Engineer

Hastighetsbegrensning

et tak på antall forespørsler fra én klient per tidsrom

Hva det er

Hastighetsbegrensning, på engelsk rate limiting, er et tak på hvor mange forespørsler én klient får sende til en tjeneste innenfor et gitt tidsrom, håndhevet av tjenesten selv eller av noe som står foran den. Ordet ratebegrensning brukes også. Enhver grense må svare på tre spørsmål: hvem som regnes som klienten, hvordan forespørslene telles, og hva som skjer med forespørselen som går over. HTTP-statuskoden for det siste, 429 Too Many Requests, ble definert i RFC 6585 i april 2012, og den RFC-en lar bevisst de to første stå åpne. En server kan telle per ressurs, for hele serveren eller på tvers av en gruppe servere, og den kan kjenne igjen klienten på påloggingsopplysningene eller på en informasjonskapsel som bærer tilstand.

I praksis er nøkkelen en av en håndfull ting: klientens IP-adresse, den innloggede kontoen, en API-nøkkel eller en av dem kombinert med ruten, slik at et innloggingsskjema og en produktside får hver sin grense. Svaret på en forespørsel over grensen kommer i tre varianter. Serveren kan avvise den med 429, helst med en `Retry-After` -header som sier hvor lenge klienten skal vente, enten som et antall sekunder eller som en HTTP-dato. Den kan holde forespørselen tilbake og behandle den et øyeblikk senere, slik at en topp jevnes ut i stedet for å avvises. Eller, når selve mengden er angrepet, kan den bryte tilkoblingen uten å svare. Det er nettopp det RFC 6585 foreslår, fordi det også koster ressurser å skrive et høflig 429-svar til hver eneste forespørsel i en flom.

Måten det telles på, avgjør hvordan en grense oppfører seg i kantene. Et fast vindu teller forespørsler per kalenderminutt eller -time og starter på null igjen når vinduet skifter. Det er billig og lett å forklare, og det lar en klient bruke hele kvoten sin i det siste sekundet av ett vindu og igjen i det første sekundet av det neste: med en grense på 100 i minuttet blir det 200 forespørsler på to sekunder. Et glidende vindu ser alltid like langt tilbake fra nå, så det finnes ikke noe vindusskifte å utnytte; stick tables i HAProxy holder tellere av den typen, og

manualen beskriver et av eksemplene sine som en tilkoblingsrate beregnet over et glidende vindu på 30 sekunder. Leaky bucket-metoden, den «lekke bøtta», som nginx bruker, slipper forespørsler gjennom i et fast gjennomsnittlig tempo, holder en liten kø for dem som kommer raskere, og avviser det som ville fått køen til å renne over. En token bucket virker motsatt vei: tokener hoper seg opp i et jevnt tempo opp til et tak, hver forespørsel bruker ett, og en klient som har vært stille en stund, kan bruke sparepengene sine i én topp.

Bruk begrepet i en beslutning

Bruk «Hva er hastighetsbegrensning?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Kontoer, passord og tilganger ordnet slik at folk har nøyaktig den tilgangen jobben krever, og de som slutter, mister den samme dag.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

Skrev 1 000 RBAC-regler for å sikre Linux-servere Ta kontakt →

Hvorfor det er viktig

Et offentlig endepunkt har ingen innflytelse på hvem som kaller det, eller hvor ofte. En skraper som kopierer en katalog, et skript som har hengt seg opp i en løkke av nye forsøk, en angriper som prøver lekkede passord mot innloggingsskjemaet, og en partner med en nattlig jobb som ble skrevet i all hast, ser like ut for serveren: flere forespørsler enn den var dimensjonert for, fra noen få kilder. Uten en grense er det den travleste klienten som bestemmer hvor mye kapasitet alle andre får. Med en grense kan én klient bare bruke sin egen andel, og resten av tjenesten går videre i normalt tempo mens den klienten venter.

Noen forespørsler er langt dyrere enn andre. En produktside kan komme fra en cache på et millisekund, mens et søk, en eksport eller alt som kaller et betalt eksternt API, koster ekte CPU-tid eller ekte penger ved hvert kall. Det er de rutene en misbruker går etter, nettopp fordi de ikke koster den som kaller, noe som helst. Én global grense som er lav nok til å beskytte eksporten, ville kvele vanlig surfing, så nyttig hastighetsbegrensning settes per rute, med hver grense valgt etter hva ruten koster og hvordan den brukes legitimt.

En grense er også en del av kontrakten til et API. Klienter som kjenner reglene, kan tilpasse tempoet og aldri treffe dem, og det er billigere for begge parter enn å oppdage grensen gjennom avvisninger. HTTP har ingen standard header for dette. Mange API-er sender `X-RateLimit-Limit`, `X-RateLimit-Remaining` og `X-`

`RateLimit-Reset`, men det samme navnet bærer ulike verdier fra én implementasjon til den neste, slik et IETF-utkast som vil standardisere ideen, gjør rede for. Utkastet definerer to felt, `RateLimit-Policy` for kvoten og vinduet og `RateLimit` for det som er igjen; den ellevte versjonen er datert 23. mai 2026 og er, som ethvert Internet-Draft, et arbeid som pågår og ikke en standard.

Slik ser det ut i praksis

I nginx krever en grense to direktiver: ett i `http`-blokken som setter opp telleren, og ett der grensen skal gjelde:

```
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;
    }
}
```

Nøkkelen er klientens adresse i binær form, som tar 4 byte for IPv4 og 16 for IPv6, og tellerne ligger i en delt minnesone på 10 megabyte kalt `perip`. Dokumentasjonen oppgir tilstanden for én nøkkel til 128 byte på en 64-bits plattform, så en megabyte rommer rundt 8 tusen adresser og denne sonen rundt 80 tusen; når den blir full, kaster nginx ut den adressen som er brukt minst nylig. `rate=1r/s` tillater én forespørsel i sekundet i snitt. `burst=5` lar inntil fem forespørsler over den raten vente på tur i stedet for å bli avvist, og `nodelay` behandler dem med en gang i stedet for å spre dem ut i den konfigurerte raten. Alt utover burst avvises. nginx ville som standard svart på den avvisningen med 503, og derfor setter `limit_req_status` 429. Rater under én i sekundet skrives per minutt, så `30r/m` er én forespørsel hvert andre sekund. Før en ny grense tas i bruk, teller `limit_req_dry_run on` som vanlig og merker forespørslene som ville blitt avvist, som `REJECTED_DRY_RUN` i variabelen `$limit_req_status`, uten å avvise noe.

På mottakersiden bør en klient som treffer grensen, lese svaret i stedet for å prøve igjen med en gang. Et 429-svar fra et API som følger IETF-utkastet, kan se slik ut:

```
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
```

Policyen tillater 100 forespørsler per vindu på 60 sekunder, det er ingen igjen, og kvoten kommer tilbake om 20 sekunder. Der begge feltene finnes, sier utkastet at `Retry-After` går foran. En klient som oppfører seg ordentlig, venter minst så lenge, legger til en liten tilfeldig forsinkelse, og hvis det ikke finnes noen `Retry-After` i det hele tatt, faller den tilbake på eksponentiell backoff i stedet for et fast intervall.

De 22 hastighetsbegrenserne som stenger ved feil fra 2025, bygget for en offentlig katalog over bedrifter og fagfolk, følger regelen om én grense per rute: hver er bygget for ruten den beskytter, fordi et innloggingsforsøk, et søk og en masseeksport blir misbruk ved helt ulike rater. Tilstanden ligger i Redis, slik at en grense deles på tvers av instanser i stedet for å gjelde per prosess. Den avgjørende beslutningen var hva som skjer når Redis er nede: ratebegrenserne avviser trafikken i stedet for å slippe den gjennom. Siden legger ikke skjul på prisen, at et Redis-problem blir et problem brukerne ser, eller på hvorfor den ble akseptert: alternativet var at et Redis-problem ble et faktureringsproblem. Rundt dem ligger Cloudflare Turnstile, en opprinnelseslås og 351 linjer idempotensmellomvare, slik at en skriving som prøves på nytt, ikke blir til to.

Hvor det går galt

Hver server teller for seg. En grense som holdes i prosessminnet, ganges opp med hver kopi av applikasjonen: fire instanser bak en lastbalanserer slipper en klient gjennom med fire ganger den tiltenkte raten, og hver omstart nullstiller tellerne. Sonen i nginx er delt minne på én maskin, og å holde soner i takt på tvers av servere hører til det kommersielle abonnementet; peers-protokollen i HAProxy kopierer oppføringer i stick tables mellom instanser, men legger dem, som manualen påpeker, ikke sammen. Hold tellerne i et lager som alle instansene spør, slik katalogen over gjør med Redis, eller håndhev grensen på det ene stedet all trafikk passerer.

Alle kommer fra samme adresse. Bak en omvendt proxy, et CDN eller en lastbalanserer ser applikasjonen én klientadresse for alle brukere med mindre den opprinnelige sendes med, og en grense per IP-adresse struper da alle på én

gang. Et helt kontor eller mobilnett bak NAT har samme virkning i mildere form. Bygg grensen på den virkelige klientadressen fra en header som bare din egen proxy får sette, og foretrekk kontoen eller API-nøkkelen framfor adressen for innlogget trafikk.

Begrenseren melder en serverfeil. Standardsvaret til nginx på en avvist forespørsel er 503 Service Unavailable, som i HTTP betyr at serveren er overbelastet eller nede for vedlikehold. Dashbord og varsler som behandler 5xx-svar som serverfeil, vekker da noen fordi én klient var grådig, og klienten lærer ingenting om sin egen oppførsel. Svar med 429, som legger årsaken hos klienten, og logg avvisningene på et eget nivå.

Alle avviste klienter kommer tilbake i samme sekund. Hvis en kvote nullstilles på et fast tidspunkt, eller hver avvist klient får samme `Retry-After`, kommer alle tilbake samtidig; IETF-utkastet gir eksemplet med en daglig nullstilling kl. 18:00:00, da alle klientene sannsynligvis dukker opp kl. 18:00:00. Toppen som følger, kan bli verre enn den som ble begrenset. Legg til jitter i begge ender: varier ventetiden serveren oppgir, og la klientene legge til en tilfeldig forsinkelse selv.

Ingen har bestemt hva som skjer når lageret går ned. En begrenser som holder tellerne i Redis eller en database, må gjøre noe når lageret ikke kan nås. Kode som fanger tilkoblingsfeilen og slipper forespørselen gjennom, har valgt å feile åpent (fail open), noe som gjør driftsavbruddet til et vindu uten grenser for den som følger med; kode som avviser, har valgt å feile lukket (fail closed), noe som gjør det til et driftsavbrudd for alle. Ingen av delene er feil overalt, men valget hører hjemme i en designbeslutning per rute, slik katalogen over tok den, og ikke i det et bibliotek tilfeldigvis gjør som standard.

Å telle avvisninger lekker informasjon. Hvis svar med 401 og 403 teller mot en kvote, advarer IETF-utkastet, kan en ondsinnet klient sondere et endepunkt og få informasjon om trafikken til en annen bruker. Bestem hvilke svar som teller, gi mislykket autentisering en egen grense, og ikke vis kvotedetaljer til klienter som ikke er autentisert.

Les mer

- RFC 6585: Additional HTTP Status Codes — definerer 429 Too Many Requests og sier hva koden overlater til serveren: hvem brukeren er, og hvordan forespørsler telles.

- RFC 9110: HTTP Semantics — feltet `Retry-After` og de to formene det har, og hva 503 betyr når serveren sender det.
- RateLimit header fields for HTTP — utkastet fra IETF-arbeidsgruppen HTTPAPI til `RateLimit-Policy` og `RateLimit`, med et sikkerhetsavsnitt om klienter som kommer tilbake samtidig, og om informasjonslekkasjer.
- Module ngx_http_limit_req_module — `limit_req`, `burst`, `nodelay`, `delay`, prøvekjøringsmodus (dry run) og størrelsene på den delte minnesonen.
- HAProxy 3.4 Configuration Manual — stick tables og ratetellerne deres, og peers-protokollen som kopierer dem mellom instanser.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er hastighetsbegrensning?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/rate-limiting/