Engineer

Rate limiting

et loft over antal kald fra én klient pr. tidsrum

Hvad det er

Rate limiting er et loft over, hvor mange anmodninger én klient må sende til en tjeneste inden for et givet tidsrum, håndhævet af tjenesten selv eller af noget, der står foran den. På dansk ses også ratebegrænsning og hastighedsbegrænsning, men i dokumentationen til de værktøjer, der gør arbejdet, er det det engelske udtryk, man møder. Enhver grænse skal besvare tre spørgsmål: hvem der tæller som klienten, hvordan anmodningerne tælles, og hvad der sker med den anmodning, der går over. HTTP-statuskoden til det sidste, 429 Too Many Requests, blev defineret i RFC 6585 i april 2012, og den RFC lader med vilje de to første stå åbne. En server kan tælle pr. ressource, for hele serveren eller på tværs af en gruppe servere, og den kan genkende klienten på dens loginoplysninger eller på en cookie, der bærer tilstand.

I praksis er nøglen en af en håndfuld ting: klientens IP-adresse, den indloggede konto, en API-nøgle eller en af dem kombineret med stien, så en loginformular og en produktside får hver sin grænse. Svaret på en anmodning over grænsen findes i tre udgaver. Serveren kan afvise den med 429, helst med en `Retry-After` -header, der fortæller, hvor længe klienten skal vente, enten som et antal sekunder eller som en HTTP-dato. Den kan holde anmodningen tilbage og behandle den et øjeblik senere, så et udbrud bliver jævnet ud i stedet for afvist. Eller, når selve mængden er angrebet, kan den lukke forbindelsen uden at svare. Det foreslår RFC 6585 netop, fordi det også koster ressourcer at skrive et høfligt 429-svar til hver eneste anmodning i en flodbølge.

Måden, der tælles på, afgør, hvordan en grænse opfører sig ved kanterne. Et fast vindue tæller anmodninger pr. kalenderminut eller -time og starter forfra ved nul, når vinduet skifter. Det er billigt og let at forklare, og det lader en klient bruge hele sin kvote i det sidste sekund af ét vindue og igen i det første sekund af det næste: med en grænse på 100 i minuttet bliver det til 200 anmodninger på to sekunder. Et glidende vindue ser altid lige langt tilbage fra nuet, så der er intet

vinduesskifte at udnytte; HAProxys stick tables holder tællere af den slags, og manualen beskriver et af sine eksempler som en forbindelsesrate beregnet over et glidende vindue på 30 sekunder. Leaky bucket-metoden, som nginx bruger, lukker anmodninger igennem i et fast gennemsnitligt tempo, holder en lille kø til dem, der kommer hurtigere, og afviser det, der ville få køen til at løbe over. En token bucket virker omvendt: tokens hober sig op i et fast tempo op til et loft, hver anmodning bruger én, og en klient, der har været stille et stykke tid, kan bruge sin opsparing i ét udbrud.

Brug begrebet i en beslutning

Brug »Hvad er rate limiting?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Konti, adgangskoder og rettigheder sat op, så folk har præcis den adgang, jobbet kræver, og fratrådte mister den samme dag.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har skrevet 1.000 RBAC-regler til Linux-servere — 1,000 adgangsregler skrevet for at lukke et sikkerhedshul

Har skrevet 1.000 RBAC-regler til Linux-servere Kontakt os →

Hvorfor det betyder noget

Et offentligt endpoint har ingen indflydelse på, hvem der kalder det, eller hvor tit. En scraper, der kopierer et katalog, et script, der sidder fast i en løkke af genforsøg, en angriber, der prøver lækkede adgangskoder mod loginformularen, og en partner, hvis natlige job blev skrevet i en fart, ser ens ud for serveren: flere anmodninger, end den var dimensioneret til, fra få kilder. Uden en grænse er det den travleste kalder, der bestemmer, hvor meget kapacitet alle andre får. Med en grænse kan én klient kun bruge sin egen andel, og resten af tjenesten kører videre i normalt tempo, mens den klient venter.

Nogle anmodninger er langt dyrere end andre. En produktside kan komme fra en cache på et millisekund, mens en søgning, en eksport eller alt, der kalder et betalt eksternt API, koster rigtig CPU-tid eller rigtige penge ved hvert kald. Det er de stier, en misbruger går efter, netop fordi de ikke koster kalderen noget. Én global grænse, der er lav nok til at beskytte eksporten, ville kvæle almindelig browsing, så brugbar rate limiting sættes pr. sti, med hver grænse valgt efter, hvad stien koster, og hvordan den bruges legitimt.

En grænse er også en del af kontrakten for et API. Klienter, der kender reglerne, kan afpasse tempoet og aldrig ramme dem, og det er billigere for begge parter end at opdage grænsen gennem afvisninger. HTTP har ingen standardheader til

formålet. Mange API'er sender `X-RateLimit-Limit`, `X-RateLimit-Remaining` og `X-RateLimit-Reset`, men det samme navn bærer forskellige værdier fra den ene implementering til den næste, som et IETF-udkast, der vil standardisere idéen, gør rede for. Udkastet definerer to felter, `RateLimit-Policy` til kvoten og dens vindue og `RateLimit` til det, der er tilbage; dets ellevte version er dateret 23. maj 2026 og er, som ethvert Internet-Draft, et arbejde i gang og ikke en standard.

Sådan ser det ud i praksis

I nginx kræver en grænse to direktiver: ét i `http`-blokken, der opretter tælleren, og ét der, hvor den skal gælde:

```
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;
    }
}
```

Nøglen er klientens adresse i binær form, som fylder 4 byte for IPv4 og 16 for IPv6, og tællerne ligger i en delt hukommelseszone på 10 megabyte ved navn `perip`. Dokumentationen sætter tilstanden for én nøgle til 128 byte på en 64-bit-platform, så en megabyte rummer omkring 8.000 adresser og denne zone omkring 80.000; når den bliver fuld, smider nginx den adresse ud, der har været brugt mindst for nylig. `rate=1r/s` tillader én anmodning i sekundet i gennemsnit. `burst=5` lader op til fem anmodninger over den rate vente på deres tur i stedet for at blive afvist, og `nodelay` behandler dem med det samme i stedet for at sprede dem ud i den konfigurerede rate. Alt ud over burst afvises. nginx ville som standard svare på den afvisning med 503, og derfor sætter `limit_req_status` 429. Rater under én i sekundet skrives pr. minut, så `30r/m` er én anmodning hvert andet sekund. Før en ny grænse går i drift, tæller `limit_req_dry_run on` som sædvanligt og markerer de anmodninger, der ville være blevet afvist, som `REJECTED_DRY_RUN` i variablen `$limit_req_status` uden at afvise noget.

I den anden ende bør en klient, der rammer grænsen, læse svaret i stedet for at prøve igen med det samme. Et 429-svar fra et API, der følger IETF-udkastet, kunne se sådan ud:

```
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
```

Politikken tillader 100 anmodninger pr. vindue på 60 sekunder, der er ingen tilbage, og kvoten kommer igen om 20 sekunder. Hvor begge felter er til stede, siger udkastet, at `Retry-After` går forud. En klient, der opfører sig ordentligt, venter mindst så længe, lægger en lille tilfældig forsinkelse oveni, og hvis der slet ikke er nogen `Retry-After`, falder den tilbage på eksponentiel backoff frem for et fast interval.

De 22 rate limitere, der lukker ved fejl fra 2025, som blev bygget til et offentligt katalog over virksomheder og fagfolk, følger reglen om grænser pr. sti: hver er bygget til den sti, den beskytter, fordi et loginforsøg, en søgning og en bulkeksport bliver til misbrug ved vidt forskellige hastigheder. Tilstanden ligger i Redis, så en grænse deles på tværs af instanser i stedet for at gælde pr. proces. Den afgørende beslutning var, hvad der sker, når Redis er nede: limiterne afviser trafikken frem for at vinke den igennem. Siden lægger ikke skjul på prisen, at et Redis-problem bliver et problem, brugeren kan se, eller på, hvorfor den blev accepteret: alternativet var, at et Redis-problem blev et problem på regningen. Omkring dem sidder Cloudflare Turnstile, en origin-lås og 351 linjers idempotency-middleware, så en skrivning, der bliver forsøgt igen, ikke bliver til to.

Hvor det går galt

Hver server fører sit eget regnskab. En grænse, der holdes i proceshukommelsen, ganges op med hver kopi af applikationen: fire instanser bag en load balancer lukker en klient igennem med fire gange den tilsigtede rate, og hver genstart nulstiller tællerne. nginx' zone er delt hukommelse på én maskine, og at holde zoner i takt på tværs af servere hører til det kommercielle abonnement; HAProxys peers-protokol kopierer poster i stick tables mellem instanser, men lægger dem, som manualen bemærker, ikke sammen. Hold tællerne i et lager, som alle instanser spørger, sådan som kataloget ovenfor gør med Redis, eller håndhæv grænsen på det ene sted, al trafik passerer.

Alle kommer fra samme adresse. Bag en reverse proxy, et CDN eller en load balancer ser applikationen én klientadresse for alle brugere, medmindre den oprindelige sendes med, og en grænse pr. IP-adresse bremser så alle på én gang. Et helt kontor eller mobilnet bag NAT har samme virkning i mildere form. Byg grænsen på den rigtige klientadresse fra en header, som kun din egen proxy må sætte, og foretræk kontoen eller API-nøglen frem for adressen, når trafikken er logget ind.

Limiteren melder en serverfejl. nginx' standardsvar på en afvist anmodning er 503 Service Unavailable, som i HTTP betyder, at serveren er overbelastet eller nede til vedligeholdelse. Dashboards og alarmer, der behandler 5xx-svar som serverfejl, vækker så nogen, fordi én klient var grådig, og klienten lærer intet om sin egen opførsel. Svar med 429, som lægger årsagen hos klienten, og log afvisningerne på et niveau for sig.

Alle afviste klienter kommer tilbage i samme sekund. Hvis en kvote nulstilles på et fast tidspunkt, eller hver afvist klient får den samme `Retry-After`, vender de alle tilbage på én gang; IETF-udkastet giver eksemplet med en daglig nulstilling kl. 18:00:00, hvor alle klienter sandsynligvis dukker op kl. 18:00:00. Det udbrud, der følger, kan være værre end det, der blev begrænset. Tilføj jitter i begge ender: variér den ventetid, serveren melder ud, og lad klienterne selv lægge en tilfældig forsinkelse til.

Ingen har besluttet, hvad der sker, når lageret går ned. En limiter, der holder sine tællere i Redis eller en database, må gøre noget, når lageret ikke kan nås. Kode, der fanger forbindelsesfejlen og lukker anmodningen igennem, har valgt at fejle åbent (fail open), hvilket gør udfaldet til et vindue uden grænser for den, der holder øje; kode, der afviser, har valgt at fejle lukket (fail closed), hvilket gør det til et udfald for alle. Ingen af delene er forkert overalt, men valget hører hjemme i en designbeslutning pr. sti, sådan som kataloget ovenfor traf den, og ikke i det, et bibliotek tilfældigvis gør som standard.

At tælle afvisninger lækker oplysninger. Hvis svar med 401 og 403 tæller mod en kvote, advarer IETF-udkastet, kan en ondsindet klient sondere et endpoint og få oplysninger om en anden brugers trafik. Beslut, hvilke svar der tæller, giv fejlslagen autentificering sin egen grænse, og vis ikke kvotedetaljer til klienter, der ikke er autentificeret.

Læs mere

- RFC 6585: Additional HTTP Status Codes — definerer 429 Too Many Requests og siger, hvad koden overlader til serveren: hvem brugeren er, og hvordan anmodninger tælles.
- RFC 9110: HTTP Semantics — feltet `Retry-After` og dets to former, og hvad 503 betyder, når serveren sender det.
- RateLimit header fields for HTTP — udkastet fra IETF-arbejdsgruppen HTTPAPI til `RateLimit-Policy` og `RateLimit`, med et sikkerhedsafsnit om klienter, der vender tilbage samtidig, og om informationslæk.
- Module ngx_http_limit_req_module — `limit_req`, `burst`, `nodelay`, `delay`, dry run-tilstanden og størrelserne på den delte hukommelseszone.
- HAProxy 3.4 Configuration Manual — stick tables og deres ratetællere, og peers-protokollen, der kopierer dem mellem instanser.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er rate limiting?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/rate-limiting/