Engineer

limitación de solicitudes

Máximo de peticiones de un cliente por intervalo

Qué es

La limitación de solicitudes establece un máximo de peticiones que un cliente puede enviar a un servicio durante un intervalo determinado, aplicado por el servicio o por algo situado delante de él. Todo límite responde a tres preguntas: quién cuenta como cliente, cómo se cuentan las peticiones y qué ocurre con la petición que lo supera. El código de estado HTTP creado para la última de ellas, 429 Too Many Requests, se definió en RFC 6585 en abril de 2012, y ese RFC deja deliberadamente abiertas las dos primeras. Un servidor puede contar por recurso, en todo el servidor o en un grupo de servidores, y puede reconocer al cliente por sus credenciales de autenticación o por una cookie con estado.

En la práctica, la clave suele elegirse entre unas pocas opciones: la dirección IP del cliente, la cuenta que ha iniciado sesión, una clave de API o una de ellas combinada con la ruta, para que un formulario de acceso y una página de producto tengan límites distintos. Hay tres tipos de respuesta a una petición que supera el límite. El servidor puede rechazarla con un 429, preferiblemente con una cabecera `Retry-After` que indique cuánto esperar, como número de segundos o como fecha HTTP. Puede retener la petición y procesarla un momento después, lo que suaviza una ráfaga en lugar de rechazarla. O, cuando el propio volumen constituye el ataque, puede cerrar la conexión sin responder. RFC 6585 propone precisamente eso, porque escribir un 429 cortés por cada petición de una avalancha también consume recursos.

La forma de contar determina cómo se comporta un límite en los casos límite. Una ventana fija cuenta peticiones por minuto u hora del reloj y vuelve a empezar desde cero al cambiar el intervalo. Es barata y fácil de explicar, y permite a un cliente gastar toda su cuota en el último segundo de una ventana y otra vez en el primero de la siguiente: con un límite de 100 por minuto, son 200 peticiones en dos segundos. Una ventana deslizante siempre abarca el mismo periodo hacia atrás desde el momento actual, por lo que no hay un cambio de intervalo que

aprovechar; las tablas de seguimiento, o stick tables, de HAProxy mantienen contadores de este tipo, y un ejemplo de su manual cuenta la tasa de conexiones durante una ventana deslizante de 30 segundos. El algoritmo de cubo con fuga, que utiliza nginx, deja pasar peticiones a una tasa media fija y mantiene una pequeña cola para las que llegan más rápido, rechazando las que la desbordarían. Un cubo de tokens funciona a la inversa: los tokens se acumulan a un ritmo constante hasta un máximo, cada petición consume uno y un cliente que lleva un tiempo inactivo puede gastar lo acumulado en una ráfaga.

Utilice el concepto en una decisión

Aplique «¿Qué es la limitación de solicitudes?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cuentas, contraseñas y permisos organizados para que cada persona tenga exactamente el acceso que necesita y lo pierda el mismo día que deja la empresa.

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. 1.000 reglas RBAC para servidores Linux — 1,000 reglas de acceso escritas para cerrar una brecha de seguridad

1.000 reglas RBAC para servidores Linux Póngase en contacto →

Por qué importa

Un punto de acceso público no decide quién lo llama ni con qué frecuencia. Un extractor que copia un catálogo, un script atrapado en un bucle de reintentos, un atacante que prueba contraseñas filtradas en el formulario de acceso y un socio cuyo trabajo nocturno se escribió con prisas parecen lo mismo para el servidor: más peticiones de las que puede atender según su dimensionamiento, procedentes de unas pocas fuentes. Sin un límite, quien más llama decide cuánta capacidad queda para los demás. Con uno, un cliente solo puede usar su parte y el resto del servicio continúa a velocidad normal mientras ese cliente espera.

Algunas peticiones son mucho más caras que otras. Una página de producto puede servirse desde una caché en un milisegundo, mientras que una búsqueda, una exportación o cualquier operación que llame a una API externa de pago consume tiempo de CPU o supone un gasto económico en cada llamada. Quien abusa apunta a esas rutas precisamente porque no le cuestan nada. Un único límite global lo bastante bajo para proteger la exportación ahogaría la navegación normal, por lo que una limitación útil se establece por ruta, eligiendo cada límite según lo que cuesta esa ruta y cómo se utiliza legítimamente.

Un límite también forma parte del contrato de una API. Los clientes que conocen las reglas pueden regular su ritmo sin alcanzarlo nunca, lo que resulta más barato para ambos lados que descubrirlo a base de rechazos. HTTP no tiene una cabecera estándar para ello. Muchas API envían `X-RateLimit-Limit`, `X-RateLimit-Remaining` y `X-RateLimit-Reset`, pero un mismo nombre contiene valores distintos según la implementación, como recoge un borrador del IETF que pretende estandarizar la idea. Ese borrador define dos campos, `RateLimit-Policy` para la cuota y su ventana, y `RateLimit` para lo que queda; su undécima versión está fechada el 23 de mayo de 2026 y, como todo Internet-Draft, es un trabajo en curso y no un estándar.

Cómo se aplica en la práctica

En nginx, un límite necesita dos directivas: una en el bloque `http` que configura el contador y otra donde se aplica:

```
limit_req_zone $binary_remote_addr zone=perip:10m rate=1r/s;
limit_req_status 429;

server {
    location /search/ {
        limit_req zone=perip burst=5 nodelay;
        proxy_pass http://app;
    }
}
```

La clave es la dirección del cliente en forma binaria, de 4 bytes para una dirección IPv4 y de 16 para una IPv6, y los contadores residen en una zona de memoria compartida de 10 megabytes llamada `perip`. La documentación cifra el estado de una clave en 128 bytes en una plataforma de 64 bits, por lo que un megabyte contiene unas 8 mil direcciones y esta zona unas 80 mil; cuando se llena, nginx expulsa la utilizada menos recientemente. `rate=1r/s` permite una petición por segundo de media. `burst=5` permite que hasta cinco peticiones por encima de esa tasa esperen su turno en lugar de ser rechazadas, y `nodelay` las atiende de inmediato en lugar de espaciarlas a la tasa configurada. Todo lo que exceda la ráfaga se rechaza. nginx respondería a ese rechazo con 503 de forma predeterminada, y por eso `limit_req_status` fija 429. Las tasas inferiores a una por segundo se escriben por minuto, de modo que `30r/m` es una petición cada dos segundos. Antes de activar un límite nuevo, `limit_req_dry_run on` sigue contando como siempre y marca los rechazos que se habrían producido como `REJECTED_DRY_RUN` en la variable `$limit_req_status`, sin rechazar nada.

En el otro extremo, un cliente que alcanza el límite debería leer la respuesta en lugar de reintentar de inmediato. Un 429 de una API que siga el borrador del IETF podría tener este aspecto:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/problem+json
Retry-After: 20
RateLimit-Policy: "default";q=100;w=60
RateLimit: "default";r=0;t=20
```

La política permite 100 peticiones por ventana de 60 segundos, no queda ninguna y la cuota vuelve en 20 segundos. Cuando aparecen ambos, el borrador indica que prevalece `Retry-After`. Un cliente que se comporte correctamente espera al menos ese tiempo, añade un pequeño retraso aleatorio y, si no hay ningún `Retry-After`, recurre a la espera exponencial entre reintentos en vez de a un intervalo fijo.

Los 22 limitadores construidos para rechazar tráfico ante un fallo para un directorio público de empresas y profesionales, desde 2025, siguen la regla por ruta: cada uno se construyó para la ruta que protege, porque un intento de acceso, una búsqueda y una exportación masiva se convierten en abuso a tasas muy distintas. Su estado reside en Redis, por lo que un límite se mantiene en todas las instancias en lugar de conservarse por proceso. La decisión determinante fue qué ocurre cuando Redis no está disponible: los limitadores rechazan el tráfico en vez de dejarlo pasar. La página expone el coste con franqueza: un problema de Redis pasa a ser visible para los usuarios. Explica también por qué se aceptó: la alternativa era que un problema de Redis se convirtiera en uno de facturación. A su alrededor se sitúan Cloudflare Turnstile, un bloqueo del origen y 351 líneas de middleware de idempotencia para que una escritura reintentada no se aplique dos veces.

Dónde surgen los problemas

Cada servidor lleva su propia cuenta. Un límite conservado en la memoria del proceso se multiplica con cada copia de la aplicación: cuatro instancias detrás de un equilibrador de carga permiten a un cliente pasar a cuatro veces la tasa prevista, y cada reinicio pone los contadores a cero. La zona de nginx es memoria compartida de un único equipo, y mantener las zonas sincronizadas entre servidores forma parte de su suscripción comercial; el protocolo de pares de HAProxy copia las entradas de las tablas de seguimiento entre instancias, pero,

como señala su manual, sin sumarlas. Mantenga los contadores en un almacén que consulte cada instancia, como hace el directorio anterior con Redis, o aplique el límite en el único punto por el que pasa todo el tráfico.

Todo el mundo llega desde la misma dirección. Detrás de un proxy inverso, una CDN o un equilibrador de carga, la aplicación ve una misma dirección de cliente para todos los usuarios salvo que se transmita la original, y entonces un límite por IP restringe a todos a la vez. Una oficina o red móvil completa detrás de NAT tiene el mismo efecto de forma más leve. Use como clave la dirección real del cliente tomada de una cabecera que solo su propio proxy pueda establecer y, para el tráfico autenticado, prefiera la cuenta o la clave de API a la dirección.

El limitador comunica un fallo del servidor. La respuesta predeterminada de nginx a una petición rechazada es 503 Service Unavailable, que en HTTP significa que el servidor está sobrecargado o fuera de servicio por mantenimiento. Los paneles y alertas que consideran las respuestas 5xx errores del servidor despertarán a alguien porque un solo cliente abusó, y el cliente no aprenderá nada sobre su propio comportamiento. Responda con 429, que sitúa la causa del lado del cliente, y registre los rechazos en un nivel propio.

Todos los clientes limitados vuelven en el mismo segundo. Si una cuota se restablece en un momento fijo o se indica el mismo `Retry-After` a todos los clientes rechazados, todos vuelven juntos; el borrador del IETF pone el ejemplo de un restablecimiento diario a las 18:00:00, cuando es probable que todos los clientes aparezcan a las 18:00:00. La ráfaga posterior puede ser peor que la que se limitó. Añada variación aleatoria en ambos lados: varíe la espera que anuncia el servidor y deje que los clientes añadan su propio retraso aleatorio.

Nadie decidió qué ocurre cuando el almacén deja de funcionar. Un limitador que conserva sus cuentas en Redis o en una base de datos debe hacer algo cuando el almacén es inaccesible. El código que captura el error de conexión y deja pasar la petición ha elegido permitir el tráfico ante fallos, lo que convierte la interrupción en una ventana sin límites para quien esté pendiente; el que rechaza ha elegido cerrar el paso ante fallos, lo que convierte el problema en una interrupción para todos. Ninguna de las dos opciones es siempre incorrecta, pero la elección corresponde a una decisión de diseño por ruta, como hizo el directorio anterior, no a lo que haga una biblioteca de forma predeterminada.

Contar los rechazos filtra información. Si las respuestas 401 y 403 consumen cuota, advierte el borrador del IETF, un cliente malicioso podría sondear un punto de acceso para conocer el tráfico de otro usuario. Decida qué respuestas cuentan, asigne un límite propio a los intentos fallidos de autenticación y mantenga los detalles de cuota fuera del alcance de los clientes que no se hayan autenticado.

Lecturas adicionales

- RFC 6585: códigos de estado HTTP adicionales — Define 429 Too Many Requests e indica qué deja el código a criterio del servidor: quién es el usuario y cómo se cuentan las peticiones.
- RFC 9110: semántica HTTP — El campo `Retry-After` y sus dos formas, y qué significa 503 cuando el servidor lo envía.
- Campos de cabecera RateLimit para HTTP — El borrador del grupo de trabajo HTTPAPI del IETF sobre `RateLimit-Policy` y `RateLimit`, con una sección de seguridad sobre regresos sincronizados y filtraciones de información.
- Módulo ngx_http_limit_req_module — `limit_req`, `burst`, `nodelay`, `delay`, el modo de simulación y los tamaños de la zona de memoria compartida.
- Manual de configuración de HAProxy 3.4 — Tablas de seguimiento y sus contadores de tasa, y el protocolo de pares que las copia entre instancias.



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 la limitación de solicitudes?» 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/rate-limiting/