**Engineer**

# eventos enviados por el servidor

"A" Flujo unidireccional de eventos de un servidor por HTTP

## Qué es

Los eventos enviados por el servidor permiten que un servidor web envíe mensajes a una página mediante una única respuesta HTTP normal, que mantiene abierta y en la que escribe cuando hay novedades. Los define el Estándar HTML, mantenido por WHATWG, que describe tanto el formato de texto del flujo como `EventSource`, la interfaz del navegador que lo lee. El nombre suele abreviarse como SSE, y el formato suele recibir el nombre de su tipo de contenido, `text/event-stream`.

Una página abre un flujo con una sola línea, `new EventSource(url)`. El navegador envía una solicitud `GET`, que puede incluir `Accept: text/event-stream`, y espera una respuesta con estado 200 y ese tipo de contenido. El servidor envía las cabeceras y después no cierra el cuerpo. Cada vez que ocurre algo, escribe unas líneas de texto, y el navegador convierte cada bloque de líneas en un evento que la página puede escuchar.

El formato es deliberadamente sencillo. Un flujo siempre usa UTF-8 y cada línea es un campo. Una línea `data` contiene los datos, y varias líneas `data` seguidas se unen con saltos de línea. Una línea `event` asigna un tipo al evento; sin ella, el tipo es `message`. Una línea `id` establece el identificador del último evento, y una línea `retry` establece cuántos milisegundos espera el navegador antes de reconectarse. Una línea que empieza por dos puntos es un comentario que el navegador ignora. Una línea en blanco termina el bloque y despacha el evento: si un bloque se corta antes de su línea en blanco, se descarta.

La reconexión forma parte del estándar. Cuando se interrumpe la conexión, el navegador espera el tiempo de reconexión, cuyo valor inicial elige él mismo y que, según el estándar, probablemente sea de unos segundos; después vuelve a enviar la solicitud. Si el servidor ha asignado identificadores a sus eventos, la

nueva solicitud incluye el último en una cabecera `Last-Event-ID`, y el servidor puede continuar desde ahí. Otras respuestas terminan el flujo definitivamente: un estado distinto de 200 u otro tipo de contenido hacen que el navegador desista y deje de intentarlo. El estándar indica 204 No Content como la forma de que un servidor comunique que el flujo ha terminado.

El flujo es unidireccional. Tras la solicitud inicial, nada viaja de la página al servidor por esa conexión. Cuando la página tiene algo que comunicar, envía otra solicitud HTTP independiente.

Utilice el concepto en una decisión

Aplique «¿Qué son los eventos enviados por el servidor?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

**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. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

**Diseño y operación de 6 flujos ETL/ELT** Póngase en contacto →

## Por qué importa

Muchas funciones que parecen funcionar en tiempo real solo necesitan una dirección: el progreso de un trabajo largo, una notificación, una cifra en un panel o una respuesta que llega por partes. Sin un flujo, la página tiene que consultar periódicamente. Pregunta cada pocos segundos, la mayoría de las respuestas dicen que nada ha cambiado y puede transcurrir hasta un intervalo entero antes de que alguien vea las novedades. Con un flujo, el servidor escribe en cuanto ocurre algo.

La alternativa habitual para enviar datos desde el servidor es WebSocket, que cambia la conexión de HTTP a un protocolo propio. Los eventos enviados por el servidor permanecen dentro de HTTP de principio a fin: pasan por los mismos proxies y balanceadores de carga que cualquier otra solicitud, usan las mismas cookies y reglas para solicitudes entre orígenes y pueden compartir una conexión HTTP/2 con el resto de la página. El navegador se encarga de la reconexión y de reanudar desde el último evento, en lugar de que cada equipo tenga que escribir ese código. WebSocket sigue siendo la opción adecuada cuando el cliente tiene tanto que comunicar como el servidor, como en un chat o un editor compartido.

El formato tampoco está ligado al navegador. Cualquier cliente HTTP puede leerlo, y lo utilizan especificaciones ajenas a la Web. En Model Context Protocol, mediante el que los agentes de IA llaman a herramientas, el transporte Streamable HTTP permite que un servidor responda a una solicitud con un objeto JSON o con una respuesta `text/event-stream`. En ese flujo puede enviar notificaciones y solicitudes propias antes de la respuesta final y adjuntar identificadores de evento para que un cliente que pierda la conexión reanude con `Last-Event-ID`. Streamable HTTP sustituyó a HTTP+SSE, el transporte de la versión anterior del protocolo, 2024-11-05.

Conviene conocer las limitaciones antes de elegirlo. El flujo solo transporta texto, por lo que los datos binarios deben codificarse primero como texto. El constructor `EventSource` del navegador acepta una URL y una opción, `withCredentials`, y nada más: la página no puede añadir una cabecera `Authorization` a la solicitud.

## Cómo se aplica en la práctica

El siguiente intercambio muestra un trabajo de exportación prolongado. El navegador ha perdido la conexión después del evento 41 y se está reconectando, por lo que su solicitud incluye ese identificador y el servidor continúa desde el evento 42:

```
GET /jobs/4471/events HTTP/1.1
Host: example.com
Accept: text/event-stream
Last-Event-ID: 41

HTTP/1.1 200 OK
Content-Type: text/event-stream
Cache-Control: no-store

retry: 5000

: still working

id: 42
event: progress
data: {"done": 120, "total": 400}

id: 43
event: progress
data: {"done": 400, "total": 400}
```

```
id: 44
event: finished
data: {"report": "/jobs/4471/report.pdf"}
```

La línea `retry` pide al navegador que espere cinco segundos antes de cualquier reconexión posterior. La línea de comentario no contiene ningún evento: mantiene activa la conexión entre las actualizaciones del trabajo. La página escucha los dos tipos de evento por su nombre:

```
const source = new EventSource("/jobs/4471/events");
source.addEventListener("progress", (event) => {
  const { done, total } = JSON.parse(event.data);
  showProgress(done / total);
});
source.addEventListener("finished", (event) => {
  showReport(JSON.parse(event.data).report);
  source.close();
});
source.addEventListener("error", () => {
  if (source.readyState === EventSource.CLOSED) showRetryButton();
});
```

La llamada a `close()` importa. Cuando el servidor termina una respuesta normalmente, el navegador la trata como una conexión interrumpida y se reconecta. Sin esa llamada, el flujo de un trabajo terminado volvería a abrirse al cabo de cinco segundos. El manejador de `error` distingue los dos tipos de fallo: mientras `readyState` sea `CONNECTING`, el navegador sigue intentándolo; cuando pasa a `CLOSED`, ha desistido.

El trabajo que continúa después de que el agente haya recibido su respuesta aparece en la página de este sitio sobre convertir el software existente en herramientas para agentes de IA. Describe cómo ofrecer las operaciones de una empresa como herramientas, con un servidor MCP que ofrece las mismas operaciones a los agentes que usan ese protocolo, y trasladar el trabajo lento a una cola para que el agente reciba una respuesta mientras el trabajo continúa. En la especificación MCP, un servidor puede utilizar un flujo de eventos como el anterior para enviar solicitudes o notificaciones propias antes de la respuesta final a la solicitud del cliente.

# Dónde surgen los problemas

**Los eventos llegan en ráfagas o todos al final.** Algo entre el servidor y el navegador retiene la respuesta. nginx, por ejemplo, almacena por defecto en búfer las respuestas del servidor que tiene detrás y solo las transmite a medida que llegan cuando ese almacenamiento está desactivado. El Estándar HTML también advierte de que la transferencia HTTP por bloques realizada por una capa que desconoce los requisitos temporales del flujo puede volverlo poco fiable. Desactive el almacenamiento en búfer para la ubicación del flujo; en nginx, con `proxy_buffering off` o una cabecera de respuesta `X-Accel-Buffering: no`. Vacíe el búfer de salida después de cada evento en el código del servidor y pruebe la ruta completa, incluido el proxy, en lugar del servidor aislado.

**Un tiempo de espera corta los flujos sin actividad.** Un proxy que no ve bytes durante un tiempo supone que la conexión ha muerto. En nginx, `proxy_read_timeout` vale por defecto 60 segundos entre dos lecturas; después, la conexión se cierra. El Estándar HTML advierte de que algunos proxies antiguos cortan las conexiones tras un breve tiempo de espera. Sugiere una línea de comentario aproximadamente cada 15 segundos, que cuesta unos pocos bytes y que el navegador ignora.

**Se pierden eventos durante una reconexión.** El navegador solo envía `Last-Event-ID` si el servidor asignó identificadores a sus eventos, y la cabecera solo ayuda si el servidor aún puede encontrar lo que vino después de ese identificador. Asigne a cada evento un identificador que sirva de cursor en el flujo, conserve una ventana de eventos recientes para reproducirlos y envíe el estado actual completo cuando un identificador sea demasiado antiguo para reanudar desde él. La especificación MCP añade una regla que conviene adoptar: los identificadores pertenecen a un flujo, y un servidor no debe reproducir en un flujo mensajes destinados a otro.

**La reconexión no se detiene nunca o se detiene para siempre.** Un flujo que termina normalmente vuelve a abrirse una y otra vez tras el tiempo de reconexión. Un trabajo terminado sigue reabriendo su flujo salvo que la página llame a `close()` o el servidor responda a la reconexión con 204. Durante un despliegue ocurre lo contrario. Un proxy que responde 502 o 503 mientras se reinician los servidores da a cada `EventSource` abierto un estado distinto de 200, y todos dejan de intentarlo definitivamente. Escuche `error` y, cuando `readyState` sea `CLOSED`, cree un nuevo `EventSource` tras una demora aleatoria que crezca con cada fallo.

**Hay demasiados flujos para un solo navegador.** Los navegadores limitan cuántas conexiones mantienen abiertas con un servidor, y un flujo servido mediante HTTP/1.1 ocupa una durante toda su vida. El Estándar HTML advierte de que un sitio en el que cada página abierta tenga su propio `EventSource` puede alcanzar ese límite. Sirva el flujo mediante HTTP/2, donde los flujos comparten una conexión y la RFC 9113 recomienda permitir al menos 100 a la vez, o comparta un solo `EventSource` entre las pestañas mediante un worker compartido.

**El token está en la dirección.** Como `EventSource` no puede enviar una cabecera `Authorization`, la solución rápida es poner un token en la URL, donde los proxies inversos y los registros de acceso lo guardan. Autentique el flujo con la cookie de sesión que ya tiene la página, añadiendo `withCredentials` si el flujo procede de otro origen, o con un ticket de acceso de corta duración y de un solo uso que la página obtenga mediante una solicitud autenticada justo antes de abrir el flujo.

## Lecturas adicionales

- Estándar HTML: eventos enviados por el servidor — el formato del flujo, la interfaz `EventSource`, la reconexión y las notas para autores sobre proxies y límites de conexiones.
- Transportes de Model Context Protocol — Streamable HTTP, con el que un servidor puede responder mediante un flujo de eventos, y cómo se reanuda un flujo interrumpido.
- RFC 9113: HTTP/2 — flujos que comparten una conexión y el ajuste que limita cuántos pueden estar abiertos a la vez.
- nginx: módulo ngx_http_proxy_module — `proxy_buffering`, la cabecera `X-Accel-Buffering` y `proxy_read_timeout`.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

**Vea cuánto cuesta** Póngase en contacto →

# Haga práctica la terminología

Pregunte cómo se aplica «¿Qué son los eventos enviados por el servidor?» 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/server-sent-events/