Engineer

WebSocket

↔ en protokoll for en toveisforbindelse som holdes åpen

Hva det er

WebSocket er en protokoll for en forbindelse begge parter kan skrive til når som helst. Et HTTP-håndtrykk oppretter den, og den holdes åpen til en part lukker den eller forbindelsen svikter. IETF publiserte den som RFC 6455 i desember 2011, og den delen en nettleser gjør tilgjengelig for skript, objektet `WebSocket`, er definert i WHATWGs WebSockets Standard. I kode og dokumentasjon ser man også navnet skrevet «websocket» eller «web socket», og flertallsformen WebSockets brukes som regel om teknologien som helhet.

I det klassiske HTTP/1.1-håndtrykket starter forbindelsen med en vanlig `GET`. Klienten legger til `Upgrade: websocket` og `Connection: Upgrade`, en `Sec-WebSocket-Version` på 13 og en `Sec-WebSocket-Key`, som er en tilfeldig verdi på 16 byte kodet i base64. En server som sier ja, svarer med statuskoden 101 Switching Protocols og headerfeltet `Sec-WebSocket-Accept`. Verdien beviser at serveren har lest akkurat dette håndtrykket og ikke spiller av et gammelt svar: den er klientens nøkkel med en fast GUID hengt på, hashet med SHA-1 og kodet i base64. Fra det øyeblikket er HTTP ferdig på forbindelsen, og partene utveksler rammer i stedet. HTTP/2 og HTTP/3 fører i stedet WebSocket over én strøm ved hjelp av extended CONNECT, slik RFC 8441 og RFC 9220 beskriver.

En ramme er liten. Headeren er minst to byte, vokser med to eller åtte når innholdet er lengre enn 125 byte, og med ytterligere fire på rammer fra klienten, som bærer en maskeringsnøkkel. Hver ramme har en opcode: tekst, binær eller fortsettelse når en melding er delt over flere rammer, og i tillegg tre kontrollrammer for lukking, ping og pong. En fullstendig tekstmelding må være gyldig UTF-8; ett enkelt fragment kan dele en UTF-8-sekvens. Hver ramme en klient sender, maskeres med en ny tilfeldig 32-bits nøkkel, og en server maskerer aldri. Maskering er ikke kryptering. Den finnes for at et fiendtlig skript ikke skal kunne velge nøyaktig hvilke byte som krysser nettverket, og bruke dem til å forgifte en mellomlagrende proxy et sted på veien.

Adresser begynner med `ws://` eller `wss://`, med port 80 og port 443 som standard, og `wss` kjører hele utvekslingen inne i TLS. Det er de samme portene HTTP allerede bruker, og det er med vilje: for brannmurer og proxyer underveis ser håndtrykket ut som en hvilken som helst annen nettforespørsel.

Bruk begrepet i en beslutning

Bruk «Hva er WebSocket?» 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

Med korte HTTP-svar snakker serveren bare når den blir spurt. Mange produkter trenger det motsatte: en chatmelding, en pris som nettopp har endret seg, en markør i et delt dokument, en bakgrunnsjobb som er ferdig. Uten en kanal serveren kan skrive til, må klienten polle. Den spør med noen sekunders mellomrom, de fleste svarene sier bare at ingenting har endret seg, hvert av dem bærer et fullt sett headerfelt, og en nyhet venter opptil et helt intervall før noen ser den. En åpen WebSocket-forbindelse fjerner begge kostnadene. Serveren kan sende når noe skjer, uten å vente på neste polling. Hver melding trenger bare noen få byte rammeinformasjon i stedet for et fullt sett HTTP-headere.

Prisen er tilstand. En HTTP-forespørsel kan havne på hvilken som helst server bak en lastbalanserer, mens en WebSocket-forbindelse tilhører én prosess så lenge den lever. Den prosessen holder av minne til hver åpen forbindelse, en produksjonssetting eller omstart kutter alle på én gang, og hver proxy og lastbalanserer på veien må støtte håndtrykket og tåle en forbindelse som kanskje er stille i flere minutter. Kjører man mer enn én server, oppstår et spørsmål vanlig HTTP aldri stiller: når noe skjer på server A, hvordan kommer det fram til en klient som har forbindelsen sin på server B? Det vanlige svaret er en felles publish/subscribe-kanal som alle serverne lytter på, og det er enda et system å drifte.

WebSocket er altså et bevisst valg, ikke et utgangspunkt. Når bare serveren har noe å fortelle, kan server-sent events, som er definert i HTML-standarden, sende hendelser over et vanlig HTTP-svar av typen `text/event-stream`, og nettleseren

kobler seg til igjen av seg selv når svaret brytes. WebSocket fortjener de ekstra bevegelige delene sine når klienten har like mye å si som serveren, som i chat, spill og samtidig redigering av det samme dokumentet.

Slik ser det ut i praksis

Utvekslingen nedenfor åpner en chatforbindelse. Nøkkelen er eksempelverdien fra selve RFC 6455, og `Sec-WebSocket-Accept` -verdien er den standarden selv viser, og som enhver korrekt server må sende tilbake for akkurat den nøkkelen.

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Origin: https://example.com

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Etter den tomme linjen som avslutter serverens svar, er ingenting på forbindelsen HTTP lenger. En nettleser gjør hele håndtrykket bak ett kall til konstruktøren og overlater resten til hendelseslyttere:

```
const socket = new WebSocket("wss://example.com/chat");
socket.addEventListener("open", () => socket.send("hei"));
socket.addEventListener("message", (event) => console.log(event.data));
socket.addEventListener("close", (event) => {
  console.log(event.code, event.reason, event.wasClean);
});
```

Lukkehendelsen forteller hvordan forbindelsen endte. Kode 1000 er en normal lukking, og 1001 betyr at den ene parten går sin vei, for eksempel en server som slås av, eller en nettleser som forlater siden. Kode 1006 betyr at forbindelsen forsvant helt uten noen lukkeramme. Ingen av partene har lov til å sende den; den finnes bare for at programmet over skal kunne skille en unormal slutt fra en ren.

Grensen mellom forespørsel og svar på den ene siden og en åpen socket på den andre vises tydelig i et prosjekt i nettstedets porteføljе. Da et API ble flyttet over på Huma v2, ble alle rutene migrert over på Humas modell for forespørsel og svar, med nøyaktig to unntak: WebSocket-endepunktene ble igjen på ren Fiber, fordi koden der faktisk vil ha selve socketen, og Humas modell ikke passer. Et rammeverk bygd rundt formen til HTTP beskriver og validerer forespørsler og svar godt, og en langlivet toveisforbindelse er rett og slett noe annet.

Hvor det går galt

En side på et annet nettsted åpner en forbindelse med cookiene til de besøkende. Nettleseren sender håndtrykksforespørselen til serverens vert og kan ta med vertens cookies, avhengig av nettleserens cookiepolicy. Der denne policyen tillater autentiserte forespørsler på tvers av nettsteder, kan en server som autentiserer med cookies og ignorerer `Origin`, godta en forbindelse åpnet av en annen nettside på vegne av den besøkende. RFC 6455 sier at en server som bare er ment for bestemte nettsteder, bør sjekke `Origin` mot dem den venter, og avvise resten med 403 Forbidden. Gjør det i håndtrykket, før forbindelsen godtas.

Tilgangstokenet havner i loggene. Nettleserens `WebSocket`-konstruktør tar en URL og en valgfri liste med subprotokoller og ingenting annet, så et skript kan ikke legge til et `Authorization`-felt. Den raske omveien er et token i URL-en, der omvendte proxyer og tilgangslogger skriver det ned. RFC-en lar autentisering stå åpen og nevner cookies, HTTP-autentisering og TLS-autentisering som muligheter. En cookie sammen med `Origin`-sjekken ovenfor fungerer, og det gjør også en kortlivet engangsbillett som siden henter over et autentisert HTTP-kall og bruker på håndtrykket.

En omstart gjør alle klientene til en flodbølge. En brå produksjonssetting eller omstart kan lukke alle forbindelser samtidig. Klienter som ikke mottar noen lukkeramme, melder kode 1006; hvis alle kobler seg til igjen med en gang og fortsetter å prøve, kommer serveren opp til noe RFC-en sammenligner med et tjenestenektangrep. RFC 6455 ber klienter vente en tilfeldig tid før første forsøk, foreslår noe mellom 0 og 5 sekunder, og ber dem deretter vente lenger etter hver feil, for eksempel med avkortet binær eksponentiell backoff. En server som lukker forbindelsene sine med 1001 før den stopper, forteller i det minste klientene at bruddet var planlagt.

```
function reconnectDelay(attempt) {
  if (attempt === 0) return Math.random() * 5000;
  const ceiling = Math.min(60_000, 1000 * 2 ** attempt);
  return Math.random() * ceiling;
}
```

Forbindelser dør uten at noen merker det. En forbindelse uten trafikk gir ingen av partene noe tegn på at veien fortsatt finnes, og en motpart som forsvant uten en lukkeramme, ser nøyaktig ut som en stille forbindelse helt til neste skriving feiler. Det er dette ping og pong er til for: et endepunkt må svare på en ping med en pong med mindre det allerede har mottatt en lukkeramme. Hvis flere pinger er ubesvart, kan det nøye seg med å svare på den siste. RFC-en nevner at en ping kan fungere som keepalive eller som en sjekk på at den andre siden fortsatt svarer. Nettleserens API har ingen måte å sende en ping på, så timeren hører hjemme på serveren: ping hver forbindelse med fast intervall, og lukk dem som ikke svarer med pong.

Én melding spiser opp serverens minne. Lengdefeltet i en rammeheader gir plass til langt større innhold enn noen server har minne til, og RFC-en advarer om at en fiendtlig motpart kan sende én enorm ramme eller en endeløs rekke små fragmenter. Implementasjoner må beskytte seg selv, så sett en maksimal meldingsstørrelse i serverbiblioteket, og lukk med kode 1009, message too big, når en motpart går over den. Det samme etterslepet kan hope seg opp i motsatt retning. I nettleseren viser `bufferedAmount` hvor mange byte fra `send()` som fortsatt venter på å komme ut på nettverket, og WHATWGs eksempel sender neste oppdatering først når tallet er null. På serveren fyller en klient som leser sakte, den utgående bufferen på samme måte, så hver forbindelse trenger et tak for den og en regel for hva som skal skje når taket er nådd.

Les mer

- RFC 6455: The WebSocket Protocol — håndtrykket, rammene, maskeringen og lukkekodene, og sikkerhetsbegrunnelsen bak hver av dem.
- WebSockets Standard — nettleserens API: konstruktøren, `send()`, `close()`, `bufferedAmount` og lukkehendelsen.
- RFC 9110: HTTP Semantics — headerfeltet `Upgrade` og statuskoden 101 Switching Protocols, som håndtrykket bygger på.
- Server-sent events — HTML-standardens enveisalternativ med innebygd gjenoppkobling.
- RFC 8441: Bootstrapping WebSockets with HTTP/2 — extended CONNECT fører WebSocket over én multiplekset HTTP/2-strøm.

- RFC 9220: Bootstrapping WebSockets with HTTP/3 — den tilsvarende extended-CONNECT-mekanismen for HTTP/3.

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 WebSocket?» 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/websocket/