Engineer

Load Balancer

⚖ Server, der Anfragen auf mehrere andere Server verteilt

Was es ist

Ein Load Balancer ist ein Server, der Anfragen von Clients entgegennimmt und auf einen Pool anderer Server verteilt, die alle dieselbe Aufgabe erfüllen. So trägt kein einzelner Rechner die gesamte Last, und sein Ausfall legt den Dienst nicht lahm. In Konfigurationsdateien und Diagrammen wird er oft zu LB abgekürzt. Seine Arbeit heißt Lastverteilung oder Load Balancing. Clients sehen eine Adresse, die des Load Balancers. Die Server dahinter bilden einen Pool, den nginx Upstream-Gruppe und HAProxy Backend nennt.

Jeder Load Balancer trifft für jede Anfrage zwei Entscheidungen: Welche Server sind geeignet, sie zu empfangen, und welcher davon bekommt sie? Die zweite Entscheidung ist das Verteilungsverfahren. Round Robin verteilt Anfragen reihum und ist nginx' Verfahren, wenn nichts anderes konfiguriert ist. Least Connections schickt die nächste Anfrage an den Server mit den wenigsten aktiven Verbindungen. Das gleicht die Last aus, wenn manche Anfragen wesentlich länger dauern als andere. Das HAProxy-Handbuch empfiehlt es für lange Sitzungen wie LDAP oder SQL und bezeichnet es als wenig geeignet für kurze wie HTTP. Hashing macht aus einem Schlüssel, etwa der IP-Adresse des Clients oder der angefragten URL, eine Serverauswahl, sodass derselbe Schlüssel weiter auf demselben Rechner landet. nginx bietet auch ein Zufallsverfahren, das zwei Server zufällig auswählen und die Anfrage an den weniger beschäftigten senden kann. Gewichte verschieben die Auswahl zugunsten größerer Rechner: Im eigenen Beispiel des nginx-Leitfadens erhalten ein Server mit `weight=3` und zwei mit dem Standardgewicht 1 jeweils drei, eine und eine von fünf Anfragen.

Load Balancer unterscheiden sich auch darin, wie viel vom Datenverkehr sie lesen. HAProxys `mode tcp`, der Standard, verbindet Client und Server in einer Vollduplexverbindung und nimmt überhaupt keine Untersuchung auf Schicht 7 vor. Das eignet sich für direkt durchgereichtes TLS, SSH oder SMTP. In `mode http` parst es jede Anfrage vor der Serverwahl, lehnt Protokollverstöße ab und kann

nach Pfad oder Header weiterleiten. Ein Load Balancer auf HTTP-Ebene beendet die Verbindung des Clients und öffnet eine eigene zum Server. Die Anwendung sieht ihn deshalb als einzigen Client, sofern er die ursprüngliche Adresse nicht weitergibt, gewöhnlich in einem Header `X-Forwarded-For`. Deshalb sind Load Balancer und Reverse Proxy so oft derselbe Prozess: nginx ist beides, und der eigene Leitfaden beschreibt Lastverteilung als Teil seiner Reverse-Proxy-Implementierung.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Load Balancer?" für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Server werden nach dem tatsächlichen Bedarf dimensioniert und in die Cloud oder aus ihr heraus verlagert — die monatlichen Kosten stehen vorher fest.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 20 Docker-Apps auf Google Cloud und AWS betrieben — 20 bereitgestellte und überwachte Containeranwendungen

20 Docker-Apps auf Google Cloud und AWS betrieben Kontakt aufnehmen →

Warum es wichtig ist

Ein Server hat eine Obergrenze. Wächst ein Dienst darüber hinaus, lässt ein Load Balancer ein Team eine zweite und dritte Kopie hinzufügen, statt einen größeren Rechner zu kaufen. Jede Kopie dahinter bringt zusätzliche Kapazität, ohne dass Clients von der Änderung des Pools wissen müssen. Dieselbe Anordnung macht Wartung zum gewöhnlichen Vorgang: einen Server aus der Verteilung nehmen, aktualisieren, prüfen, wieder aufnehmen und beim nächsten wiederholen, während die übrigen weiter Nutzer bedienen.

Sie macht auch aus einem Serverausfall ein kleineres Ereignis statt eines Dienstausfalls. Antwortet ein Server nicht mehr, wählt der Load Balancer ihn nicht mehr aus, und die übrigen übernehmen seinen Anteil. Das funktioniert nur, wenn sie dafür Reserven haben. Drei Server, die jeweils mit 80 % ihrer Kapazität laufen, können einander nicht ersetzen. Fällt einer aus, trägt jeder der beiden übrigen eine Last von 120 % seiner Kapazität.

Ein Load Balancer verändert, wie die Anwendung geschrieben werden muss. Bei Round Robin oder Least Connections können aufeinanderfolgende Anfragen desselben Clients laut nginx-Leitfaden verschiedene Server erreichen. Alles, was ein Server zwischen Anfragen in seinem eigenen Speicher oder auf seiner eigenen Platte hält, fehlt, sobald die nächste Anfrage auf einem anderen landet:

etwa eine Anmeldesitzung, eine auf Verarbeitung wartende hochgeladene Datei oder ein Cache des Warenkorbs eines Nutzers. Entweder hält die Anwendung diesen Zustand dort, wo jeder Server ihn erreicht, in einer Datenbank oder einem gemeinsamen Cache, oder der Load Balancer bindet jeden Client an einen Server. Das tut nginx' `ip_hash`, außer wenn dieser Server nicht verfügbar ist.

Zuletzt wird der Load Balancer selbst zum einen Rechner, durch den jede Anfrage geht. Ein Pool mit zehn Anwendungsservern hinter einem einzigen Load Balancer hat einen einzelnen Ausfallpunkt; er ist nur umgezogen. Ein solider Aufbau betreibt den Load Balancer als Paar mit einer zwischen beiden verschiebbaren Adresse oder verwendet einen verwalteten Load Balancer, dessen Anbieter den redundanten Betrieb übernimmt.

Wie es in der Praxis aussieht

Hier verteilt ein nginx-Server Anfragen auf drei Anwendungsserver, mit einem vierten in 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` schickt jede neue Anfrage an den Server mit den wenigsten aktiven Verbindungen und löst Gleichstände per gewichtetem Round Robin auf. Der dritte Server ist der größere Rechner; `weight=2` zählt ihn deshalb doppelt. `max_fails=3 fail_timeout=30s` bedeutet, dass drei fehlgeschlagene Versuche innerhalb von 30 Sekunden einen Server für 30 Sekunden aus dem Pool nehmen.

Danach versucht nginx es mit echten Anfragen erneut und nimmt ihn wieder auf, wenn diese gelingen. Ohne diese Parameter gelten die Standardwerte: Ein Fehler innerhalb von 10 Sekunden genügt. Der Server mit `backup` erhält nur Anfragen, solange alle drei primären Server nicht verfügbar sind.

In der `location` entscheidet `proxy_next_upstream`, wann eine Anfrage an einen anderen Server weitergegeben wird: bei einem Verbindungsfehler, einer Zeitüberschreitung oder einer Antwort mit 502 oder 503. `proxy_next_upstream_tries 2` begrenzt die Zahl der Versuche für eine Anfrage; der Standard 0 bedeutet unbegrenzt. Die beiden Zeilen mit `proxy_set_header` erhalten den vom Client angefragten Hostnamen, den nginx sonst durch den Upstream-Namen `app` ersetzen würde, und ergänzen die Clientadresse in `X-Forwarded-For`. Die Anwendung kann damit echte Besucher statt des Load Balancers protokollieren. `proxy_connect_timeout 2s` gibt bei einem Server auf, der die Verbindung nicht innerhalb von zwei Sekunden angenommen hat; standardmäßig würde er 60 Sekunden warten.

Diese Prüfungen sind passiv: nginx erfährt erst durch fehlgeschlagene echte Anfragen, dass ein Server gestört ist. HAProxy kann jeden Server auch nach eigenem Zeitplan prüfen. `option httpchk GET /healthz` in einem Backend, zusammen mit `check` in jeder `server`-Zeile, sendet diese Anfrage standardmäßig alle zwei Sekunden. Drei Fehler in Folge markieren den Server als ausgefallen, zwei Erfolge nehmen ihn wieder auf. Ohne `check` führt HAProxy keine Zustandsprüfungen aus und behandelt den Server als immer verfügbar.

Bevor nginx ein Load Balancer ist, ist es ein gewöhnlicher Webserver. Es ist einer der beiden hinter den vierzig auf Apache und Nginx verwalteten Websites zwischen 2003 und 2014. Diese liefen auf Hosting-Servern mit Ubuntu Linux und wurden abgestimmt und gepflegt, um unter echtem Datenverkehr zuverlässig zu bleiben. Die Zeile mit `backup` hat einen Verwandten eine Ebene darüber, wo die Server ganze Anbieter sind. E-Mail mit Failover über drei Anbieter hält SendGrid als primären Anbieter vor, dahinter SMTP2GO und Azure Communication Services, wechselt automatisch zwischen ihnen und machte aus dem schlechten Nachmittag eines Anbieters einen beeinträchtigten Pfad statt eines Ausfalls.

Wo es schiefgeht

Ein Server, der jede Anfrage mit 500 beantwortet, bleibt im Pool.
Standardmäßig zählt nginx nur Kommunikationsfehler mit dem Server, Zeitüberschreitungen und fehlerhafte Antwortheader als Fehlschläge. Ein Server,

dessen Anwendung die Datenbank verloren hat und überall 500 zurückgibt, antwortet schnell und höflich. Der Load Balancer schickt ihm deshalb weiterhin einen Anteil des Datenverkehrs. Die Antworten 500, 502, 503, 504 und 429 werden nur als Fehlschläge des Servers gezählt, wenn sie in `proxy_next_upstream` aufgeführt sind; 403 und 404 zählen nie als Fehlschläge. Führen Sie die Codes auf, die „dieser Server ist defekt" bedeuten. Geben Sie den Servern eine aktive Prüfung gegen einen Endpunkt, der die Komponenten nutzt, die die Anwendung zum Ausliefern einer Seite benötigt.

Eine langsame Anfrage läuft nacheinander auf jedem Server. `timeout` gehört zur Standardeinstellung von `proxy_next_upstream`, und `proxy_read_timeout` beträgt standardmäßig 60 Sekunden. Ein Bericht, dessen Erstellung 70 Sekunden dauert, läuft auf dem ersten Server in eine Zeitüberschreitung, wird an den zweiten weitergegeben, überschreitet dort die Zeit und geht an den dritten. Der Client wartet damit drei Minuten, während jeder Server dieselbe Arbeit erledigt. Eine Zeitüberschreitung zählt immer als fehlgeschlagener Versuch. Beim Standardwert 1 für `max_fails` wird deshalb außerdem jeder kontaktierte Server für 10 Sekunden als nicht verfügbar markiert. Begrenzen Sie die Weitergaben mit `proxy_next_upstream_tries` und `proxy_next_upstream_timeout`, geben Sie dem langsamen Endpunkt eine eigene `location` mit längerer Frist oder verschieben Sie die Arbeit aus dem Anfragepfad in eine Hintergrundaufgabe.

Eine Zahlung wird zweimal ausgeführt. Standardmäßig weigert sich nginx, eine bereits bei einem Server eingetroffene POST-, LOCK- oder PATCH-Anfrage an einen anderen weiterzusenden, weil der erste Server schon gehandelt haben könnte. Wer `non_idempotent` zu `proxy_next_upstream` hinzufügt, um Zeitüberschreitungen verschwinden zu lassen, entfernt diesen Schutz. Eine vom ersten Server bereits abgeschlossene, aber zu langsam beantwortete Zahlung wird beim zweiten noch einmal abgeschlossen. Lassen Sie die Option aus, sofern nicht jeder Endpunkt dahinter idempotent ist, etwa weil er einen Idempotenzschlüssel entgegennimmt und eine bereits gesehene Anfrage nicht wiederholt.

Jeder Besucher scheint vom Load Balancer zu kommen. Ohne die Header des Beispiels protokolliert die Anwendung dessen Adresse für jede Anfrage. Jede Grenze je IP-Adresse gilt dann für alle Nutzer zugleich. `X-Forwarded-For` hinzuzufügen bringt eine eigene Falle mit. `$proxy_add_x_forwarded_for` hängt die Clientadresse an alles an, was der Client bereits in `X-Forwarded-For` gesendet hat. Die Einträge ganz links sind deshalb das, was der Client selbst hineingeschrieben hat. Die Anwendung sollte nur dem von ihrem eigenen Load Balancer ergänzten Eintrag vertrauen und den Header nur von dessen Adresse akzeptieren.

Ein Büro landet auf einem Server. nginx' `ip_hash` bildet den Hash aus den ersten drei Oktetten einer IPv4-Adresse. Alle Clients mit denselben drei Oktetten landen deshalb auf demselben Server, und ein ganzes Unternehmen hinter einem NAT-Gateway kommt als einzelner Client an. Änderungen am Pool sorgen für die entgegengesetzte Überraschung. Bei einfachem `hash` kann das Hinzufügen oder Entfernen eines Servers die meisten Schlüssel auf einen anderen Server verschieben. Jeder dadurch gebundene Nutzer verliert dann das, was sein alter Server vorgehalten hat. Die nginx-Dokumentation empfiehlt, einen Server als `down` zu markieren, statt ihn aus einer `ip_hash` -Gruppe zu löschen. Sie bietet außerdem `hash ... consistent` , das bei Änderungen am Pool nur wenige Schlüssel verschiebt. Noch besser ist es, Sitzungszustand außerhalb der Server zu halten; dann verliert die Bindung ihre Bedeutung.

Weiterführende Lektüre

- nginx als HTTP-Load-Balancer verwenden — die drei grundlegenden Verfahren, Gewichte, Sitzungsbindung und passive Zustandsprüfungen, jeweils mit kurzer Konfiguration.
- Modul ngx_http_upstream_module — jeder Parameter einer `server` -Zeile, einschließlich `max_fails` , `backup` und der Ausnahme für einen einzelnen Server, sowie die Verfahren `hash` , `ip_hash` , `least_conn` und `random` .
- Modul ngx_http_proxy_module — `proxy_next_upstream` , was als fehlgeschlagener Versuch zählt, die Zeitgrenzen und die Header, die nginx auf dem Weg zum Server setzt.
- HAProxy 3.4: Konfigurationshandbuch — `balance` und seine Algorithmen, TCP- und HTTP-Modus sowie aktive Zustandsprüfungen mit `check` , `inter` , `rise` und `fall` .

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Load Balancer?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/load-balancer/