Engineer

Server-sent events

"Å" en enveisstrøm av hendelser fra en server over HTTP

Hva det er

Server-sent events er en måte en webserver kan skyve meldinger ut til en side på, gjennom ett vanlig HTTP-svar som den holder åpent og skriver i hver gang det er noe nytt. De er definert i HTML-standarden, som WHATWG vedlikeholder, og den beskriver både tekstformatet for strømmen og `EventSource`, grensesnittet i nettleseren som leser strømmen. Navnet forkortes ofte til SSE, og formatet kalles ofte etter medietypen sin, `text/event-stream`.

En side åpner en strøm med én linje, `new EventSource(url)`. Nettleseren sender en `GET`-forespørsel, som kan ha med seg `Accept: text/event-stream`, og forventer et svar med status 200 og den innholdstypen. Serveren sender headerfeltene og gjør så ikke svaret ferdig. Hver gang noe skjer, skriver den noen linjer tekst, og nettleseren gjør hver blokk med linjer om til en hendelse som siden kan lytte etter.

Formatet er bevisst enkelt. En strøm er alltid UTF-8, og hver linje er ett felt. En `data`-linje bærer innholdet, og flere `data`-linjer etter hverandre settes sammen med linjeskift. En `event`-linje gir hendelsen en type, og uten den er typen `message`. En `id`-linje setter ID-en til den siste hendelsen, og en `retry`-linje angir hvor mange millisekunder nettleseren venter før den gjenoppretter forbindelsen. En linje som begynner med kolon, er en kommentar, som nettleseren ignorerer. En tom linje avslutter blokken og sender hendelsen av sted, så en blokk som blir avbrutt før den tomme linjen sin, blir kastet.

Gjenoppkobling er en del av standarden. Når forbindelsen brytes, venter nettleseren i gjenoppkoblingstiden, som starter på en verdi nettleseren selv velger, ifølge standarden trolig noen sekunder, og sender så forespørselen på nytt. Har serveren gitt hendelsene sine ID-er, har den nye forespørselen med seg den siste av dem i et `Last-Event-ID`-headerfelt, og serveren kan fortsette derfra.

Noen svar avslutter i stedet strømmen for godt: en annen status enn 200 eller en annen innholdstype får nettleseren til å gi opp og slutte å prøve, og standarden peker ut 204 No Content som måten en server sier at strømmen er slutt på.

Retningen går bare én vei. Etter den første forespørselen går ingenting fra siden til serveren over den forbindelsen; når siden har noe å si, sender den en egen HTTP-forespørsel.

Bruk begrepet i en beslutning

Bruk «Hva er server-sent events?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Delen brukerne aldri ser: hvor dataene ligger, hvilke regler som gjelder, og hvordan andre systemer får lov til å snakke med den.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Designet og driftet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tvers av tre databasemotorer

Designet og driftet 6 ETL/ELT-pipelines Ta kontakt →

Hvorfor det er viktig

Mange funksjoner som ser levende ut, trenger bare én retning: framdriften i en lang jobb, en ny melding, et tall på et dashbord, et svar som kommer i deler. Uten en strøm må siden polle. Den spør med noen sekunders mellomrom, de fleste svarene sier at ingenting er endret, og nyheter venter opptil et helt intervall før noen ser dem. Med en strøm skriver serveren i samme øyeblikk som noe skjer.

Det vanlige alternativet for å skyve data ut er WebSocket, som bytter forbindelsen over fra HTTP til en egen protokoll. Server-sent events holder seg i HTTP fra start til slutt, så de passerer de samme proxyene og lastbalanserere som enhver annen forespørsel, de bruker de samme informasjonskapslene og de samme reglene for forespørsler på tvers av opphav, og de kan dele én HTTP/2-forbindelse med resten av siden. Gjenoppkobling og gjenopptakelse fra den siste hendelsen følger med nettleseren i stedet for å være kode hvert team skriver selv. WebSocket er fortsatt det riktige valget når klienten har like mye å si som serveren, som i en chat eller en delt editor.

Formatet er heller ikke bundet til nettleseren. Enhver HTTP-klient kan lese det, og spesifikasjoner utenfor nettet bruker det. I Model Context Protocol, som AI-agenter kaller verktøy gjennom, lar transporten Streamable HTTP en server svare på en forespørsel enten med ett JSON-objekt eller med et `text/event-stream`-svar. På den strømmen kan serveren sende varslinger og egne forespørsler før

det endelige svaret, og den kan sette hendelses-ID-er på, slik at en klient som mister forbindelsen, kan fortsette med `Last-Event-ID`. Streamable HTTP erstattet HTTP+SSE, transporten i den tidligere protokollversjonen 2024-11-05.

Begrensningene er verdt å kjenne før man velger det. Strømmen bærer bare tekst, så binære data må først kodes som tekst. Nettleserens `EventSource`-konstruktør tar en URL og én innstilling, `withCredentials`, og ingenting annet, så en side kan ikke legge til et `Authorization`-headerfelt i forespørselen.

Slik ser det ut i praksis

Utvekslingen nedenfor følger en lang eksportjobb. Nettleseren har mistet forbindelsen etter hendelse 41 og kobler seg til igjen, så forespørselen har med seg den ID-en, og serveren fortsetter fra hendelse 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"}
```

`retry`-linjen ber nettleseren vente fem sekunder før hver senere gjenoppkobling, og kommentarlinjen bærer ingen hendelse i det hele tatt; den er der for å holde forbindelsen i aktivitet mens jobben er stille. Siden lytter etter de to hendelsestypene ved navn:

```
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();
});
```

Kallet til `close()` betyr noe. Når serveren avslutter et svar på vanlig måte, behandler nettleseren det som en brutt forbindelse og kobler seg til igjen, så en strøm for en ferdig jobb ellers ville åpnet seg igjen etter fem sekunder. `error`-handleren skiller mellom de to typene feil: så lenge `readyState` er `CONNECTING`, prøver nettleseren fortsatt, og når den er `CLOSED`, har den gitt opp.

Arbeid som fortsetter etter at agenten har fått svaret sitt, er et av temaene på siden om programvaren din som verktøy for AI-agenter her på nettstedet. Den beskriver hvordan hver operasjon en agent skal ha tilgang til, tilbys som et verktøy, med en MCP-server som tilbyr de samme operasjonene til agenter som bruker MCP, og hvordan tidkrevende jobber legges i en kø, så agenten får et svar mens arbeidet pågår. I MCP-spesifikasjonen kan en server bruke en hendelsesstrøm som den ovenfor til å sende egne forespørsler eller varslinger før det endelige svaret på klientens forespørsel.

Hvor det går galt

Hendelser som kommer i klumper, eller alle til slutt. Noe mellom serveren og nettleseren holder svaret tilbake. nginx, for eksempel, bufrer som standard svarene fra serveren bak seg og sender bare et svar videre etter hvert som det kommer, når bufring er slått av. HTML-standarden advarer også om at HTTP-chunking utført av et lag som ikke vet noe om timingen i strømmen, kan gjøre den upålitelig. Slå av bufring for strømmens location, i nginx med `proxy_buffering off` eller med headerfeltet `X-Accel-Buffering: no` i svaret, tøm skrivebufferen etter hver hendelse i serverkoden, og test gjennom hele veien, proxy medregnet, i stedet for mot serveren alene.

Stille strømmer som blir kuttet av en timeout. En proxy som ikke ser en eneste byte på en stund, går ut fra at forbindelsen er død. I nginx er standardverdien for `proxy_read_timeout` 60 sekunder mellom to lesinger, og deretter blir forbindelsen lukket, og HTML-standarden advarer om at noen eldre proxyer bryter forbindelser etter en kort timeout. Den foreslår en kommentarlinje omtrent hvert 15. sekund, som koster noen få byte, og som nettleseren ignorerer.

Hendelser som går tapt ved en gjenoppkobling. Nettleseren sender bare `Last-Event-ID` hvis serveren har gitt hendelsene sine ID-er, og headerfeltet hjelper bare hvis serveren fortsatt kan finne det som kom etter den ID-en. Gi hver hendelse en ID som fungerer som en markør i strømmen, behold et vindu med nylige hendelser som kan spilles av på nytt, og send hele den gjeldende tilstanden når en ID er for gammel å fortsette fra. MCP-spesifikasjonen legger til en regel som er verdt å kopiere: ID-er hører til én strøm, og en server må ikke spille av meldinger på nytt på én strøm når de var ment for en annen.

Gjenoppkobling som aldri stopper, eller stopper for godt. En strøm som slutter normalt, kommer tilbake etter gjenoppkoblingstiden, igjen og igjen, så en ferdig jobb fortsetter å åpne strømmen sin på nytt med mindre siden kaller `close()` eller serveren svarer på gjenoppkoblingen med 204. Det motsatte skjer under en utrulling. En proxy som svarer 502 eller 503 mens serverne starter på nytt, gir hver åpen `EventSource` en annen status enn 200, og hver eneste av dem stopper for godt. Lytt etter `error`, og når `readyState` er `CLOSED`, opprett en ny `EventSource` etter en tilfeldig forsinkelse som vokser for hver feil.

For mange strømmer til én nettleser. Nettlesere begrenser hvor mange forbindelser de holder åpne til én server, og en strøm over HTTP/1.1 opptar en av dem så lenge den varer. HTML-standarden advarer om at et nettsted der hver åpen side har sin egen `EventSource`, kan møte den grensen. Lever strømmen over HTTP/2, der strømmer deler én forbindelse, og RFC 9113 anbefaler å tillate minst 100 om gangen, eller del én `EventSource` mellom fanene gjennom en shared worker.

Tokenet i adressen. Fordi `EventSource` ikke kan sende et `Authorization`-headerfelt, er den raske løsningen et token i URL-en, der reverse proxyer og tilgangslogger skriver det ned. Autentiser strømmen med sesjonens informasjonskapsel, som siden allerede har, og legg til `withCredentials` når strømmen kommer fra et annet opphav, eller med en kortlivet engangsbillett som siden henter med en autentisert forespørsel rett før den åpner strømmen.

Les mer

- HTML Standard: Server-sent events — formatet for strømmen, `EventSource` -grensesnittet, gjenoppkobling og standardens merknader til forfattere om proxyer og grenser for antall forbindelser.
- Transports - Model Context Protocol — Streamable HTTP, der en server kan svare med en hendelsesstrøm, og hvordan en avbrutt strøm blir gjenopptatt.
- RFC 9113: HTTP/2 — strømmer som deler én forbindelse, og innstillingen som begrenser hvor mange som kan være åpne samtidig.
- nginx: Module ngx_http_proxy_module — `proxy_buffering` , headerfeltet `X-Accel-Buffering` og `proxy_read_timeout` .

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er server-sent events?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/server-sent-events/