Engineer

WebSocket

↔ ein Protokoll für dauerhaft offene Zweiwegeverbindungen

Was es ist

WebSocket ist ein Protokoll für eine Verbindung, auf der beide Seiten jederzeit Daten senden können. Ein HTTP-Handshake stellt sie her. Sie bleibt offen, bis eine Seite sie schließt oder die Verbindung ausfällt. Die IETF veröffentlichte das Protokoll im Dezember 2011 als RFC 6455. Den Teil, den ein Browser Skripten bereitstellt, das `WebSocket`-Objekt, definiert der WebSockets-Standard der WHATWG. In Code und Dokumentation stehen häufig auch die Schreibweisen „websocket“ und „web socket“. Der Plural WebSockets bezeichnet meist die Technologie insgesamt.

Beim klassischen HTTP/1.1-Handshake beginnt die Verbindung mit einem gewöhnlichen `GET`. Der Client ergänzt `Upgrade: websocket` und `Connection: Upgrade`, eine `Sec-WebSocket-Version` von 13 und einen `Sec-WebSocket-Key`: einen zufälligen, in base64 kodierten 16-Byte-Wert. Ein Server, der den Wechsel akzeptiert, antwortet mit dem Status 101 Switching Protocols und einem `Sec-WebSocket-Accept`-Header. Dessen Wert belegt, dass der Server genau diesen Handshake gelesen hat, statt eine alte Antwort erneut abzuspielen: An den Schlüssel des Clients wird eine feste GUID angehängt, das Ergebnis mit SHA-1 gehasht und in base64 kodiert. Ab dann endet HTTP auf dieser Verbindung; beide Seiten tauschen stattdessen Frames aus. HTTP/2 und HTTP/3 übertragen WebSocket hingegen über einen einzelnen Stream mit erweitertem CONNECT, wie RFC 8441 und RFC 9220 es festlegen.

Ein Frame ist klein. Sein Header beginnt bei zwei Bytes. Ist die Nutzlast länger als 125 Bytes, kommen zwei oder acht Bytes hinzu. Bei Frames vom Client kommen weitere vier Bytes für einen Maskierungsschlüssel dazu. Jeder Frame trägt einen Opcode: Text, Binärdaten oder Fortsetzung für eine über mehrere Frames verteilte Nachricht. Dazu kommen drei Steuerframes für Schließen, Ping und Pong. Eine vollständige Textnachricht muss gültiges UTF-8 sein; ein einzelnes Fragment darf eine UTF-8-Sequenz aufteilen. Jeden gesendeten Frame maskiert

der Client mit einem neuen zufälligen 32-Bit-Schlüssel. Ein Server maskiert nie. Maskierung ist keine Verschlüsselung. Sie verhindert, dass ein bösartiges Skript die genauen über das Netz übertragenen Bytes festlegt und damit den Zwischenspeicher eines Proxys auf dem Weg manipuliert.

Adressen verwenden die Schemata `ws://` und `wss://`, mit Port 80 und Port 443 als Standardwerten. Bei `wss` läuft der gesamte Austausch innerhalb von TLS. Dass dies die schon von HTTP verwendeten Ports sind, ist Absicht: Für Firewalls und Proxys auf dem Weg sieht der Handshake wie jede andere Webanfrage aus.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist WebSocket?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Bei kurzen HTTP-Antworten spricht der Server nur, wenn er angesprochen wird. Viele Produkte brauchen das Gegenteil: eine Chatnachricht, einen gerade geänderten Preis, einen Cursor in einem gemeinsam bearbeiteten Dokument oder einen abgeschlossenen Hintergrundauftrag. Ohne einen Kanal, auf dem der Server von sich aus senden kann, muss der Client regelmäßig nachfragen. Er fragt alle paar Sekunden; die meisten Antworten besagen nur, dass sich nichts geändert hat. Jede bringt einen vollständigen Satz Header mit, und Neuigkeiten warten bis zu einem Abfrageintervall, bevor jemand sie sieht. Eine offene WebSocket-Verbindung beseitigt beide Kosten. Der Server kann senden, sobald etwas passiert, ohne die nächste Abfrage abzuwarten. Jede Nachricht braucht nur wenige Bytes zur Rahmung statt eines vollständigen Satzes HTTP-Header.

Der Preis ist der Verbindungszustand. Eine HTTP-Anfrage kann bei jedem Server hinter einem Lastverteiler landen. Eine WebSocket-Verbindung gehört dagegen für ihre gesamte Lebensdauer zu einem Prozess. Dieser hält für jede offene Verbindung Speicher vor. Ein Deployment oder Neustart trennt sie alle auf einmal, und jeder Proxy und Lastverteiler auf dem Weg muss den Handshake unterstützen und eine möglicherweise minutenlang stille Verbindung dulden.

Mit mehr als einem Server stellt sich eine Frage, die gewöhnliches HTTP nicht kennt: Wie erreicht ein Ereignis auf Server A einen Client, dessen Verbindung auf Server B liegt? Die übliche Antwort ist ein gemeinsamer Publish-Subscribe-Kanal, den alle Server abonnieren. Damit kommt ein weiteres zu betreibendes System hinzu.

WebSocket ist deshalb eine bewusste Wahl, keine Standardeinstellung. Hat nur der Server Neuigkeiten zu senden, übertragen die im HTML-Standard definierten Server-Sent Events Ereignisse über eine gewöhnliche HTTP-Antwort vom Typ `text/event-stream` . Bricht die Antwort ab, verbindet sich der Browser von selbst neu. Die zusätzlichen Bestandteile von WebSocket lohnen sich, wenn der Client ebenso viel mitzuteilen hat wie der Server, etwa bei Chats, Spielen und der gemeinsamen Bearbeitung von Dokumenten.

Wie es in der Praxis aussieht

Der folgende Austausch öffnet eine Chatverbindung. Der Schlüssel ist der Beispielwert aus RFC 6455 selbst. Auch der Accept-Wert stammt aus dem Standard; jeder korrekte Server muss ihn für diesen Schlüssel zurückgeben.

```
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=
```

Nach der Leerzeile, die die Serverantwort beendet, ist nichts auf der Verbindung mehr HTTP. Ein Browser erledigt den gesamten Handshake mit einem einzigen Konstruktoraufruf und überlässt den Rest den Ereignisbehandlern:

```
const socket = new WebSocket("wss://example.com/chat");
socket.addEventListener("open", () => socket.send("hello"));
socket.addEventListener("message", (event) => console.log(event.data));
```

```
socket.addEventListener("close", (event) => {
  console.log(event.code, event.reason, event.wasClean);
});
```

Das Schließereignis meldet, wie die Verbindung endete. Code 1000 steht für ein normales Schließen. 1001 bedeutet, dass eine Seite geht, etwa weil ein Server herunterfährt oder der Browser die Seite verlässt. Code 1006 bedeutet, dass die Verbindung ohne jeglichen Schließframe abbrach. Kein Endpunkt darf ihn senden. Er dient nur dazu, dass das darüberliegende Programm einen ungewöhnlichen Abbruch von einem sauberen Ende unterscheiden kann.

Der Unterschied zwischen Anfrage und Antwort einerseits und einem Socket andererseits zeigt sich deutlich in einem Projekt dieses Portfolios. Beim Umstieg einer API auf Huma v2 wurden alle Routen bis auf genau zwei auf Humas Anfrage- und Antwortmodell umgestellt. Die WebSocket-Endpunkte blieben direkt auf Fiber, weil der Code dort tatsächlich den Socket selbst braucht und Humas Modell nicht passt. Ein am Aufbau von HTTP ausgerichtetes Framework beschreibt und validiert Anfragen und Antworten gut. Eine langlebige Verbindung in beide Richtungen ist schlicht etwas anderes.

Wo es schiefgeht

Eine fremde Seite öffnet eine Verbindung mit den Cookies Ihrer Besucher. Der Browser sendet die Handshake-Anfrage an den Host des Servers und kann dabei nach Maßgabe seiner Cookie-Regeln dessen Cookies mitsenden. Erlauben diese Regeln authentifizierte Anfragen von fremden Websites, kann ein per Cookie authentifizierender Server, der den `Origin` -Header ignoriert, eine von einer fremden Seite geöffnete Verbindung als die des Besuchers akzeptieren. Laut RFC 6455 sollte ein nur für bestimmte Websites vorgesehener Server `Origin` mit den erwarteten Websites abgleichen und alle anderen mit 403 Forbidden abweisen. Prüfen Sie dies während des Handshakes, bevor Sie die Verbindung annehmen.

Das Zugriffstoken landet in den Protokollen. Der `WebSocket` -Konstruktor des Browsers nimmt eine URL und eine optionale Liste von Unterprotokollen entgegen, sonst nichts. Ein Skript kann daher keinen `Authorization` -Header ergänzen. Der schnelle Umweg ist ein Token in der URL, das Reverse-Proxys und Zugriffsprotokolle mitschreiben. Der RFC lässt die Authentifizierung offen und nennt Cookies, HTTP- und TLS-Authentifizierung als Möglichkeiten. Ein Cookie

mit der oben beschriebenen `Origin`-Prüfung funktioniert ebenso wie ein kurzlebiges, nur einmal verwendbares Ticket, das die Seite über einen authentifizierten HTTP-Aufruf abruft und beim Handshake einlöst.

Ein Neustart lässt sämtliche Clients den Server überfluten. Ein abruptes Deployment oder ein Neustart kann alle Verbindungen zugleich schließen. Clients, die keinen Schließframe erhalten, melden Code 1006. Verbinden sich alle sofort neu und versuchen es immer weiter, wird der wieder startende Server mit etwas konfrontiert, das der RFC mit einem Denial-of-Service-Angriff vergleicht. RFC 6455 fordert Clients auf, vor dem ersten Versuch eine zufällige Zeit zu warten, und schlägt zwischen 0 und 5 Sekunden vor. Nach jedem Fehlschlag sollen sie länger warten, etwa mit begrenztem binärem exponentiellem Backoff. Ein Server, der seine Verbindungen vor dem Anhalten mit 1001 schließt, teilt den Clients zumindest mit, dass die Unterbrechung geplant war.

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

Verbindungen brechen ab, und niemand merkt es. Eine Verbindung ohne Verkehr gibt keiner Seite ein Zeichen, dass der Weg noch besteht. Eine ohne Schließframe verschwundene Gegenstelle sieht bis zum nächsten fehlgeschlagenen Schreibzugriff genauso aus wie eine stille. Dafür gibt es Ping und Pong: Ein Endpunkt muss auf einen Ping mit einem Pong antworten, sofern er nicht bereits einen Schließframe erhalten hat. Sind mehrere Pings unbeantwortet, darf er nur auf den jüngsten antworten. Der RFC hält fest, dass ein Ping als Lebenszeichen oder zur Prüfung dienen kann, ob die Gegenstelle noch reagiert. Die Browser-API bietet keine Möglichkeit, einen Ping zu senden. Der Timer gehört daher auf den Server: Pingen Sie jede Verbindung in festen Abständen und schließen Sie diejenigen, deren Pong ausbleibt.

Eine Nachricht verbraucht den gesamten Serverspeicher. Das Längenfeld im Frame-Header lässt Nutzlasten zu, die viel größer sind als der Speicher eines Servers. Der RFC warnt vor einer bösartigen Gegenstelle, die einen riesigen Frame oder eine endlose Folge kleiner Fragmente sendet. Implementierungen müssen sich schützen. Setzen Sie deshalb in der Serverbibliothek eine maximale Nachrichtengröße und schließen Sie bei Überschreitung mit Code 1009, Nachricht zu groß. Derselbe Rückstau entsteht in der Gegenrichtung. Im Browser

meldet `bufferedAmount` die Bytes, die `send()` eingereiht hat und die noch auf ihre Übertragung warten. Das WHATWG-Beispiel sendet die nächste Aktualisierung erst, wenn diese Zahl null ist. Auf dem Server füllt ein langsam lesender Client den Ausgabepuffer ebenso. Jede Verbindung braucht daher eine Obergrenze dafür und eine Regel, was beim Erreichen dieser Grenze geschieht.

Weiterführende Lektüre

- RFC 6455: The WebSocket Protocol — Handshake, Rahmung, Maskierung und Schließcodes sowie die Sicherheitsgründe hinter jeder dieser Entscheidungen.
- WebSockets-Standard — die Browser-API: Konstruktor, `send()`, `close()`, `bufferedAmount` und das Schließereignis.
- RFC 9110: HTTP Semantics — das Headerfeld `Upgrade` und der Status 101 Switching Protocols, auf denen der Handshake beruht.
- Server-Sent Events — die einseitige Alternative des HTML-Standards mit eingebautem Neuverbinden.
- RFC 8441: Bootstrapping WebSockets with HTTP/2 — erweitertes CONNECT überträgt WebSocket auf einem multiplexierten HTTP/2-Stream.
- RFC 9220: Bootstrapping WebSockets with HTTP/3 — der entsprechende Mechanismus mit erweitertem CONNECT für HTTP/3.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist WebSocket?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/websocket/