Engineer

Балансувальник навантаження

⚖ сервер, що розподіляє запити між кількома іншими

Що це таке

Балансувальник навантаження — це сервер, який приймає запити клієнтів і розподіляє їх між пулом інших серверів, що виконують ту саму роботу. Жодна окрема машина не несе всього навантаження, і відмова однієї не забирає із собою весь сервіс. Англійською це load balancer, у конфігураціях і на схемах його часто скорочують до LB, а в розмові кажуть просто «балансер». Сама робота називається балансуванням навантаження. Клієнти бачать одну адресу — адресу балансувальника. Сервери за ним утворюють пул, який nginx називає групою upstream, а HAProxy — бекендом.

Для кожного запиту балансувальник ухвалює два рішення: які сервери досить здорові, щоб його отримати, і котрий із них його отримає. Друге рішення — це метод балансування. Round robin віддає запити серверам по черзі, і саме так поводиться nginx, коли нічого іншого не налаштовано. Least connections, тобто «найменше з'єднань», надсилає наступний запит серверу з найменшою кількістю активних з'єднань і вирівнює навантаження, коли одні запити тривають набагато довше за інші. Посібник HAProxy радить цей метод для довгих сесій на кшталт LDAP чи SQL і вважає його невдалим вибором для коротких, як-от HTTP. Хешування перетворює ключ, наприклад IP-адресу клієнта чи запитаний URL, на вибір сервера, тож той самий ключ щоразу потрапляє на ту саму машину. nginx має й випадковий метод: він може навмання вибрати два сервери й віддати запит менш завантаженому з них. Ваги зсувають вибір у бік більших машин: у прикладі з власного посібника nginx один сервер із `weight=3` і два з типовою вагою 1 отримують відповідно три, один і один запит із кожних п'яти.

Балансувальники різняться й тим, наскільки глибоко вони читають трафік. У режимі `mode tcp`, який у HAProxy є типовим, клієнт і сервер з'єднуються повнодуплексним з'єднанням без жодного аналізу на рівні 7. Це підходить для TLS, який проходить крізь балансувальник нерозшифрованим, для SSH і

SMTP. У режимі `mode http` HAProxy розбирає кожен запит, перш ніж вибрати сервер, відхиляє запити, що порушують протокол, і може маршрутизувати за шляхом чи заголовком. Балансувальник, що працює на рівні HTTP, завершує з'єднання клієнта й відкриває власне до сервера. Через це застосунок бачить балансувальник своїм єдиним клієнтом, якщо той не передає справжню адресу далі, зазвичай у заголовку `X-Forwarded-For`. З тієї ж причини балансувальник і зворотний проксі так часто є одним і тим самим процесом: nginx — і те, і те, а його власний посібник описує балансування навантаження як частину реалізації зворотного проксі.

Застосуйте поняття до рішення

Застосуйте «Що таке балансувальник навантаження?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Сервери під те навантаження, яке у вас справді є, перенесені в хмару чи назад, із порахованим наперед місячним рахунком.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Вела 20 застосунків Docker у Google Cloud і AWS — 20 контейнеризованих застосунків розгорнуто й поставлено під моніторинг

Вела 20 застосунків Docker у Google Cloud і AWS Зв'язатися з нами →

Чому це важливо

Один сервер має стелю. Коли сервіс її переростає, саме балансувальник дає команді змогу поставити другу й третю копію замість купувати більшу машину, і кожна нова копія додає потужності так, що клієнти навіть не помічають змін у пулі. Та сама схема робить обслуговування буденним: виведіть один сервер із ротації, оновіть його, перевірте, поверніть назад і повторіть із наступним, поки користувачів обслуговують інші.

Вона ж перетворює відмову сервера з аварії на меншу подію. Коли сервер перестає відповідати, балансувальник перестає його обирати, а решта серверів перебирає на себе його частку. Це працює лише тоді, коли в них є запас потужності. Три сервери, кожен із яких завантажений на 80%, не здатні підстраховувати один одного: якщо один випаде, кожен із двох інших отримає 120% роботи одного сервера.

Балансувальник змінює й те, як треба писати застосунок. Як зауважує посібник nginx, за методів round robin і least connections два послідовні запити одного клієнта можуть потрапити на різні сервери. Усе, що сервер тримає у своїй пам'яті чи на своєму диску між запитами, — сесію входу,

завантажений файл, що чекає на обробку, кеш кошика користувача — буде недоступне, якщо наступний запит потрапить на інший сервер. Тож або застосунок зберігає такий стан там, куди дотягуються всі сервери, у базі даних чи спільному кеші, або балансувальник закріплює кожного клієнта за одним сервером. Так робить `ip_hash` у nginx, за винятком випадків, коли цей сервер недоступний.

І нарешті, сам балансувальник стає тією єдиною машиною, крізь яку проходить кожен запит. Десять серверів застосунку за одним балансувальником усе ще мають єдину точку відмови — вона просто перемістилася. У серйозній інсталяції балансувальники працюють парою зі спільною адресою, яка може переходити від одного до іншого, або ж використовують керований балансувальник, за резервування якого відповідає провайдер.

Як це виглядає на практиці

Тут сервер nginx розподіляє запити між трьома серверами застосунку й тримає четвертий у резерві:

```
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` надсилає кожен новий запит серверу з найменшою кількістю активних з’єднань, а нічию розв’язує зваженим round robin. Третій сервер — найпотужніша машина, тому `weight=2` рахує його подвійно. `max_fails=3`

`fail_timeout=30s` означає, що три невдалі спроби протягом 30 секунд виводять сервер із пулу на 30 секунд. Після цього nginx знову пробує його на живих запитах і повертає до пулу, якщо вони вдаються. Без цих параметрів діють типові значення, і тоді досить однієї невдачі за 10 секунд. Сервер із позначкою `backup` отримує запити лише тоді, коли недоступні всі три основні.

У блоці `location` директива `proxy_next_upstream` визначає, коли запит передається іншому серверу: у разі помилки під час обміну із сервером, тайм-ауту або відповіді 502 чи 503. `proxy_next_upstream_tries 2` обмежує, скільки разів таке може статися з одним запитом; типове значення 0 означає «без обмежень». Два рядки `proxy_set_header` зберігають ім'я хоста, яке запитав клієнт, — інакше nginx підставив би замість нього ім'я групи `app`, — і додають адресу клієнта до `X-Forwarded-For`, щоб застосунок записував у журнали справжніх відвідувачів, а не балансувальник. `proxy_connect_timeout 2s` відмовляється від сервера, який не прийняв з'єднання за дві секунди, тоді як за замовчуванням nginx чекав би 60.

Ці перевірки пасивні: nginx дізнається, що сервер хворий, лише коли на ньому падають справжні запити. HAProxy вміє ще й сам опитувати кожен сервер за власним розкладом. `option httpchk GET /healthz` у бекенді разом із `check` у кожному рядку `server` за замовчуванням надсилає цей запит кожні дві секунди; три невдачі поспіль позначають сервер як непрацездатний, а дві успішні відповіді повертають його назад. Без `check` HAProxy не виконує перевірок справності й вважає сервер завжди доступним.

Перш ніж стати балансувальником, nginx є вебсервером, і досить звичайним. Він — один із двох вебсерверів, на яких трималися 40 вебсайтів на Apache і Nginx у 2003–2014 роках: сайти розміщувалися на серверах хостингу з Ubuntu Linux, а їхню конфігурацію, продуктивність і обслуговування вели так, щоб вони надійно витримували реальний трафік. Рядок `backup` має родича рівнем вище, де серверами є цілі провайдери. Пошта з failover між трьома провайдерами тримає SendGrid як основний, SMTP2GO та Azure Communication Services у резерві за ним і перемикається між ними автоматично. Це перетворило невдалий день у провайдера на роботу в деградованому режимі замість аварії.

Де виникають проблеми

Сервер, що відповідає 500 на кожен запит, лишається в пулі. За замовчуванням nginx вважає невдалою спробою лише помилку під час обміну із сервером, тайм-аут і некоректні заголовки відповіді. Сервер, чий застосунок утратив зв'язок із базою даних і на все повертає 500, відповідає швидко й чемно, тож балансувальник і далі надсилає йому його частку трафіку. Відповіді 500, 502, 503, 504 і 429 зараховуються серверу як невдачі лише тоді, коли їх перелічено в `proxy_next_upstream`, а 403 і 404 — ніколи. Перелічіть коди, що означають «сервер зламався», і дайте серверам активну перевірку на ендпоінт, який зачіпає все, що потрібно застосунку, щоб віддати сторінку.

Один повільний запит по черзі виконується на кожному сервері. `timeout` входить до типового значення `proxy_next_upstream`, а `proxy_read_timeout` за замовчуванням дорівнює 60 секундам. Звіт, на побудову якого йде 70 секунд, отримує тайм-аут на першому сервері, переходить на другий, знову отримує тайм-аут і йде на третій. Клієнт чекає три хвилини, а кожен сервер тим часом виконує ту саму роботу. Тайм-аут завжди вважається невдалою спробою, тому за типового `max_fails`, що дорівнює 1, кожен сервер, якого торкнувся запит, ще й позначається недоступним на 10 секунд. Обмежте передачі директивами `proxy_next_upstream_tries` і `proxy_next_upstream_timeout`, виділіть повільному ендпоінту окремий `location` із довшим тайм-аутом або винесіть роботу із запиту у фонове завдання.

Платіж проходить двічі. nginx за замовчуванням не передає запит POST, LOCK чи PATCH іншому серверу, якщо запит уже дійшов до одного з них, бо перший сервер міг устигнути його виконати. Якщо додати `non_idempotent` до `proxy_next_upstream`, щоб позбутися тайм-аутів, цей захист зникає: платіж, який перший сервер провів, але відповів надто повільно, другий проведе знову. Не вмикайте цей параметр, доки кожен ендпоінт за ним не стане ідемпотентним, наприклад приймаючи ключ ідемпотентності й відмовляючись повторювати запит, який уже бачив.

Усі відвідувачі ніби приходять із балансувальника. Без заголовків із прикладу застосунок записує в журнал адресу балансувальника для кожного запиту, а будь-яке обмеження за IP-адресою відтепер діє на всіх користувачів одночасно. `X-Forwarded-For` має й власну пастку. `$proxy_add_x_forwarded_for` дописує адресу клієнта до того `X-Forwarded-For`, який клієнт уже надіслав сам,

тож крайні ліві записи містять те, що клієнт вирішив туди написати. Застосунку варто довіряти лише запису, який додав його власний балансувальник, і приймати цей заголовок лише з адреси балансувальника.

Цілий офіс опиняється на одному сервері. `ip_hash` у nginx хешує перші три октети адреси IPv4, тож усі клієнти, у яких вони збігаються, потрапляють на той самий сервер, а ціла компанія за одним шлюзом NAT приходить як один клієнт. Зміни в пулі дають протилежну несподіванку. Зі звичайним `hash` додавання чи вилучення сервера може перенести більшість ключів на інші сервери, і кожен закріплений так користувач утрачає все, що тримав його старий сервер. Документація nginx радить позначати сервер як `down`, а не видаляти його з групи `ip_hash`, і пропонує `hash ... consistent`, що за зміни пулу переносить лише кілька ключів. А ще краще тримати стан сесій поза серверами — тоді вже неважливо, куди потрапить клієнт.

Що почитати далі

- Using nginx as HTTP load balancer — три базові методи, ваги, закріплення сесій і пасивні перевірки справності, кожне з короткою конфігурацією.
- Module ngx_http_upstream_module — усі параметри рядка `server`, зокрема `max_fails`, `backup` і виняток для груп з одним сервером, а також методи `hash`, `ip_hash`, `least_conn` і `random`.
- Module ngx_http_proxy_module — `proxy_next_upstream` і те, що вважається невдалою спробою, тайм-аути та заголовки, які nginx виставляє дорогою до сервера.
- HAProxy 3.4 Configuration Manual — `balance` та його алгоритми, режими TCP і HTTP, а також активні перевірки справності з `check`, `inter`, `rise` і `fall`.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке балансувальник навантаження?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/load-balancer/