Engineer

Load balancer

⚖ en server, der fordeler anmodninger på flere andre

Hvad det er

En load balancer er en server, der tager imod anmodninger fra klienter og fordeler dem på en pulje af andre servere, som alle udfører det samme arbejde. Ingen enkelt maskine bærer hele belastningen, og når én falder ud, tager den ikke tjenesten med sig. Selve arbejdet hedder load balancing, af og til belastningsfordeling på dansk, og i konfigurationsfiler og diagrammer forkortes det tit til LB. Klienterne ser kun én adresse, nemlig load balancerens. Serverne bag den udgør en pulje, som nginx kalder en upstream-gruppe og HAProxy en backend.

For hver anmodning træffer en load balancer to beslutninger: hvilke servere der er raske nok til at få den, og hvem af dem der får den. Den anden beslutning er balanceringsmetoden. Round robin giver serverne anmodningerne på skift og er det, nginx gør, når intet andet er sat op. Least connections sender den næste anmodning til den server, der har færrest aktive forbindelser, og udjævner dermed belastningen, når nogle anmodninger tager langt længere tid end andre. HAProxys manual anbefaler metoden til lange sessioner som LDAP og SQL og kalder den dårligt egnet til korte sessioner som HTTP. Hashing gør en nøgle, for eksempel klientens IP-adresse eller den URL, der bliver bedt om, til et valg af server, så den samme nøgle bliver ved med at lande på den samme maskine. nginx har også en tilfældig metode, der kan trække to servere og sende anmodningen til den af dem, der har mindst at lave. Vægte trækker valget mod de større maskiner: i nginx' egen vejledning får én server med `weight=3` og to med standardvægten 1 henholdsvis tre, én og én af hver fem anmodninger.

Load balancere adskiller sig også i, hvor meget af trafikken de læser. I HAProxys `mode tcp`, som er standard, kobles klient og server sammen i en fuld duplex-forbindelse uden nogen undersøgelse på lag 7, og det egner sig til TLS, der sendes uåbnet igennem, til SSH og til SMTP. I `mode http` analyseres hver anmodning, før der vælges server: anmodninger, der bryder protokollen, afvises,

og trafikken kan styres efter sti eller header. En load balancer, der arbejder på HTTP-niveau, afslutter klientens forbindelse og åbner sin egen til serveren, så applikationen ser load balanceren som sin eneste klient, medmindre den oprindelige adresse bliver sendt med, som regel i headeren `X-Forwarded-For`. Derfor er en load balancer og en reverse proxy så ofte den samme proces: nginx er begge dele, og dens egen vejledning beskriver load balancing som en del af dens reverse proxy-implementering.

Brug begrebet i en beslutning

Brug »Hvad er en load balancer?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Servere dimensioneret efter det, I rent faktisk bruger, flyttet op i skyen eller hjem igen, med den månedlige regning regnet ud først.

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 drevet 20 Docker-apps på Google Cloud og AWS — 20 containeriserede applikationer udrullet og overvåget

Har drevet 20 Docker-apps på Google Cloud og AWS Kontakt os →

Hvorfor det betyder noget

En server har et loft. Når en tjeneste vokser ud over det, er det load balanceren, der lader et team sætte en anden og en tredje kopi op i stedet for at købe en større maskine, og hver ny kopi bag den giver mere kapacitet, uden at klienterne opdager, at puljen har ændret sig. Den samme opstilling gør vedligeholdelse til hverdag: tag én server ud af rotationen, opdater den, kontrollér den, sæt den ind igen, og gentag med den næste, mens brugerne bliver betjent af resten.

Den gør også en server, der falder ud, til en mindre hændelse end et nedbrud. Når en server holder op med at svare, holder load balanceren op med at vælge den, og de øvrige servere overtager dens andel. Det virker kun, hvis de har luft til det. Tre servere, der hver kører med 80 % af deres kapacitet, kan ikke dække for hinanden: falder én ud, får hver af de to andre 120 % af en servers arbejde.

En load balancer ændrer også, hvordan applikationen skal skrives. Med round robin eller least connections kan to anmodninger i træk fra den samme klient ramme to forskellige servere, som nginx' vejledning gør opmærksom på. Alt, hvad en server gemmer i sin egen hukommelse eller på sin egen disk mellem to anmodninger, for eksempel en login-session, en uploadet fil, der venter på behandling, eller en cache af brugerens indkøbskurv, er væk, så snart den næste anmodning lander på en anden server. Enten lægger applikationen den tilstand

et sted, som alle servere kan nå, i en database eller en fælles cache, eller også binder load balanceren hver klient til én server, sådan som nginx' `ip_hash` gør, undtagen når den server er utilgængelig.

Til sidst bliver load balanceren selv den ene maskine, som hver eneste anmodning skal igennem. Ti applikationsservere bag én load balancer har stadig et enkelt fejlpunkt; det er bare flyttet. En seriøs opsætning kører load balanceren som et par med en adresse, der kan flytte mellem de to, eller bruger en administreret load balancer, hvor udbyderen står for redundansen.

Sådan ser det ud i praksis

Her fordeler en nginx-server anmodninger på tre applikationsservere 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 ny anmodning til den server, der har færrest aktive forbindelser, og afgør stillingen med vægtet round robin, når to står lige. Den tredje server er den største maskine, så `weight=2` tæller den dobbelt. `max_fails=3 fail_timeout=30s` betyder, at tre mislykkede forsøg inden for 30 sekunder tager en server ud af puljen i 30 sekunder, hvorefter nginx prøver den igen med rigtige anmodninger og tager den ind igen, hvis de lykkes. Uden de

parametre gælder standardværdierne, og så er én fejl inden for 10 sekunder nok. `backup` -serveren får kun anmodninger, mens alle tre primære servere er utilgængelige.

I `location` -blokken afgør `proxy_next_upstream` , hvornår en anmodning gives videre til en anden server: ved en fejl i kommunikationen med serveren, ved en timeout og ved et 502- eller 503-svar. `proxy_next_upstream_tries 2` begrænser, hvor mange gange det kan ske for én anmodning; standardværdien 0 betyder ingen grænse. De to `proxy_set_header` -linjer bevarer det værtsnavn, klienten bad om, som nginx ellers ville erstatte med upstream-navnet `app` , og føjer klientens adresse til `X-Forwarded-For` , så applikationen kan logge de rigtige besøgende i stedet for load balanceren. `proxy_connect_timeout 2s` opgiver en server, der ikke har taget imod forbindelsen inden for to sekunder, hvor standarden ville vente i 60.

Disse tjek er passive: nginx opdager først, at en server er syg, når rigtige anmodninger til den fejler. HAProxy kan også selv undersøge hver server efter sin egen tidsplan. `option httpchk GET /healthz` i en backend, sammen med `check` på hver `server` -linje, sender den anmodning hvert andet sekund som standard; tre fejl i træk markerer serveren som nede, og to vellykkede svar tager den ind igen. Uden `check` kører HAProxy ingen health checks og betragter serveren som altid tilgængelig.

Før nginx er en load balancer, er det en webserver, og en helt almindelig af slagsen. Det er en af de to webservere bag 40 websites administreret på Apache og Nginx fra 2003 til 2014, som kørte på Ubuntu Linux-hostingservere og blev konfigureret, optimeret og vedligeholdt, så de forblev pålidelige under reel trafik. `backup` -linjen har en fætter et niveau højere oppe, hvor serverne er hele udbydere. E-mail med failover mellem tre udbydere har SendGrid som den primære med SMTP2GO og Azure Communication Services i reserve og skifter automatisk mellem dem, så en udbyder med en dårlig eftermiddag nu betyder en forringet levering snarere end et nedbrud.

Hvor det går galt

En server, der svarer 500 på alle anmodninger, bliver i puljen. Som standard tæller nginx kun fejl i kommunikationen med serveren, timeouts og ugyldige svarheadere som mislykkede forsøg. En server, hvis applikation har mistet forbindelsen til databasen og svarer 500 på alt, svarer hurtigt og høfligt, så load balanceren bliver ved med at sende den sin andel af trafikken. Svarene 500, 502,

503, 504 og 429 tæller kun imod en server, når de står i `proxy_next_upstream`, og 403 og 404 tæller aldrig. Skriv de koder på, der betyder »serveren er i stykker«, og giv serverne et aktivt tjek mod et endpoint, der bruger alt det, applikationen skal have for at levere en side.

Én langsom anmodning kører på alle servere efter tur. `timeout` er med i standardværdien for `proxy_next_upstream`, og `proxy_read_timeout` er som standard 60 sekunder. En rapport, der tager 70 sekunder at bygge, får timeout på den første server, sendes videre til den anden, får timeout igen og går videre til den tredje, så klienten venter tre minutter, mens hver server laver det samme arbejde. En timeout tæller altid som et mislykket forsøg, så med standardværdien 1 for `max_fails` bliver hver server, anmodningen har været forbi, også markeret som utilgængelig i 10 sekunder. Begræns videresendelserne med `proxy_next_upstream_tries` og `proxy_next_upstream_timeout`, giv det langsomme endpoint sin egen `location` med en længere timeout, eller flyt arbejdet væk fra selve anmodningen og over i et baggrundsjob.

En betaling går igennem to gange. nginx nægter som standard at sende en POST-, LOCK- eller PATCH-anmodning videre til en anden server, når den først har nået én, fordi den første server allerede kan have handlet på den. Tilføjer man `non_idempotent` til `proxy_next_upstream` for at få timeouts til at forsvinde, fjerner man den beskyttelse: en betaling, som den første server gennemførte, men svarede for langsomt på, bliver gennemført igen af den anden. Lad indstillingen være slået fra, medmindre hvert endpoint bag den er idempotent, for eksempel fordi det tager imod en idempotensnøgle og afviser at gentage en anmodning, det allerede har set.

Alle besøgende ser ud til at komme fra load balanceren. Uden headerne fra eksemplet logger applikationen load balancerens adresse for hver anmodning, og enhver grænse, der holdes pr. IP-adresse, rammer nu alle brugere på én gang. `X-Forwarded-For` har sin egen fælde. `$proxy_add_x_forwarded_for` føjer klientens adresse til den `X-Forwarded-For`, klienten selv sendte med, så de forreste poster er det, klienten valgte at skrive. Applikationen bør kun stole på den post, dens egen load balancer har tilføjet, og kun tage imod headeren fra load balancerens adresse.

Et helt kontor havner på én server. nginx' `ip_hash` hasher de første tre oktetter af en IPv4-adresse, så alle klienter, der deler dem, lander på samme server, og en hel virksomhed bag én NAT-gateway ankommer som én klient. Ændringer i puljen giver den modsatte overraskelse. Med et almindeligt `hash`

kan det at tilføje eller fjerne en server flytte de fleste nøgler til en anden server, og hver bruger, der var bundet på den måde, mister det, den gamle server havde. nginx' dokumentation råder til at markere en server med `down` i stedet for at slette den fra en `ip_hash` -gruppe og tilbyder `hash ... consistent` , som kun flytter nogle få nøgler, når puljen ændres. Endnu bedre er det at holde sessionstilstanden væk fra serverne, for så er det ligegyldigt, hvor en klient lander.

Læs mere

- Using nginx as HTTP load balancer — de tre grundlæggende metoder, vægte, sessionspersistens og de passive health checks, hver med en kort konfiguration.
- Module ngx_http_upstream_module — alle parametre på en `server` -linje, herunder `max_fails` , `backup` og undtagelsen for grupper med én server, samt metoderne `hash` , `ip_hash` , `least_conn` og `random` .
- Module ngx_http_proxy_module — `proxy_next_upstream` , og hvad der tæller som et mislykket forsøg, timeouts og de headere, nginx sætter på vejen til serveren.
- HAProxy 3.4 Configuration Manual — `balance` og dens algoritmer, TCP- og HTTP-tilstanden og aktive health checks med `check` , `inter` , `rise` og `fall` .

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 en load balancer?« 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/load-balancer/