Engineer

Équilibreur de charge

⚖ Serveur répartissant les requêtes entre plusieurs autres

Ce que c’est

Un équilibreur de charge est un serveur qui reçoit les requêtes des clients et les répartit entre plusieurs autres serveurs accomplissant le même travail : aucune machine ne porte ainsi toute la charge, et la perte de l’une n’emporte pas le service. Les configurations et schémas l’abrègent souvent en LB, et son activité s’appelle équilibrage de charge. Les clients ne voient qu’une adresse, celle de l’équilibreur. Les serveurs situés derrière lui forment un pool, que nginx appelle groupe upstream et HAProxy backend.

Chaque équilibreur prend deux décisions par requête : quels serveurs sont aptes à la recevoir, et lequel d’entre eux la reçoit. La seconde est la méthode de répartition. Le tourniquet transmet les requêtes à tour de rôle ; nginx l’utilise sans autre configuration. La méthode du plus petit nombre de connexions envoie la prochaine requête au serveur ayant le moins de connexions actives : elle équilibre la charge lorsque certaines requêtes durent beaucoup plus longtemps que d’autres. Le manuel de HAProxy la recommande pour les longues sessions comme LDAP ou SQL et la juge peu adaptée aux courtes, comme HTTP. Le hachage transforme une clé, telle l’adresse IP du client ou l’URL demandée, en choix de serveur : la même clé aboutit donc toujours à la même machine. nginx propose aussi une méthode aléatoire pouvant tirer deux serveurs au hasard et envoyer la requête au moins occupé. Les poids favorisent les machines plus grandes : dans l’exemple du guide nginx, un serveur avec `weight=3` et deux au poids par défaut de 1 reçoivent respectivement trois, une et une des cinq requêtes.

Les équilibreurs diffèrent aussi par la part du trafic qu’ils lisent. Le `mode tcp` de HAProxy, valeur par défaut, relie client et serveur par une connexion bidirectionnelle simultanée sans examiner la couche 7 : il convient à TLS transmis directement, SSH ou SMTP. En `mode http`, il analyse chaque requête avant de choisir un serveur, rejette celles qui enfreignent le protocole et peut router selon

le chemin ou l'en-tête. Un équilibreur HTTP termine la connexion du client et en ouvre une vers le serveur : l'application ne voit donc que l'équilibreur comme client, à moins qu'il transmette l'adresse d'origine, généralement dans `X-Forwarded-For`. C'est aussi pourquoi un équilibreur et un proxy inverse sont si souvent le même processus : nginx assure les deux, et son guide décrit l'équilibrage comme une partie de son implémentation de proxy inverse.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un équilibreur de charge ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des serveurs dimensionnés selon vos usages réels, migrés vers le cloud ou hors du cloud, avec une estimation préalable de la facture mensuelle.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 20 applications Docker sur Google Cloud et AWS — 20 applications conteneurisées déployées et surveillées

20 applications Docker sur Google Cloud et AWS Contactez-nous →

Pourquoi c'est important

Un serveur a un plafond. Lorsqu'un service le dépasse, un équilibreur permet à l'équipe d'ajouter une deuxième puis une troisième copie plutôt que d'acheter une machine plus grande, et chaque copie augmente la capacité sans que les clients sachent que le pool a changé. La maintenance devient aussi une opération courante : retirer un serveur de la rotation, le mettre à jour, le vérifier, le réintégrer, puis passer au suivant, pendant que les autres continuent de servir les utilisateurs.

La panne d'un serveur devient aussi un événement plus limité qu'une interruption du service. Lorsqu'un serveur ne répond plus, l'équilibreur cesse de le choisir et les autres absorbent sa part. Cela ne fonctionne que s'ils ont la réserve nécessaire. Trois serveurs utilisés chacun à 80 % ne peuvent pas se couvrir mutuellement : la perte de l'un impose à chacun des deux autres 120 % du travail d'un serveur.

Un équilibreur modifie la façon d'écrire l'application. Avec le tourniquet ou le plus petit nombre de connexions, précise le guide nginx, les requêtes consécutives d'un client peuvent atteindre des serveurs différents. Tout ce qu'un serveur garde en mémoire ou sur son disque entre requêtes — session de connexion, fichier téléversé attendant son traitement, cache du panier — manque lorsque la suivante arrive ailleurs. L'application conserve alors cet état à

un endroit accessible à tous, base de données ou cache partagé, ou l'équilibreur fixe chaque client sur un serveur, ce que fait `ip_hash` de nginx sauf lorsque ce serveur est indisponible.

Enfin, l'équilibreur devient lui-même la machine par laquelle passe chaque requête. Dix serveurs applicatifs derrière un équilibreur unique ont un point de défaillance unique : il a simplement changé d'endroit. Une installation sérieuse utilise une paire d'équilibreurs avec une adresse pouvant passer de l'un à l'autre ou un équilibreur géré dont le fournisseur assure la redondance.

À quoi cela ressemble en pratique

Ici, un serveur nginx répartit les requêtes entre trois serveurs applicatifs et en garde un quatrième en réserve :

```
upstream app {
    least_conn;
    server 10.0.0.11:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.12:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.13:8080 max_fails=3 fail_timeout=30s weight=2;
    server 10.0.0.20:8080 backup;
}

server {
    listen 80;
    location / {
        proxy_pass http://app;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_next_upstream error timeout http_502 http_503;
        proxy_next_upstream_tries 2;
        proxy_connect_timeout 2s;
    }
}
```

`least_conn` envoie chaque nouvelle requête au serveur ayant le moins de connexions actives et départage les égalités par un tourniquet pondéré. Le troisième serveur est la machine plus grande : `weight=2` le compte donc double. `max_fails=3 fail_timeout=30s` indique que trois tentatives échouées en 30 secondes retirent un serveur du pool pendant 30 secondes ; nginx le réessaie ensuite avec des requêtes réelles et le réintègre si elles réussissent. Sans ces

paramètres, les valeurs par défaut s'appliquent : un échec en 10 secondes suffit. Le serveur `backup` ne reçoit des requêtes que lorsque les trois principaux sont indisponibles.

Dans `location`, `proxy_next_upstream` détermine quand une requête passe à un autre serveur : erreur de connexion, expiration du délai ou réponse 502 ou 503. `proxy_next_upstream_tries 2` borne les tentatives pour une même requête ; la valeur par défaut, 0, ne fixe aucune limite. Les deux lignes `proxy_set_header` conservent le nom d'hôte demandé par le client, que nginx remplacerait sinon par le nom du groupe upstream `app`, et ajoutent l'adresse du client à `X-Forwarded-For` pour que l'application journalise les vrais visiteurs plutôt que l'équilibreur. `proxy_connect_timeout 2s` abandonne un serveur n'ayant pas accepté la connexion en deux secondes, là où la valeur par défaut attendrait 60.

Ces contrôles sont passifs : nginx apprend qu'un serveur est défaillant seulement lorsque des requêtes réelles échouent. HAProxy peut aussi sonder chaque serveur selon son propre calendrier. `option httpchk GET /healthz` dans un backend, avec `check` sur chaque ligne `server`, envoie cette requête toutes les deux secondes par défaut ; trois échecs consécutifs déclarent le serveur indisponible, et deux succès le rétablissent. Sans `check`, HAProxy n'effectue aucun contrôle de santé et considère le serveur toujours disponible.

Avant d'être un équilibreur, nginx est un serveur web ordinaire : l'un des deux utilisés pour les quarante sites administrés sur Apache et Nginx entre 2003 et 2014, hébergés sur Ubuntu Linux, ajustés et maintenus pour rester fiables sous un trafic réel. La ligne `backup` a un équivalent à l'échelle supérieure, lorsque des fournisseurs entiers jouent le rôle des serveurs. Une messagerie avec basculement entre trois fournisseurs garde SendGrid comme principal, avec SMTP2GO et Azure Communication Services derrière. Elle passe automatiquement de l'un à l'autre et a transformé le mauvais après-midi d'un fournisseur en chemin dégradé plutôt qu'en interruption.

Où les problèmes surviennent

Un serveur répondant 500 à tout reste dans le pool. Par défaut, nginx ne compte comme échecs que les erreurs de communication avec le serveur, les délais expirés et les en-têtes de réponse mal formés. Un serveur dont l'application a perdu sa base et renvoie toujours 500 répond vite et poliment : l'équilibreur continue donc de lui envoyer sa part du trafic. Les réponses 500, 502, 503, 504 et 429 ne comptent contre lui que si elles figurent dans

`proxy_next_upstream` ; 403 et 404 ne comptent jamais. Listez les codes qui signifient « ce serveur est défaillant » et donnez aux serveurs un contrôle actif sur un point de terminaison sollicitant ce dont l'application a besoin pour servir une page.

Une requête lente s'exécute sur tous les serveurs à tour de rôle. `timeout` figure dans la valeur par défaut de `proxy_next_upstream` , et `proxy_read_timeout` vaut 60 secondes par défaut. Un rapport dont la construction prend 70 secondes dépasse le délai sur le premier serveur, passe au deuxième, y dépasse le délai puis passe au troisième : le client attend trois minutes pendant que chaque serveur effectue le même travail. Une expiration compte toujours comme tentative échouée : avec `max_fails` à 1 par défaut, chaque serveur concerné devient aussi indisponible pendant 10 secondes. Limitez les transferts avec `proxy_next_upstream_tries` et `proxy_next_upstream_timeout` , donnez au point de terminaison lent sa propre `location` avec un délai plus long, ou sortez le travail du chemin de la requête pour en faire une tâche d'arrière-plan.

Un paiement passe deux fois. Par défaut, nginx refuse de renvoyer une requête POST, LOCK ou PATCH à un autre serveur après qu'elle en a atteint un, car le premier peut déjà l'avoir traitée. Ajouter `non_idempotent` à `proxy_next_upstream` pour faire disparaître les expirations retire cette protection : un paiement terminé par le premier serveur mais dont la réponse est trop lente est exécuté de nouveau par le deuxième. Laissez l'option désactivée sauf si tous les points de terminaison derrière sont idempotents, par exemple grâce à une clé d'idempotence et au refus de répéter une requête déjà vue.

Chaque visiteur semble venir de l'équilibreur. Sans les en-têtes de l'exemple, l'application journalise l'adresse de l'équilibreur à chaque requête, et toute limite par adresse IP s'applique désormais à tous les utilisateurs à la fois. Ajouter `X-Forwarded-For` apporte son propre piège. `$proxy_add_x_forwarded_for` ajoute l'adresse du client à tout `X-Forwarded-For` déjà envoyé par celui-ci : les entrées les plus à gauche sont donc celles qu'il a choisies. L'application ne devrait faire confiance qu'à l'entrée ajoutée par son propre équilibreur et n'accepter l'en-tête que depuis l'adresse de celui-ci.

Tout un bureau se retrouve sur un serveur. `ip_hash` de nginx hache les trois premiers octets d'une adresse IPv4 : tous les clients les partageant aboutissent donc au même serveur, et toute une entreprise derrière une passerelle NAT arrive comme un seul client. Changer le pool réserve la surprise inverse. Avec un simple `hash` , ajouter ou retirer un serveur peut déplacer la plupart des clés vers

un autre serveur ; chaque utilisateur ainsi rattaché à un serveur perd ce que son ancien serveur conservait. La documentation nginx conseille de marquer un serveur `down` plutôt que de le supprimer d'un groupe `ip_hash`, et propose `hash ... consistent`, qui ne déplace que quelques clés lorsque le pool change. Mieux encore, gardez l'état des sessions hors des serveurs : l'affinité cesse alors d'importer.

Pour aller plus loin

- Utiliser nginx comme équilibreur HTTP — les trois méthodes de base, les poids, la persistance des sessions et les contrôles de santé passifs, chacun avec une configuration courte.
- Module ngx_http_upstream_module — chaque paramètre d'une ligne `server`, dont `max_fails`, `backup` et l'exception du serveur unique, ainsi que les méthodes `hash`, `ip_hash`, `least_conn` et `random`.
- Module ngx_http_proxy_module — `proxy_next_upstream`, ce qui constitue une tentative échouée, les délais et les en-têtes définis par nginx vers le serveur.
- Manuel de configuration de HAProxy 3.4 — `balance` et ses algorithmes, modes TCP et HTTP, et contrôles de santé actifs avec `check`, `inter`, `rise` et `fall`.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un équilibreur de charge ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/load-balancer/