Engineer

WebSocket

↔ en protokol til en tovejsforbindelse, der forbliver åben

Hvad det er

WebSocket er en protokol til en forbindelse, som begge parter kan skrive på når som helst. Et HTTP-håndtryk opretter den, og den forbliver åben, indtil en part lukker den, eller forbindelsen svigter. IETF udgav den som RFC 6455 i december 2011, og den del, en browser stiller til rådighed for scripts, objektet `WebSocket`, er defineret i WHATWG's WebSockets Standard. I kode og dokumentation ser man også navnet stavet »websocket« eller »web socket«, og flertalsformen WebSockets bruges som regel om teknologien som helhed.

I det klassiske HTTP/1.1-håndtryk begynder forbindelsen med en almindelig `GET`. Klienten tilføjer `Upgrade: websocket` og `Connection: Upgrade`, en `Sec-WebSocket-Version` på 13 og en `Sec-WebSocket-Key`, som er en tilfældig værdi på 16 byte kodet i base64. En server, der siger ja, svarer med statuskoden 101 Switching Protocols og headerfeltet `Sec-WebSocket-Accept`. Værdien beviser, at serveren har læst netop dette håndtryk og ikke genbruger et gammelt svar: den er klientens nøgle med et fast GUID hængt på, hashet med SHA-1 og kodet i base64. Fra det øjeblik er HTTP færdig på forbindelsen, og parterne udveksler rammer i stedet. HTTP/2 og HTTP/3 bærer i stedet WebSocket over én strøm med extended CONNECT, som RFC 8441 og RFC 9220 beskriver.

En ramme er lille. Headeren fylder mindst to byte, vokser med to eller otte, når indholdet er længere end 125 byte, og med yderligere fire på rammer fra klienten, som bærer en maskeringsnøgle. Hver ramme har en opcode: tekst, binær eller fortsættelse, når en besked er delt over flere rammer, og dertil tre kontrolrammer til lukning, ping og pong. En komplet tekstbesked skal være gyldig UTF-8; et enkelt fragment kan dele en UTF-8-sekvens. Hver ramme, en klient sender, er maskeret med en ny tilfældig 32-bit-nøgle, og en server maskerer aldrig. Maskering er ikke kryptering. Den findes, så et fjendtligt script ikke kan vælge præcis de byte, der krydser netværket, og bruge dem til at forgifte en cachende proxy et sted på vejen.

Adresser begynder med `ws://` eller `wss://`, med port 80 og port 443 som standard, og `wss` kører hele udvekslingen inde i TLS. Det er de porte, HTTP allerede bruger, og det er med vilje: håndtrykket ligner enhver anden webforespørgsel for de firewalls og proxyer, det kommer igennem.

Brug begrebet i en beslutning

Brug »Hvad er WebSocket?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Den del, brugerne aldrig ser: hvor data ligger, hvilke regler der gælder, og hvordan andre systemer må tale med den.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har designet og drevet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tre databasemotorer

Har designet og drevet 6 ETL/ELT-pipelines Kontakt os →

Hvorfor det betyder noget

Med korte HTTP-svar taler serveren kun, når den bliver spurgt. Mange produkter har brug for det modsatte: en chatbesked, en pris, der lige har ændret sig, en markør i et delt dokument, et baggrundsjob, der er blevet færdigt. Uden en kanal, serveren kan skrive på, må klienten polle. Den spørger med få sekunders mellemrum, de fleste svar siger blot, at intet er ændret, hvert svar bærer et helt sæt headerfelter, og en nyhed venter op til et helt interval, før nogen ser den. En åben WebSocket-forbindelse fjerner begge omkostninger. Serveren kan sende, når noget sker, uden at vente på næste polling. Hver besked behøver kun nogle få byte til rammen i stedet for et helt sæt HTTP-headerfelter.

Prisen er, at forbindelsen bærer tilstand. En HTTP-forespørgsel kan lande på en hvilken som helst server bag en load balancer, mens en WebSocket-forbindelse hører til én proces, så længe den lever. Den proces holder hukommelse for hver åben forbindelse, en udrulning eller genstart kapper dem alle på én gang, og hver proxy og load balancer på vejen skal understøtte håndtrykket og finde sig i en forbindelse, der måske er stille i flere minutter. Kører man mere end én server, opstår et spørgsmål, almindelig HTTP aldrig stiller: når noget sker på server A, hvordan når det så frem til en klient, hvis forbindelse lever på server B? Det sædvanlige svar er en fælles publish/subscribe-kanal, som alle serverne lytter på, og det er endnu et system at drive.

WebSocket er altså noget, man vælger til, ikke noget, man starter med. Når kun serveren har noget at fortælle, kan server-sent events, som er defineret i HTML-standarden, skubbe hændelser over et almindeligt HTTP-svar af typen

`text/event-stream`, og browseren forbinder selv igen, når svaret bliver afbrudt. WebSocket gør sig fortjent til sine ekstra bevægelige dele, når klienten har lige så meget at sige som serveren, som i chat, spil og samtidig redigering af det samme dokument.

Sådan ser det ud i praksis

Udvekslingen nedenfor åbner en chatforbindelse. Nøglen er eksempelværdien fra selve RFC 6455, og accept-værdien er den, standarden selv viser, og som enhver korrekt server skal sende tilbage for netop den nøgle.

```
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=
```

Efter den tomme linje, der afslutter serverens svar, er intet på forbindelsen HTTP længere. En browser klarer hele håndtrykket bag ét kald til konstruktøren og overlader resten til hændelseslyttere:

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

Lukkehændelsen fortæller, hvordan forbindelsen sluttede. Kode 1000 er en normal lukning, og 1001 betyder, at den ene part går sin vej, for eksempel en server, der lukker ned, eller en browser, der forlader siden. Kode 1006 betyder, at forbindelsen forsvandt helt uden en lukkeramme. Intet endpoint må sende den; den findes kun, så programmet ovenpå kan skelne en unormal afslutning fra en ren.

Grænsen mellem forespørgsel og svar på den ene side og en åben socket på den anden ses tydeligt i et projekt i sitets portefølje. Da et API flyttede over på Huma v2, kom alle ruter over på Humas model med forespørgsel og svar, undtagen præcis to: WebSocket-endpointene blev på rå Fiber, fordi koden dér reelt har brug for selve socketten, og fordi Humas model ikke passer. Et framework, der er bygget op om HTTP's egen form, beskriver og validerer forespørgsler og svar godt, og en langlivet tovejsforbindelse er simpelthen noget andet.

Hvor det går galt

En side på et andet site åbner en forbindelse med dine besøgendes cookies. Browseren sender håndtryksforespørgslen til serverens vært og kan medtage værtens cookies, afhængigt af browserens cookiepolitik. Hvor den politik tillader autentificerede forespørgsler på tværs af sites, kan en server, der autentificerer med cookies og ignorerer `Origin`, acceptere en forbindelse åbnet af en anden sides script i den besøgendes navn. RFC 6455 siger, at en server, der kun er beregnet til bestemte sites, bør tjekke `Origin` mod dem, den forventer, og afvise resten med 403 Forbidden. Gør det i håndtrykket, før forbindelsen accepteres.

Adgangstokenet ender i loggene. Browserens `WebSocket`-konstruktør tager en URL og en valgfri liste af subprotokoller og intet andet, så et script kan ikke tilføje et `Authorization`-felt. Den hurtige omvej er et token i URL'en, hvor reverse proxyer og adgangslogge skriver det ned. RFC'en lader autentificering stå åben og nævner cookies, HTTP-autentificering og TLS-autentificering som muligheder. En cookie kombineret med `Origin`-tjekket ovenfor virker, og det gør en kortlivet engangsbillet også, som siden henter over et autentificeret HTTP-kald og bruger på håndtrykket.

En genstart gør alle klienter til en stormflod. En brat udrulning eller genstart kan lukke alle forbindelser på én gang. Klienter, der ikke modtager nogen lukkeramme, melder kode 1006; hvis de alle forbinder igen med det samme og bliver ved, kommer serveren op til noget, som RFC'en sammenligner med et denial-of-service-angreb. RFC 6455 beder klienter vente et tilfældigt stykke tid før første forsøg, foreslår noget mellem 0 og 5 sekunder og beder dem derefter vente længere efter hver fejl, for eksempel med afkortet binær eksponentiel backoff. En server, der lukker sine forbindelser med 1001, før den stopper, fortæller i det mindste klienterne, at afbrydelsen 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, uden at nogen opdager det. En forbindelse uden trafik giver ingen af parterne tegn på, at vejen stadig findes, og en modpart, der forsvandt uden en lukkeramme, ligner præcis en stille forbindelse, indtil næste skrivning fejler. Det er det, ping og pong er til: et endpoint skal besvare et ping med et pong, medmindre det allerede har modtaget en lukkeramme. Er flere pings ubesvarede, må det nøjes med at besvare det seneste. RFC'en nævner, at et ping kan tjene som keepalive eller som et tjek af, at den anden side stadig svarer. Browserens API har ingen måde at sende et ping på, så timeren hører hjemme på serveren: ping hver forbindelse med et fast interval, og luk dem, der ikke svarer med pong.

Én besked æder serverens hukommelse. Længdefeltet i en rammes header giver plads til langt større indhold, end nogen server har hukommelse til, og RFC'en advarer om, at en fjendtlig modpart kan sende én enorm ramme eller en endeløs strøm af små fragmenter. Implementeringer skal beskytte sig selv, så sæt en maksimal beskedstørrelse i serverbiblioteket, og luk med kode 1009, message too big, når en modpart går over den. Det samme efterslæb kan hobe sig op den anden vej. I browseren fortæller `bufferedAmount`, hvor mange byte fra `send()` der stadig venter på at komme ud på netværket, og WHATWG's eksempel sender først den næste opdatering, når tallet er nul. På serveren fylder en klient, der læser langsomt, den udgående buffer på samme måde, så hver forbindelse har brug for et loft over den og en regel for, hvad der sker, når loftet er nået.

Læs mere

- RFC 6455: The WebSocket Protocol — håndtrykket, rammerne, maskeringen og lukkekoderne, og sikkerhedsbegrundelsen bag hver af dem.
- WebSockets Standard — browserens API: konstruktøren, `send()`, `close()`, `bufferedAmount` og lukkehændelsen.
- RFC 9110: HTTP Semantics — headerfeltet `Upgrade` og statuskoden 101 Switching Protocols, som håndtrykket bygger på.
- Server-sent events — HTML-standardens envejsalternativ med indbygget genforbindelse.

- RFC 8441: Bootstrapping WebSockets with HTTP/2 — extended CONNECT bærer WebSocket over én multiplekset HTTP/2-strøm.
- RFC 9220: Bootstrapping WebSockets with HTTP/3 — den tilsvarende extended-CONNECT-mekanisme til HTTP/3.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er WebSocket?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/websocket/