Engineer

Lastbalanserare

⚖ en server som fördelar anrop mellan flera andra

Vad det är

En lastbalanserare är en server som tar emot begäranden från klienter och fördelar dem över en pool av andra servrar som alla gör samma jobb. Ingen enskild maskin bär hela lasten, och när en faller bort tar den inte med sig tjänsten. På engelska heter den load balancer, och många svenska utvecklare använder det engelska ordet också; i konfigurationsfiler och diagram förkortas det ofta till LB. Själva arbetet kallas lastbalansering. Klienterna ser bara en adress, nämligen lastbalanserarens. Servrarna bakom den bildar en pool, som nginx kallar en upstream-grupp och HAProxy en backend.

För varje begäran fattar en lastbalanserare två beslut: vilka servrar som är friska nog att ta emot den, och vilken av dem som får den. Det andra beslutet är balanseringsmetoden. Round robin ger servrarna begäranden i tur och ordning och är vad nginx gör när inget annat är inställt. Least connections skickar nästa begäran till den server som har minst antal aktiva anslutningar och jämnar på så sätt ut lasten när vissa begäranden tar mycket längre tid än andra. HAProxys manual rekommenderar metoden för långa sessioner som LDAP och SQL och kallar den illa lämpad för korta sessioner som HTTP. Hashning gör en nyckel, till exempel klientens IP-adress eller den begärda URL:en, till ett val av server, så att samma nyckel fortsätter att hamna på samma maskin. nginx har också en slumpmässig metod som kan dra två servrar och skicka begäran till den av dem som har minst att göra. Vikter drar valet mot de större maskinerna: i exemplet i nginx egen guide får en server med `weight=3` tre av fem begäranden, medan två servrar med standardvikten 1 får en var.

Lastbalanserare skiljer sig också åt i hur mycket av trafiken de läser. I HAProxys `mode tcp`, som är standard, kopplas klient och server ihop i en anslutning med full duplex, utan någon granskning alls på lager 7, och det passar för TLS som skickas igenom oöppnad, för SSH och för SMTP. I `mode http` tolkas varje begäran innan servern väljs: begäranden som bryter mot protokollet avvisas, och trafiken

kan styras efter sökväg eller header. En lastbalanserare som arbetar på HTTP-nivå avslutar klientens anslutning och öppnar en egen mot servern, så applikationen ser lastbalanseraren som sin enda klient om inte den ursprungliga adressen skickas vidare, oftast i headern `X-Forwarded-For`. Därför är en lastbalanserare och en omvänd proxy så ofta samma process: nginx är båda, och dess egen guide beskriver lastbalansering som en del av dess implementation av omvänd proxy.

Använd begreppet i ett beslut

Använd ”Vad är en lastbalanserare?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Servrar dimensionerade efter vad du faktiskt använder, flyttade till molnet eller tillbaka ut ur det, med månadskostnaden uträknad först.

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. Drev 20 Docker-appar på Google Cloud och AWS — 20 containerbaserade applikationer, driftsatta och övervakade

Drev 20 Docker-appar på Google Cloud och AWS Hör av er →

Varför det spelar roll

En server har ett tak. När en tjänst växer ur det är det lastbalanseraren som låter ett team ställa upp en andra och en tredje kopia i stället för att köpa en större maskin, och varje ny kopia bakom den ger mer kapacitet utan att klienterna märker att poolen har ändrats. Samma upplägg gör underhåll till vardag: ta en server ur rotationen, uppdatera den, kontrollera den, sätt in den igen och upprepa med nästa, medan användarna betjänas av resten.

Det gör också att en server som faller bort blir en mindre händelse än ett avbrott. När en server slutar svara slutar lastbalanseraren att välja den, och de övriga servrarna tar över dess andel. Det fungerar bara om de har kapacitet över. Tre servrar som var och en kör på 80 % av sin kapacitet kan inte täcka upp för varandra: faller en bort får var och en av de två andra 120 % av en servers arbete.

En lastbalanserare ändrar också hur applikationen måste skrivas. Med round robin eller least connections kan två begäranden i följd från samma klient hamna på två olika servrar, vilket nginx guide påpekar. Allt en server håller i sitt eget minne eller på sin egen disk mellan två begäranden, till exempel en inloggningssession, en uppladdad fil som väntar på bearbetning eller en cache av användarens varukorg, är borta så fort nästa begäran landar på en annan server. Antingen lägger applikationen det tillståndet någonstans där alla servrar når det, i en databas eller en delad cache, eller så binder lastbalanseraren varje klient till en server, så som nginx `ip_hash` gör, utom när den servern inte går att nå.

Till sist blir lastbalanseraren själv den enda maskin som varje begäran måste passera. Tio applikationsservrar bakom en enda lastbalanserare har fortfarande en enskild felpunkt; den har bara flyttat. Ett seriöst upplägg kör lastbalanseraren som ett par med en adress som kan flytta mellan de två, eller använder en hanterad lastbalanserare där leverantören står för redundansen.

Så ser det ut i praktiken

Här fördelar en nginx-server begäranden över tre applikationsservrar och håller en fjärde i reserv:

```
upstream app {
    least_conn;
    server 10.0.0.11:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.12:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.13:8080 max_fails=3 fail_timeout=30s weight=2;
    server 10.0.0.20:8080 backup;
}

server {
    listen 80;
    location / {
        proxy_pass http://app;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_next_upstream error timeout http_502 http_503;
        proxy_next_upstream_tries 2;
        proxy_connect_timeout 2s;
    }
}
```

`least_conn` skickar varje ny begäran till den server som har minst antal aktiva anslutningar och avgör lika läge med viktad round robin. Den tredje servern är den största maskinen, så `weight=2` räknar den dubbelt. `max_fails=3 fail_timeout=30s` betyder att tre misslyckade försök inom 30 sekunder tar en server ur poolen i 30 sekunder, varefter nginx provar den igen med riktiga begäranden och tar tillbaka den om de lyckas. Utan de parametrarna gäller standardvärdena, och då räcker ett enda fel inom 10 sekunder. `backup` -servern får begäranden bara medan alla tre primära servrar är otillgängliga.

I `location` -blocket avgör `proxy_next_upstream` när en begäran lämnas vidare till en annan server: vid ett fel i kommunikationen med servern, vid en timeout och vid ett 502- eller 503-svar. `proxy_next_upstream_tries 2` begränsar hur många

gånger det kan hända för en begäran; standardvärdet 0 betyder ingen gräns. De två `proxy_set_header` -raderna behåller det värdnamn som klienten bad om, som nginx annars skulle byta ut mot upstream-namnet `app` , och lägger till klientens adress i `X-Forwarded-For` så att applikationen kan logga de verkliga besökarna i stället för lastbalanseraren. `proxy_connect_timeout 2s` ger upp en server som inte har tagit emot anslutningen inom två sekunder, där standardvärdet skulle ha väntat i 60.

De här kontrollerna är passiva: nginx märker att en server är sjuk först när riktiga begäranden till den misslyckas. HAProxy kan dessutom själv undersöka varje server enligt ett eget schema. `option httpchk GET /healthz` i en backend, tillsammans med `check` på varje `server` -rad, skickar den begäran varannan sekund som standard; tre fel i rad markerar servern som nere, och två lyckade svar tar tillbaka den. Utan `check` kör HAProxy inga hälsokontroller och betraktar servern som alltid tillgänglig.

Innan nginx är en lastbalanserare är det en webbserver, och en helt vanlig sådan. Det är en av de två webbservrarna bakom 40 webbplatser administrerade på Apache och Nginx mellan 2003 och 2014, som kördes på hostingservrar med Ubuntu Linux och konfigurerades, prestandaoptimerades och underhölls för att hålla sig tillförlitliga under verklig trafik. `backup` -raden har en släkting en nivå upp, där servrarna är hela leverantörer. E-post med failover mellan tre leverantörer har SendGrid som primär med SMTP2GO och Azure Communication Services bakom och växlar automatiskt mellan dem, så att en dålig eftermiddag hos en leverantör ledde till ett nedsatt driftläge i stället för ett avbrott.

Var det går fel

En server som svarar 500 på alla begäranden stannar kvar i poolen. Som standard räknar nginx bara fel i kommunikationen med servern, timeouter och ogiltiga svarsheaders som misslyckade försök. En server vars applikation har tappat kontakten med databasen och returnerar 500 på allt svarar snabbt och artigt, så lastbalanseraren fortsätter att skicka den dess andel av trafiken. Svaren 500, 502, 503, 504 och 429 räknas mot en server bara när de står i `proxy_next_upstream` , och 403 och 404 räknas aldrig. Skriv upp de koder som betyder ”servern är trasig”, och ge servrarna en aktiv kontroll mot en endpoint som använder allt applikationen behöver för att leverera en sida.

En långsam begäran körs på varje server i tur och ordning. `timeout` ingår i standardvärdet för `proxy_next_upstream`, och `proxy_read_timeout` är 60 sekunder som standard. En rapport som tar 70 sekunder att bygga får timeout på den första servern, lämnas vidare till den andra, får timeout igen och går vidare till den tredje, så klienten väntar i tre minuter medan varje server gör samma arbete. En timeout räknas alltid som ett misslyckat försök, så med standardvärdet 1 för `max_fails` markeras också varje server som begäran har passerat som otillgänglig i 10 sekunder. Begränsa vidarebefordringarna med `proxy_next_upstream_tries` och `proxy_next_upstream_timeout`, ge den långsamma endpointen en egen `location` med längre timeout, eller flytta arbetet bort från själva begäran och in i ett bakgrundsjobb.

En betalning går igenom två gånger. nginx skickar som standard inte en POST-, LOCK- eller PATCH-begäran vidare till en annan server när den väl har nått en, eftersom den första servern redan kan ha agerat på den. Lägger man till `non_idempotent` i `proxy_next_upstream` för att få timeouterna att försvinna tar man bort det skyddet: en betalning som den första servern genomförde men svarade för långsamt på genomförs igen av den andra. Låt inställningen vara avstängd om inte varje endpoint bakom den är idempotent, till exempel för att den tar emot en idempotensnyckel och vägrar upprepa en begäran den redan har sett.

Alla besökare ser ut att komma från lastbalanseraren. Utan headrarna från exemplet loggar applikationen lastbalanserarens adress för varje begäran, och varje gräns som hålls per IP-adress drabbar nu alla användare på en gång. `X-Forwarded-For` har en egen fälla. `$proxy_add_x_forwarded_for` lägger till klientens adress efter den `X-Forwarded-For` som klienten själv skickade med, så de första posterna är vad klienten valde att skriva. Applikationen bör bara lita på den post som dess egen lastbalanserare har lagt till, och bara ta emot headern från lastbalanserarens adress.

Ett helt kontor hamnar på en server. nginx `ip_hash` hashar de tre första oktetterna i en IPv4-adress, så alla klienter som delar dem hamnar på samma server, och ett helt företag bakom en NAT-gateway kommer in som en enda klient. Ändringar i poolen ger den motsatta överraskningen. Med en vanlig `hash` kan det att lägga till eller ta bort en server flytta de flesta nycklar till en annan server, och varje användare som var bunden på det sättet förlorar det som den gamla servern höll. nginx dokumentation råder till att markera en server med `down` i stället för att ta bort den ur en `ip_hash`-grupp, och erbjuder `hash ... consistent`, som bara flyttar några få nycklar när poolen ändras. Ännu bättre är att hålla sessionstillståndet borta från servrarna, för då spelar det ingen roll var en klient hamnar.

Läs mer

- Using nginx as HTTP load balancer — de tre grundläggande metoderna, vikter, sessionsbeständighet och de passiva hälsokontrollerna, var och en med en kort konfiguration.
- Module ngx_http_upstream_module — alla parametrar på en `server`-rad, bland dem `max_fails`, `backup` och undantaget för grupper med en enda server, samt metoderna `hash`, `ip_hash`, `least_conn` och `random`.
- Module ngx_http_proxy_module — `proxy_next_upstream` och vad som räknas som ett misslyckat försök, timeouterna och de headers som nginx sätter på vägen till servern.
- HAProxy 3.4 Configuration Manual — `balance` och dess algoritmer, TCP- och HTTP-läget och aktiva hälsokontroller med `check`, `inter`, `rise` och `fall`.

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 en lastbalanserare?” 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/load-balancer/