Engineer

Lastbalanserer

⚖ en server som fordeler forespørsler på flere andre

Hva det er

En lastbalanserer er en server som tar imot forespørsler fra klienter og fordeler dem på en pool av andre servere som alle gjør den samme jobben. Ingen enkelt maskin bærer hele lasten, og når én faller ut, tar den ikke tjenesten med seg. På engelsk heter den load balancer, og mange norske utviklere bruker det engelske ordet også; i konfigurasjonsfiler og diagrammer forkortes det gjerne til LB. Selve arbeidet kalles lastbalansering. Klientene ser bare én adresse, nemlig lastbalanserernes. Serverne bak den utgjør en pool, som nginx kaller en upstream-gruppe og HAProxy en backend.

For hver forespørsel tar en lastbalanserer to avgjørelser: hvilke servere som er friske nok til å få den, og hvem av dem som får den. Den andre avgjørelsen er balanseringsmetoden. Round robin gir serverne forespørslene etter tur og er det nginx gjør når ingenting annet er satt opp. Least connections sender neste forespørsel til serveren med færrest aktive tilkoblinger og jevner dermed ut lasten når noen forespørsler tar mye lengre tid enn andre. HAProxy-manualen anbefaler metoden for lange økter som LDAP og SQL og kaller den dårlig egnet for korte økter som HTTP. Hashing gjør en nøkkel, for eksempel klientens IP-adresse eller den forespurte URL-en, om til et valg av server, slik at den samme nøkkelen fortsetter å havne på den samme maskinen. nginx har også en tilfeldig metode som kan trekke to servere og sende forespørselen til den av dem som har minst å gjøre. Vekter trekker valget mot de større maskinene: i nginx' egen veiledning får én server med `weight=3` og to med standardvekten 1 henholdsvis tre, én og én av hver fem forespørsler.

Lastbalanserere skiller seg også i hvor mye av trafikken de leser. I HAProxys `mode tcp`, som er standard, kobles klient og server sammen i en tilkobling med full dupleks, uten noen undersøkelse på lag 7, og det passer for TLS som sendes uåpnet gjennom, for SSH og for SMTP. I `mode http` analyseres hver forespørsel før serveren velges: forespørsler som bryter protokollen, avvises, og trafikken

kan rutes etter sti eller header. En lastbalanserer som jobber på HTTP-nivå, avslutter klientens tilkobling og åpner sin egen mot serveren, så applikasjonen ser lastbalansereren som sin eneste klient, med mindre den opprinnelige adressen sendes videre, som regel i headeren `X-Forwarded-For`. Derfor er en lastbalanserer og en omvendt proxy så ofte den samme prosessen: nginx er begge deler, og dens egen veiledning beskriver lastbalansering som en del av implementasjonen av omvendt proxy.

Bruk begrepet i en beslutning

Bruk «Hva er en lastbalanserer?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Servere dimensjonert etter det du faktisk bruker, flyttet inn i skyen eller ut av den igjen, med månedsregningen regnet ut på forhånd.

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. Driftet 20 Docker-apper på Google Cloud og AWS — 20 containeriserte applikasjoner produksjonssatt og overvåket

Driftet 20 Docker-apper på Google Cloud og AWS Ta kontakt →

Hvorfor det er viktig

En server har et tak. Når en tjeneste vokser ut av det, er det lastbalansereren som lar et team sette opp en andre og en tredje kopi i stedet for å kjøpe en større maskin, og hver ny kopi bak den gir mer kapasitet uten at klientene merker at poolen har endret seg. Det samme oppsettet gjør vedlikehold til hverdagskost: ta én server ut av rotasjonen, oppdater den, kontroller den, sett den inn igjen og gjenta med den neste, mens brukerne betjenes av resten.

Det gjør også at en server som faller ut, blir en mindre hendelse enn et driftsavbrudd. Når en server slutter å svare, slutter lastbalansereren å velge den, og de øvrige serverne overtar andelen dens. Det fungerer bare hvis de har kapasitet til overs. Tre servere som hver kjører på 80 % av kapasiteten sin, kan ikke dekke for hverandre: faller én ut, får hver av de to andre 120 % av én servers arbeid.

En lastbalanserer endrer også hvordan applikasjonen må skrives. Med round robin eller least connections kan to forespørsler på rad fra den samme klienten havne på to forskjellige servere, som nginx' veiledning påpeker. Alt en server holder i sitt eget minne eller på sin egen disk mellom to forespørsler, for eksempel en innloggingsøkt, en opplastet fil som venter på behandling eller en cache av brukerens handlekurv, er borte så snart neste forespørsel lander på en

annen server. Enten legger applikasjonen den tilstanden et sted alle serverne når, i en database eller en felles cache, eller så binder lastbalansereren hver klient til én server, slik nginx' `ip_hash` gjør, unntatt når den serveren er utilgjengelig.

Til slutt blir lastbalansereren selv den ene maskinen som hver eneste forespørsel må gjennom. Ti applikasjonsservere bak én lastbalanserer har fortsatt et enkelt feilpunkt; det har bare flyttet seg. Et seriøst oppsett kjører lastbalansereren som et par med en adresse som kan flytte mellom de to, eller bruker en administrert lastbalanserer der leverandøren står for redundansen.

Slik ser det ut i praksis

Her fordeler en nginx-server forespørsler på tre applikasjonsservere og holder en fjerde i reserve:

```
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` sender hver nye forespørsel til serveren med færrest aktive tilkoblinger og avgjør uavgjort med vektet round robin. Den tredje serveren er den største maskinen, så `weight=2` teller den dobbelt. `max_fails=3` `fail_timeout=30s` betyr at tre mislykkede forsøk innen 30 sekunder tar en server ut av poolen i 30 sekunder, og deretter prøver nginx den igjen med ekte forespørsler og tar den inn igjen hvis de lykkes. Uten disse parameterne gjelder standardverdiene, og da er én feil innen 10 sekunder nok. `backup` -serveren får bare forespørsler mens alle de tre primærserverne er utilgjengelige.

I `location`-blokken avgjør `proxy_next_upstream` når en forespørsel gis videre til en annen server: ved en feil i kommunikasjonen med serveren, ved tidsavbrudd og ved et 502- eller 503-svar. `proxy_next_upstream_tries 2` begrenser hvor mange ganger det kan skje for én forespørsel; standardverdien 0 betyr ingen grense. De to `proxy_set_header`-linjene bevarer vertsnavnet klienten ba om, som nginx ellers ville erstattet med upstream-navnet `app`, og legger klientens adresse til i `X-Forwarded-For`, slik at applikasjonen kan logge de virkelige besøkende i stedet for lastbalansereren. `proxy_connect_timeout 2s` gir opp en server som ikke har tatt imot tilkoblingen innen to sekunder, der standarden ville ventet i 60.

Disse kontrollene er passive: nginx oppdager først at en server er syk når ekte forespørsler til den feiler. HAProxy kan også selv undersøke hver server etter sin egen timeplan. `option httpchk GET /healthz` i en backend, sammen med `check` på hver `server`-linje, sender den forespørselen hvert annet sekund som standard; tre feil på rad markerer serveren som nede, og to vellykkede svar tar den inn igjen. Uten `check` kjører HAProxy ingen helsesjekker og regner serveren som alltid tilgjengelig.

Før nginx er en lastbalanserer, er den en webserver, og en helt vanlig en. Den er en av de to webserverne bak 40 nettsteder administrert på Apache og Nginx fra 2003 til 2014, som kjørte på hostingservere med Ubuntu Linux og ble konfigurert, ytelsesjustert og vedlikeholdt for å holde seg pålitelige under reell trafikk. `backup`-linjen har en slektning ett nivå opp, der serverne er hele leverandører. E-post med failover mellom tre leverandører har SendGrid som primær med SMTP2GO og Azure Communication Services bak og bytter automatisk mellom dem, slik at en dårlig ettermiddag hos en leverandør førte til degradert drift i stedet for et driftsavbrudd.

Hvor det går galt

En server som svarer 500 på alle forespørsler, blir i poolen. Som standard teller nginx bare feil i kommunikasjonen med serveren, tidsavbrudd og ugyldige svarheadere som mislykkede forsøk. En server der applikasjonen har mistet forbindelsen til databasen og returnerer 500 på alt, svarer raskt og høflig, så lastbalansereren fortsetter å sende den sin andel av trafikken. Svarene 500, 502, 503, 504 og 429 teller bare mot en server når de står i `proxy_next_upstream`, og 403 og 404 teller aldri. Før opp kodene som betyr «serveren er i stykker», og gi serverne en aktiv kontroll mot et endepunkt som bruker alt applikasjonen trenger for å levere en side.

Én treg forespørsel kjører på hver server etter tur. `timeout` er med i standardverdien for `proxy_next_upstream`, og `proxy_read_timeout` er 60 sekunder som standard. En rapport som tar 70 sekunder å bygge, får tidsavbrudd på den første serveren, sendes videre til den andre, får tidsavbrudd igjen og går videre til den tredje, så klienten venter i tre minutter mens hver server gjør det samme arbeidet. Et tidsavbrudd teller alltid som et mislykket forsøk, så med standardverdien 1 for `max_fails` blir hver server forespørselen har vært innom, også markert som utilgjengelig i 10 sekunder. Begrens videresendingene med `proxy_next_upstream_tries` og `proxy_next_upstream_timeout`, gi det trege endepunktet sin egen `location` med lengre tidsavbrudd, eller flytt arbeidet ut av selve forespørselen og over i en bakgrunnsjobb.

En betaling går gjennom to ganger. nginx sender som standard ikke en POST-, LOCK- eller PATCH-forespørsel videre til en annen server når den først har nådd én, fordi den første serveren allerede kan ha handlet på den. Legger man til `non_idempotent` i `proxy_next_upstream` for å få tidsavbruddene til å forsvinne, fjerner man den beskyttelsen: en betaling som den første serveren gjennomførte, men svarte for sakte på, blir gjennomført på nytt av den andre. La innstillingen være av med mindre hvert endepunkt bak den er idempotent, for eksempel fordi det tar imot en idempotensnøkkel og nekter å gjenta en forespørsel det allerede har sett.

Alle besøkende ser ut til å komme fra lastbalansereren. Uten headerne fra eksempelet logger applikasjonen lastbalanserernes adresse for hver forespørsel, og enhver grense som holdes per IP-adresse, rammer nå alle brukerne på én gang. `X-Forwarded-For` har sin egen felle. `$proxy_add_x_forwarded_for` legger klientens adresse til den `X-Forwarded-For` klienten selv sendte med, så de første oppføringene er det klienten valgte å skrive. Applikasjonen bør bare stole på oppføringen dens egen lastbalanserer har lagt til, og bare godta headeren fra lastbalanserernes adresse.

Et helt kontor havner på én server. nginx' `ip_hash` hasher de tre første oktettene i en IPv4-adresse, så alle klienter som deler dem, havner på samme server, og en hel bedrift bak én NAT-gateway kommer inn som én klient. Endringer i poolen gir den motsatte overraskelsen. Med en vanlig `hash` kan det å legge til eller fjerne en server flytte de fleste nøklene til en annen server, og hver bruker som var bundet på den måten, mister det den gamle serveren hadde. nginx-dokumentasjonen råder til å markere en server med `down` i stedet for å slette den fra en `ip_hash` -gruppe, og tilbyr `hash ... consistent`, som bare flytter noen få nøkler når poolen endres. Enda bedre er det å holde økttilstanden borte fra serverne, for da spiller det ingen rolle hvor en klient havner.

Les mer

- Using nginx as HTTP load balancer — de tre grunnleggende metodene, vekter, øktpersistens og de passive helsesjekkene, hver med en kort konfigurasjon.
- Module ngx_http_upstream_module — alle parameterne på en `server`-linje, blant dem `max_fails`, `backup` og unntaket for grupper med én server, og metodene `hash`, `ip_hash`, `least_conn` og `random`.
- Module ngx_http_proxy_module — `proxy_next_upstream` og hva som teller som et mislykket forsøk, tidsavbruddene og headerne nginx setter på veien til serveren.
- HAProxy 3.4 Configuration Manual — `balance` og algoritmene dens, TCP- og HTTP-modusen og aktive helsesjekker med `check`, `inter`, `rise` og `fall`.

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 en lastbalanserer?» 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/load-balancer/