Engineer

Hastighetsbegränsning

ett tak för antalet anrop från en klient per tidsenhet

Vad det är

Hastighetsbegränsning, på engelska rate limiting, är ett tak för hur många begäranden en klient får skicka till en tjänst inom en viss tidsrymd, upprätthållet av tjänsten själv eller av något som står framför den. Varje gräns måste besvara tre frågor: vem som räknas som klienten, hur begärandena räknas och vad som händer med den begäran som går över. HTTP-statuskoden för den sista frågan, 429 Too Many Requests, definierades i RFC 6585 i april 2012, och den RFC:n lämnar medvetet de två första öppna. En server kan räkna per resurs, för hela servern eller över en grupp servrar, och den kan känna igen klienten på dess autentiseringsuppgifter eller på en kaka (cookie) som bär tillstånd.

I praktiken är nyckeln en av en handfull saker: klientens IP-adress, det inloggade kontot, en API-nyckel eller någon av dem kombinerad med anropsvägen, så att ett inloggningsformulär och en produktsida får olika gränser. Svaret på en begäran över gränsen finns i tre varianter. Servern kan neka den med 429, helst med en `Retry-After` -header som säger hur länge klienten ska vänta, antingen som ett antal sekunder eller som ett HTTP-datum. Den kan hålla tillbaka begäran och behandla den en stund senare, så att en topp jämnas ut i stället för att nekas. Eller, när själva mängden är attacken, kan den stänga anslutningen utan att svara. Det är just vad RFC 6585 föreslår, eftersom det också kostar resurser att skriva ett artigt 429-svar till varenda begäran i en flod.

Hur räkningen går till avgör hur en gräns beter sig i kanterna. Ett fast fönster räknar begäranden per kalenderminut eller -timme och börjar om från noll när fönstret byts. Det är billigt och lätt att förklara, och det låter en klient göra av med hela sin kvot under den sista sekunden av ett fönster och igen under den första sekunden av nästa: med en gräns på 100 i minuten blir det 200 begäranden på två sekunder. Ett glidande fönster ser alltid lika långt bakåt från nuet, så det finns inget fönsterbyte att utnyttja; stick tables i HAProxy håller räknare av det slaget, och manualen beskriver ett av sina exempel som en

anslutningsfrekvens beräknad över ett glidande fönster på 30 sekunder. Leaky bucket-metoden, den ”läckande hinken”, som nginx använder, släpper igenom begäranden i en fast genomsnittlig takt, håller en liten kö för dem som kommer snabbare och nekar det som skulle få kön att svämma över. En token bucket fungerar tvärtom: polletter (tokens) samlas i jämn takt upp till ett tak, varje begäran förbrukar en, och en klient som har varit tyst ett tag kan använda sina besparingar i en enda topp.

Använd begreppet i ett beslut

Använd ”Vad är hastighetsbegränsning?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Konton, lösenord och behörigheter ordnade så att alla har exakt den åtkomst deras arbete kräver, och de som slutar förlorar den samma dag.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

En publik endpoint kan inte påverka vem som anropar den eller hur ofta. En skrapbot som kopierar en katalog, ett skript som har fastnat i en loop av nya försök, en angripare som provar läckta lösenord mot inloggningsformuläret och en partner vars nattliga jobb skrevs i all hast ser likadana ut för servern: fler begäranden än den var dimensionerad för, från några få källor. Utan en gräns är det den flitigaste anroparen som avgör hur mycket kapacitet alla andra får. Med en gräns kan en enskild klient bara använda sin egen andel, och resten av tjänsten fortsätter i normal takt medan den klienten väntar.

Vissa begäranden är mycket dyrare än andra. En produktsida kan levereras från en cache på en millisekund, medan en sökning, en export eller allt som anropar ett betalt externt API kostar riktig processortid eller riktiga pengar vid varje anrop. Det är de anropsvägarna en missbrukare går efter, just för att de inte kostar anroparen någonting. En enda global gräns som är låg nog att skydda exporten skulle strypa vanlig surfning, så användbar hastighetsbegränsning sätts per anropsväg, med varje gräns vald efter vad vägen kostar och hur den används legitimt.

En gräns är också en del av kontraktet för ett API. Klienter som känner till reglerna kan anpassa takten och aldrig slå i dem, vilket är billigare för båda parter än att upptäcka gränsen genom att bli nekade. HTTP har ingen standardheader

för det här. Många API:er skickar `X-RateLimit-Limit`, `X-RateLimit-Remaining` och `X-RateLimit-Reset`, men samma namn bär olika värden från en implementation till nästa, vilket ett IETF-utkast som vill standardisera idén redogör för. Utkastet definierar två fält, `RateLimit-Policy` för kvoten och fönstret och `RateLimit` för det som återstår; den elfte versionen är daterad 23 maj 2026 och är, som varje Internet-Draft, ett pågående arbete och inte en standard.

Så ser det ut i praktiken

I nginx kräver en gräns två direktiv: ett i `http`-blocket som sätter upp räknaren, och ett där gränsen ska gälla:

```
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;
    }
}
```

Nyckeln är klientens adress i binär form, som tar 4 byte för IPv4 och 16 för IPv6, och räknarna ligger i en delad minneszon på 10 megabyte som heter `perip`. Dokumentationen anger tillståndet för en nyckel till 128 byte på en 64-bitarsplattform, så en megabyte rymmer cirka åttatusen adresser och den här zonen cirka åttiotusen; när den blir full kastar nginx ut den minst nyligen använda adressen. `rate=1r/s` tillåter en begäran per sekund i genomsnitt. `burst=5` låter upp till fem begäranden över den takten vänta på sin tur i stället för att nekas, och `nodelay` behandlar dem direkt i stället för att sprida ut dem i den konfigurerade takten. Allt utöver burst nekas. nginx skulle som standard svara på det med 503, och därför sätter `limit_req_status` 429. Takter under en per sekund skrivs per minut, så `30r/m` är en begäran varannan sekund. Innan en ny gräns tas i bruk räknar `limit_req_dry_run on` som vanligt och markerar de begäranden som skulle ha nekats som `REJECTED_DRY_RUN` i variabeln `$limit_req_status`, utan att neka något.

På mottagarsidan bör en klient som slår i gränsen läsa svaret i stället för att försöka igen direkt. Ett 429-svar från ett API som följer IETF-utkastet kan se ut så här:

```
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
```

Policyn tillåter 100 begäranden per fönster på 60 sekunder, inga återstår, och kvoten kommer tillbaka om 20 sekunder. Där båda fälten finns säger utkastet att `Retry-After` går före. En väluppfostrad klient väntar minst så länge, lägger till en liten slumpmässig fördröjning och faller, om det inte finns någon `Retry-After` alls, tillbaka på exponentiell backoff i stället för ett fast intervall.

De 22 hastighetsbegränsarna som stänger vid fel från 2025, byggda för en publik katalog över företag och yrkespersoner, följer regeln om en gräns per anropsväg: var och en är konstruerad för den väg den skyddar, eftersom ett inloggningsförsök, en sökning och en massexport blir missbruk vid helt olika frekvenser. Tillståndet ligger i Redis, så att en gräns delas mellan instanser i stället för att gälla per process. Det avgörande beslutet var vad som händer när Redis inte svarar: anropsbegränsarna nekar trafiken i stället för att vinka igenom den. Sidan döljer inte priset, att ett Redis-problem blir ett problem som användarna ser, och inte heller varför det accepterades: alternativet var att ett Redis-problem blev ett faktureringsproblem. Runt dem finns Cloudflare Turnstile, ett ursprungslås och 351 rader middleware för idempotens, så att en skrivning som görs om inte blir två.

Var det går fel

Varje server räknar för sig. En gräns som hålls i processminnet multipliceras med varje kopia av applikationen: fyra instanser bakom en lastbalanserare släpper igenom en klient i fyra gånger den avsedda takten, och varje omstart nollställer räknarna. Zonen i nginx är delat minne på en maskin, och att hålla zoner i takt mellan servrar ingår i den kommersiella prenumerationen; peers-protokollet i HAProxy kopierar poster i stick tables mellan instanser men, som manualen påpekar, utan att lägga ihop dem. Håll räknarna i ett lager som alla instanser frågar, som katalogen ovan gör med Redis, eller upprätthåll gränsen på den enda punkt som all trafik passerar.

Alla kommer från samma adress. Bakom en omvänd proxy, ett CDN eller en lastbalanserare ser applikationen en enda klientadress för alla användare om inte den ursprungliga skickas vidare, och en gräns per IP-adress stryper då alla på en

gång. Ett helt kontor eller mobilnät bakom NAT ger samma effekt i mildare form. Bygg gränsen på den verkliga klientadressen från en header som bara din egen proxy får sätta, och föredra kontot eller API-nyckeln framför adressen för inloggad trafik.

Begränsaren rapporterar ett serverfel. Standardsvaret från nginx på en nekad begäran är 503 Service Unavailable, som i HTTP betyder att servern är överbelastad eller nere för underhåll. Instrumentpaneler och larm som behandlar 5xx-svar som serverfel väcker då någon för att en enda klient var girig, och klienten lär sig ingenting om sitt eget beteende. Svara med 429, som lägger orsaken hos klienten, och logga nekandena på en egen nivå.

Alla nekade klienter kommer tillbaka i samma sekund. Om en kvot nollställs vid en fast tidpunkt, eller om varje nekad klient får samma `Retry-After`, kommer alla tillbaka samtidigt; IETF-utkastet ger exemplet med en daglig nollställning kl. 18:00:00, då alla klienter sannolikt dyker upp kl. 18:00:00. Toppen som följer kan bli värre än den som begränsades. Lägg till jitter i båda ändar: variera väntetiden som servern anger, och låt klienterna lägga till en egen slumpmässig fördröjning.

Ingen har bestämt vad som händer när lagret går ner. En begränsare som håller sina räknare i Redis eller en databas måste göra något när lagret inte går att nå. Kod som fångar anslutningsfelet och släpper igenom begäran har valt att stå öppen vid fel (fail open), vilket gör avbrottet till ett fönster utan gränser för den som håller utkik; kod som nekar har valt att stänga vid fel (fail closed), vilket gör det till ett avbrott för alla. Inget av dem är fel överallt, men valet hör hemma i ett designbeslut per anropsväg, så som katalogen ovan fattade det, och inte i vad ett bibliotek råkar göra som standard.

Att räkna nekanden läcker information. Om svar med 401 och 403 räknas mot en kvot, varnar IETF-utkastet, kan en illvillig klient sondera en endpoint för att få veta något om en annan användares trafik. Bestäm vilka svar som räknas, ge misslyckad autentisering en egen gräns och visa inte kvotdetaljer för klienter som inte har autentiserat sig.

Läs mer

- RFC 6585: Additional HTTP Status Codes — definierar 429 Too Many Requests och säger vad koden lämnar åt servern: vem användaren är och hur begäranden räknas.

- RFC 9110: HTTP Semantics — fältet `Retry-After` och dess två former, och vad 503 betyder när servern skickar det.
- RateLimit header fields for HTTP — IETF-arbetsgruppen HTTPAPI:s utkast till `RateLimit-Policy` och `RateLimit`, med ett säkerhetsavsnitt om klienter som återvänder samtidigt och om informationsläckor.
- Module ngx_http_limit_req_module — `limit_req`, `burst`, `nodelay`, `delay`, testkörningsläget (dry run) och storlekarna på den delade minneszonen.
- HAProxy 3.4 Configuration Manual — stick tables och deras frekvensräknare, och peers-protokollet som kopierar dem mellan instanser.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är hastighetsbegränsning?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/rate-limiting/