Engineer

Equilibrador de carga

⚖ Servidor que reparte peticiones entre varios servidores

Qué es

Un equilibrador de carga es un servidor que recibe peticiones de clientes y las distribuye entre un grupo de otros servidores que hacen el mismo trabajo, para que ninguna máquina soporte toda la carga y la pérdida de una no haga caer el servicio. Los archivos de configuración y los diagramas suelen abreviarlo como LB, y su trabajo se llama equilibrado de carga. Los clientes ven una dirección: la del equilibrador. Los servidores que hay detrás forman un grupo, que nginx denomina grupo upstream y HAProxy, backend.

Todo equilibrador toma dos decisiones por petición: qué servidores están en condiciones de recibirla y cuál de ellos lo hará. La segunda es el método de equilibrado. Round robin entrega las peticiones a los servidores por turnos y es lo que hace nginx cuando no se configura otra cosa. El método de menos conexiones envía la siguiente petición al servidor con menos conexiones activas, lo que iguala la carga cuando algunas peticiones duran mucho más que otras; el manual de HAProxy lo recomienda para sesiones largas, como LDAP o SQL, y lo considera poco adecuado para las cortas, como HTTP. El cálculo de un hash convierte una clave, como la dirección IP del cliente o la URL solicitada, en una elección de servidor, de modo que la misma clave sigue llegando a la misma máquina. nginx también ofrece un método aleatorio que puede elegir dos servidores al azar y enviar la petición al menos ocupado. Los pesos inclinan la elección hacia las máquinas más grandes: en el ejemplo de la propia guía de nginx, un servidor con `weight=3` y dos con el peso predeterminado de 1 reciben tres, una y una de cada cinco peticiones.

Los equilibradores también difieren en cuánto examinan el contenido del tráfico. El `mode tcp` de HAProxy, su valor predeterminado, une cliente y servidor en una conexión de dúplex completo y no examina en absoluto la capa 7, lo que sirve para TLS sin terminarlo en el equilibrador, SSH o SMTP. En `mode http`, analiza cada petición antes de elegir servidor, rechaza las que incumplen el protocolo y

puede enrutar por ruta o cabecera. Un equilibrador que trabaja en el nivel HTTP termina la conexión del cliente y abre la suya con el servidor, por lo que la aplicación ve al equilibrador como su único cliente, salvo que este transmita la dirección original, normalmente en una cabecera `X-Forwarded-For`. Esa es también la razón por la que un equilibrador de carga y un proxy inverso son tan a menudo el mismo proceso: nginx es ambas cosas, y su propia guía describe el equilibrado de carga como parte de su implementación de proxy inverso.

Utilice el concepto en una decisión

Aplique «¿Qué es un equilibrador de carga?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Servidores dimensionados para el uso real, trasladados a la nube o fuera de ella, con la factura mensual calculada de antemano.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 20 aplicaciones Docker en Google Cloud y AWS — 20 aplicaciones en contenedores desplegadas y monitorizadas

20 aplicaciones Docker en Google Cloud y AWS Póngase en contacto →

Por qué importa

Un servidor tiene un límite. Cuando un servicio lo supera, un equilibrador permite al equipo añadir una segunda y una tercera copia en vez de comprar una máquina más grande, y cada copia que hay detrás añade capacidad sin que los clientes sepan que el grupo ha cambiado. La misma organización convierte el mantenimiento en una operación corriente: retirar un servidor de la rotación, actualizarlo, comprobarlo, devolverlo y repetir con el siguiente, mientras los demás siguen atendiendo a los usuarios.

También convierte el fallo de un servidor en un incidente menor, en lugar de una interrupción del servicio. Cuando un servidor deja de responder, el equilibrador deja de elegirlo y los restantes absorben su parte. Eso solo funciona si tienen capacidad libre para hacerlo. Tres servidores al 80 % de su capacidad no pueden cubrirse entre sí: perder uno impone a cada uno de los otros dos un trabajo equivalente al 120 % de la capacidad de un servidor.

Un equilibrador cambia cómo debe escribirse la aplicación. Con round robin o menos conexiones, señala la guía de nginx, las peticiones consecutivas de un cliente pueden llegar a servidores diferentes. Cualquier cosa que un servidor guarde en su propia memoria o disco entre peticiones, como una sesión de usuario autenticado, un archivo subido pendiente de procesamiento o una caché de la cesta del usuario, faltará cuando la siguiente petición llegue a otro servidor.

O bien la aplicación guarda ese estado donde todos los servidores puedan acceder a él, en una base de datos o una caché compartida, o bien el equilibrador fija cada cliente a un servidor, que es lo que hace `ip_hash` de nginx salvo cuando ese servidor no está disponible.

Por último, el propio equilibrador se convierte en la única máquina por la que pasa cada petición. Un grupo de diez servidores de aplicaciones detrás de un solo equilibrador tiene un único punto de fallo; solo lo ha trasladado. Una instalación sólida utiliza una pareja de equilibradores con una dirección que puede pasar de uno a otro, o usa uno gestionado cuyo proveedor se encarga de la redundancia.

Cómo se aplica en la práctica

Aquí un servidor nginx reparte las peticiones entre tres servidores de aplicaciones, con un cuarto en reserva:

```
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` envía cada nueva petición al servidor con menos conexiones activas y resuelve los empates mediante round robin ponderado. El tercer servidor es la máquina más grande, por lo que `weight=2` lo cuenta doble. `max_fails=3 fail_timeout=30s` indica que tres intentos fallidos en 30 segundos retiran un servidor del grupo durante 30 segundos; después, nginx vuelve a probarlo con

peticiones reales y lo reincorpora si tienen éxito. Sin esos parámetros se aplican los valores predeterminados: basta un fallo en 10 segundos. El servidor `backup` solo recibe peticiones mientras los tres principales están fuera de servicio.

En `location` , `proxy_next_upstream` decide cuándo se pasa una petición a otro servidor: ante un error de conexión, un tiempo de espera agotado o una respuesta 502 o 503. `proxy_next_upstream_tries 2` limita los intentos para una petición; el valor predeterminado, 0, no impone límite. Las dos líneas `proxy_set_header` conservan el nombre de host que pidió el cliente, que nginx sustituiría de otro modo por el nombre upstream `app` , y añaden la dirección del cliente a `X-Forwarded-For` para que la aplicación registre a los visitantes reales en vez de al equilibrador. `proxy_connect_timeout 2s` abandona un servidor que no haya aceptado la conexión en dos segundos, cuando el valor predeterminado esperaría 60 segundos.

Estas comprobaciones son pasivas: nginx solo descubre que un servidor tiene problemas cuando fallan peticiones reales dirigidas a él. HAProxy también puede sondear cada servidor según su propia programación. `option httpchk GET /healthz` en un backend, con `check` en cada línea `server` , envía esa petición cada dos segundos de forma predeterminada; tres fallos seguidos marcan el servidor como no disponible y dos éxitos lo reincorporan. Sin `check` , HAProxy no realiza comprobaciones de salud y considera el servidor siempre disponible.

Antes de ser un equilibrador, nginx es un servidor web corriente: es uno de los dos utilizados para los cuarenta sitios web administrados con Apache y Nginx entre 2003 y 2014, que funcionaban en servidores de alojamiento con Ubuntu Linux y se ajustaban y mantenían para conservar la fiabilidad con tráfico real. La línea `backup` tiene un equivalente un nivel más arriba, donde los servidores son proveedores completos. El correo con conmutación ante fallos entre tres proveedores mantiene SendGrid como principal, con SMTP2GO y Azure Communication Services detrás, conmuta entre ellos automáticamente y convirtió una mala tarde de un proveedor en una ruta degradada, en lugar de una interrupción.

Dónde surgen los problemas

Un servidor que responde a cada petición con un 500 permanece en el grupo. De forma predeterminada, nginx solo cuenta como fallos los errores al comunicarse con el servidor, los tiempos de espera agotados y las cabeceras de respuesta mal formadas. Un servidor cuya aplicación ha perdido su base de datos

y devuelve 500 a todo responde deprisa y respetando el protocolo, por lo que el equilibrador sigue enviándole parte del tráfico. Las respuestas 500, 502, 503, 504 y 429 solo se computan como fallos del servidor cuando se enumeran en `proxy_next_upstream` ; 403 y 404 nunca se computan como fallos. Enumere los códigos que significan «este servidor está averiado» y dote a los servidores de una comprobación activa contra un endpoint que ponga a prueba los componentes que necesita la aplicación para servir una página.

Una petición lenta se ejecuta en todos los servidores por turnos. `timeout` forma parte del valor predeterminado de `proxy_next_upstream` , y `proxy_read_timeout` tiene un valor predeterminado de 60 segundos. Un informe que tarda 70 segundos en generarse agota el tiempo de espera en el primer servidor, pasa al segundo, también lo agota y continúa al tercero, de modo que el cliente espera tres minutos mientras todos los servidores hacen el mismo trabajo. Un tiempo de espera agotado siempre cuenta como intento fallido, así que, con el valor predeterminado de 1 para `max_fails` , cada servidor al que llegó la petición también queda marcado como no disponible durante 10 segundos. Limite los traspasos con `proxy_next_upstream_tries` y `proxy_next_upstream_timeout` , dé al endpoint lento su propio `location` con más tiempo de espera o saque el trabajo de la ruta de la petición y páselo a una tarea en segundo plano.

Un pago se procesa dos veces. De forma predeterminada, nginx se niega a reenviar una petición POST, LOCK o PATCH a otro servidor una vez que ha llegado a uno, porque el primero puede haber actuado ya. Añadir `non_idempotent` a `proxy_next_upstream` para hacer desaparecer los tiempos de espera agotados elimina esa protección: un pago que el primer servidor completó, pero al que respondió demasiado despacio, vuelve a completarlo el segundo. Deje la opción desactivada salvo que cada endpoint que haya detrás sea idempotente, por ejemplo, porque acepta una clave de idempotencia y se niega a repetir una petición que ya haya visto.

Todos los visitantes parecen venir del equilibrador. Sin las cabeceras del ejemplo, la aplicación registra la dirección del equilibrador en cada petición, y cualquier límite por dirección IP pasa a aplicarse a todos los usuarios a la vez. Añadir `X-Forwarded-For` tiene su propia trampa. `$proxy_add_x_forwarded_for` añade la dirección del cliente a lo que este ya hubiera enviado en `X-Forwarded-For` , de modo que las entradas de la izquierda son lo que el cliente eligió escribir. La aplicación solo debe confiar en la entrada añadida por su propio equilibrador y aceptar la cabecera únicamente desde la dirección de este.

Una oficina acaba en un único servidor. `ip_hash` de nginx calcula el hash de los tres primeros octetos de una dirección IPv4, por lo que todos los clientes que los comparten llegan al mismo servidor, y toda una empresa detrás de una pasarela NAT llega como un solo cliente. Cambiar el grupo provoca la sorpresa opuesta. Con un `hash` simple, añadir o retirar un servidor puede trasladar la mayoría de las claves a otro servidor, y cada usuario fijado por esa clave pierde lo que guardaba el anterior. La documentación de nginx aconseja marcar un servidor como `down` en vez de eliminarlo de un grupo `ip_hash` y ofrece `hash ... consistent`, que solo mueve unas pocas claves cuando cambia el grupo. Mejor aún: mantenga el estado de las sesiones fuera de los servidores y la afinidad dejará de importar.

Lecturas adicionales

- Usar nginx como equilibrador de carga HTTP — los tres métodos básicos, los pesos, la persistencia de sesión y las comprobaciones pasivas de salud, cada uno con una configuración breve.
- Módulo ngx_http_upstream_module — todos los parámetros de una línea `server`, incluidos `max_fails`, `backup` y la excepción de un único servidor, y los métodos `hash`, `ip_hash`, `least_conn` y `random`.
- Módulo ngx_http_proxy_module — `proxy_next_upstream`, qué cuenta como intento fallido, los tiempos de espera y las cabeceras que nginx establece de camino al servidor.
- Manual de configuración de HAProxy 3.4 — `balance` y sus algoritmos, los modos TCP y HTTP y las comprobaciones activas de salud con `check`, `inter`, `rise` y `fall`.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un equilibrador de carga?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/load-balancer/