Engineer

Обмеження частоти запитів

ліміт на кількість запитів від клієнта за певний час

Що це таке

Обмеження частоти запитів (англійською rate limiting) — це верхня межа того, скільки запитів один клієнт може надіслати сервісу за певний проміжок часу. Стежить за нею сам сервіс або щось, що стоїть перед ним. Кожен ліміт відповідає на три запитання: кого вважати клієнтом, як рахувати запити і що робити із запитом, який виходить за межу. Код стану HTTP для останнього випадку, 429 Too Many Requests, визначено в RFC 6585 у квітні 2012 року, і цей RFC свідомо залишає перші два запитання відкритими. Сервер може рахувати запити для кожного ресурсу окремо, для всього сервера або для групи серверів, а впізнавати клієнта — за обліковими даними автентифікації чи за cookie, що зберігає стан.

На практиці ключем стає щось одне з небагатьох варіантів: IP-адреса клієнта, обліковий запис, у який він увійшов, ключ API або щось із цього в поєднанні зі шляхом, щоб форма входу і сторінка товару мали різні ліміти. Відповідь на запит понад ліміт буває трьох видів. Сервер може відхилити його з кодом 429, бажано із заголовком `Retry-After`, який каже, скільки чекати: або кількістю секунд, або датою у форматі HTTP. Він може притримати запит і обробити його трохи згодом, щоб згладити сплеск, а не відкинути його. Або, коли атакою є сам обсяг, він може розірвати з'єднання без відповіді. Саме так радить RFC 6585, бо ввічлива відповідь 429 на кожен запит у потоці теж коштує ресурсів.

Те, як ведеться підрахунок, визначає поведінку ліміту на межах. Фіксоване вікно рахує запити за календарну хвилину чи годину й обнуляється, коли вікно змінюється. Це дешево й легко пояснити, але клієнт може витратити весь свій ліміт в останню секунду одного вікна і ще раз у першу секунду наступного: за ліміту 100 запитів на хвилину це 200 запитів за дві секунди. Ковзне вікно завжди дивиться назад на однаковий відрізок від поточної миті, тож межі між вікнами, якою можна скористатися, у нього немає; такі

лічильники зберігають stick-таблиці HAProxy, і посібник описує один зі своїх прикладів як частоту з’єднань, обчислену в ковзному вікні тривалістю 30 секунд. Метод «дірявого відра» (leaky bucket), який використовує nginx, пропускає запити з фіксованою середньою швидкістю, тримає невелику чергу для тих, що надходять швидше, і відкидає все, що переповнило б її. Відро з токенами (token bucket) працює навпаки: токени накопичуються з постійною швидкістю до певної стелі, кожен запит витрачає один, і клієнт, який деякий час мовчав, може витратити заощаджене одним сплеском.

Застосуйте поняття до рішення

Застосуйте «Що таке обмеження частоти запитів?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Облікові записи, паролі й права налаштовані так, щоб доступ відповідав роботі, а звільнені втрачали його того ж дня.

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. Написала 1 000 правил RBAC для Linux-серверів — 1,000 правил доступу написано, щоб усунути вразливість

Написала 1 000 правил RBAC для Linux-серверів Зв'язатися з нами →

Чому це важливо

Публічний ендпоінт не вибирає, хто і як часто його викликає. Скрапер, що копіює каталог, скрипт, який застряг у циклі повторних спроб, зловмисник, що перебирає злиті паролі на формі входу, і партнер, чиє нічне завдання писали поспіхом, для сервера виглядають однаково: запитів більше, ніж він розрахований витримати, і надходять вони з кількох джерел. Без ліміту саме найактивніший клієнт вирішує, скільки потужності дістанеться всім іншим. З лімітом один клієнт може використати тільки свою частку, а решта сервісу працює зі звичайною швидкістю, поки цей клієнт чекає.

Одні запити набагато дорожчі за інші. Сторінку товару можна віддати з кешу за мілісекунду, тоді як пошук, експорт чи будь-що, що звертається до платного зовнішнього API, щоразу коштує справжнього процесорного часу або справжніх грошей. Саме на ці шляхи й цілиться зловмисник, бо тому, хто викликає, вони нічого не коштують. Один глобальний ліміт, досить низький, щоб захистити експорт, задушив би звичайний перегляд сайту, тож корисне обмеження частоти задають для кожного шляху окремо й добирають кожен ліміт під те, скільки шлях коштує і як ним користуються законно.

Ліміт — це ще й частина контракту API. Клієнти, які знають правила, можуть самі тримати темп і ніколи не впиратися в ліміт, а це дешевше для обох сторін, ніж дізнаватися про нього з відмов. Стандартного заголовка для цього в HTTP немає. Багато API надсилають `X-RateLimit-Limit`, `X-RateLimit-Remaining` і `X-RateLimit-Reset`, але в різних реалізаціях та сама назва означає різне, як фіксує чернетка IETF, що має на меті стандартизувати цю ідею. Чернетка визначає два поля: `RateLimit-Policy` для квоти та її вікна і `RateLimit` для залишку; її одинадцята версія датована 23 травня 2026 року і, як і будь-який Internet-Draft, є незавершеною роботою, а не стандартом.

Як це виглядає на практиці

У nginx ліміт задають двома директивами: одна в блоці `http` створює лічильник, друга стоїть там, де ліміт має діяти:

```
limit_req_zone $binary_remote_addr zone=perip:10m rate=1r/s;
limit_req_status 429;

server {
    location /search/ {
        limit_req zone=perip burst=5 nodelay;
        proxy_pass http://app;
    }
}
```

Ключем є адреса клієнта у двійковому вигляді, що займає 4 байти для IPv4 і 16 для IPv6, а лічильники живуть у зоні спільної пам'яті на 10 мегабайтів під назвою `perip`. За документацією стан для одного ключа займає 128 байтів на 64-бітній платформі, тож мегабайт вміщує близько 8 тисяч адрес, а ця зона — близько 80 тисяч; коли вона заповнюється, nginx вилучає адресу, яку найдовше не використовували. `rate=1r/s` дозволяє в середньому один запит на секунду. `burst=5` дозволяє щонайбільше п'ятьом запитам понад цю швидкість дочекатися своєї черги замість відмови, а `nodelay` обробляє їх одразу, не розтягуючи в часі за налаштованою швидкістю. Усе, що виходить за межі burst, відхиляється. За замовчуванням nginx відповів би на таку відмову кодом 503, тому `limit_req_status` задає 429. Швидкості, менші за один запит на секунду, записують на хвилину, тож `30r/m` — це один запит кожні дві секунди. Перш ніж новий ліміт запрацює по-справжньому, `limit_req_dry_run on` рахує як зазвичай і позначає запити, які було б відхилено, значенням `REJECTED_DRY_RUN` у змінній `$limit_req_status`, але пропускає їх усі.

На іншому боці клієнт, який уперся в ліміт, має прочитати відповідь, а не повторювати запит негайно. Відповідь 429 від API, що дотримується чернетки IETF, може виглядати так:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/problem+json
Retry-After: 20
RateLimit-Policy: "default";q=100;w=60
RateLimit: "default";r=0;t=20
```

Політика дозволяє 100 запитів за 60-секундне вікно, залишок дорівнює нулю, а квота відновиться за 20 секунд. Якщо присутні обидва поля, чернетка каже, що пріоритет має `Retry-After` . Клієнт, який поводиться коректно, чекає щонайменше стільки, додає невелику випадкову паузу, а якщо `Retry-After` немає взагалі, переходить на експоненційну затримку замість фіксованого інтервалу.

22 обмежувачі частоти, які в разі збою блокують доступ 2025 року, побудовані для публічного каталогу компаній і фахівців, дотримуються правила «окремий ліміт для кожного шляху»: кожен налаштований під той шлях, який захищає, бо спроба входу, пошук і масовий експорт стають зловживанням за дуже різної частоти. Стан зберігається в Redis, тож ліміт спільний для всіх інстансів, а не окремий у кожному процесі. Вирішальним було рішення про те, що відбувається, коли Redis недоступний: обмежувачі відхиляють трафік, а не пропускають його. Сторінка прямо називає ціну — проблема з Redis стає видимою для користувачів — і пояснює, чому її прийняли: інакше проблема з Redis стала б проблемою з рахунками. Навколо них стоять Cloudflare Turnstile, блокування за origin і 351 рядок проміжного шару ідемпотентності, щоб повторений запис не перетворився на два.

Де виникають проблеми

Кожен сервер рахує сам по собі. Ліміт, що зберігається в пам’яті процесу, множиться з кожною копією застосунку: чотири інстанси за балансувальником навантаження пропускають клієнта з учетверо більшою швидкістю, ніж задумано, а кожен перезапуск обнуляє лічильники. Зона nginx — це спільна пам’ять на одній машині, а синхронізація зон між серверами входить до комерційної підписки; протокол peers у HAProxy копіює записи stick-таблиць між інстансами, але, як зазначає посібник, не

підсумовує їх. Тримайте лічильники в сховищі, до якого звертаються всі інстанси, як це робить згаданий вище каталог із Redis, або застосовуйте ліміт у єдиній точці, через яку проходить увесь трафік.

Усі приходять з однієї адреси. За зворотним проксі, CDN чи балансувальником навантаження застосунок бачить одну адресу клієнта для всіх користувачів, якщо оригінальну не передають далі, і ліміт за IP-адресою тоді душить усіх одночасно. Цілий офіс чи мобільна мережа за NAT дають той самий ефект у м'якшій формі. Будуйте ліміт на справжній адресі клієнта із заголовка, який може виставляти лише ваш власний проксі, а для трафіку користувачів, що увійшли в систему, віддавайте перевагу обліковому запису чи ключу API, а не адресі.

Обмежувач повідомляє про збій сервера. За замовчуванням nginx відповідає на відхилений запит кодом 503 Service Unavailable, що в HTTP означає перевантаження сервера або планове обслуговування. Дашборди й сповіщення, які вважають відповіді 5xx помилками сервера, будять когось через те, що один клієнт виявився жадібним, а сам клієнт так і не дізнається, що поводиться погано. Відповідайте кодом 429, який покладає причину на клієнта, і записуйте відмови в журнал на окремому рівні.

Усі відхилені клієнти повертаються в ту саму секунду. Якщо квота скидається в певний момент або кожен відхилений клієнт отримує однаковий `Retry-After`, усі вони повертаються разом; чернетка IETF наводить приклад щоденного скидання о 18:00:00, коли всі клієнти, імовірно, з'являться саме о 18:00:00. Сплеск, що настає слідом, може виявитися гіршим за той, який обмежили. Додавайте джитер з обох боків: варіюйте час очікування, який оголошує сервер, і дайте клієнтам додавати власну випадкову паузу.

Ніхто не вирішив, що робити, коли сховище впаде. Обмежувач, що тримає лічильники в Redis чи базі даних, мусить якось діяти, коли сховище недоступне. Код, який перехоплює помилку з'єднання і пропускає запит, обрав відкриватися при збої (fail open), і тоді збій стає вікном без лімітів для того, хто за цим стежить; код, який відхиляє запит, обрав закриватися при збої (fail closed), і тоді збій стає відмовою для всіх. Жоден із варіантів не хибний скрізь, але вибір має бути проєктним рішенням для кожного шляху, як його ухвалили для каталогу вище, а не тим, що бібліотека робить за замовчуванням.

Підрахунок відмов розкриває інформацію. Якщо відповіді 401 і 403 зараховуються до квоти, попереджає чернетка IETF, зловмисний клієнт може промацати ендпоінт і дізнатися про трафік іншого користувача. Вирішіть, які відповіді рахуються, дайте невдалій автентифікації окремий ліміт і не показуйте подробиць квоти клієнтам, які не пройшли автентифікацію.

Що почитати далі

- RFC 6585: Additional HTTP Status Codes — визначає 429 Too Many Requests і каже, що цей код залишає на розсуд сервера: хто такий користувач і як рахувати запити.
- RFC 9110: HTTP Semantics — поле `Retry-After` і дві його форми, а також що означає 503, коли його надсилає сервер.
- RateLimit header fields for HTTP — чернетка робочої групи HTTPAPI в IETF з полями `RateLimit-Policy` і `RateLimit` та розділом про безпеку, де йдеться про одночасне повернення клієнтів і витоки інформації.
- Module ngx_http_limit_req_module — `limit_req`, `burst`, `nodelay`, `delay`, пробний режим (dry run) і розміри зони спільної пам’яті.
- HAProxy 3.4 Configuration Manual — stick-таблиці та їхні лічильники частоти, а також протокол peers, що копіює їх між інстансами.

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 обмеження частоти запитів?» 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/rate-limiting/