Engineer

Ratenbegrenzung

Obergrenze für Clientanfragen pro Zeitintervall

Was es ist

Ratenbegrenzung setzt eine Obergrenze dafür, wie viele Anfragen ein Client innerhalb eines bestimmten Zeitraums an einen Dienst senden darf. Durchgesetzt wird sie vom Dienst oder von einer vorgeschalteten Komponente. Jede Begrenzung beantwortet drei Fragen: Wer gilt als Client, wie werden Anfragen gezählt und was geschieht mit der überschreitenden Anfrage? Der für die letzte Frage geschaffene HTTP-Statuscode 429 Too Many Requests wurde im April 2012 in RFC 6585 definiert. Die ersten beiden Fragen lässt dieser RFC bewusst offen. Ein Server kann pro Ressource, über den gesamten Server oder über eine Servergruppe zählen und den Client an Zugangsdaten oder an einem zustandsbehafteten Cookie erkennen.

In der Praxis wird meist einer von wenigen Schlüsseln gewählt: die IP-Adresse des Clients, das angemeldete Konto, ein API-Schlüssel oder eines davon zusammen mit dem Pfad, damit ein Anmeldeformular und eine Produktseite unterschiedliche Grenzen erhalten. Für eine Anfrage oberhalb der Grenze gibt es drei Reaktionen. Der Server kann sie mit 429 ablehnen, vorzugsweise mit einem `Retry-After` -Header, der die Wartezeit als Sekundenzahl oder HTTP-Datum nennt. Er kann die Anfrage zurückhalten und etwas später bearbeiten, was einen Schub glättet, statt ihn abzulehnen. Oder er kann die Verbindung unbeantwortet abbrechen, wenn die Menge selbst der Angriff ist. Genau dies schlägt RFC 6585 vor: Auch eine höfliche 429-Antwort auf jede Anfrage einer Flut kostet Ressourcen.

Das Zählverfahren bestimmt das Verhalten an den Grenzen. Ein festes Fenster zählt Anfragen je Kalenderminute oder -stunde und beginnt an der Grenze wieder bei null. Das ist billig und leicht zu erklären. Es lässt einen Client aber sein gesamtes Kontingent in der letzten Sekunde eines Fensters und gleich darauf in der ersten Sekunde des nächsten verbrauchen: Bei 100 pro Minute sind das 200 Anfragen in zwei Sekunden. Ein gleitendes Fenster blickt vom aktuellen

Zeitpunkt immer dieselbe Zeitspanne zurück. Es gibt daher keine ausnutzbare Grenze. Die Stick Tables von HAProxy führen solche Zähler; ein Handbuchbeispiel zählt die Verbindungsrate über ein gleitendes Fenster von 30 Sekunden. Der von nginx verwendete Leaky Bucket lässt Anfragen mit fester durchschnittlicher Rate durch und hält eine kleine Warteschlange für schneller eintreffende Anfragen. Was sie überlaufen ließe, wird abgelehnt. Ein Token Bucket arbeitet umgekehrt: Tokens sammeln sich mit gleichbleibender Rate bis zu einer Obergrenze an. Jede Anfrage verbraucht eines. Ein zuvor stiller Client kann sein angespartes Kontingent auf einmal ausgeben.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Ratenbegrenzung?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Konten, Passwörter und Berechtigungen werden so geordnet, dass alle genau den beruflich erforderlichen Zugriff haben und ihn am Tag ihres Ausscheidens verlieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 1.000 RBAC-Regeln für sichere Linux-Server — 1,000 Zugriffsregeln zum Schließen einer Sicherheitslücke

1.000 RBAC-Regeln für sichere Linux-Server Kontakt aufnehmen →

Warum es wichtig ist

Ein öffentlicher Endpunkt kann sich weder seine Aufrufer noch deren Häufigkeit aussuchen. Ein Katalogscraper, ein in einer Wiederholungsschleife steckendes Skript, ein Angreifer mit geleakten Passwörtern am Anmeldeformular und ein Partner mit hastig geschriebenem Nachtjob sehen für den Server gleich aus: mehr Anfragen als vorgesehen, aus wenigen Quellen. Ohne Begrenzung bestimmt der aktivste Aufrufer die für alle anderen verfügbare Kapazität. Mit Begrenzung kann ein einzelner Client nur seinen Anteil verbrauchen. Der übrige Dienst arbeitet normal weiter, während dieser Client wartet.

Manche Anfragen sind deutlich teurer als andere. Eine Produktseite kann in einer Millisekunde aus dem Cache kommen. Eine Suche, ein Export oder ein Aufruf einer kostenpflichtigen externen API kostet dagegen jedes Mal echte Prozessorzeit oder echtes Geld. Genau diese Pfade greifen missbräuchliche Nutzer an, weil der Aufruf sie selbst nichts kostet. Eine globale Grenze, niedrig genug zum Schutz des Exports, würde gewöhnliches Surfen abwürgen. Nützliche Ratenbegrenzung wird deshalb pro Pfad festgelegt, jeweils nach dessen Kosten und legitimer Nutzung.

Eine Grenze ist auch Teil des API-Vertrags. Clients, die die Regeln kennen, können ihr Tempo anpassen, ohne die Grenze zu erreichen. Das ist für beide Seiten billiger, als sie durch Ablehnungen zu entdecken. HTTP hat dafür keinen standardisierten Header. Viele APIs senden `X-RateLimit-Limit`, `X-RateLimit-Remaining` und `X-RateLimit-Reset`. Derselbe Name trägt aber je nach Implementierung unterschiedliche Werte, wie ein IETF-Entwurf zur Standardisierung festhält. Er definiert zwei Felder: `RateLimit-Policy` für Kontingent und Fenster sowie `RateLimit` für den Rest. Seine elfte Fassung stammt vom 23. Mai 2026 und ist wie jeder Internet-Draft ein Arbeitsstand, kein Standard.

Wie es in der Praxis aussieht

In nginx braucht eine Begrenzung zwei Direktiven: eine im Block `http`, die den Zähler einrichtet, und eine dort, wo er angewendet wird:

```
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;
    }
}
```

Der Schlüssel ist die Clientadresse in Binärform, 4 Bytes lang bei IPv4 und 16 bei IPv6. Die Zähler liegen in einer gemeinsamen Speicherzone von 10 Megabyte namens `perip`. Laut Dokumentation braucht der Zustand pro Schlüssel auf einer 64-Bit-Plattform 128 Bytes. Ein Megabyte fasst daher etwa 8 Tausend Adressen, diese Zone etwa 80 Tausend. Ist sie voll, entfernt nginx den am längsten nicht verwendeten Eintrag. `rate=1r/s` erlaubt durchschnittlich eine Anfrage pro Sekunde. `burst=5` lässt bis zu fünf Anfragen oberhalb dieser Rate warten, statt sie abzulehnen. `nodelay` bedient sie sofort, statt sie im konfigurierten Abstand zu verteilen. Alles jenseits dieses Schubs wird abgelehnt. nginx würde dafür standardmäßig 503 senden; deshalb setzt `limit_req_status` den Wert 429. Raten unter einer Anfrage pro Sekunde werden pro Minute geschrieben: `30r/m` bedeutet eine Anfrage alle zwei Sekunden. Vor dem Aktivieren einer neuen Grenze zählt `limit_req_dry_run on` normal weiter und markiert sonst abgelehnte Anfragen mit `REJECTED_DRY_RUN` in der Variablen `$limit_req_status`, ohne etwas abzulehnen.

Auf der Empfangsseite sollte ein Client beim Erreichen der Grenze die Antwort lesen, statt sofort erneut anzufragen. Ein 429 von einer dem IETF-Entwurf folgenden API könnte so aussehen:

```
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
```

Die Regel erlaubt 100 Anfragen je Fenster von 60 Sekunden. Keine bleibt übrig; das Kontingent kehrt in 20 Sekunden zurück. Sind beide vorhanden, hat laut Entwurf `Retry-After` Vorrang. Ein korrekt implementierter Client wartet mindestens so lange, ergänzt eine kleine zufällige Verzögerung und fällt ohne `Retry-After` auf exponentiellen Backoff statt eines festen Intervalls zurück.

Die 22 bei Ausfällen geschlossen bleibenden Ratenbegrenzer für ein öffentliches Verzeichnis von Unternehmen und Fachleuten, entstanden in einer seit 2025 ausgeübten Rolle, folgen der Regel pro Pfad. Jeder wurde für seinen geschützten Pfad gebaut, denn Anmeldeversuch, Suche und Massenexport werden bei sehr unterschiedlichen Raten missbräuchlich. Ihr Zustand liegt in Redis, sodass die Grenze über alle Instanzen hält statt nur pro Prozess. Entscheidend war das Verhalten bei einem Redis-Ausfall: Die Begrenzer lehnen Verkehr ab, statt ihn durchzuwinken. Die Seite benennt den Preis offen: Ein Redis-Problem wird für Nutzer sichtbar. Akzeptiert wurde das, weil andernfalls aus einem Redis-Problem ein Abrechnungsproblem würde. Cloudflare Turnstile, eine Sperre am Ursprung und 351 Zeilen Idempotenzmiddleware, damit eine wiederholte Schreiboperation nicht zweimal wirkt, liegen darum herum.

Wo es schiefgeht

Jeder Server zählt für sich. Eine im Prozessspeicher gehaltene Grenze vervielfacht sich mit jeder Anwendungskopie. Vier Instanzen hinter einem Lastverteiler lassen einen Client mit der vierfachen beabsichtigten Rate durch; jeder Neustart setzt die Zähler zurück. nginx teilt seine Speicherzone auf einem Rechner. Zonen zwischen Servern abzugleichen ist Teil seines kommerziellen Abonnements. HAProxys Peers-Protokoll kopiert Stick-Table-Einträge zwischen Instanzen, aber laut Handbuch ohne sie zu addieren. Halten Sie die Zähler

in einem von jeder Instanz abgefragten Speicher wie Redis im genannten Verzeichnis oder begrenzen Sie am einzigen Punkt, den der gesamte Verkehr passiert.

Alle kommen von derselben Adresse. Hinter Reverse Proxy, CDN oder Lastverteiler sieht die Anwendung für alle Nutzer dieselbe Clientadresse, sofern die ursprüngliche nicht weitergegeben wird. Eine Grenze pro IP drosselt dann alle zugleich. Ein ganzes Büro oder Mobilfunknetz hinter NAT hat denselben Effekt in milderer Form. Verwenden Sie die echte Clientadresse aus einem Header, den nur Ihr eigener Proxy setzen darf. Bevorzugen Sie bei angemeldetem Verkehr Konto oder API-Schlüssel statt Adresse.

Der Begrenzer meldet einen Serverfehler. nginx antwortet auf eine abgelehnte Anfrage standardmäßig mit 503 Service Unavailable. In HTTP bedeutet dies, dass der Server überlastet oder wegen Wartung nicht verfügbar ist. Dashboards und Alarme, die 5xx-Antworten als Serverfehler behandeln, wecken dann jemanden, weil ein einzelner Client gierig war. Der Client erfährt nichts über sein eigenes Verhalten. Antworten Sie mit 429, das die Ursache auf der Clientseite verortet, und protokollieren Sie Ablehnungen auf einer eigenen Stufe.

Alle gedrosselten Clients kehren in derselben Sekunde zurück. Wird ein Kontingent zu einem festen Zeitpunkt zurückgesetzt oder jedem abgelehnten Client dasselbe `Retry-After` genannt, kommen alle gemeinsam wieder. Der IETF-Entwurf nennt eine tägliche Rücksetzung um 18:00:00, bei der alle Clients wahrscheinlich um 18:00:00 auftauchen. Der folgende Schub kann schlimmer sein als der begrenzte. Ergänzen Sie auf beiden Seiten Zufall: Variieren Sie die vom Server genannte Wartezeit und lassen Sie Clients eine eigene zufällige Verzögerung addieren.

Niemand hat entschieden, was beim Speicherausfall geschieht. Ein Begrenzer mit Zählern in Redis oder einer Datenbank muss bei deren Unerreichbarkeit etwas tun. Code, der den Verbindungsfehler abfängt und die Anfrage durchlässt, hat sich für Fail-open entschieden. Aus dem Ausfall wird ein Zeitfenster ohne Ratenbegrenzung für jemanden, der darauf achtet. Ablehnender Code hat Fail-closed gewählt und macht daraus einen Ausfall für alle. Keine der beiden Möglichkeiten ist in jedem Fall falsch. Die Wahl muss aber wie beim genannten Verzeichnis für jeden Pfad bewusst getroffen werden, statt dem Standardverhalten einer Bibliothek überlassen zu bleiben.

Das Zählen von Ablehnungen gibt Informationen preis. Werden 401- und 403-Antworten auf ein Kontingent angerechnet, kann ein bösartiger Client laut IETF-Entwurf einen Endpunkt sondieren und etwas über den Verkehr eines anderen Nutzers erfahren. Legen Sie fest, welche Antworten zählen, geben Sie fehlgeschlagenen Authentifizierungen eine eigene Grenze und halten Sie Kontingentdetails vor nicht authentifizierten Clients zurück.

Weiterführende Lektüre

- RFC 6585: Zusätzliche HTTP-Statuscodes — definiert 429 Too Many Requests und benennt den Spielraum des Servers: Wer gilt als Nutzer, und wie werden Anfragen gezählt?
- RFC 9110: HTTP-Semantik — das Feld `Retry-After`, seine zwei Formen und die Bedeutung von 503, wenn der Server es sendet.
- RateLimit-Headerfelder für HTTP — der Entwurf der IETF-Arbeitsgruppe HTTPAPI zu `RateLimit-Policy` und `RateLimit`, mit Sicherheitsabschnitt zu gleichzeitiger Rückkehr und Informationslecks.
- Modul ngx_http_limit_req_module — `limit_req`, `burst`, `nodelay`, `delay`, Probelaufmodus und Größen der gemeinsamen Speicherzone.
- HAProxy 3.4: Konfigurationshandbuch — Stick Tables und ihre Ratenzähler sowie das Peers-Protokoll zum Kopieren zwischen Instanzen.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Ratenbegrenzung?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/rate-limiting/