hosts web3 | awk '{print $1}'); echo "add server default/web3 $IP:80 check"
> | socat stdio /run/haproxy/admin.sock
> ```

enable (server / health)

Par défaut, un serveur ajouté dynamiquement est en mode maintenance (maint). enable server le rend opérationnel. enable health active spécifiquement les tests de santé (health checks) pour ce serveur.

publish (backend)

C'est une étape de validation. Tant qu'un backend n'est pas **publié**, il est invisible pour les frontends, même s'il existe en mémoire. Cela permet de configurer entièrement un pool avant de l'exposer au trafic.

map (add / show / del)

Les **Maps** sont des tables de correspondance (ex: URL -> Backend) stockées en mémoire. add map ajoute une règle de routage, del map la supprime, et show map permet de déboguer le contenu de la table sans recharger HAProxy.

3. Ajouter un backend et un serveur

Voici le workflow pour créer un nouveau pool de serveurs et le rendre disponible.

Création et activation

Le processus se décompose en 4 étapes clés : l'ajout du backend, l'ajout du serveur, l'activation de la santé et la publication.

```
# 1. Ajouter le backend en héritant d'une section defaults
echo "add backend test-backend from mydefaults mode http" | socat stdio unix-
connect:/run/haproxy/admin.sock

# 2. Ajouter un serveur au backend
echo "add server test-backend/server1 127.0.0.1:3000 check" | socat stdio unix-
connect:/run/haproxy/admin.sock

# 3. Activer le serveur et ses health checks
echo "enable server test-backend/server1" | socat stdio unix-connect:/run/haproxy/admin.sock
echo "enable health test-backend/server1" | socat stdio unix-connect:/run/haproxy/admin.sock
```

```
# 4. Publier le backend pour le rendre utilisable par les frontends
echo "publish backend test-backend" | socat stdio unix-connect:/run/haproxy/admin.sock
```

Routage dynamique via Map files

Pour diriger du trafic vers ce nouveau backend sans modifier le frontend, utilisez les fichiers de mapping en mémoire :

```
# Associer le chemin /test au nouveau backend
echo "add map virt@paths.map /test test-backend" | socat stdio unix-
connect:/run/haproxy/admin.sock
```

4. Supprimer un backend proprement

La suppression nécessite de respecter un ordre précis pour ne pas rompre les connexions en cours.

Procédure de retrait

Il est impératif de passer le serveur en maintenance et d'attendre qu'il soit 'removable' avant la suppression physique.

```
# 1. Passer le serveur en maintenance
echo "set server test-backend/server1 state maint" | socat stdio unix-
connect:/run/haproxy/admin.sock

# 2. Attendre et supprimer le serveur
echo "wait 2s srv-removable test-backend/server1; del server test-
backend/server1" | socat stdio unix-connect:/run/haproxy/admin.sock

# 3. Dé-publier et supprimer le backend
echo "unpublish backend test-backend" | socat stdio unix-connect:/run/haproxy/admin.sock
echo "wait 2s be-removable test-backend; del backend test-backend" | socat stdio unix-
connect:/run/haproxy/admin.sock
```

5. Points d'attention importants

Quelques règles de comportement du moteur HAProxy à connaître.

Publication et Force Switch

Un backend référencé par un use_backend sera ignoré s'il n'est pas publié. Pour forcer l'utilisation même s'il est 'unpublished', utilisez la directive force-be-switch.

Gestion de la mémoire

Pour permettre l'ajout dynamique, HAProxy conserve les sections *defaults* en mémoire. Si vous n'utilisez pas de backends dynamiques, vous pouvez libérer cette mémoire avec la directive globale suivante :

```
global
    tune.defaults.purge on
```

Conclusion

Vers une automatisation totale

Cette approche est idéale pour coupler HAProxy à des outils comme Consul ou des contrôleurs personnalisés, permettant de scaler votre infrastructure sans jamais générer de micro-coupures liées aux rechargements de processus.

HAProxy : Guide détaillé des directives de configuration

Une exploration complète des directives essentielles de HAProxy : de bind à http-check, apprenez à configurer votre load balancer avec précision.

1. acl

Listes de contrôle d'accès pour les tests logiques. Disponible depuis la version 1.1.

Structure

La syntaxe générale est : acl <nom> <critère> <valeurs>. Le nom permet de réutiliser le test plus tard.

is_api

Utilise path_beg pour tester le début du chemin de l'URL.

```
acl is_api path_beg /api
```

Cette ligne crée une règle nommée 'is_api' qui est validée si l'URL commence par /api.

is_static

Utilise path_end pour identifier les fichiers par leurs extensions.

```
acl is_static path_end .jpg .png
```

Cette commande définit la règle 'is_static' pour filtrer les requêtes vers des fichiers d'images.

2. balance

Définit l'algorithme de répartition de charge. Disponible depuis la version 1.0.

Structure

La syntaxe est : balance <algorithme>. Elle se place généralement dans une section backend.

roundrobin

L'algorithme par défaut pour une distribution égale.

```
balance roundrobin
```

Chaque nouveau client est envoyé au serveur suivant dans la liste de manière circulaire.

leastconn

Optimisé pour les traitements longs.

```
balance leastconn
```

HAProxy choisit le serveur ayant le plus petit nombre de sessions ouvertes au moment de la requête.

3. bind

Définit les adresses et ports d'écoute du frontend. Disponible depuis la version 1.0.

Structure

La syntaxe est : bind [<adresse>]:<port_range> [options].

Interface et Port

Spécifie le point d'entrée réseau.

```
bind *:80
```

HAProxy acceptera les connexions sur toutes les interfaces réseau disponibles sur le port 80.

SSL/TLS

Active le chiffrement sur le point d'entrée.

```
bind *:443 ssl crt /etc/ssl/certs/site.pem
```

Écoute sur le port 443 en utilisant le fichier certificat PEM spécifié pour la terminaison SSL.

4. http-check

Configuration avancée des Health Checks applicatifs. Disponible depuis la version 2.2.

Structure

Utilisé en complément de option httpchk pour personnaliser le test de santé.

send

Définit la requête à envoyer au serveur.

```
http-check send meth GET uri /health
```

HAProxy enverra une requête HTTP GET sur /health pour tester la vitalité de l'application.

expect

Vérifie la réponse du serveur.

```
http-check expect status 200
```

Le serveur n'est considéré comme 'UP' que s'il répond avec un code d'état 200.

5. http-request

Manipule ou filtre les requêtes HTTP entrantes. Disponible depuis la version 1.5.

Structure

La syntaxe est : http-request <action> [if/unless <condition>].

deny

Bloque l'accès selon une condition.

```
http-request deny if { src 192.168.1.50 }
```

Rejette immédiatement la requête si l'IP source est 192.168.1.50.

set-header

Ajoute ou modifie des entêtes HTTP.

```
http-request set-header X-Proxy-By HAProxy
```

Insère l'entête 'X-Proxy-By' dans la requête transmise au serveur backend.

6. mode

Définit le protocole de fonctionnement de l'instance. Disponible depuis la version 1.0.

Structure

La syntaxe est simple : mode <http|tcp>.

http

Analyse de la couche 7.

```
mode http
```

Permet à HAProxy de comprendre et modifier le contenu des paquets HTTP.

tcp

Analyse de la couche 4.

```
mode tcp
```

HAProxy agit comme un relais de flux binaire sans lire le contenu protocolaire applicatif.

7. server

Déclare un serveur backend et ses options de santé. Disponible depuis la version 1.0.

Structure

La syntaxe est : server <nom> <adresse>[:port] [options].

Nom et Adresse

Déclaration de base d'un serveur.

```
server web1 10.0.0.1:80
```

Définit un serveur nommé 'web1' localisé à l'adresse IP 10.0.0.1 sur le port 80.

check

Activation du monitoring.

```
server web1 10.0.0.1:80 check
```

Le mot-clé 'check' active les tests de santé réguliers pour s'assurer que le serveur est opérationnel.

8. stats

Configure l'interface de monitoring. Disponible depuis la version 1.1.

Structure

S'utilise avec plusieurs options pour définir le comportement de la page de rapport.

enable / uri

Activation et point d'accès.

```
stats enable
stats uri /haproxy?stats
```

Active la page de statistiques et la rend accessible via l'URI spécifiée.

auth

Sécurisation par mot de passe.

```
stats auth admin:rousseltm
```

Restreint l'accès aux statistiques à l'utilisateur 'admin' avec le mot de passe 'rousseltm'.

9. timeout

Gère les délais d'expiration des connexions. Disponible depuis la version 1.2.

Structure

La syntaxe est : timeout <type> <durée>. Les durées peuvent être en ms, s, m, h.

connect

```
timeout connect 5s
```

Définit le temps maximum d'attente pour que HAProxy établisse une connexion TCP avec le serveur backend.

client

```
timeout client 30s
```

Définit le temps maximum d'inactivité autorisé côté client une fois la connexion établie.

server

```
timeout server 30s
```

Définit le temps maximum d'attente pour que le serveur backend traite la requête et commence à répondre.

10. use_backend

Route dynamiquement vers un groupe de serveurs. Disponible depuis la version 1.1.

Structure

Définit le choix du backend. Syntaxe : use_backend <nom> [if/unless <condition>].

if

Routage basé sur une ACL.

```
use_backend api_cluster if is_api
```

Envoie la requête au pool de serveurs 'api_cluster' si l'ACL 'is_api' est vérifiée.

11. server-template

Permet de générer dynamiquement des serveurs backend sans les déclarer un par un. Très utilisé avec Docker, DNS et Kubernetes.

Structure

La syntaxe est : server-template <base-name> <start-end> <dns-template> [options].

base-name : nom de base utilisé pour générer les serveurs (ex : web ? web1, web2, web3).

start-end : nombre d'éléments à sélectionner. Si c'est un entier, correspond au nombre d'instances souhaitées dans l'ordre des résultats de la résolution. Si plage numérique, correspond à la plage souhaitée (ex : 2-5 ? 4 serveurs mais du 2e au 5e).

dns-template : nom DNS ou template résolu dynamiquement par HAProxy (ex : web.service.local ou backend.internal).

options : paramètres additionnels comme check, resolvers, resolve-prefer pour gérer santé et résolution DNS.

Exemple Docker / DNS

```
server-template web 1-5 web.service.local:80 check resolvers docker
```

Permet à HAProxy de créer automatiquement les serveurs web1 à web5 et en déduisant leurs noms/IP via résolution DNS.

12. option

Active des comportements globaux ou spécifiques dans frontend ou backend.

Structure

La directive option active des fonctionnalités additionnelles dans HAProxy.

Exemples courants

```
option forwardfor
option httpchk
option redispatch
option httplog
option dontlognull
```

forwardfor

Ajoute automatiquement l'IP client dans X-Forwarded-For.

httpchk

Active les health checks HTTP sur les serveurs backend.

11. Autres directives

Paramètres globaux et comportementaux. Disponibles depuis la version 1.0.

Structure

Ensemble de paramètres de configuration système et de comportements par défaut.

log

```
log global
```

Indique à HAProxy d'utiliser la configuration de journalisation définie dans la section 'global'.

maxconn

```
maxconn 4096
```

Limite le nombre total de connexions simultanées pour éviter la saturation des ressources système.

user / group

```
user haproxy
group haproxy
```

Définit les privilèges système sous lesquels le processus HAProxy doit s'exécuter une fois démarré.

option forwardfor

```
option forwardfor
```

Ajoute l'entête HTTP 'X-Forwarded-For' contenant l'IP réelle du client avant de transmettre la requête au serveur.

retries

```
retries 3
```

Définit le nombre de tentatives de reconnexion au serveur backend en cas d'échec de connexion TCP.

Health Checks et Haute Disponibilité

Comment HAProxy détecte les pannes et gère le basculement automatique vers les serveurs sains.

1. Le check TCP basique

Vérifier simplement si le port répond.

Option check

Par défaut, HAProxy tente d'établir une connexion TCP sur le port du serveur. S'il ne répond pas, le serveur est marqué comme DOWN.

```
backend app_servers
    server s1 10.0.0.1:80 check inter 2s fall 3 rise 2
```

Ici : test toutes les 2s, retrait après 3 échecs, remise en service après 2 succès.

2. Le check HTTP (Layer 7)

Vérifier que l'application renvoie un contenu spécifique.

Utilisation de http-check

Un port peut être ouvert mais l'application peut renvoyer une erreur 500. Le check HTTP est plus précis.

```
backend app_servers
    option httpchk GET /health
    http-check expect status 200
    server s1 10.0.0.1:80 check
```

3. Serveurs de Backup

Gérer un serveur de secours uniquement si tous les autres tombent.

Le mot-clé backup

Idéal pour afficher une page de maintenance ou utiliser un serveur de secours moins puissant.

```
backend app_servers
    server s1 10.0.0.1:80 check
    server s2 10.0.0.2:80 check
    server s_backup 10.0.0.99:80 check backup
```

Conclusion

Zéro interruption de service.

Une vigilance constante

En combinant des checks applicatifs fins et des serveurs de backup, vous garantissez une disponibilité maximale à vos utilisateurs.

Installation de HAProxy

Guide complet pour installer HAProxy sur les serveurs Linux (Debian, RedHat) et via des environnements conteneurisés (Docker, Kubernetes).

1. Différences entre HAProxy Community et Enterprise

Comprendre les distinctions clés entre les deux éditions de HAProxy pour choisir celle qui correspond le mieux à vos besoins.

HAProxy Community

La version Community est le cœur open-source de HAProxy, développée et maintenue par une communauté active.

Avantages :

- **Gratuite et Open Source :** Accès libre au code source et aucune licence requise.
- **Innovation :** Bénéficie des dernières fonctionnalités et innovations rapidement.
- **Flexibilité :** Peut être compilée à partir des sources avec des options spécifiques pour des besoins très précis.
- **Communauté :** Support via forums, listes de diffusion et contributions de la communauté.

Inconvénients :

- **Support :** Pas de support commercial officiel ou de SLA (Service Level Agreement).
- **Cycles de vie :** Les versions peuvent avoir des cycles de support plus courts, nécessitant des mises à jour plus fréquentes.
- **Fonctionnalités avancées :** Certaines fonctionnalités (WAF avancé, protection bot) ne sont pas incluses nativement ou nécessitent une intégration manuelle complexe.
- **Gestion :** Pas de tableau de bord de gestion intégré, s'appuie sur des outils tiers pour la surveillance et la configuration.

Comment la reconnaître ?

Via la ligne de commande haproxy -v, la sortie affichera généralement HAProxy version X.Y.Z sans mention spécifique d'« Enterprise » ou « HAPEE ». Les noms de paquets sont souvent haproxy.

HAProxy Enterprise (HAPEE)

HAProxy Enterprise est une version commerciale de HAProxy, conçue pour les environnements de production critiques.

Avantages :

- **Support Commercial :** Accès à un support technique 24/7 avec des SLA.
- **Stabilité et LTS :** Versions Long Term Support (LTS) avec des correctifs de sécurité et de bugs rétroportés, garantissant une grande stabilité.
- **Fonctionnalités exclusives :** Intègre des modules avancés tels qu'un WAF (Web Application Firewall) sophistiqué, une protection contre les bots, des capacités d'API Gateway, et des optimisations de performance spécifiques.
- **Gestion Centralisée :** Fournit une interface de gestion graphique (HAProxy Enterprise Suite) pour la configuration, la surveillance et l'analyse.

- **Intégration :** Optimisée pour l'intégration avec les écosystèmes Cloud et les outils d'orchestration.

Inconvénients :

- **Coût :** Nécessite une licence commerciale.
- **Innovation :** Peut introduire les toutes dernières fonctionnalités de la version Community avec un léger décalage, privilégiant la stabilité.

Comment la reconnaître ?

La commande haproxy -v affichera HAProxy Enterprise Edition ou HAPEE. Les noms de paquets suivent le format hapee-X.Y-lb.

En résumé

Avant de procéder à l'installation, il est important de choisir l'édition adaptée à vos besoins. HAProxy est disponible en deux versions distinctes : **Community** et **Enterprise**.

Édition Community vs Enterprise

- **HAProxy Community :** Version open-source de pointe. Elle contient les dernières fonctionnalités et innovations. Elle est idéale pour les environnements de test ou les infrastructures gérées par des équipes ayant une forte expertise technique.
- **HAProxy Enterprise (HAPEE) :** Version stabilisée et optimisée pour la production. Elle inclut des modules exclusifs (WAF avancé, Dashboard global), un support technique 24/7, et des correctifs de sécurité rétroportés sur des versions LTS (Long Term Support).

2. Installation sur serveurs (Linux)

HAProxy est disponible nativement dans les dépôts de la plupart des distributions. La méthode diffère selon la famille d'OS.

Famille Debian (Debian, Ubuntu, Mint)

L'installation sur Debian ou Ubuntu utilise le gestionnaire apt. Pour bénéficier des dernières versions stables (LTS), il est recommandé d'utiliser les dépôts officiels de la communauté maintenus par *Vincent Bernat*.

Pour Ubuntu :

```
# Définir la version souhaitée (ex: 3.0, 3.1)
HAPROXY_VERSION="3.1"

sudo apt update
sudo apt install --no-install-recommends software-properties-common -y
sudo add-apt-repository ppa:vbernat/haproxy-${HAPROXY_VERSION} -y
sudo apt update
sudo apt install haproxy -y
```

Pour Debian :

```
# Définir la version souhaitée (ex: 3.0, 3.1)
HAPROXY_VERSION="3.1"

# Installation des prérequis
sudo apt update && sudo apt install curl gpg lsb-release -y
```

```
# Ajout de la clé et du dépôt officiel debian.haproxy.org
curl https://haproxy.debian.net/bernat.gpg | gpg --dearmor | sudo tee /usr/share/keyrings/hapr
oxy.debian.net.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/haproxy.debian.net.gpg] http://haproxy.debian.net $(l
sb_release -cs)-backports-${HAPROXY_VERSION} main" | sudo tee /etc/apt/sources.list.d/haproxy.
list
sudo apt update
sudo apt install haproxy -y
```

Famille RedHat (RHEL, CentOS, AlmaLinux, Rocky)

Sur les systèmes RedHat récents (RHEL 8/9+), on utilise dnf. Bien que HAProxy soit présent dans les dépôts **AppStream**, l'activation du dépôt **EPEL** est recommandée pour obtenir des versions plus récentes :

```
# Installation d'EPEL
sudo dnf install epel-release -y
# Installation de HAProxy
sudo dnf install haproxy -y
# Activation et démarrage immédiat
sudo systemctl enable --now haproxy
```

Pour valider l'installation et consulter les options de compilation, utilisez haproxy -v.

3. Installation via Conteneurs

La conteneurisation facilite le déploiement et l'isolation du load balancer.

Sans orchestrateur (Docker)

Lancer HAProxy avec Docker est idéal pour des besoins simples ou du test. On utilise généralement l'image officielle disponible sur Docker Hub. Il est crucial de monter le fichier de configuration localement.

```
# Définir la version souhaitée (ex: 3.0, 3.1, latest)
HAPROXY_VERSION="3.1"

docker run -d --name my-haproxy \
    -v /path/to/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro \
    -p 80:80 -p 443:443 \
    haproxy:${HAPROXY_VERSION}
```

L'argument :ro garantit que le conteneur ne peut pas modifier votre fichier de configuration d'origine.

Avec orchestrateur (Kubernetes / Swarm)

Dans un environnement orchestré comme Kubernetes, HAProxy est souvent déployé en tant que **Ingress Controller**. Il ne se contente pas de charger une configuration fixe, mais observe l'API Kubernetes pour mettre à jour ses routes dynamiquement.

L'installation recommandée se fait via Helm :

```
helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts
helm install ingress-controller haproxy-ingress/haproxy-ingress
```

Sous Docker Swarm, on utilisera un fichier docker-compose.yml déployé en tant que *stack*, en s'appuyant sur le réseau overlay pour joindre les services applicatifs.

4. HAProxy Enterprise Edition (HAPEE)

HAProxy Enterprise est une version optimisée, stabilisée et supportée pour les environnements critiques (Ubuntu, Debian, RHEL, CentOS, Rocky, AlmaLinux).

Installation via le script officiel

La méthode la plus simple consiste à utiliser le script d'installation unifié. Il configure automatiquement les dépôts GPG et les sources de paquets pour votre distribution :

```
# Définir la version souhaitée (ex: 2.9, 3.0, 3.1)
HAPROXY_VERSION="3.1"

# Téléchargement et exécution du script d'installation officiel
curl https://www.haproxy.com/static/install_haproxy_enterprise.sh | sudo bash -s -- \
    --key YOUR_LICENSE_KEY \
    --version ${HAPROXY_VERSION}
```

Le script installera le paquet hapee-${HAPROXY_VERSION}-lb. Vous pouvez ensuite démarrer le service avec systemctl enable --now hapee-${HAPROXY_VERSION}-lb.

Alternative : Installation manuelle

L'installation manuelle via apt, dnf ou **Docker**, ... est possible. Elle nécessite la création manuelle d'un fichier de dépôt pointant vers https://www.haproxy.com/download/hapee/key/... en utilisant votre clé comme paramètre d'authentification.

5. Conclusion

Choisir la bonne stratégie.

Verdict

L'installation sur serveur (Bare metal / VM) reste la norme pour les load balancers de bordure (Edge) nécessitant des performances maximales. La conteneurisation, via les Ingress Controllers, est devenue indispensable pour la flexibilité du Cloud et des microservices.

Maîtriser les ACLs dans HAProxy

Guide complet sur les ACLs HAProxy : apprenez à configurer le routage intelligent, le contrôle d'accès par IP, la géolocalisation, et l'analyse des Headers et User-Agent pour sécuriser et optimiser vos flux.

1. Le concept d'ACL

Les **Access Control Lists (ACL)** sont le cœur de l'intelligence de HAProxy. Elles permettent de définir des conditions logiques basées sur presque n'importe quel aspect d'une requête HTTP ou d'une connexion TCP. Une fois qu'une ACL est définie, elle peut être utilisée pour prendre des décisions de routage, bloquer du trafic, ou modifier des headers.

Syntaxe de base

La syntaxe standard d'une ACL suit ce schéma : acl <nom> <critère> [flags] <valeurs>. Elle renvoie un résultat booléen (vrai/faux).

```
# Définition : le chemin se termine par une extension d'image
acl est_image path_end .jpg .png .gif

# Utilisation : router vers le backend statique si l'ACL est vraie
use_backend static_servers if est_image
```

Dans cet exemple, path_end analyse la fin de l'URL. Si l'URL se termine par l'une des extensions listées, l'ACL devient 'vraie'. La directive use_backend n'est exécutée que si cette condition est remplie.

1.2 Configuration détaillée de l'ACL

Chaque élément de la syntaxe d'une ACL joue un rôle précis dans la définition de la condition.

C'est un identifiant unique et descriptif pour votre ACL. Il est recommandé d'utiliser des noms clairs qui reflètent la condition testée (ex: is_mobile, host_api, network_interne). Ce nom sera ensuite utilisé dans les règles if ou unless.

(Fetch Method)

Le critère, ou 'fetch method', indique à HAProxy quelle partie de la requête ou de la connexion il doit analyser. Voici une liste de 50 critères incontournables classés par usage :

1. Couche 4 (Réseau et Transport) :

- src : Adresse IP source du client.
- src_port : Port TCP source du client.
- dst : Adresse IP de destination (IP de l'interface HAProxy).
- dst_port : Port TCP de destination.
- fe_ip : Adresse IP d'écoute du frontend.
- fe_port : Port d'écoute du frontend.
- be_id : Identifiant interne du backend traité.
- fc_err : Détecte si une erreur de connexion est survenue.

2. Couche 7 (HTTP - URI, Chemin et Méthode) :

- method : Méthode HTTP (GET, POST, PUT, DELETE, etc.).
- path : Chemin complet de l'URI (ex: /index.html).
- path_beg : Vérifie si le chemin commence par une chaîne spécifique.
- path_end : Vérifie si le chemin se termine par une chaîne.
- path_sub : Vérifie si une chaîne est contenue dans le chemin.
- path_dir : Vérifie si une chaîne correspond à un répertoire dans le chemin.
- path_len : Nombre de caractères composant le chemin.
- path_reg : Correspondance par expression régulière sur le chemin.
- url : L'URL complète de la requête (incluant les paramètres).
- url_beg : Début de l'URL.
- url_sub : Sous-chaîne présente dans l'URL.
- base : Concaténation du Host et du Path (ex: example.com/api).

3. En-têtes (Headers) et Cookies :

- hdr(<nom>) : Valeur exacte d'un en-tête (ex: Host, User-Agent).
- hdr_beg(<nom>) : L'en-tête commence par...
- hdr_end(<nom>) : L'en-tête se termine par...
- hdr_sub(<nom>) : L'en-tête contient la sous-chaîne...
- hdr_cnt(<nom>) : Nombre d'occurrences d'un en-tête spécifique.
- hdr_found(<nom>) : Vrai si l'en-tête est présent, quelle que soit sa valeur.
- hdr_val(<nom>) : Extrait la valeur entière d'un en-tête.
- res.hdr(<nom>) : Analyse les en-têtes de la réponse serveur.
- cook(<nom>) : Valeur d'un cookie spécifique.
- cook_beg(<nom>) : Le cookie commence par...
- cook_sub(<nom>) : Le cookie contient...
- cook_cnt(<nom>) : Nombre de cookies présents dans la requête.

4. Paramètres d'URL (Query String) :

- urlp(<nom>) : Vérifie la présence d'un paramètre spécifique dans la requête.
- urlp_val(<nom>) : Extrait la valeur entière d'un paramètre d'URL.
- query : La chaîne de requête complète située après le '?'.
- url_param_found : Détecte si des paramètres d'URL sont présents.

5. SSL / TLS :

- ssl_fc : Vrai si la connexion est sécurisée (SSL/TLS).
- ssl_fc_has_crt : Vrai si le client a présenté un certificat.
- ssl_fc_sni : Valeur du SNI (Server Name Indication) envoyé par le client.
- ssl_fc_protocol : Version du protocole TLS utilisé (ex: TLSv1.3).
- ssl_fc_cipher : Nom de l'algorithme de chiffrement négocié.
- ssl_fc_alg_keysize : Taille de la clé de chiffrement symétrique utilisée.
- ssl_bc : Vrai si la connexion vers le backend est sécurisée.
- ssl_fc_is_resumed : Détecte si la session SSL a été reprise.

6. État de l'Infrastructure (Health & Load) :

- nbsrv(<backend>) : Nombre de serveurs 'UP' dans un backend spécifique.
- srv_is_up(<serveur>) : Vérifie l'état de santé d'un serveur précis.
- be_conn(<backend>) : Nombre total de connexions actives sur un backend.
- fe_conn : Nombre de connexions actives sur le frontend.
- connslots(<backend>) : Nombre de slots de connexion encore disponibles.
- queue(<backend>) : Nombre de requêtes en attente dans la file du backend.

[flags]

Les flags sont des modificateurs optionnels qui affinent le comportement de la comparaison. Les plus courants sont :

- -i : Rend la comparaison insensible à la casse (ex: hdr(Host) -i example.com correspondra à Example.com).
- -m <méthode> : Spécifie la méthode de correspondance. Les valeurs possibles incluent str (chaîne exacte), sub (sous-chaîne), beg (commence par), end (finit par), reg (expression régulière), found (présence de l'élément), bool (valeur booléenne).
- -f <fichier> : Permet de charger les valeurs à comparer depuis un fichier externe, utile pour les longues listes d'IP ou de User-Agents.

Ce sont les données avec lesquelles le critère doit être comparé. Elles peuvent être une ou plusieurs chaînes de caractères, des adresses IP/CIDR, des nombres, ou des expressions régulières, en fonction du critère et des flags utilisés. Si plusieurs valeurs sont spécifiées, l'ACL est vraie si le critère correspond à **l'une** des valeurs (logique OU).

2. Les critères de sélection courants

HAProxy propose des centaines de méthodes d'extraction (fetch methods). Voici les plus couramment utilisées pour l'analyse des requêtes.

2.1 Nom de domaine (Host)

Permet le virtual hosting. L'ACL extrait la valeur du header Host de la requête.

```
acl host_api hdr(host) -i api.domaine.com
use_backend api_cluster if host_api
```

Le flag -i est crucial ici : il rend la comparaison **insensible à la casse**. Ainsi, 'API.domaine.com' ou 'api.domaine.com' seront tous deux acceptés.

2.2 Chemin d'URL (Path)

Idéal pour le routage de microservices basé sur l'URI. path_beg teste le début du chemin.

```
acl is_blog path_beg /blog
use_backend blog_servers if is_blog
```

Ici, path_beg (Path Begin) vérifie si l'URL commence par /blog. Cela englobe /blog/article1 ou /blog/images/logo.png.

2.3 Headers HTTP

Vous pouvez tester n'importe quel header (Standard ou Custom). Cela permet un routage fin basé sur le contexte applicatif.

```
# Vérifier si la requête attend du JSON
acl wants_json hdr(Accept) -i application/json

# Vérifier la présence d'un header d'authentification custom
```

```
acl has_auth_token hdr(X-Auth-Token) -m found

use_backend json_api if wants_json has_auth_token
```

L'utilisation de -m found permet de valider la simple **existence** du header X-Auth-Token, sans se soucier de sa valeur. C'est très utile pour forcer la présence de jetons de sécurité avant même de les traiter.

2.4 User-Agent

Le header User-Agent permet d'identifier le client. C'est essentiel pour la redirection mobile ou la lutte contre les bots.

```
# Détecter les mobiles pour redirection
acl is_mobile hdr_sub(user-agent) -i android iphone
redirect prefix https://m.monsite.com if is_mobile

# Bloquer un bot par son nom
acl is_bad_bot hdr_sub(user-agent) -i badbot
http-request deny if is_bad_bot
```

La méthode hdr_sub (Header Substring) cherche si la valeur spécifiée est contenue n'importe où dans la chaîne du User-Agent. C'est plus flexible que hdr qui exigerait une correspondance exacte.

3. Contrôle d'accès et Sécurité

Les ACLs sont fondamentales pour sécuriser vos applications en filtrant le trafic entrant.

Limitation par IP (Whitelist/Blacklist)

Vous pouvez autoriser ou bloquer des accès basés sur l'adresse IP source (src).

```
acl network_interne src 10.0.0.0/8
acl is_admin path_beg /admin

# Bloquer l'accès admin si l'IP n'est pas dans le réseau interne
http-request deny if is_admin !network_interne
```

Le critère src identifie l'adresse IP du client qui initie la connexion. L'usage du point d'exclamation ! devant network_interne signifie "Si la requête provient de n'importe où SAUF du réseau interne".

Authentification HTTP

Combinez les ACLs avec des userlist pour forcer une authentification sur des chemins sensibles.

```
userlist admins
  user martin password password123

acl auth_ok http_auth(admins)
http-request auth realm Protection if is_admin !auth_ok
```

La fonction http_auth vérifie si les identifiants envoyés dans le header Authorization correspondent à ceux définis dans la section userlist. Si ce n'est pas le cas (!auth_ok), HAProxy renvoie un code 401 au navigateur.

4. Géolocalisation

Prendre des décisions basées sur l'origine géographique du visiteur.

Routage par pays

En utilisant des bases GeoIP2 ou des headers injectés par un CDN amont, vous pouvez cibler des zones géographiques.

```
# Exemple basé sur un header injecté par un CDN amont
acl is_france hdr(X-Country-Code) -i FR
use_backend fr_cluster if is_france

# Bloquer certains pays pour des raisons légales ou de sécurité
acl blocked_countries hdr(X-Country-Code) -i CN RU
http-request deny if blocked_countries
```

Cette technique repose sur le fait que le service en amont (comme Cloudflare ou Akamai) a déjà identifié l'origine géographique et l'a transmise via un header HTTP spécifique. Si vous gérez votre propre base MaxMind nativement dans HAProxy, vous utiliserez plutôt le fetch src_get_gpc0 ou des convertisseurs dédiés.

5. Opérateurs logiques

Combiner plusieurs ACLs pour créer des règles de décision complexes.

ET, OU et NON

La logique d'application des ACLs suit des règles simples :

- **ET (Implicite) :** Juxtaposer des ACLs sur la même ligne (ex: if acl1 acl2).
- **OU (Explicite) :** Utiliser l'opérateur || ou définir plusieurs lignes d'action.
- **NON :** Utiliser le point d'exclamation ! devant le nom de l'ACL.

```
# Si (Admin OU Maintenance) ET (Pas Local)
acl is_admin path_beg /admin
acl is_maint nbsrv(maint_backend) lt 1
acl is_local src 127.0.0.1

http-request deny if { is_admin || is_maint } !is_local
```

L'utilisation des **accolades** { ... } permet de créer des ACLs anonymes ou de grouper des conditions complexes sans avoir à les définir au préalable. Dans cet exemple, HAProxy bloque la requête si l'on tente d'accéder à l'admin OU si le serveur est en maintenance, SAUF si l'on se connecte depuis la machine locale.

Conclusion

La puissance au service de l'architecture.

Un couteau suisse pour le DevOps

La maîtrise des ACLs transforme HAProxy d'un simple répartiteur de charge en un véritable moteur de contrôle de trafic intelligent. Que ce soit pour implémenter du Blue/Green deployment, sécuriser des endpoints ou optimiser l'expérience utilisateur par la géolocalisation, les ACLs sont votre outil principal.

Sécurité et SSL avec HAProxy

Guide complet sur la terminaison SSL, le chiffrement des flux et la sécurisation du serveur HAProxy.

1. Terminaison SSL (Offloading)

HAProxy gère le déchiffrement pour soulager les serveurs web.

Configuration du certificat

C'est la méthode la plus courante. HAProxy possède le certificat SSL et communique en HTTP clair avec les backends sur un réseau privé sécurisé.

```
frontend https-in
    bind *:443 ssl crt /etc/ssl/certs/mon-site.pem
    default_backend app_servers
```

Note : Le fichier .pem doit contenir le certificat et la clé privée.

2. SSL Pass-through

Transmettre le flux chiffré sans inspection.

Utilisation du mode TCP

Si vous ne voulez pas que HAProxy déchiffre le trafic (pour des raisons de confidentialité totale), utilisez le mode TCP (Layer 4).

```
frontend stream-in
    mode tcp
    bind *:443
    default_backend secure_web
```

3. Forcer le HTTPS

Rediriger automatiquement les utilisateurs vers le port sécurisé.

Redirection HTTP vers HTTPS

Une règle simple dans le frontend HTTP suffit.

```
frontend http-in
    bind *:80
    http-request redirect scheme https unless { ssl_fc }
```

Conclusion

Un point central pour la sécurité.

Performance et Sécurité

En centralisant le SSL, vous simplifiez la gestion des certificats (ex: Let's Encrypt) et optimisez les performances de vos serveurs d'application.

Monitoring et Statistiques dans HAProxy

Comment activer le tableau de bord de statistiques natif et exposer des métriques pour Prometheus.

1. La page de statistiques native

Une interface web légère pour surveiller vos serveurs.

Configuration du bloc Listen

Il est recommandé d'écouter sur un port séparé et de sécuriser l'accès par mot de passe.

```
listen stats
    bind *:8404
    stats enable
    stats uri /admin?stats
    stats refresh 10s
    stats auth admin:MonMotDePasse
```

2. Intégration avec Prometheus

Exposer les métriques pour une surveillance à long terme.

Utilisation du module prometheus-exporter

Depuis la version 2.0, HAProxy peut exposer directement des métriques au format Prometheus sans agent tiers.

```
frontend stats
    bind *:8405
    http-request use-service prometheus-exporter if { path /metrics }
```

3. Support natif OpenTelemetry (OTLP)

HAProxy s'intègre désormais au standard de l'industrie pour les traces et les métriques.

Exportation vers un collecteur OTLP

Depuis la **version 2.9**, HAProxy supporte nativement **OpenTelemetry**. Cette fonctionnalité permet d'envoyer des données de télémétrie directement vers un collecteur OTel (ou des outils comme Jaeger, Honeycomb ou Grafana Tempo) sans agent intermédiaire.

Ce que cela apporte :

- **Standardisation :** Un format unique pour les traces et les métriques, compatible avec tous les outils modernes.
- **Traces distribuées :** Permet de corréler une requête passant par HAProxy avec les traces de vos microservices pour un débogage de bout en bout.
- **Richesse des données :** Exportation de métadonnées précises sur le cycle de vie des requêtes HTTP.

```
# Exemple de configuration (nécessite le module otlp.so)
```

```
module-load /usr/lib/haproxy/otlp.so

otlp
    endpoint my-otel-collector:4317
    interval 5s
    add-header x-otel-traceid
```

4. Interpréter les indicateurs clés

Quelles métriques surveiller en priorité ?

Sessions et Erreurs

- **Cur :** Nombre de connexions actives.
- **Rate :** Nombre de nouvelles connexions par seconde.
- **Check status :** État de santé des serveurs backends.
- **Errors (Req/Resp) :** Taux d'erreurs 4xx ou 5xx.

Conclusion

L'observabilité est la clé.

Anticiper les pannes

Grâce aux statistiques, vous pouvez détecter un serveur qui ralentit avant qu'il ne tombe complètement en panne.

Persistance de session (Stickiness)

Comprendre comment garantir qu'un utilisateur reste sur le même serveur backend durant toute sa session.

1. Persistance par Cookie

La méthode la plus fiable pour les applications HTTP.

Insertion de Cookie

HAProxy insère un cookie dans la réponse du serveur. Au retour, il lit ce cookie pour savoir vers quel serveur diriger la requête.

```
backend app_servers
    balance roundrobin
    cookie SERVERID insert indirect nocache
    server s1 10.0.0.1:80 check cookie s1
    server s2 10.0.0.2:80 check cookie s2
```

2. Persistance par IP Source

Utilisée quand l'application ne gère pas les cookies ou pour du trafic non-HTTP.

L'algorithme source

L'IP de l'utilisateur est hachée pour désigner un serveur. Simple, mais problématique si beaucoup d'utilisateurs partagent la même IP (ex: entreprise via un proxy).

```
backend app_servers
    balance source
    server s1 10.0.0.1:80 check
```

3. Les Stick-Tables

La mémoire interne de HAProxy pour une persistance avancée.

Stockage en mémoire

Les stick-tables permettent de stocker des informations (IP, ID de session) et de les partager entre plusieurs nœuds HAProxy via *Peers*.

```
backend myapp
    stick-table type ip size 100k expire 30m
    stick on src
```

Conclusion

Maintenir l'état utilisateur.

Choisir la bonne méthode

Le cookie est privilégié pour le Web, tandis que l'IP source ou les stick-tables sont indispensables pour les flux persistants complexes (TCP, RDP, etc.).

Protection des Applications

Découvrez comment sécuriser vos backends contre les abus et les attaques courantes. Guide pratique sur le Rate Limiting, le WAF, Fail2ban et CrowdSec avec HAProxy et Traefik.

1. Le Rate Limiting (Limitation de débit)

Le Rate Limiting consiste à limiter le nombre de requêtes qu'un utilisateur ou une IP peut effectuer dans un intervalle de temps donné.

Concept

Indispensable pour protéger les APIs, le Rate Limiting empêche un client unique de saturer vos ressources, qu'il s'agisse d'un bug dans un script client ou d'une tentative malveillante de déni de service (DoS).

Exemple HAProxy

HAProxy utilise des **stick-tables** pour mémoriser les compteurs de requêtes.

```
frontend http-in
    # Table de 100k entrées, expiration après 30s
    stick-table type ip size 100k expire 30s store http_req_rate(10s)

    # Suivre l'IP source
    http-request track-sc0 src

    # Refuser (429 Too Many Requests) si > 20 requêtes en 10s
    http-request deny deny_status 429 if { sc0_http_req_rate gt 20 }
```

Exemple Traefik

Traefik implémente cela via le middleware rateLimit.

```
# Configuration File (YAML)
http:
  middlewares:
    limit-api:
      rateLimit:
        average: 100
        period: 1m
        burst: 20
```

2. Les Stick Tables (HAProxy)

Une fonctionnalité avancée de HAProxy pour stocker des états en mémoire.

Pourquoi les utiliser ?

Les stick tables permettent de suivre bien plus que le simple débit. Elles peuvent stocker le nombre de connexions simultanées, le taux d'erreurs HTTP générées par un client, ou encore des informations de session.

Exemple : Blocage basé sur le taux d'erreurs

Si une IP génère trop d'erreurs 404 ou 403, elle est probablement en train de scanner votre application.

```
backend app-backend
    # Stocke le taux d'erreurs sur 10s
    stick-table type ip size 1m expire 1h store http_err_rate(10s)

    http-request track-sc1 src
    # Bloquer l'IP si elle génère plus de 10 erreurs en 10s
    http-request deny if { sc1_http_err_rate gt 10 }
```

3. Protection contre le Brute Force

Identifier les tentatives répétées de deviner des mots de passe.

Stratégie

La protection brute force cible généralement les points d'entrée sensibles comme /login ou /wp-login.php. On applique une limite beaucoup plus stricte sur ces URLs que sur le reste du trafic.

Exemple HAProxy

```
frontend https-in
    stick-table type ip size 100k expire 1m store http_req_rate(20s)

    acl is_login path_beg /login
    http-request track-sc2 src if is_login

    # Max 3 tentatives de login toutes les 20 secondes
    http-request deny if is_login { sc2_http_req_rate gt 3 }
```

Exemple Traefik

On crée un middleware spécifique pour le routeur gérant l'authentification.

```
http:
  middlewares:
    auth-bruteforce:
      rateLimit:
        average: 2
        period: 1m
        burst: 1
```

4. Anti-DDoS (Attaques par déni de service)

Prévenir la saturation de l'infrastructure par des flots de connexions.

Limitation des connexions simultanées

Une attaque DDoS classique consiste à ouvrir un maximum de connexions TCP sans jamais les fermer (Slowloris). Limiter le nombre de connexions simultanées par IP est une protection fondamentale.

Exemple HAProxy

Utilisation de conn_cur dans une stick table.

```
frontend http-in
    stick-table type ip size 100k expire 30s store conn_cur

    tcp-request connection track-sc0 src
    # Rejeter la connexion si l'IP a déjà 15 connexions ouvertes
    tcp-request connection reject if { sc0_conn_cur gt 15 }
```

Exemple Traefik

Traefik propose le middleware inFlightConn.

```
http:
  middlewares:
    limit-simultaneous-conn:
      inFlightConn:
        amount: 10
        sourceCriterion:
          ipStrategy:
            depth: 2
```

5. Web Application Firewall (WAF)

Le WAF analyse le trafic HTTP en profondeur pour bloquer les attaques applicatives (OWASP Top 10).

Concept

Contrairement au Rate Limiting qui s'appuie sur la quantité, le WAF inspecte la **qualité** des requêtes. Il cherche des motifs d'attaque (Payloads) comme l'injection SQL ou le Cross-Site Scripting (XSS).

Exemple HAProxy

L'intégration se fait souvent via **SPOE** (Stream Processing Offload Engine) pour déléguer l'analyse à un agent ModSecurity externe sans ralentir le processus principal.

```
frontend http-in
    # Appel de l'agent SPOE ModSecurity
    filter spoe engine modsecurity config /etc/haproxy/modsec.conf

    # Bloquer (403 Forbidden) si l'agent retourne un score d'attaque
    http-request deny deny_status 403 if { var(txn.modsec.code) -m int gt 0 }
```

Exemple Traefik

Traefik peut intégrer un WAF via des plugins (comme modsecurity) ou en utilisant un sidecar comme Coraza.

```
# Exemple via un plugin de middleware
http:
  middlewares:
    waf-modsec:
      plugin:
        modsecurity:
```

```
        modsecurityUrl: http://modsecurity-agent:8080
```

6. Fail2ban et CrowdSec (Analyse de logs)

Automatisez le bannissement des IPs malveillantes en analysant les logs en temps réel.

Concept

Fail2ban et **CrowdSec** agissent comme des systèmes de détection d'intrusion (IDS) basés sur l'analyse de logs. Leur rôle est d'identifier des comportements malveillants automatisés (brute force, scans de vulnérabilités, déni de service applicatif) qui auraient pu passer à travers les premières couches de défense, et d'appliquer une sanction automatique (le bannissement IP).

Comparatif : Fail2ban vs CrowdSec

Bien qu'ils partagent le même objectif, leur approche diffère radicalement :

- **Fail2ban :** L'outil historique. Il lit les fichiers de logs locaux et utilise des expressions régulières (regex) pour compter les échecs. S'il dépasse un seuil, il appelle une action (souvent iptables) pour bloquer l'IP.
    - **Avantages :** Très léger, stable, sans dépendance externe, simple à configurer pour des besoins basiques.
    - **Inconvénients :** Analyse uniquement locale (ne profite pas des attaques subies par les autres), les regex peuvent être gourmandes en CPU sur de très gros volumes de logs.
- **CrowdSec :** Une approche moderne et collaborative. Il utilise un moteur de scénarios pour détecter les comportements et repose sur deux composants clés :
    - **LAPI (Local API) :** Elle centralise les détections de vos agents locaux et instruit les bouncers (pare-feu, proxy) sur les actions de blocage à prendre au sein de votre infrastructure.
    - **CAPI (Central API) :** Elle reçoit vos signaux d'attaque anonymisés et vous renvoie en temps réel la base de réputation mondiale alimentée par la communauté.
    - **Avantages :** Protection proactive, bouncers multi-couches, idéal pour les architectures distribuées.
    - **Inconvénients :** Installation plus complexe et dépendance à Internet pour la CAPI.

Fail2ban (Exemple avec HAProxy)

Fail2ban surveille le fichier de log de HAProxy et bannit les IPs qui génèrent trop d'erreurs d'authentification.

```
# /etc/fail2ban/jail.local
[haproxy-http-auth]
enabled  = true
filter   = haproxy-http-auth
logpath  = /var/log/haproxy.log
maxretry = 3
bantime  = 3600
```

CrowdSec (Exemple avec HAProxy)

Pour HAProxy, CrowdSec utilise souvent un bouncer LUA qui vérifie l'IP en temps réel auprès de l'agent CrowdSec.

```
frontend http-in
    # Chargement du script LUA du bouncer CrowdSec
    lua-load /var/lib/crowdsec/lua/bouncer.lua
```

```
    # Appel du bouncer pour vérification avant traitement
    http-request lua.crowdsec_allow
```

CrowdSec (Exemple avec Traefik)

CrowdSec est une alternative moderne et collaborative. Pour Traefik, on utilise généralement un **Bouncer** sous forme de middleware pour bloquer les requêtes avant qu'elles n'atteignent le backend.

```
# Configuration Traefik (Middleware via Plugin)
http:
  middlewares:
    crowdsec-bouncer:
      plugin:
        crowdsec-bouncer:
          enabled: true
          crowdsecLapiHost: crowdsec-agent:8080
          crowdsecLapiKey: "votre_api_key"
```

Conclusion

L'importance d'une défense en profondeur.

Aller plus loin

En plus des protections applicatives locales, il est recommandé de coupler votre architecture à des services tiers (Cloudflare, AWS Shield) pour les attaques volumétriques massives au niveau réseau (Couche 3 et 4).

Visualiser votre infrastructure avec HAProxy Topology Visualizer

Apprenez à utiliser HAProxy Topology Visualizer pour transformer vos fichiers haproxy.cfg complexes en schémas interactifs et faciliter le débogage de votre architecture.

1. Qu'est-ce que HAProxy Topology Visualizer ?

Un outil indispensable pour comprendre la logique de routage de votre load balancer.

Le concept

HAProxy Topology Visualizer est un produit développé et maintenu par **RousselTM** qui analyse vos fichiers de configuration haproxy.cfg pour générer une représentation graphique. Il permet de voir instantanément les relations entre les **Frontends**, les **Backends**, les règles **ACL** et les serveurs finaux.

Pourquoi l'utiliser ?

- **Audit de configuration :** Identifier des backends orphelins ou des erreurs de routage.
- **Documentation :** Générer des schémas d'architecture à jour pour vos équipes.
- **Onboarding :** Aider les nouveaux collaborateurs à comprendre le flux de trafic sans lire des milliers de lignes de code.

2. Comment déployer l'outil ?

Le déploiement s'effectue via Docker Compose à partir du dépôt officiel de la formation.

Pré-requis

- Disposer de Docker installé sur votre machine.
- Avoir des droits suffisants pour pouvoir déployer un conteneur.
- Le réseau haproxy_net doit être présent. Si ce n'est pas le cas, créez-le manuellement :

```
docker network create haproxy_net
```

Procédure de déploiement

Exécutez les commandes suivantes dans votre terminal :

```
# 1. Cloner le repo
git clone https://github.com/rousseltm/haproxy-formation.git

# 2. Accéder au dossier
cd haproxy-formation

# 3. Lancer le déploiement
docker compose -f compose-parser.yaml up -d
```

Accès à l'interface

Une fois le conteneur démarré, vous avez deux méthodes pour accéder à l'outil :

- **Option 1 : Routage via HAProxy (Recommandé)**
  Utilisez votre instance HAProxy (ex: port 8404) pour rediriger le flux vers le service parser. Pour éviter tout conflit avec l'interface de statistiques, assurez-vous que l'URI des stats est spécifique (ex: /stats) et utilisez la directive use_backend :

```
# Dans votre section listen ou frontend
stats uri /stats
use_backend parser_backend if { path_beg /parser }

backend parser_backend
    http-request set-path %[path,regsub(^/parser,/)]
    server visualizer_srv parser:80 check
```

- **Option 2 : Exposition directe de port**
  Modifiez le fichier compose-parser.yaml pour exposer le port 80 du conteneur sur un port de votre machine (ex: 8555) :

```
services:
  parser:
    ports:
      - "8555:80"
```

3. Comment l'utiliser ?

Trois étapes simples pour visualiser votre topologie.

Importation de la configuration

Sur l'interface web, sélectionnez simplement le dossier contenant vos fichiers de configuration HAProxy. L'outil analysera l'ensemble des fichiers pour reconstituer la topologie complète. Si vous souhaitez corréler la vue avec l'état réel de votre noeud, vous pouvez également renseigner les informations d'accès aux statistiques (URL, utilisateur et mot de passe).

Analyse du graphe

L'outil génère un diagramme dirigé :

- **Nœuds Frontends :** Points d'entrée avec leurs ports d'écoute (bind).
- **Arêtes (Lignes) :** Représentent les décisions de routage (ACL, use_backend).
- **Nœuds Backends :** Groupes de serveurs avec leurs algorithmes de balance.

Filtrage et interaction

La plupart des versions permettent de cliquer sur un composant pour mettre en évidence son chemin spécifique, ce qui est extrêmement utile pour isoler le flux d'une application précise dans un cluster mutualisé.

4. Limites et bonnes pratiques

Ce qu'il faut savoir avant de l'utiliser en production.

Sécurité des données

> **ATTENTION :** Ne téléversez jamais votre configuration sur des versions en ligne publiques si elle contient des secrets, des mots de passe ou des adresses IP sensibles. Utilisez toujours une instance locale ou conteneurisée.

Versions de HAProxy

L'outil supporte généralement les directives standards jusqu'à la version 3.x. Certaines directives très récentes ou spécifiques à la version Enterprise pourraient ne pas être rendues graphiquement, mais l'essentiel de la topologie restera valide.

ACME : Le guide du déploiement SSL/TLS automatisé

Comprendre le protocole ACME, ses avantages et comment configurer le renouvellement automatique des certificats avec Traefik, HAProxy et Nginx.

1. Qu'est-ce que le protocole ACME ?

Le protocole ACME (RFC 8555) est un standard permettant d'automatiser les interactions entre une autorité de certification (CA) et les serveurs web.

Origine et Fonctionnement

Développé par l'ISRG (Internet Security Research Group) pour le lancement de **Let's Encrypt**, ACME permet à un serveur de prouver qu'il contrôle un domaine afin d'obtenir un certificat sans intervention humaine. Cette preuve repose sur des 'défis' (challenges) :

- **HTTP-01 :** Placer un fichier spécifique sur le serveur web.
- **DNS-01 :** Créer un enregistrement TXT spécifique dans la zone DNS.

Les avantages majeurs

- **Gratuité :** La plupart des autorités ACME (Let's Encrypt, ZeroSSL) offrent des certificats gratuits.
- **Sécurité :** Les certificats expirent tous les 90 jours, limitant l'impact en cas de compromission.
- **Fiabilité :** L'automatisation élimine le risque d'oubli de renouvellement (première cause d'indisponibilité HTTPS).

2. Automatisation avec Traefik

Traefik est le champion de l'ACME grâce à son support natif sans outil tiers.

Configuration (traefik.yml)

Traefik gère lui-même le stockage et le renouvellement via des *CertificatesResolvers* :

```
certificatesResolvers:
  myresolver:
    acme:
      email: admin@votre-domaine.fr
      storage: acme.json
      httpChallenge:
        entryPoint: web # Port 80 obligatoire pour le défi
```

Ensuite, sur vos conteneurs, il suffit d'ajouter un label :
traefik.http.routers.monapp.tls.certresolver=myresolver

3. Automatisation avec HAProxy

HAProxy nécessite généralement un client externe (Certbot ou acme.sh) pour gérer le challenge.

Intégration avec Certbot

La méthode recommandée consiste à dédier un backend au challenge ACME pour ne pas couper le trafic :

```
frontend http-in
    bind *:80
    acl is_acme path_beg /.well-known/acme-challenge/
    use_backend acme_backend if is_acme

backend acme_backend
    server certbot 127.0.0.1:8888
```

Le renouvellement se fait via un script qui concatène le certificat et la clé privée dans un fichier .pem, format attendu par HAProxy.

4. Automatisation avec Nginx

Nginx utilise massivement Certbot via un plugin dédié qui automatise la lecture et l'écriture de la configuration.

Commande de mise en place

Sur la plupart des systèmes Linux, une seule commande suffit pour transformer un site HTTP en HTTPS :

```
sudo certbot --nginx -d mon-domaine.fr
```

Certbot va alors :

1. Modifier votre nginx.conf pour ajouter les blocs SSL.
2. Gérer le challenge HTTP-01.
3. Installer une tâche *Cron* pour renouveler le certificat automatiquement tous les 60 jours.

Conclusion

Quel outil choisir ?

Si vous travaillez dans un environnement cloud/conteneurs, **Traefik** offre l'expérience la plus fluide. Pour des infrastructures traditionnelles plus statiques, le couple **Nginx + Certbot** reste la référence mondiale par sa simplicité.

Aide-mémoire HAProxy : Les commandes essentielles

Un guide pratique regroupant toutes les commandes indispensables pour installer, gérer, tester et déboguer HAProxy au quotidien (CLI, Systemd, Docker et Runtime API).

1. Validation et CLI

Identifier la version et valider l'intégrité de votre configuration.

Vérification de la version

Il est essentiel de vérifier les capacités de votre binaire, notamment pour le support SSL/TLS ou Lua.

```
# Version simple
haproxy -v

# Version détaillée (indispensable pour le débug)
haproxy -vv
```

La commande -vv affiche les bibliothèques liées (OpenSSL, PCRE, etc.) et les options de compilation qui peuvent restreindre certaines fonctionnalités.

Test de configuration

Règle d'or : Ne jamais recharger un service en production sans valider la syntaxe. Une erreur de virgule peut arrêter le service.

```
# -c : check mode, -f : spécifie le fichier
haproxy -c -f /etc/haproxy/haproxy.cfg
```

HAProxy permet de charger plusieurs fichiers. L'ordre des -f est important car les directives sont lues séquentiellement.

2. Gestion du Service

Piloter le cycle de vie selon votre environnement.

Systemd (Linux)

Privilégiez le **reload** au **restart**. Le reload utilise le signal USR2 pour lancer un nouveau processus tout en laissant l'ancien terminer les connexions en cours (Hitless Reload).

```
# Rechargement sans coupure
sudo systemctl reload haproxy

# Vérification du statut du service
sudo systemctl status haproxy
```

Docker & Docker Compose

Dans un conteneur, redémarrer le conteneur coupe les connexions. Utilisez le signal SIGHUP pour déclencher le rechargement interne de HAProxy.

```
# Envoyer le signal HUP au processus 1 du conteneur
docker kill -s HUP <container_name>

# Consulter les logs de santé et de trafic
docker logs --tail 100 -f <container_name>
```

3. Runtime API via Socat

Interagir avec HAProxy en temps réel via la socket UNIX.

Lecture d'informations

La socket permet de parler directement au moteur. Elle doit être définie dans global avec un niveau admin.

```
# Extraire les stats au format CSV
echo "show stat" | socat stdio /var/run/haproxy.stat

# Voir les infos de santé globale (Uptime, Process ID, etc.)
echo "show info" | socat stdio /var/run/haproxy.stat
```

Actions à chaud

Ces modifications sont immédiates et ne nécessitent pas de reload, mais elles sont **volatiles** (perdues au prochain restart si non reportées dans le .cfg).

```
# Sortir un serveur du pool (Drain)
echo "set server web_servers/srv1 state maint" | socat stdio /var/run/haproxy.stat

# Changer dynamiquement le ratio de trafic (Weight)
echo "set server web_servers/srv1 weight 10" | socat stdio /var/run/haproxy.stat
```

4. Débogage

Outils pour l'analyse réseau.

Mode interactif

Le mode -db (disable background) est parfait pour le développement. Il affiche toutes les alertes et erreurs directement dans le terminal.

```
haproxy -db -f /etc/haproxy/haproxy.cfg
```

Vérification des ports

Vérifiez que HAProxy écoute bien sur les interfaces et ports attendus (80, 443, 8404).

```
sudo ss -tlpn | grep haproxy
```

Conclusion

Astuce de production

Privilégiez toujours le signal **HUP** (ou systemctl reload) plutôt qu'un restart pour éviter de rejeter les connexions clients en cours lors d'une mise à jour de configuration.

Détails du Programme Pratique

HAProxy : de la découverte à l'expertise **(Travaux Pratiques)**

Testez vos connaissances avec ce QCM

Introduction à la répartition de charge (Load Balancing) **(Session Vidéo)**

Comprendre les enjeux de la haute disponibilité et le positionnement d'HAProxy.

Installation et Gestion du service **(Session Vidéo)**

Installer HAProxy sur Linux et maîtriser les commandes de rechargement à chaud.

Anatomie de la configuration: Global, Frontend, Backend et Defaults **(Session Vidéo)**

Étude anatomique du fichier de configuration haproxy.cfg.

Algorithmes : Round Robin, Leastconn et Hash **(Session Vidéo)**

Comment HAProxy choisit le serveur de destination.

Maîtriser les ACL **(Session Vidéo)**

Routage basé sur le chemin, l'hôte ou les cookies.

Terminaison SSL et Offloading **(Session Vidéo)**

Centraliser vos certificats SSL sur HAProxy.

Le tableau de bord des statistiques **(Session Vidéo)**

Activer et sécuriser l'interface web de monitoring.

HAProxy et Docker : construire un cluster load balancé de haute disponibilité
(Travaux Pratiques)

Mise en place d'une infrastructure complète avec répartition de charge et monitoring.

Mise en place de l'infrastructure

Déploiement de 3 serveurs web, d'un serveur Nginx et d'un cluster HAProxy via Docker Compose.

IMPORTANT : Vous devez lire cet article avant de commencer ce TP pour bien maîtriser les fondamentaux.

Nous vous recommandons aussi de suivre ces formations :

- Docker : Pour mieux comprendre l'importance de la conteneurisation
- Prometheus : Pour mieux comprendre la notion de métriques et leurs gestions
- Opentelemetry : Pour mieux comprendre la notion de métriques et leurs gestions
- Grafana : Pour mieux comprendre la visualisation des données de télémétrie

. Mais cela n'est pas indispensable, car nous fournirons pour ce TP toutes les commandes pour ces produits.

Obtention des sources

Vous devez cloner les sources du projet depuis ce REPO https://github.com/rousseltm/haproxy-formation.git dans le dossier de votre choix.

Correction :

```
git clone https://github.com/rousseltm/haproxy-formation.git
```

Mise en place de l'infrastructure

Pour tester nos configurations, on aura besoin d'une infrastructure minimale. Vous devez lancer les commandes suivantes pour la déployer sur Docker :

```
cd haproxy-formation
sudo docker compose -f compose-infra.yaml up -d
```

Attention : Toutes les commandes docker compose devront être lancées dans le dossier haproxy-formation.

Prise en main de la configuration

Exploration de la structure de configuration et mise en place de la modularité.

Accéder au fichier de base haproxy.cfg

Le cœur de HAProxy réside dans son fichier de configuration principal : haproxy.cfg. Dans notre environnement Docker, ce fichier est monté en volume.

> Le fichier se situe dans config/haproxy/haproxy.cfg. Vous pouvez l'ouvrir avec votre éditeur de texte habituel pour observer les sections déjà présentes.

Démarrage de l'instance de base

Pour tester nos configurations, on aura besoin d'une infrastructure **HAProxy 3.4** avec une configuration de base que l'on va modifier par la suite. Dans un premier temps, on va tester son fonctionnement avec la configuration de base avec les commandes suivantes :

```
sudo docker compose up -d
```

Vérification du démarrage

L'objectif est d'apprendre à identifier les erreurs de configuration à travers les logs système et Docker et de vérifier l'absence d'erreurs dans les logs générés par HAProxy.

> Installation sur machine virtuelle ou machine physique : Les logs sont généralement disponibles dans /var/log/haproxy.log ou via la commande
>
> ```
> journalctl -u haproxy
> ```
>
> Installation via Docker Compose (notre cas) : Utilisez la commande suivante pour consulter les logs de toutes les instance :
>
> ```
> sudo docker compose logs
> ```
>
> ou cette commande pour consulter celui d'une instance en particulier (nom conteneur 3: haproxy-formation-haproxy-3)
>
> ```
> docker logs <container_name>
> ```

Avec une configuration par défaut ou incorrecte, vous pouvez rencontrer des erreurs comme :

> haproxy-1 | [ALERT] (1) : config : parsing [/usr/local/etc/haproxy/haproxy.cfg:71]: Missing LF on last line, file might have been truncated at position 32.

Vous devez comprendre le message et le corriger

Correction :
L'erreur vous informe simplement qu'il manque un saut de ligne à la fin du fichier (LF).

> BONNE PRATIQUE : Toujours tester vos configurations avant de relancer votre instance en Production: option -c

L'importance de la modularité

> À grande échelle, maintenir un fichier haproxy.cfg monolithique devient complexe et risqué. La modularité permet de séparer les responsabilités (ex. : un fichier par application, un pour le monitoring, un pour la sécurité, etc.), facilitant ainsi le déploiement continu, les revues de code et la limitation des erreurs humaines lors des mises à jour.

> Par défaut, le service HAProxy est lancé avec l'option -f haproxy.cfg. Vous pouvez ajouter autant d'options -f que vous avez de fichiers, mais vous pouvez aussi utiliser la même option et passer un dossier en paramètre : HAProxy chargera tous les fichiers se terminant par .cfg dans ce dossier.
>
> Attention à bien prendre en compte la dépendance entre les modules !

Dans notre cas, avec notre configuration Docker Compose, utilisez la commande suivante pour basculer vers l'utilisation du dossier conf.d (pensez à déplacer le fichier de conf dans ce dossier), qui se trouve au même niveau que le fichier haproxy.cfg. Les blocs de configuration des exercices suivants devront donc être placés dans un fichier .cfg de ce dossier, par exemple config/haproxy/conf.d/tp.cfg :

```
sudo docker compose down; export HAPROXY_CONFIG=conf.d; sudo -E docker compose up -d
```

Relance de la configuration

Chaque modification du fichier nécessite une prise en compte par le moteur HAProxy.

> **Reload vs Restart :** En production, n'utilisez jamais le restart car il coupe les connexions. Le **Reload** (signal HUP) permet de charger la nouvelle conf sans aucune interruption de service.

Correction :
Pour relancer avec systemctl :

```
sudo systemctl reload haproxy
```

Pour relancer proprement avec Docker Compose (Reload) :

```
sudo docker compose kill -s HUP haproxy
```

Pour un redémarrage complet (Restart) :

```
sudo docker compose restart haproxy
```

Pour en savoir plus, consultez l'article sur les bonnes pratiques de configuration.

Gestion des Frontends

Configuration des points d'entrée pour les statistiques et le trafic HTTP.

Point d'entrée pour l'administration

Créez un point d'entrée dédié au monitoring nommé 'admin'. Ce point d'entrée doit écouter sur le port 8404 (port par défaut). Vous devez activer l'interface de statistiques, définir la racine (/) comme URI d'accès et configurer un rafraîchissement automatique toutes les 5 secondes. Notez que vous pouvez réaliser cette configuration avec un bloc frontend ou un bloc listen, le résultat sera identique.

> Comme nous avons basculé vers le chargement du dossier conf.d, ajoutez cette configuration dans un fichier .cfg de ce dossier. Pour appliquer vos modifications, utilisez la commande export HAPROXY_CONFIG=conf.d; sudo -E docker compose restart haproxy (bien faire un reload en Production).

Correction :

Vous pouvez éditer le fichier de configuration HAProxy pour ajouter cette configuration :

```
frontend admin
    bind :8404
    stats enable
    stats uri /
    stats refresh 5s
```

Et ensuite, relancer le service.

Test de l'accès aux statistiques

Après avoir configuré le frontend 'admin', vérifiez que l'interface de monitoring est bien opérationnelle. Vous pouvez utiliser votre navigateur ou une commande terminal pour tester l'accessibilité du service sur le port dédié.

Correction :

Vérifiez l'accès en ouvrant l'URL suivante dans votre navigateur :

```
http://localhost:8404
```

Vous pouvez également valider la connectivité avec curl :

```
curl http://localhost:8404
```

Frontend HTTP

Configurez le frontend principal nommé 'http_in' pour recevoir le trafic web des utilisateurs. Il doit écouter sur le port 80 et rediriger par défaut toutes les requêtes vers le groupe de serveurs 'default' (qui sera créé par la suite).

Correction :

```
frontend http_in
    bind :80
    default_backend default
```

Gestion des Backends

Configuration des serveurs de destination pour traiter les requêtes.

Le Backend par défaut

Créez un groupe de serveurs nommé 'default'. Ce backend sera utilisé pour traiter les requêtes ne correspondant à aucune règle spécifique. Il doit fonctionner en mode HTTP, utiliser l'algorithme 'roundrobin' et pointer vers le serveur 'web1'. Configurez une règle pour que ce backend serve systématiquement le fichier 'default.html'.

Correction :

```
backend default
    mode http
    balance roundrobin
    # Réécriture du chemin pour afficher la page de maintenance
    http-request set-path /default.html
```

```
    # Déclaration des nœuds applicatifs
    server web1 web1:80 check
```

Reload à chaud via fichier de conf

Ouvrez votre interface de statistiques (port 8404). Vous devriez normalement voir 1 serveur actif dans le backend **default**. L'objectif est d'ajouter le serveur **web2** à ce pool à chaud, c'est-à-dire sans redémarrer le conteneur HAProxy, afin de préserver les connexions en cours. Les étapes :

- Voir la date de création des conteneurs HAProxy : L'information est dans la colonne 'CREATED'

  ```
  sudo docker compose ps haproxy
  ```
- Vous devez éditer le fichier pour ajouter le second serveur et relancer le service :

  ```
  sudo docker compose kill -s HUP haproxy
  ```
- Vous devez vérifier dans l'interface que le nouveau serveur est bien présent et que la date de création du serveur n'a pas changé

Correction :

1. Ajoutez le serveur **web2** dans le bloc backend default de votre fichier de configuration chargé par HAProxy:

```
server web2 web2:80 check
```

2. Pour recharger la configuration sans redémarrer le conteneur, envoyez un signal de rechargement (SIGHUP) au processus HAProxy :

```
sudo docker compose kill -s HUP haproxy
```

Vérifiez sur le dashboard de statistiques : le backend **default** doit maintenant afficher 2 serveurs sans que l'Uptime du service n'ait été réinitialisé.

Reload à chaud via socat

HAProxy dispose d'une **Runtime API** accessible via une socket Unix. Elle permet de modifier l'état du load balancer (poids, état des serveurs, ajout dynamique) sans aucun rechargement.

> Pour inspecter l'état interne ou passer des commandes directement dans votre conteneur, connectez-vous au shell via :
> sudo docker exec -it <nom_du_conteneur> sh
>
> Pour utiliser l'API, vous devez avoir activé la socket dans la section **global** de la configuration chargée par HAProxy :
> stats socket /tmp/admin.sock mode 600 level admin
>
> **Recommandation de lecture :** Pour approfondir la gestion dynamique des backends et des serveurs via la Runtime API, consultez notre article dédié : HAProxy : Ajout et suppression dynamique de backends.
> En ajoutant, vous devez voir le serveur apparaitre dans votre page de stats Mais comme vous avez peu le constater la commande socat est une commande locale donc il faut la jouter sur l'ensemble des instances de votre cluster HAProxy. De plus, comme indiqué dans la documentation, ajouter un serveur ne suffit pas pour qu'il reçoive le trafic, il faut l'activer ...

Correction :

1. Connectez-vous au conteneur HAProxy :

```
docker exec -it <nom_du_conteneur> sh
```

2. Utilisez **socat** pour envoyer la commande d'ajout au moteur :

```
echo "add server default/web3 web3:80 check" | socat stdio /tmp/admin.sock
```

3. Par défaut, un serveur ajouté dynamiquement est en mode maintenance. Activez-le :

```
echo "enable server default/web3" | socat stdio /tmp/admin.sock
```

4. Il faut aussi activer le healthcheck :

```
echo "enable health default/web3" | socat stdio /tmp/admin.sock
```

Vérifiez sur le dashboard : le serveur web3 apparaît instantanément. Mais attention à socat qui ne gère pas la persistence et a une exécution locale

Le Backend web_servers

Créez un groupe de serveurs nommé 'web_servers'. Ce backend doit fonctionner en mode HTTP et utiliser l'algorithme de répartition 'least connectioin' avec chaque backend qui a un poids qui correspond à son id de serveur. Vous devez également ajouter une règle pour transmettre l'adresse IP réelle du client au backend via l'en-tête 'X-Real-IP' et déclarer les 3 nœuds applicatifs (web1, web2, web3) sur le port 80 avec une vérification de santé (check). Enfin, mettez à jour le frontend http_in pour envoyer le trafic par défaut vers ce backend.

Correction :

```
frontend http_in
    bind :80
    default_backend web_servers

backend web_servers
    mode http
    balance leastconn
    http-request set-header X-Real-IP %[src]
    # Déclaration des 3 nœuds applicatifs du cluster
    server web1 web1:80 weight 1 check
    server web2 web2:80 weight 2 check
    server web3 web3:80 weight 3 check
```

Scaling de HAProxy

Le passage à l'échelle horizontal (scaling) consiste à multiplier les instances de HAProxy pour garantir la **haute disponibilité (HA)**. Dans une architecture réelle, avoir une seule instance de load balancer crée un 'Single Point of Failure' (SPOF) : si HAProxy tombe, toute l'application est inaccessible, même si les serveurs web sont sains. En environnement Docker, le scaling permet de répartir la charge réseau et d'assurer une continuité de service lors des mises à jour (rolling updates). Vous devez lancer la commande Docker Compose avec le paramètre --scale haproxy=3 pour faire passer le nombre d'instances HAProxy à 3.

Note : Multiplier le nombre de nœuds HAProxy ne suffit pas à le mettre en haute disponibilité. Il faut qu'il dispose d'une VIP : dans notre cas, c'est Nginx qui fera le load balancing pour assurer une haute disponibilité complète.

Correction :

Pour passer votre point d'entrée à l'échelle dans le monde Docker sans modifier votre fichier de configuration compose.yml, nous utilisons le paramètre d'exécution --scale :

Déploiement : Appliquez la nouvelle topologie :

```
sudo -E docker compose up -d --scale haproxy=3
```

Pourquoi est-ce important ?

- **Résilience :** Si un conteneur HAProxy plante, les autres instances continuent de répondre.
- **Maintenance à chaud :** Vous pouvez redémarrer ou mettre à jour une instance sans interrompre le trafic global.
- **Performance :** En production, ces instances seraient réparties sur différents nœuds physiques (via Docker Swarm ou Kubernetes), ce qui permettrait d'absorber des débits réseau massifs.

Note : En local avec Docker Compose, les instances peuvent partager le même mappage de port si vous utilisez un réseau de type 'overlay' ou un load balancer amont.

Routage avancé avec ACL

Rediriger les requêtes commençant par /api vers le serveur 2.

Modification ACL

Le routage intelligent permet de diriger le trafic vers différents backends selon des critères précis. Dans cet exercice, vous devez reconfigurer HAProxy pour qu'il identifie les requêtes destinées à l'API (commençant par le chemin /api) et les envoie exclusivement vers le backend secure_servers qui contient les serveurs du service **web** mais uniquement de la 2e à la 10e occurence. Le trafic restant doit continuer à être distribué par le backend par défaut.

Par defaut HAProxy va faire des tests de niveau 4 (TCP - IP:PORT) mais vous devez modifier le backend pour faire un test de vie de niveau 7 (HTTP - page web) qui vérifie que la page 'secure.html' retourne bien la chaîne 'Service\ '

Correction :
Voici un exemple de configuration (il est conseillé de le dispatcher en plusieurs fichiers) avec un bloc pour le frontend, la résolution des services docker et un pour le backend

```
frontend http_in
    bind *:80
    # On définit une ACL qui vérifie si le chemin commence par /api
    acl is_api path_beg /api
    # On demande à HAProxy d'utiliser un backend spécifique si l'ACL est vraie
    use_backend secure_servers if is_api
    default_backend web_servers
```

```
resolvers docker
    # DNS interne Docker (permet resolution des noms de containers)
    nameserver dns 127.0.0.11:53
    # Nombre de tentatives en cas d'echec DNS
   resolve_retries 3
    # Timeout global pour une requete DNS
   timeout resolve 1s
    # Temps d'attente entre deux tentatives de resolution
    timeout retry 1s
    # Duree pendant laquelle une resolution est consideree valide
    # Important pour eviter les re-resolutions trop frequentes
    hold valid 10s

backend secure_servers
    # Healthcheck HTTP personnalise
    option httpchk GET /secure.html
    # Verifie que la reponse contient bien 'Service'
    http-check expect string Service\
    # Pool dynamique securise
    server-template web 2-10 web:80 check resolvers docker init-addr libc,none
```

Whitelisting des IP

Le filtrage par adresse IP est un pilier de la sécurité. Vous devez maintenant restreindre l'accès au dashboard de statistiques (port 8404) pour n'autoriser que les sources internes qui ont besoin de lire l'interface ou l'endpoint /metrics, par exemple le serveur Prometheus et le serveur Nginx d'administration.

> **IMPORTANT :** Veillez à bien comprendre le schéma d'architecture pour vous assurer que vos règles de routage permettent toujours l'accès à la console de statistiques et à l'endpoint /metrics.
>
> **Whitelisting vs Blacklisting :**
>
> HAProxy est compatible avec les deux approches via ses règles d'ACL.
>
> - **Whitelisting (liste blanche) :** Stratégie *'Default Deny'*. On bloque tout par défaut et on n'autorise que les sources connues. C'est l'approche la plus robuste pour la sécurité.
>   *Cas d'usage :* accès à une console d'administration, API interne réservée à des partenaires, restriction d'accès aux IP d'un VPN d'entreprise.
> - **Blacklisting (liste noire) :** Stratégie *'Default Allow'*. On laisse passer tout le trafic sauf les IP identifiées comme malveillantes.
>   *Cas d'usage :* blocage de bots connus (scrapers), protection temporaire contre une IP réalisant une attaque par force brute, ou Geo-blocking (blocage de régions entières).

Correction :

En environnement réel, n'oubliez pas d'ajouter les IP de vos passerelles VPN ou de vos bureaux. Dans notre infrastructure de TP, les services internes Docker utilisent le sous-réseau du projet ; on l'autorise donc pour conserver l'accès depuis Prometheus et Nginx d'administration. Voici comment appliquer ce filtrage sur le frontend des statistiques :

```
frontend admin
    bind :8404
    # Définition de l'ACL basée sur l'IP source (src)
    acl network_allowed src 172.20.0.0/16
    # On refuse l'accès si l'IP n'est pas autorisée
```

```
    http-request deny if !network_allowed

    stats enable
    stats uri /
    stats refresh 5s
```

Astuce de factorisation : Si vous avez une liste d'IP que vous souhaitez réutiliser dans plusieurs frontends, évitez de les copier-coller. Utilisez un fichier externe (ex. : /etc/haproxy/whitelist.lst) contenant une IP ou un CIDR par ligne. Votre ACL devient alors simple et centralisée :

```
acl network_allowed src -f /etc/haproxy/whitelist.lst
```

Sécurité et Durcissement

Protection du cluster contre les abus et les attaques réseau.

Protection contre le DoS et le DDoS

Configurez le frontend http_in pour limiter chaque IP source à un maximum de 15 connexions TCP simultanées (couche 4). Une fois la configuration appliquée, tentez d'ouvrir plusieurs connexions en parallèle pour vérifier que HAProxy rejette les suivantes.

Vérification : Simulez une charge avec une boucle shell pour ouvrir de nombreuses connexions en arrière-plan :

```
for i in $(seq 1 20); do curl -s http://localhost & done
```

Regardez vos logs HAProxy ou le dashboard : les connexions au-delà de 15 doivent être rejetées immédiatement.

DoS vs DDoS : quelle différence ?

- **DoS (Denial of Service) :** l'attaque provient d'une seule machine. HAProxy peut facilement la bloquer en limitant les connexions par IP.
- **DDoS (Distributed DoS) :** l'attaque provient de milliers de sources (botnet). Bloquer une IP ne suffit plus, car le trafic est distribué.

Recommandations de HAProxy (Lire l'article officiel) :

- **Défense en profondeur :** pour les attaques volumétriques massives (couches 3 et 4), utilisez des services de protection Cloud (Cloudflare, AWS Shield) en amont.
- **Filtrage couche 7 :** utilisez HAProxy pour identifier des signatures d'attaque spécifiques dans les en-têtes ou les URL via les ACL.
- **Stick Tables :** elles restent essentielles pour limiter l'impact de chaque nœud du botnet sur vos ressources applicatives.

Correction :

Mode opératoire :

1. Modifiez le frontend http_in existant pour intégrer le suivi des connexions :

```
frontend http_in
    bind :80
    # Création de la table de suivi
    stick-table type ip size 100k expire 30s store conn_cur
    # Suivi de l'IP source
    tcp-request connection track-sc0 src
    # Rejet au-delà de 15 connexions
    tcp-request connection reject if { sc0_conn_cur gt 15 }
    default_backend web_servers
```

2. Rechargez HAProxy.

Protection contre le Slowloris

L'attaque Slowloris sature le serveur en envoyant des en-têtes HTTP très lentement. Implémentez une protection en limitant le temps de réception des en-têtes à 5 secondes.

Simulation d'attaque : Lancez cette commande **avant** la configuration (la connexion restera bloquée), puis **après** (HAProxy coupera la connexion après 5 s) :

```
curl -X GET "http://localhost:80/" \
     -H "X-Slow: $(head -c 5000      --limit-rate 1 \
     --verbose
```

Explication des paramètres :

- -H "X-Slow: ..." : génère un en-tête massif de 5000 octets (via head et tr) pour forcer un envoi long.
- --limit-rate 1 : bride l'envoi à 1 octet/s. Sans protection, la connexion resterait ouverte plus d'une heure.
- --verbose : permet d'observer en direct le moment exact où HAProxy interrompt la session.

Recommandations de HAProxy (Lire l'article officiel**) :** Pour une protection optimale, HAProxy préconise de combiner plusieurs réglages :

- timeout http-request 5s : limite le temps d'attente des en-têtes (défense principale).
- timeout http-keep-alive 1s : libère rapidement les slots de connexion inactifs.
- maxconn : limite le nombre global de connexions pour protéger les descripteurs de fichiers du système contre l'épuisement.

Correction :

Mode opératoire :

1. Dans la section defaults (ou le frontend), ajoutez la directive de timeout :

```
defaults
    # Coupe la connexion si les en-têtes mettent plus de 5 s à arriver
    timeout http-request 5s
```

2. Rechargez HAProxy et relancez la commande de simulation. Vous devriez voir un message Empty reply from server ou une fermeture de socket après environ 5 secondes.

Limiter les requêtes (Rate Limiting)

Mettez en place un quota de 50 requêtes maximum toutes les 10 secondes par IP. Testez le dépassement du seuil et vérifiez que HAProxy retourne bien un code d'erreur HTTP 429.

Vérification : Exécutez une rafale de requêtes curl pour saturer le quota :

```
for i in $(seq 1 60); do curl -I http://localhost; done
```

Vous devez observer des réponses HTTP/1.1 429 Too Many Requests après la 50e requête.

Correction :

Mode opératoire :

1. Mettez à jour la stick-table du frontend http_in pour suivre à la fois les connexions en cours et le débit de requêtes (http_req_rate) :

```
frontend http_in
    bind :80
    # On conserve conn_cur et on ajoute le stockage du débit sur 10s
    stick-table type ip size 100k expire 30s store conn_cur,http_req_rate(10s)
    tcp-request connection track-sc0 src
    tcp-request connection reject if { sc0_conn_cur gt 15 }
    # Suivi au niveau HTTP (sc1)
    http-request track-sc1 src
    # Blocage avec code 429 (Too Many Requests)
    http-request deny deny_status 429 if { sc1_http_req_rate gt 50 }
    default_backend web_servers
```

2. Rechargez la configuration.

Modules Anti-bot et Détection du scraping

Bloquez les aspirateurs de données (scrapers) et les bots malveillants en identifiant leurs signatures User-Agent suspectes.

Simulation de bot : Tentez d'accéder au site en simulant un bot connu via curl :

```
curl -I -H "User-Agent: Googlebot-Scraper" http://localhost/
```

Correction :

Mode opératoire :

1. Ajoutez ces ACL et cette règle dans le frontend http_in existant, en conservant les protections déjà configurées :

```
    # Détection de mots-clés dans le User-Agent (insensible à la casse)
    acl is_bot hdr_sub(user-agent) -i bot crawler scraper spider
    # Rejet des requêtes identifiées comme bots
    http-request deny deny_status 403 if is_bot
```

2. Rechargez HAProxy et vérifiez que votre requête curl avec l'en-tête suspect est désormais rejetée par un code 403.

Terminaison silencieuse de requête

Le 'Silent Drop' est une technique de défense avancée : au lieu d'envoyer un code d'erreur (403/429) qui confirme au pirate l'existence d'une protection, HAProxy ignore simplement la requête. L'attaquant gaspille ses ressources à attendre une réponse qui ne viendra jamais.

> **Vérification :** Testez l'accès à un chemin interdit et constatez que curl reste en attente (hang) jusqu'au timeout :
>
> ```
> curl -v http://localhost/admin/config
> ```

Correction :

Mode opératoire :

1. Ajoutez l'action silent-drop dans le frontend http_in existant pour les chemins ou flux interdits :

```
    acl is_forbidden path_beg /admin
    # Fermeture de la connexion sans envoyer le moindre bit de réponse
    http-request silent-drop if is_forbidden
```

2. Rechargez la configuration. Le client ne recevra aucune donnée (pas même un TCP RST), ce qui est particulièrement efficace contre les scanners automatiques.

Observabilité

Mise en place du monitoring, de l'exposition des métriques et de la télémétrie.

Stats Dashboard

Vérifiez que le point d'entrée d'administration configuré plus tôt expose toujours le dashboard de statistiques sur la racine du port 8404.

Correction :

```
frontend admin
    bind :8404
    stats enable
    stats uri /
    stats refresh 5s
```

Prometheus : Exposition des métriques

Configurez HAProxy pour exposer les métriques de performance au format Prometheus sur l'URI /metrics. Cela permet à un serveur Prometheus de scraper l'état du load balancer.

> **Vérification :** Testez la récupération des métriques via curl :
>
> ```
> curl http://localhost:8404/metrics
> ```
>
> Vous devriez voir défiler les métriques au format texte brut (HELP et TYPE).

Correction :

Mode opératoire :

1. Dans votre bloc frontend admin, utilisez la directive http-request use-service :

```
frontend admin
    bind :8404
    stats enable
    stats uri /
    stats refresh 5s
    # Exposition des métriques pour Prometheus
    # Cette règle intercepte les requêtes sur l'URI '/metrics' et renvoie le format Prometheus
    http-request use-service prometheus-exporter if { path /metrics }
```

2. Rechargez la configuration. HAProxy sert désormais les métriques sur le même port que le dashboard.

Prise en charge d'OpenTelemetry (OTel)

Configurez l'exportation native de données de télémétrie vers un collecteur OpenTelemetry.

Note : La prise en charge native d'OpenTelemetry est disponible à partir de la version **3.4** de HAProxy.

Correction :

Mode opératoire :

1. Déclarez l'exportateur OTel dans la section global :

```
global
    otel-exporter my-collector
        endpoint otel-collector:4317
```

2. Activez l'envoi des traces dans votre frontend pour propager le contexte :

```
frontend http_in
    http-request otel-send-trace
```

Quizz (QCM)

Information importante
Pour valider vos acquis et consulter les corrections interactives de ce quizz, nous vous invitons à vous connecter sur notre plateforme en ligne : elearning.rousseltm.fr.

Q1 : Que signifie le 'HA' dans HAProxy ?

1. High Accuracy
2. High Availability
3. Hyper Access
4. Home Automation

? Indice : *HAProxy signifie High Availability Proxy.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q2 : À quel niveau du modèle OSI HAProxy peut-il opérer ?

1. Couche 2 et 3
2. Couche 4 uniquement
3. Couche 7 uniquement
4. Couche 4 et Couche 7

? Indice : *HAProxy est un load balancer capable de traiter le trafic au niveau TCP (Layer 4) et HTTP (Layer 7).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q3 : Quelle section du fichier de configuration définit les paramètres de sécurité du processus (user, group) ?

1. defaults
2. frontend
3. global
4. backend

? Indice : *La section 'global' gère les paramètres liés au processus système lui-même.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q4 : Quelle commande permet de tester la syntaxe du fichier haproxy.cfg ?

1. haproxy -check
2. haproxy -c -f /etc/haproxy/haproxy.cfg
3. haproxy -t
4. systemctl test haproxy

? Indice : *L'option -c (check) combinée à -f (file) permet de valider la configuration.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q5 : Dans quel bloc définit-on l'adresse IP et le port d'écoute ?

1. backend
2. global
3. defaults
4. frontend

? Indice : *Le 'frontend' est la porte d'entrée où l'on utilise la directive 'bind'.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q6 : Quel algorithme de répartition distribue les requêtes tour à tour ?

1. leastconn
2. source
3. roundrobin
4. uri

? Indice : *Le Round Robin est l'algorithme cyclique par défaut jusqu'à la version 3.3 qui est passé à 'random'*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q7 : Quel algorithme est recommandé pour les sessions longues comme SQL ou WebSocket ?

1. roundrobin
2. leastconn
3. static-rr
4. first

? Indice : *Leastconn dirige le trafic vers le serveur ayant le moins de connexions actives, idéal pour les sessions persistantes.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q8 : Quelle directive permet d'activer les health checks sur un serveur ?

1. option health
2. check
3. verify
4. test-alive

? Indice : *On ajoute le mot-clé 'check' à la fin de la ligne 'server'.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q9 : Que fait la directive 'option httpchk' ?

1. Elle active le mode HTTP

2. Elle effectue une requête HTTP pour vérifier la santé du backend
3. Elle logue les requêtes HTTP
4. Elle redirige le HTTP vers le HTTPS

? Indice : *Elle permet de passer d'un simple check TCP à une vérification applicative HTTP.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q10 : Comment appelle-t-on le fait de déchiffrer le SSL sur HAProxy pour envoyer du HTTP clair aux serveurs ?

1. SSL Passthrough
2. SSL Bridging
3. SSL Termination (Offloading)
4. SSL Encrypt

? Indice : *La terminaison SSL décharge les serveurs web du calcul lié au chiffrement.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q11 : Quel mot-clé est utilisé pour créer une règle de routage basée sur l'URL ?

1. RULE
2. IF
3. ACL
4. MATCH

? Indice : *ACL (Access Control List) est le moteur de décision d'HAProxy.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q12 : Quelle ACL permet de vérifier si le chemin commence par '/api' ?

1. path_end /api
2. url_beg /api
3. path_beg /api
4. path_is /api

? Indice : *path_beg vérifie le début du chemin (path).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q13 : Quel service HAProxy permet de gérer centralement plusieurs instances Enterprise ?

1. HAProxy OSS
2. HAProxy Fusion
3. HAProxy Edge
4. HAProxy One

? Indice : *HAProxy Fusion est le Control Plane pour la gestion multi-clusters.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q14 : Dans HAProxy One, qu'est-ce que HAProxy Edge ?

1. Un logiciel de monitoring
2. Un réseau de distribution global (ADN/CDN)
3. Une version pour Raspberry Pi
4. Un plugin pour Kubernetes

? Indice : *HAProxy Edge est la couche ADN distribuée mondialement.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q15 : Quelle directive dans 'global' permet d'utiliser plusieurs coeurs CPU ?

1. nbproc
2. nbthread
3. cpu-map
4. multi-core

? Indice : *nbthread est la méthode moderne recommandée pour le multi-threading.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q16 : Comment forcer la redirection HTTP vers HTTPS ?

1. redirect scheme https if !{ ssl_fc }
2. force https
3. option https-only
4. http-request redirect https

? Indice : *redirect scheme https est la syntaxe classique pour rediriger si la connexion n'est pas sécurisée.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q17 : Que signifie le statut 'L7OK' dans les statistiques ?

1. Le serveur répond en TCP
2. Le check HTTP a réussi
3. Le serveur est en maintenance
4. La couche 7 est désactivée

? Indice : *L7OK indique que le check de niveau 7 (applicatif) est valide.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q18 : Quel port est couramment utilisé pour l'interface de statistiques HAProxy ?

1. 80
2. 443
3. 8404
4. 8080

? Indice : *Bien que configurable, le port 8404 est le standard recommandé.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q19 : Quelle directive permet de lier un utilisateur à un serveur spécifique pendant sa session ?

1. persistence
2. stick-table
3. cookie insert
4. session-bind

? Indice : *'cookie insert' permet de gérer la persistance via un cookie injecté par HAProxy.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q20 : Quelle est la principale différence entre HAProxy OSS et Enterprise ?

1. La performance
2. Le support commercial et les modules de sécurité (WAF)
3. Le langage de programmation
4. La compatibilité Linux

? Indice : *La version Enterprise apporte le support, la stabilité garantie et des modules comme le WAF avancé.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q21 : Que fait le mode 'mode tcp' ?

1. Il analyse les headers HTTP
2. Il fait du load balancing aveugle au niveau transport
3. Il active le pare-feu
4. Il accélère le téléchargement

? Indice : *Le mode TCP traite les flux sans regarder le contenu applicatif (Layer 4).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q22 : Quel mot-clé définit un serveur de secours uniquement utilisé si les autres échouent ?

1. secondary
2. backup
3. failover
4. spare

? Indice : *Le mot-clé 'backup' active le serveur uniquement en cas de panne du pool principal.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q23 : Comment limiter le nombre maximum de connexions globales ?

1. max-limit
2. conn-limit
3. maxconn
4. limit-sessions

? Indice : *La directive 'maxconn' est utilisée aussi bien en global qu'en frontend/backend.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q24 : À quoi sert la 'Runtime API' ?

1. À modifier la configuration sans recharger le service
2. À créer des applications web
3. À mettre à jour HAProxy
4. À générer des certificats

? Indice : *La Runtime API permet de changer des poids, vider des serveurs ou lire des stats dynamiquement.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q25 : Quelle section hérite ses paramètres aux frontends et backends ?

1. global
2. main
3. defaults
4. common

? Indice : *La section 'defaults' évite la répétition des timeouts et options de logs.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q26 : L'algorithme 'source' est utile pour quoi ?

1. Répartir équitablement
2. Garantir qu'une IP client va toujours vers le même serveur
3. Le streaming vidéo
4. Le trafic mobile uniquement

? Indice : *Il crée une persistance basée sur le hachage de l'adresse IP source.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q27 : Quelle directive permet d'ajouter le header 'X-Forwarded-For' ?

1. add-header xff
2. option forwardfor
3. http-request set-xff
4. forward-ip

? Indice : *C'est l'option standard pour transmettre l'IP réelle du client au serveur backend.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q28 : Que signifie 'inter 2s' dans une ligne server ?

1. Timeout de 2 secondes
2. Intervalle de 2 secondes entre deux health checks
3. Attente de 2 secondes avant reconnexion
4. Durée de vie de la session

? Indice : *C'est le délai régulier entre chaque test de santé.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q29 : Quelle directive définit le backend par défaut d'un frontend ?

1. use_backend
2. default_backend
3. backend_link
4. main_backend

? Indice : *C'est la destination utilisée si aucune règle ACL ne matche.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q30 : Peut-on utiliser plusieurs fichiers de configuration avec HAProxy ?

1. Non, un seul fichier haproxy.cfg est possible
2. Oui, en pointant vers un répertoire avec -f
3. Oui, uniquement en version Enterprise
4. Uniquement via des inclusions 'include'

? Indice : *HAProxy peut concaténer plusieurs fichiers passés via l'argument -f ou un dossier.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q31 : Que fait 'stats refresh 5s' ?

1. Redémarre HAProxy toutes les 5s
2. Rafraîchit la page de stats automatiquement toutes les 5s
3. Vérifie les serveurs toutes les 5s
4. Nettoie les logs

? Indice : *C'est une option de confort pour le monitoring temps réel via le dashboard.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q32 : Quelle est la commande pour recharger HAProxy sans interruption ?

1. systemctl restart haproxy
2. systemctl reload haproxy

3. haproxy -update
4. service haproxy stop && start

? Indice : *Le 'reload' effectue un hot-reconfigure en transférant les sockets aux nouveaux processus.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q33 : À quoi sert 'fall 3' dans un check ?

1. Le serveur tombe après 3 secondes
2. Le serveur est marqué DOWN après 3 échecs consécutifs
3. Il faut 3 serveurs pour que ça marche
4. Le poids du serveur est divisé par 3

? Indice : *C'est le seuil de tolérance avant de considérer le serveur comme défaillant.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q34 : Et 'rise 2' ?

1. Augmente le poids
2. Le serveur revient UP après 2 succès consécutifs
3. Délai de démarrage
4. Nombre de connexions

? Indice : *C'est le nombre de tests réussis nécessaires pour réintégrer un serveur dans le pool.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q35 : L'option 'http-server-close' sert à quoi ?

1. Fermer le serveur
2. Désactiver le Keep-Alive côté serveur
3. Bloquer les requêtes
4. Optimiser le SSL

? Indice : *Elle permet de libérer les connexions serveurs plus rapidement.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q36 : Quelle directive permet de bloquer une adresse IP ?

1. block ip
2. http-request deny if { src 1.2.3.4 }
3. acl ban ip
4. deny_access 1.2.3.4

? Indice : *http-request deny est l'outil standard pour le filtrage de requêtes.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q37 : Comment HAProxy Enterprise gère-t-il les mises à jour de sécurité ?

1. Il ne le fait pas
2. Via des correctifs rétroportés sur des versions LTS
3. Uniquement en changeant de version majeure
4. Automatiquement sans prévenir

? Indice : *Le support Enterprise garantit des correctifs sur des versions stables sur le long terme.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q38 : Qu'est-ce qu'une 'Stick Table' ?

1. Une table en bois
2. Une base de données en mémoire pour suivre des compteurs (IP, sessions, erreurs)
3. Un fichier de log
4. Un algorithme de hachage

? Indice : *C'est une structure de données ultra-rapide en RAM pour le tracking avancé.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q39 : Peut-on faire du load balancing de base de données MySQL avec HAProxy ?

1. Non, uniquement HTTP
2. Oui, en mode TCP (Layer 4)
3. Uniquement avec un plugin spécial
4. Oui, mais sans health check

? Indice : *HAProxy excelle dans le proxying de protocoles binaires comme MySQL ou PostgreSQL.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q40 : La directive 'bind *:443 ssl crt /etc/ssl/cert.pem' fait quoi ?

1. Elle crée un certificat
2. Elle écoute en HTTPS en utilisant le certificat spécifié
3. Elle redirige le 443
4. Elle vérifie la validité du PEM

? Indice : *C'est la syntaxe pour activer la terminaison SSL sur un port.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q41 : Quel est l'avantage de 'nbthread' sur 'nbproc' ?

1. Plus rapide
2. Partage de la mémoire entre les threads pour les Stick Tables
3. Moins de lignes de code
4. Meilleur support Windows

? Indice : *Les threads partagent le même espace mémoire, ce qui simplifie grandement la synchronisation des données.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q42 : Dans les statistiques, que signifie 'Sessions Cur' ?

1. Nombre total de sessions depuis le début
2. Nombre de sessions actuellement actives
3. Vitesse des requêtes
4. Nombre de sessions perdues

? Indice : *Cur = Current (actuel).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q43 : Quelle option permet d'enregistrer les logs HAProxy vers un serveur Syslog ?

1. log 127.0.0.1 local0
2. syslog-server 127.0.0.1
3. output-logs local0
4. send-logs

? Indice : *HAProxy n'écrit pas dans des fichiers, il envoie vers Syslog (souvent via /dev/log).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q44 : Comment HAProxy détecte-t-il qu'une application renvoie une erreur 500 ?

1. Automatiquement
2. Via 'http-check expect status 200'
3. En analysant les logs
4. Il ne peut pas

? Indice : *On peut spécifier les codes de succès attendus pour valider la santé.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q45 : L'ALOHA d'HAProxy Technologies est :

1. Une boisson
2. Une appliance (matérielle ou virtuelle) prête à l'emploi
3. Un nouvel algorithme
4. Un script de déploiement

? Indice : *ALOHA est la solution Load Balancer 'Plug and Play' d'HAProxy.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q46 : Quel mode permet d'utiliser HAProxy comme un Ingress Controller Kubernetes ?

1. Mode Standalone
2. HAProxy Ingress Controller
3. Mode Sidecar
4. Gateway Mode

? Indice : *C'est un composant spécifique qui traduit les ressources Ingress K8s en configuration HAProxy.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q47 : Que fait la directive 'balance leastconn' ?

1. Prend le serveur le moins cher
2. Prend le serveur avec le moins de connexions actives
3. Prend le dernier serveur utilisé
4. Prend le serveur le plus rapide

? Indice : *C'est l'algorithme dynamique le plus utilisé pour l'équilibrage de charge intelligent.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q48 : Peut-on modifier le poids (weight) d'un serveur sans redémarrer ?

1. Non
2. Oui, via la Runtime API
3. Oui, en modifiant le fichier
4. Uniquement en version 3.0+

? Indice : *La Runtime API permet de modifier les poids dynamiquement (ex: pour du Blue/Green deployment).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q49 : Quelle directive permet d'activer le monitoring Prometheus ?

1. prometheus-enable
2. http-request use-service prometheus-exporter
3. stats prometheus
4. option metrics

? Indice : *HAProxy expose nativement un exporteur Prometheus via cette directive de service.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q50 : Quelle est la durée de vie par défaut d'une version LTS chez HAProxy Technologies ?

1. 6 mois
2. 1 an
3. Entre 2 et 5 ans
4. Indéfinie

? Indice : *Les versions LTS (Long Term Support) offrent plusieurs années de tranquillité.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q51 : Une section 'listen' est utile pour :

1. Remplacer le global
2. Combiner frontend et backend dans un seul bloc
3. Écouter de la musique
4. Rien, elle est dépréciée

? Indice : *Elle simplifie les configurations simples (ex: TCP proxy ou stats).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q52 : Que signifie 'timeout client 50s' ?

1. Le client doit répondre en 50s
2. Inactivité maximale du client avant coupure
3. Temps de chargement de la page
4. Délai avant retry

? Indice : *C'est le délai d'inactivité autorisé côté client.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q53 : L'option 'redispatch' sert à quoi ?

1. Envoyer à nouveau la requête
2. Permettre de renvoyer une requête vers un autre serveur si celui choisi tombe
3. Nettoyer le cache
4. Changer d'algorithme

? Indice : *Elle garantit la continuité de service si un serveur devient indisponible juste après la sélection.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q54 : Comment définir une ACL qui vérifie si l'utilisateur utilise un mobile ?

1. acl is_mobile user_agent -i mobile
2. acl is_mobile hdr_sub(user-agent) -i mobile
3. match mobile
4. check device mobile

? Indice : *On inspecte le header 'User-Agent' pour y chercher la chaîne 'mobile'.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q55 : Le flag '-D' au lancement de HAProxy signifie :

1. Debug
2. Daemon (lance en arrière-plan)

3. Delete
4. Default

? Indice : *Indispensable pour que le processus tourne en tâche de fond.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q56 : Que se passe-t-il si aucun backend n'est disponible ?

1. HAProxy s'arrête
2. HAProxy renvoie une erreur 503 Service Unavailable
3. La requête attend indéfiniment
4. Le trafic va vers le frontend

? Indice : *HAProxy génère lui-même la page d'erreur 503.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q57 : Quelle directive permet de personnaliser les pages d'erreur ?

1. error-page
2. errorfile
3. custom-error
4. http-error

? Indice : *'errorfile' permet de pointer vers un fichier HTML personnalisé pour chaque code HTTP.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q58 : L'option 'log-format' permet de :

1. Changer la couleur des logs
2. Personnaliser les informations présentes dans chaque ligne de log
3. Compresser les logs
4. Désactiver les logs

? Indice : *Essentiel pour extraire des headers spécifiques ou des temps de réponse précis.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q59 : Que fait 'balance uri' ?

1. Répartit selon l'URL pour optimiser le cache des backends
2. Répartit au hasard
3. Vérifie si l'URL est valide
4. Bloque certaines URLs

? Indice : *Idéal pour les architectures de serveurs de cache (comme Varnish).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q60 : Quel header est utilisé par défaut pour la persistance par cookie ?

1. PHPSESSID
2. SERVERID
3. SRV
4. HA_COOKIE

? Indice : *SERVERID est le nom conventionnel souvent utilisé dans la documentation.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q61 : Dans HAProxy One, qu'est-ce que Fusion ?

1. Un moteur de rendu
2. Le plan de contrôle centralisé
3. Le nom du support technique
4. Un module de compression

? Indice : *Fusion permet de piloter et monitorer tout son parc HAProxy Enterprise via une seule interface.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q62 : Que signifie 'SSL Passthrough' ?

1. HAProxy ne déchiffre pas, il transmet le flux chiffré au backend
2. HAProxy supprime le SSL
3. HAProxy génère un faux certificat
4. Le SSL est optionnel

? Indice : *HAProxy travaille alors en Layer 4 (TCP) et ne voit pas le contenu HTTP.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q63 : Quelle directive permet de limiter le débit (Rate Limiting) ?

1. limit-rate
2. http-request track-sc0 src
3. rate-limit
4. conn-limit

? Indice : *Combiné avec les Stick Tables, cela permet de compter et bloquer les excès de requêtes.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q64 : Le 'WAF' intégré à HAProxy Enterprise sert à :

1. Accélérer le réseau
2. Protéger contre les attaques applicatives (SQLi, XSS)
3. Gérer les mots de passe
4. Remplacer l'antivirus

? Indice : *Web Application Firewall : une barrière de sécurité pour filtrer le trafic malveillant.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q65 : Quelle est la commande pour vérifier la version de HAProxy ?

1. haproxy -v
2. haproxy --version
3. haproxy version
4. cat /proc/haproxy/version

? Indice : *L'option -v affiche la version et les options de compilation.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q66 : À quoi sert 'stats auth admin:password' ?

1. À créer un utilisateur système
2. À protéger l'accès au dashboard de stats par mot de passe
3. À authentifier les serveurs
4. À rien

? Indice : *Sécurité élémentaire pour ne pas exposer ses stats à tout le monde.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q67 : Quelle directive 'global' permet de réduire les privilèges après le démarrage ?

1. user / group
2. reduce-privs
3. low-priv
4. chroot

? Indice : *HAProxy démarre en root pour ouvrir les ports (80/443) puis bascule sur l'utilisateur défini.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q68 : Le 'chroot' sert à quoi ?

1. Accélérer le disque
2. Isoler HAProxy dans un répertoire vide pour limiter l'impact d'une faille
3. Changer de racine
4. Optimiser les logs

? Indice : *C'est une mesure de sécurité système (jail).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q69 : Peut-on utiliser HAProxy comme un proxy sortant (Forward Proxy) ?

1. Oui
2. Non, c'est un Reverse Proxy uniquement
3. Seulement en version 1.5
4. Seulement avec Squid

? Indice : *HAProxy est conçu pour être un Reverse Proxy (devant les serveurs).*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q70 : Quelle directive permet d'activer la compression GZIP ?

1. gzip enable
2. compression algo gzip
3. http-request compress
4. option zip

? Indice : *Elle permet de réduire la bande passante consommée pour les fichiers texte.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q71 : Un 'Pod' dans Kubernetes peut contenir HAProxy via :

1. Une Ingress
2. Un Sidecar
3. Un Service
4. Les trois sont possibles

? Indice : *HAProxy est très flexible dans les architectures Kubernetes.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q72 : Laquelle de ces affirmations sur HAProxy One est vraie ?

1. C'est un moteur de base de données
2. C'est une plateforme SaaS unifiée pour l'écosystème HAProxy
3. C'est une version gratuite limitée
4. C'est un plugin navigateur

? Indice : *HAProxy One unifie Enterprise, Fusion et Edge.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q73 : L'option 'http-buffer-request' sert à :

1. Stocker la requête en mémoire
2. Attendre la fin du body avant de router (utile pour le WAF)
3. Accélérer le CPU
4. Limiter les logs

? Indice : *Indispensable pour que le WAF puisse inspecter l'intégralité d'un POST.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q74 : Que fait 'http-request set-header Server MyProxy' ?

1. Change le nom du serveur backend
2. Ajoute ou modifie le header HTTP 'Server' dans la requête
3. Définit le nom de l'instance
4. C'est une erreur de syntaxe

? Indice : *On peut manipuler tous les headers à la volée.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q75 : Le protocole PROXY (Proxy Protocol) sert à :

1. Accélérer le réseau
2. Transmettre l'IP client réelle même en mode TCP (Layer 4)
3. Authentifier les serveurs
4. Remplacer le HTTPS

? Indice : *Inventé par HAProxy, il est devenu un standard pour garder la visibilité sur l'IP source.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q76 : Quelle version de HAProxy a introduit le support natif de Prometheus ?

1. 1.0
2. 2.0
3. 1.5
4. 2.8

? Indice : *La branche 2.x a apporté énormément de fonctionnalités d'observabilité.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q77 : La directive 'external-check' permet de :

1. Vérifier le DNS
2. Lancer un script Bash/Python personnalisé pour tester un service complexe
3. Faire des checks depuis Internet
4. Vérifier le matériel

? Indice : *Très utile quand un simple check TCP ou HTTP ne suffit pas.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q78 : Dans le dashboard de stats, que signifie la couleur rouge pour un serveur ?

1. Trop de trafic
2. Serveur DOWN (échec des health checks)
3. Serveur en maintenance
4. Certificat expiré

? Indice : *Le rouge indique une panne détectée.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q79 : Et la couleur orange ?

1. Surcharge
2. Maintenance (drain mode)
3. Certificat bientôt expiré
4. Serveur en cours de démarrage

? Indice : *Généralement utilisé pour le mode Maintenance ou Drain.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q80 : Quel composant d'HAProxy Technologies est spécialisé dans l'absorption des attaques DDoS globales ?

1. HAProxy OSS
2. HAProxy Fusion
3. HAProxy Edge
4. ALOHA

? Indice : *HAProxy Edge utilise son réseau distribué pour absorber le trafic malveillant avant qu'il n'atteigne vos serveurs.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q81 : Combien de frontends peut-on définir dans un fichier ?

1. Un seul
2. Maximum 10
3. Illimité
4. Autant que de coeurs CPU

? Indice : *HAProxy est extrêmement scalable.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q82 : Que fait 'option abortonclose' ?

1. Ferme HAProxy
2. Arrête le traitement de la requête si le client ferme la connexion prématurément
3. Supprime les logs
4. Ignore les erreurs

? Indice : *Permet de ne pas gaspiller de ressources sur les serveurs si le client est déjà parti.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q83 : Le mot-clé 'ssl_fc' dans une ACL vérifie :

1. La vitesse SSL
2. Si la connexion actuelle est chiffrée (Front Connection)
3. La validité du certificat
4. Le nom du domaine

? Indice : *SSL Front Connection.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q84 : Quelle directive permet d'inclure un autre fichier ?

1. include
2. import
3. Pas possible nativement via une directive interne
4. f-include

? Indice : *HAProxy utilise l'argument de ligne de commande -f pour inclure plusieurs fichiers, il n'y a pas de directive 'include' à l'intérieur du .cfg.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q85 : Que signifie 'L4OK' ?

1. Le check TCP a réussi
2. Le check HTTP a réussi
3. Le serveur est puissant
4. C'est une erreur

? Indice : *Layer 4 OK.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q86 : Quel est le rôle d'un 'Map' file ?

1. Afficher une carte
2. Stocker des milliers de correspondances clé/valeur pour des ACLs ultra-rapides
3. Définir les routes réseau
4. Localiser les clients

? Indice : *C'est la méthode performante pour gérer du routage multi-domaines massif.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q87 : L'option 'dontlog-normal' sert à :

1. Ne rien logger
2. Ne logger que les erreurs (réduire le volume de logs)
3. Logger en binaire
4. Logger uniquement sur disque

? Indice : *Utile pour économiser de l'espace disque si tout va bien.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q88 : Que fait 'http-request capture req.hdr(Referer) len 64' ?

1. Bloque le Referer
2. Enregistre le Referer dans les logs (capture)
3. Limite la taille des requêtes
4. Compte les referers

? Indice : *Permet d'ajouter des informations spécifiques dans les logs HAProxy.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q89 : Peut-on faire du SSL Passthrough et de la Terminaison SSL sur le même HAProxy ?

1. Non
2. Oui, sur des frontends différents
3. Uniquement avec Fusion
4. Seulement en mode UDP

? Indice : *HAProxy est multi-usage.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q90 : Quelle directive permet d'activer le protocole HTTP/2 ?

1. option http2
2. alpn h2,http/1.1 (sur la ligne bind)
3. mode h2
4. http2-enable

? Indice : *L'ALPN permet au navigateur et à HAProxy de négocier le passage en HTTP/2.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q91 : Que signifie 'HAPEE' ?

1. Happy HAProxy
2. HAProxy Enterprise Edition
3. HAProxy Enhanced
4. C'est une blague

? Indice : *C'est l'acronyme utilisé pour désigner la version Enterprise.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q92 : Comment HAProxy One facilite-t-il la sécurité ?

1. En installant un antivirus
2. En centralisant les politiques de WAF et de gestion des bots
3. En chiffrant le disque dur
4. En limitant le nombre de frontends

? Indice : *Une plateforme unifiée permet d'appliquer la sécurité partout de façon cohérente.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q93 : La directive 'maxconn' dans une ligne server d'un backend sert à :

1. Limiter les connexions client
2. Limiter le nombre de requêtes simultanées envoyées à CE serveur précis
3. Définir la bande passante
4. C'est une erreur

? Indice : *Crucial pour protéger un serveur backend contre la saturation.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q94 : Le mode 'drain' sur un serveur signifie :

1. Le serveur est supprimé
2. On ne lui envoie plus de NOUVEAUX clients, mais on laisse finir les sessions en cours
3. Le serveur est vidé de ses fichiers
4. C'est un check de disque

? Indice : *Indispensable pour faire une maintenance propre sans déconnecter les gens.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q95 : Quelle est la licence de HAProxy OSS ?

1. Propriétaire
2. GPLv2
3. MIT
4. Apache 2.0

? Indice : *HAProxy est un projet Open Source historique sous licence GPL.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q96 : Le créateur de HAProxy est :

1. Linus Torvalds
2. Willy Tarreau
3. Brendan Gregg
4. Solomon Hykes

? Indice : *Willy Tarreau maintient toujours activement le projet.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q97 : Dans Kubernetes, quelle annotation active HAProxy ?

1. kubernetes.io/ingress.class: haproxy
2. proxy: true
3. use-haproxy: yes
4. service.type: lb

? Indice : *C'est la méthode standard pour sélectionner l'Ingress Controller.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q98 : L'écosystème HAProxy One est-il uniquement pour le Cloud ?

1. Oui
2. Non, il supporte le On-premise, le Multi-cloud et l'Hybride
3. Uniquement AWS
4. Uniquement Azure

? Indice : *HAProxy est conçu pour être agnostique vis-à-vis de l'infrastructure.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q99 : Quelle directive permet de compresser les réponses ?

1. option compress
2. compression algo gzip
3. http-request gzip
4. zip-on

? Indice : *On définit l'algorithme et les types MIME à compresser.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Q100 : Félicitations, vous avez fini ! Quel est le dernier conseil d'HAProxy ?

1. Testez toujours votre config avec -c
2. Utilisez le port 80 pour tout
3. Ne lisez pas les logs
4. Redémarrez toutes les heures

? Indice : *La validation de configuration est la base de toute mise en production sereine.*

Articles pour approfondir :

- Anatomie de la configuration HAProxy

Glossaire

ACL (Access Control List)

Liste de contrôle d'accès définissant les permissions accordées sur des ressources. Dans HAProxy, les ACL sont des conditions flexibles permettant de prendre des décisions de routage dynamique basées sur le flux (URL, headers, IP source).

Backend

Section de la configuration HAProxy définissant un pool de serveurs cibles vers lesquels le trafic est redirigé. C'est ici que l'on configure l'algorithme de répartition et les vérifications de santé.

DoS / DDoS

Attaque visant à rendre un service indisponible en submergeant le serveur ou le réseau de trafic. On parle de DDoS (Distributed Denial of Service) lorsque l'attaque provient de multiples sources simultanées (botnets).

Frontend

Section de la configuration HAProxy qui définit comment les requêtes sont reçues du côté client (adresse IP, port, certificats SSL) avant d'être envoyées vers un Backend.

GSLB (Global Server

Health Check

Mécanisme automatisé permettant au load balancer de tester périodiquement la disponibilité des serveurs afin d'exclure temporairement ceux qui sont en panne du pool actif.

Load Balancing

Dispositif qui distribue le trafic applicatif sur un ensemble de serveurs afin d'optimiser l'usage des ressources, de minimiser les temps de réponse et d'assurer la haute disponibilité.

Slowloris

Type d'attaque par déni de service (DoS) qui consiste à ouvrir de nombreuses connexions HTTP et à les maintenir ouvertes le plus longtemps possible en envoyant les requêtes très lentement. Cela finit par saturer la table des connexions du serveur.

SSL / TLS

Protocoles de sécurisation des échanges sur internet. TLS (Transport Layer Security) est le successeur moderne de SSL (Secure Sockets Layer). Ils permettent de chiffrer les données, d'authentifier les serveurs et de garantir l'intégrité des messages.

SSL Offloading

Load Balancing)

Technique de répartition de charge à l'échelle mondiale, utilisant généralement le DNS pour diriger les utilisateurs vers le datacenter le plus proche géographiquement ou le moins chargé.

HAProxy

Logiciel de load balancing et de proxying haute performance pour TCP et HTTP. HAProxy est le standard de l'industrie pour la gestion de trafic à très haute disponibilité.

Processus consistant à déléguer le déchiffrement du trafic HTTPS au load balancer pour libérer de la puissance CPU sur les serveurs d'application.

Stickiness

Mécanisme garantissant qu'un utilisateur est systématiquement redirigé vers le même serveur backend durant toute sa session, indispensable pour les applications stockant l'état en local.

MERCI

Nous espérons que ce support vous sera utile pour votre montée en compétences.

Retrouvez toutes nos formations sur :
https://elearning.rousseltm.fr/