Engineer

WebSocket

↔ ett protokoll för en tvåvägsanslutning som hålls öppen

Vad det är

WebSocket är ett protokoll för en anslutning som båda parter kan skriva till när som helst. En HTTP-handskakning upprättar den, och den hålls öppen tills en part stänger den eller anslutningen bryts. IETF publicerade det som RFC 6455 i december 2011, och den del som en webbläsare gör tillgänglig för skript, objektet `WebSocket`, är definierad i WHATWG:s WebSockets Standard. I kod och dokumentation stavas namnet också ofta ”websocket” eller ”web socket”, och pluralformen WebSockets brukar syfta på tekniken som helhet.

I den klassiska HTTP/1.1-handskakningen börjar anslutningen med en vanlig `GET`. Klienten lägger till `Upgrade: websocket` och `Connection: Upgrade`, en `Sec-WebSocket-Version` på 13 och en `Sec-WebSocket-Key`, som är ett slumpmässigt värde på 16 byte kodat i base64. En server som säger ja svarar med statuskoden 101 Switching Protocols och huvudfältet `Sec-WebSocket-Accept`. Värdet bevisar att servern har läst just den här handskakningen och inte spelar upp ett gammalt svar: det är klientens nyckel med ett fast GUID tillagt, hashat med SHA-1 och kodat i base64. Från och med då är det slut med HTTP på anslutningen, och parterna skickar ramar i stället. HTTP/2 och HTTP/3 för i stället WebSocket över en ström med extended CONNECT, enligt RFC 8441 och RFC 9220.

En ram är liten. Ramhuvudet är minst två byte, växer med två eller åtta när innehållet är längre än 125 byte, och med ytterligare fyra på ramar från klienten, som bär en maskeringsnyckel. Varje ram har en opcode: text, binärdata eller fortsättning när ett meddelande är uppdelat på flera ramar, och därtill tre kontrollramar för stängning, ping och pong. Ett fullständigt textmeddelande måste vara giltig UTF-8; ett enskilt fragment kan dela en UTF-8-sekvens. Varje ram som en klient skickar maskeras med en ny slumpmässig 32-bitarsnyckel, och en server maskerar aldrig. Maskering är inte kryptering. Den finns för att ett fientligt skript inte ska kunna välja exakt vilka byte som korsar nätverket och använda dem för att förgifta cachen i en proxy någonstans längs vägen.

Adresserna börjar med `ws://` eller `wss://`, med port 80 och port 443 som standard, och `wss` kör hela utbytet inuti TLS. Det är samma portar som HTTP redan använder, och det är avsiktligt: för brandväggar och proxyer längs vägen ser handskakningen ut som vilken annan webbförfrågan som helst.

Använd begreppet i ett beslut

Använd ”Vad är WebSocket?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Den del användarna aldrig ser: där data lagras, vilka reglerna är och hur andra system får kommunicera med den.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Utformade och drev 6 ETL/ELT-pipelines — 6 ETL-pipelines över tre databasmotorer

Utformade och drev 6 ETL/ELT-pipelines Hör av er →

Varför det spelar roll

Med korta HTTP-svar talar servern bara när den blir tilltalad. Många produkter behöver det motsatta: ett chattmeddelande, ett pris som just har ändrats, en markör i ett delat dokument, ett bakgrundsjobb som har blivit klart. Utan en kanal som servern kan skriva till måste klienten polla. Den frågar med några sekunders mellanrum, de flesta svaren säger bara att ingenting har ändrats, varje svar bär en full uppsättning huvudfält, och en nyhet väntar upp till ett helt intervall innan någon ser den. En öppen WebSocket-anslutning tar bort båda kostnaderna. Servern kan skicka när något händer utan att vänta på nästa pollning. Varje meddelande behöver bara några få byte ramdata i stället för en fullständig uppsättning HTTP-huvudfält.

Priset är att anslutningen bär tillstånd. En HTTP-förfrågan kan hamna på vilken server som helst bakom en lastbalanserare, medan en WebSocket-anslutning hör till en enda process så länge den lever. Den processen binder minne för varje öppen anslutning, en driftsättning eller omstart bryter alla på en gång, och varje proxy och lastbalanserare längs vägen måste stödja handskakningen och tåla en anslutning som kanske är tyst i flera minuter. Kör man mer än en server uppstår en fråga som vanlig HTTP aldrig ställer: när något händer på server A, hur når det fram till en klient vars anslutning lever på server B? Det vanliga svaret är en gemensam publish/subscribe-kanal som alla servrar lyssnar på, och det är ännu ett system att hålla i drift.

WebSocket är alltså ett medvetet val, inte en utgångspunkt. När bara servern har något att berätta kan server-sent events, som definieras i HTML-standarden, skicka händelser över ett vanligt HTTP-svar av typen `text/event-stream`, och webbläsaren ansluter på nytt av sig själv när svaret bryts. WebSocket gör skäl för sina extra rörliga delar när klienten har lika mycket att säga som servern, som i chatt, spel och samtidig redigering av samma dokument.

Så ser det ut i praktiken

Utbytet nedan öppnar en chattanslutning. Nyckeln är exempelvärdet från själva RFC 6455, och värdet i `Sec-WebSocket-Accept` är det som standarden själv visar och som varje korrekt server måste skicka tillbaka för just den nyckeln.

```
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 tomma raden som avslutar serverns svar är ingenting på anslutningen HTTP längre. En webbläsare sköter hela handskakningen bakom ett enda anrop till konstruktorn och lämnar resten åt händelselyssnare:

```
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);
});
```

Stängningshändelsen berättar hur anslutningen slutade. Kod 1000 är en normal stängning, och 1001 betyder att ena parten går därifrån, till exempel en server som stängs av eller en webbläsare som lämnar sidan. Kod 1006 betyder att

anslutningen försvann helt utan någon stängningsram. Ingen av parterna får skicka den; den finns bara för att programmet ovanpå ska kunna skilja ett onormalt slut från ett rent.

Gränsen mellan förfrågan och svar å ena sidan och en öppen socket å den andra syns tydligt i ett projekt i webbplatsens portfölj. När ett API flyttades över till Huma v2, gick de befintliga rutterna över till Humas modell med anrop och svar, utom exakt två: slutpunkterna för WebSocket blev kvar på ren Fiber, eftersom koden där verkligen vill åt själva socketen och Humas modell inte passar. Ett ramverk som är byggt kring HTTP:s egen form beskriver och validerar förfrågningar och svar väl, och en långlivad tvåvägsanslutning är helt enkelt något annat.

Var det går fel

En sida på en annan webbplats öppnar en anslutning med dina besökares kakor. Webbläsaren skickar handskakningsförfrågan till serverns värd och kan ta med värdens kakor enligt webbläsarens kakpolicy. Där policyn tillåter autentiserade förfrågningar mellan webbplatser kan en server som autentiserar med kakor och ignorerar `Origin` godta en anslutning som en sida på en annan webbplats öppnar i besökarens namn. RFC 6455 säger att en server som bara är avsedd för vissa webbplatser bör kontrollera `Origin` mot dem den väntar sig och avvisa resten med 403 Forbidden. Gör det i handskakningen, innan anslutningen godtas.

En åtkomsttoken hamnar i loggarna. Webbläsarens `WebSocket` -konstruktor tar en URL och en valfri lista med subprotokoll och inget annat, så ett skript kan inte lägga till ett `Authorization` -fält. Den snabba genvägen är en token i URL:en, där en omvänd proxy och åtkomstloggar skriver ned den. RFC:n lämnar autentiseringen öppen och nämner kakor, HTTP-autentisering och TLS-autentisering som möjligheter. En kaka tillsammans med `Origin` -kontrollen ovan fungerar, och det gör även en kortlivad engångsbiljett som sidan hämtar via ett autentiserat HTTP-anrop och använder i handskakningen.

En omstart gör alla klienter till en flodvåg. En abrupt driftsättning eller omstart kan stänga alla anslutningar på en gång. Klienter som inte får någon stängningsram rapporterar kod 1006; om alla ansluter igen direkt och fortsätter försöka kommer servern upp till något som RFC:n liknar vid en överbelastningsattack. RFC 6455 ber klienterna vänta en slumpmässig tid före första försöket, föreslår något mellan 0 och 5 sekunder och ber dem sedan vänta

längre efter varje misslyckande, till exempel med avkortad binär exponentiell backoff. En server som stänger sina anslutningar med 1001 innan den stoppar talar åtminstone om för klienterna att avbrottet var planerat.

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

Anslutningar dör utan att någon märker det. En anslutning utan trafik ger ingen av parterna något tecken på att vägen fortfarande finns, och en motpart som försvann utan en stängningsram ser precis ut som en tyst anslutning ända tills nästa skrivning misslyckas. Det är det ping och pong är till för: en slutpunkt måste svara på en ping med en pong om den inte redan har fått en stängningsram. Om flera ping är obesvarade får den nöja sig med att svara på den senaste. RFC:n nämner att en ping kan fungera som keepalive eller som en kontroll av att andra sidan fortfarande svarar. Webbläsarens API har inget sätt att skicka en ping, så timern hör hemma på servern: pinga varje anslutning med ett fast intervall och stäng dem som inte svarar med pong.

Ett meddelande äter upp serverns minne. Längdfältet i ett ramhuvud ger plats för innehåll som är långt större än någon server har minne till, och RFC:n varnar för att en fientlig motpart kan skicka en enda enorm ram eller en ändlös rad små fragment. Implementationer måste skydda sig själva, så sätt en största tillåtna meddelandestorlek i serverbiblioteket och stäng med kod 1009, message too big, när en motpart går över den. Samma eftersläpning kan byggas upp åt andra hållet. I webbläsaren visar `bufferedAmount` hur många byte från `send()` som fortfarande väntar på att komma ut på nätverket, och WHATWG:s exempel skickar nästa uppdatering först när talet är noll. På servern fyller en klient som läser långsamt den utgående bufferten på samma sätt, så varje anslutning behöver ett tak för den och en regel för vad som ska hända när taket är nått.

Läs mer

- RFC 6455: The WebSocket Protocol — handskakningen, ramarna, maskeringen och stängningskoderna, och säkerhetsresonemanget bakom var och en av dem.
- WebSockets Standard — webbläsarens API: konstruktorn, `send()`, `close()`, `bufferedAmount` och stängningshändelsen.

- RFC 9110: HTTP Semantics — huvudfältet `Upgrade` och statuskoden 101 Switching Protocols, som handskakningen bygger på.
- Server-sent events — HTML-standardens envägsalternativ med inbyggd återanslutning.
- RFC 8441: Bootstrapping WebSockets with HTTP/2 — extended CONNECT för WebSocket över en multiplexerad HTTP/2-ström.
- RFC 9220: Bootstrapping WebSockets with HTTP/3 — motsvarande extended-CONNECT-mekanism för HTTP/3.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är WebSocket?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/websocket/