Engineer

ロードバランサー

⚖ リクエストを複数の別のサーバーへ分配するサーバー

どのようなものか

ロードバランサーは、クライアントのリクエストを受け、同じ仕事をする複数のサーバーに分配するサーバーです。一台が全負荷を背負わず、一台を失ってもサービス全体が止まらないようにします。設定ファイルや図ではLBと略すことが多く、その仕事を負荷分散と呼びます。クライアントから見えるアドレスは、ロードバランサーの一つだけです。背後のサーバー群をプールと呼び、nginxはupstreamグループ、HAProxyはbackendと呼びます。

リクエストごとに、二つを決めます。受け取れる状態のサーバーはどれか、その中のどれへ送るかです。後者が負荷分散の方式です。ラウンドロビンは、サーバーへ順番に渡し、ほかに設定しなければ、nginxはこれを使います。最少接続数方式は、使用中の接続の最も少ないサーバーへ次を送り、所要時間が大きく違うリクエスト間で負荷を均します。HAProxyのマニュアルは、LDAPやSQLの長いセッションに勧め、HTTPの短いものには不向きとします。ハッシュ方式は、クライアントのIPアドレスや要求したURLなどのキーからサーバーを選び、同じキーを同じマシンへ送り続けます。nginxには、無作為に二台を選び、負荷の低い方へ送れるランダム方式もあります。重みは、大きいマシンを選びやすくします。nginxのガイドの例では、`weight=3`の一台と、既定の重み1の二台に、五件のうち三件、一件、一件を送ります。

通信をどこまで読むかも異なります。HAProxyの既定の`mode tcp`は、クライアントとサーバーを全二重の接続で結び、レイヤー7の検査を一切しません。TLSをそのまま通す場合や、SSH、SMTPに向きます。`mode http`では、選ぶ前に各リクエストを解析し、プロトコル違反を拒否し、パスやヘッダーで送り先を変えられます。HTTP層では、クライアントとの接続を終端し、サーバーへ自身の接続を開きます。そのため、通常は`X-Forwarded-For`ヘッダーで元のアドレスを渡さない限り、アプリケーションが見るクライアントは、ロードバランサーだけです。ロードバランサーとリバースプロキシーが、同じプロセスであることが多いのも、このためです。nginxは両方で、ガイドも、負荷分散をリバースプロキシーの実装の一部と説明します。

概念を判断に生かす

「ロードバランサーとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 実際の使用量に合わせてサーバーを構成し、月額費用を事前に把握したうえで、クラウドへの移行もクラウドからの移転も進めます。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Google Cloud・AWSでDockerアプリ20個を運用 — 20 デプロイし監視したコンテナアプリケーション

Google Cloud・AWSでDockerアプリ20個を運用 お問い合わせ →

なぜ重要なのか

一台のサーバーには上限があります。サービスがそれを超えたら、大きいマシンを買う代わりに、二つ目、三つ目のコピーを加えられるのが、ロードバランサーです。クライアントにプールの変更を知らせず、背後のコピーごとに容量を増やせます。同じ構成で、保守も日常の作業になります。一台を振り分けから外し、更新、確認、復帰を行い、次へ進みます。その間も、残るサーバーが利用者に応答します。

サーバー一台の故障も、サービス停止ではなく、小さい出来事にできます。応答しなくなったサーバーを選ばず、残るサーバーがその分を受け持ちます。ただし、その余力がある場合だけです。三台がそれぞれ容量の80%で動いていれば、互いを補えません。一台を失うと、残る二台それぞれに、一台分の120%の仕事がかかります。

アプリケーションの書き方も変わります。nginxのガイドが述べるとおり、ラウンドロビンや最少接続数方式では、同じクライアントの連続するリクエストが、別々のサーバーへ届くことがあります。ログインセッション、処理待ちのアップロードファイル、利用者の買い物かごのキャッシュなど、サーバーが自身のメモリーやディスクに保存したものは、次が別のサーバーに着くと存在しません。すべてのサーバーから使えるデータベースや共有キャッシュに状態を置くか、各クライアントを一台に固定します。nginxの `ip_hash` は、そのサーバーが利用不能な場合を除き、後者を行います。

最後に、ロードバランサー自身が、すべてのリクエストの通る一台になります。アプリケーションサーバーが十台でも、ロードバランサーが一台なら、単一障害点が移っただけです。本格的な構成では、ロードバランサーを一組にし、二台の間で移せるアドレスを使うか、事業者が冗長構成を運用するマネージドのロードバランサーを使います。

実際の活用例

次は、nginxが三台のアプリケーションサーバーへ分配し、四台目を予備にする例です。

```
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` は、使用中の接続が最も少ないサーバーへ新しいリクエストを送り、同数なら、重み付きラウンドロビンで選びます。三台目は大きいため、`weight=2` で二倍と数えます。 `max_fails=3 fail_timeout=30s` は、30秒以内に三回失敗したサーバーを30秒間外し、その後、実際のリクエストで再試行して、成功すれば戻す指定です。指定しなければ既定値が働き、10秒以内の一回の失敗で足ります。 `backup` のサーバーが受けるのは、主系の三台すべてが使えない間だけです。

`location` 内の `proxy_next_upstream` は、接続エラー、タイムアウト、502か503の応答で、別のサーバーへ渡すことを決めます。 `proxy_next_upstream_tries 2` は、一リクエストについての試行数を制限します。既定値0は、制限なしです。二行の `proxy_set_header` は、要求されたホスト名を、通常なら置き換わるupstream名 `app` の代わりに保ち、クライアントのアドレスを `X-Forwarded-For` へ加えます。これで、ロードバランサーでなく、実際の訪問者を記録できます。
`proxy_connect_timeout 2s` は、二秒で接続を受け付けないサーバーを諦めます。既定なら60秒待ちます。

これらは受動的なチェックです。nginxが異常を知るのは、実際のリクエストが失敗したときだけです。HAProxyは、独自の間隔でサーバーを探ることもできます。backendの `option httpchk GET /healthz` に、 `check` を付けた各 `server` 行を組み合わせると、既定では二秒ごとに送ります。三回続けて失敗すれば停止扱い、二回成功すれば復帰です。 `check` がなければ、ヘルスチェックを一切行わず、常に利用可能とみなします。

nginxは、ロードバランサーである前に、普通のウェブサーバーです。2003年から2014年に、ApacheとNginxで管理した四十のウェブサイトを支えた二つのうちの一つです。Ubuntu Linuxのホスティングサーバーで動かし、実際の通信下でも安定するよう調整し、保守しました。 `backup` 行に似た考えが、一段上の、サーバーが事業者全体になる場所にもあります。三事業者間で切り替えるメールは、SendGridを主系とし、SMTP2GOとAzure Communication Servicesを切り替え先として用意し、自動で切り替えます。事業者側で不調が起きたある午後も、全面停止でなく、一部の経路の劣化にとどめました。

問題が起きるところ

すべてのリクエストに500を返すサーバーが、プールに残ります。 nginxが既定で失敗と数えるのは、通信エラー、タイムアウト、不正な応答ヘッダーだけです。データベースに接続できず、何にでも500を返すアプリケーションも、プロトコルに沿った応答を素早く返すため、通信の一部が届き続けます。500、502、503、504、429は、 `proxy_next_upstream` に列挙したときだけ失敗と数え、403と404は決して数えません。「このサーバーは壊れている」を意味するコードを列挙し、ページを返すために必要なものを実際に試すエンドポイントで、能動チェックも行います。

一つの遅いリクエストが、順番に全サーバーで動きます。 `timeout` は、既定の `proxy_next_upstream` に含まれ、 `proxy_read_timeout` の既定値は60秒です。生成に70秒かかる報告書は、一台目でタイムアウトし、二台目へ渡ってまた失敗し、三台目へ移ります。全サーバーが同じ仕事をする間、クライアントは三分待ちます。タイムアウトは必ず失敗した試行と数えるため、 `max_fails` の既定値1では、試行先になった各サーバーも10秒間、利用不能とされます。 `proxy_next_upstream_tries` と `proxy_next_upstream_timeout` で引き渡しを制限します。遅いエンドポイントは、待ち時間の長い専用の `location` へ分けるか、処理をリクエスト経路から外し、バックグラウンドジョブに移します。

支払いが二回行われます。 nginxは、POST、LOCK、PATCHが一度サーバーに届いたら、既定では別のサーバーへ再送しません。最初のサーバーが、すでに処理したかもしれないためです。タイムアウトを解消するために `non_idempotent` を

`proxy_next_upstream` へ足すと、この保護を外します。最初が支払いを完了しても、応答が遅ければ、次でも完了します。背後のすべてのエンドポイントが、例えば冪等キーを受け、見たことのあるリクエストを繰り返さないなど、冪等でない限り、そのオプションは無効のままにします。

すべての訪問者が、ロードバランサーから来たように見えます。 例のヘッダーがなければ、アプリケーションは、各リクエストにロードバランサーのアドレスを記録し、IP単位の制限が、全利用者にまとめてかかります。 `X-Forwarded-For` を足す際にも罠があります。 `$proxy_add_x_forwarded_for` は、クライアントがすでに送った `X-Forwarded-For` に、クライアントの接続元アドレスを付け足します。つまり、左側の値はクライアントが自由に書いたものです。自身のロードバランサーが足した値だけを信用します。ヘッダーを受け入れるのも、自身のロードバランサーのアドレスから届く場合だけです。

一つのオフィスが、一台に集まります。 nginxの `ip_hash` は、IPv4アドレスの最初の三オクテットでハッシュを計算するため、そこを共有するクライアントは同じサーバーへ行きます。一つのNATゲートウェイの背後の企業全体が、一クライアントとして到着します。プールの変更は、逆の驚きも生みます。単純な `hash` では、追加や削除で大半のキーが別のサーバーへ移り、固定されていた利用者は、元のサーバーにあるものを失います。nginxの文書は、サーバーを `down` とし、 `ip_hash` 群から削除しないよう勧め、変更時に少数のキーだけ動かす `hash ... consistent` も提供します。さらによいのは、セッション状態をサーバーの外に置くことです。固定の必要がなくなります。

参考資料

- nginxをHTTPロードバランサーとして使う — 基本の三方式、重み、セッション維持、受動ヘルスチェックを、それぞれ短い設定で説明します。
- ngx_http_upstream_moduleモジュール — `server` 行のすべての引数です。 `max_fails` 、 `backup` 、一台だけの場合の例外、 `hash` 、 `ip_hash` 、 `least_conn` 、 `random` の各方式を含みます。
- ngx_http_proxy_moduleモジュール — `proxy_next_upstream` 、失敗した試行の定義、待ち時間、サーバーへ送る際に設定するヘッダーです。
- HAProxy 3.4設定マニュアル — `balance` と各方式、TCPとHTTPのモード、 `check` 、 `inter` 、 `rise` 、 `fall` による能動ヘルスチェックです。



用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「ロードバランサーとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/load-balancer/