Engineer

HTTP

⇄ Anfrage-und-Antwort-Protokoll als Grundlage des Webs

Was es ist

HTTP, das Hypertext Transfer Protocol, ist das Regelwerk, nach dem ein Browser, eine App oder ein Skript etwas von einem Server anfordert und eine Antwort erhält. Läuft dieses Gespräch innerhalb einer verschlüsselten TLS-Verbindung, heißt dasselbe Protokoll HTTPS. Jeder Austausch besteht aus zwei Hälften. Der Client sendet eine Anfrage: eine Methode wie `GET` oder `POST`, das gewünschte Ziel, eine Liste von Headerfeldern und manchmal einen Body. Der Server antwortet mit einem dreistelligen Statuscode, eigenen Headerfeldern und meist einem Body. Eine Seite laden, ein Bild abrufen und ein Programm ein anderes aufrufen lassen sind Wiederholungen desselben Austauschs.

```
GET /pricing/ HTTP/1.1
Host: example.com
Accept: text/html

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=600
```

Die Spezifikation nennt es zustandslos. Jede Anfrage muss alles enthalten, was der Server zum Verstehen benötigt; das Protokoll bewahrt keine Erinnerung von einem Austausch zum nächsten. Anmeldungen, Warenkörbe und Sitzungen werden darauf aufgebaut, meist mit Cookies: einem Headerfeld, das der Server einmal setzt und der Client mit jeder späteren Anfrage zurücksendet.

Das Vokabular ist klein und präzise. `GET` und `HEAD` sind sicher: Sie fordern eine Repräsentation an und sollen nichts verändern. `PUT` und `DELETE` sind idempotent: Zweimal dieselbe Anfrage zu senden hinterlässt den Server in demselben Zustand wie einmaliges Senden. `POST` verspricht keines von beidem. Statuscodes sind nach ihrer ersten Ziffer gruppiert: 2xx für Erfolg, 3xx für

Weiterleitung, 4xx für ein Problem mit der Anfrage und 5xx für ein Problem auf dem Server. So weiß ein Client auch bei einem unbekannten Code ungefähr, was passiert ist.

Diese Bedeutung ist gleich geblieben, während das Übertragungsformat zweimal ersetzt wurde. HTTP/1.1 schreibt Nachrichten als Textzeilen über eine TCP-Verbindung; jede Verbindung liefert ihre Antworten nacheinander in der richtigen Reihenfolge. HTTP/2 behält dieselben Methoden und Felder, verpackt sie aber in binäre Frames und überträgt Teile vieler Anfragen abwechselnd über eine Verbindung. HTTP/3 überträgt dieselbe Semantik über QUIC, einen auf UDP aufbauenden Transport, der TCPs Head-of-Line-Blocking auf der Transportebene zwischen unabhängigen Streams vermeidet. Seit 2022 veröffentlicht die IETF die Semantik in einem Dokument und das Übertragungsformat jeder Version in einem anderen. Deshalb haben „Was bedeutet 404?“ und „Wie sind die Bytes angeordnet?“ heute getrennte Antworten.

HTTP gegenüber HTTPS ist daher kein Wettbewerb zweier Protokolle: HTTPS ist HTTP mit TLS darunter, das den Austausch verschlüsselt und den Namen des Servers bestätigt. WebSocket beginnt als HTTP-Anfrage und wandelt die Verbindung dann in einen Kanal in beide Richtungen um. gRPC ist eine Konvention für Aufrufe entfernter Funktionen über HTTP/2.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist HTTP?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. WLAN, Firewalls und Fernzugriff werden so eingerichtet, dass Mitarbeitende von überall arbeiten können und nur Berechtigte Zugang erhalten.

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. 30 Ubuntu-Linux-VPS-Instanzen verwaltet — 30 verwaltete Linux-Server mit geplantem Wiederherstellungsweg

30 Ubuntu-Linux-VPS-Instanzen verwaltet Kontakt aufnehmen →

Warum es wichtig ist

Fast jedes System, das über das Internet mit einem anderen kommuniziert, spricht bereits HTTP. Die entscheidende Frage ist deshalb selten, ob es genutzt werden soll, sondern ob man seine Vorgaben nutzt oder es als reinen Übertragungskanal behandelt und den Rest darauf neu aufbaut.

Es vollständig zu nutzen lohnt sich, weil eine lange Kette von Software das Vokabular bereits liest. Eine Clientbibliothek kann ein `GET` nach einer Zeitüberschreitung wiederholen; `POST` darf sie dagegen nicht wiederholen; das gilt auch für jeden Proxy dazwischen. Ein Cache kann eine mit `max-age=600` markierte Antwort zehn Minuten behalten und dem nächsten Leser liefern, ohne den Server zu fragen. Die Überwachung zählt 5xx-Antworten als Fehler, ohne eigens darauf hingewiesen zu werden. Eine API, die jeden Aufruf als `POST` an eine einzige Adresse sendet und Fehler innerhalb einer 200-Antwort meldet, gibt all das auf. Jeder Client muss dann die besonderen Regeln dieser API lernen.

Das Modell hat Grenzen. Anfrage und Antwort passen zu einem Client, der weiß, was er will, deutlich weniger zu einem Server, der Neuigkeiten senden möchte. Deshalb laufen Liveaktualisierungen meist über WebSocket, Server-Sent Events oder Polling.

Unverschlüsseltes HTTP im öffentlichen Internet ist eine andere Frage mit kurzer Antwort. Browser kennzeichnen es als unsicher, jeder auf dem Übertragungsweg kann es lesen und verändern, und Zertifikate kosten nichts. Öffentlicher Verkehr gehört deshalb auf HTTPS. Die Wahl zwischen HTTP/1.1, HTTP/2 und HTTP/3 ist meist eine Einstellung auf dem Server oder einer vorgeschalteten Edge-Komponente. Der Anwendungscode bleibt gleich, weil die Bedeutung jeder Anfrage gleich bleibt.

Wie es in der Praxis aussieht

Das deutlichste Beispiel auf dieser Website ist eine API, die auf ein an HTTP selbst ausgerichtetes Framework umgestellt wurde. Die HTTP-API wechselte von Fiber zu Huma v2, das auf Fiber aufsetzt. Jeder Endpunkt wird dadurch mit Eingabe- und Ausgabe-Structs deklariert, und die Validierung ergibt sich aus den Typen statt aus handgeschriebenen Prüfungen. Dieselben Deklarationen erzeugen eine OpenAPI-Beschreibung. Eine CI-Prüfung vergleicht die vom Server registrierten Routen mit den in der Beschreibung angegebenen. Bei 649 Pfaden und 760 Operationen beträgt die Übereinstimmung 100 %. Sie bleibt erhalten, weil eine nirgends beschriebene Route den Build scheitern lässt. Genau zwei Endpunkte blieben direkt auf Fiber: die WebSocket-Endpunkte, für die Humas Anfrage-Antwort-Modell nicht passt.

Die Website selbst nutzt Inhaltsaushandlung. Jede Seite erscheint als HTML und Markdown. Eine Möglichkeit, Markdown zu erhalten, ist, eine Anfrage mit `Accept: text/markdown` zu senden und den Server wählen zu lassen. Die dortige

Onion-Adresse verwendet bewusst unverschlüsseltes `http://`. Da diese Adresse den öffentlichen Schlüssel des Dienstes kodiert, hat Tor den Verbindungsweg bereits authentifiziert und verschlüsselt, bevor die erste HTTP-Anfrage darüber läuft.

HTTP dient auch als Werkzeug im Infrastrukturcode. Als ein zweiter Server bereitgestellt wurde, wurden die Firewalls beider Anbieter direkt über ihre REST-APIs beschrieben, mit einer allgemeinen HTTP-Aufgabe statt einer anbieterspezifischen Collection. Das beseitigte eine Abhängigkeit und deren Versionsdrift, verlangte aber, jede Anfragestruktur von Hand zu schreiben. Die Seite nennt einen weiteren Nachteil: Die allgemeine HTTP-Aufgabe deklariert keine Unterstützung für den Check-Modus. Ein Probelauf des Plays, das die Maschine erstellt, überspringt deshalb jede Aufgabe darin; der Probelauf erfolgt stattdessen mit dem Firewall-Play.

Wo es schiefgeht

Die meisten HTTP-Fehler entstehen, wenn ein Wort aus dem Vokabular etwas anderes bedeuten soll, als der Rest der Kette darunter versteht.

Ein Fehler kommt als Erfolg an

Eine API antwortet mit 200 und schreibt `"error": "card declined"` in den Body. Caches speichern das womöglich, Dashboards zählen es als Erfolg, und die Wiederholungslogik des Clients springt nie an. Senden Sie den passenden Statuscode, 4xx oder 5xx, und beschreiben Sie das Problem im Body. RFC 9457 definiert dafür eine JSON-Struktur mit dem Medientyp `application/problem+json`.

Ein Link löscht etwas

Eine `GET`-Anfrage verändert den Zustand, etwa ein Link zum Abbestellen oder Löschen in einer E-Mail. Crawler, Linkvorschauen in Chatapps, Sicherheitsscanner und das Vorabladen des Browsers folgen Links selbstständig; jeder dieser Mechanismen kann die Aktion auslösen. Alles, was Daten ändert, gehört hinter `POST`, `PUT` oder `DELETE`, mit einer Bestätigungsseite, falls der Link existieren muss.

Eine Zahlung wird zweimal eingezogen

Der Client sendet ein `POST`, die Verbindung bricht vor Eintreffen der Antwort ab, und er versucht es erneut. Der Server hatte die erste Anfrage bereits verarbeitet. HTTP gibt dem Client keine Möglichkeit, diese Fälle

auseinanderzuhalten; die API muss deshalb damit umgehen: Akzeptieren Sie einen Idempotenzschlüssel in einem Header und geben Sie bei wiederholtem Schlüssel das gespeicherte Ergebnis zurück. Oder lassen Sie den Client die Kennung der neuen Ressource wählen und diese mit `PUT` erstellen.

Ein Formular wird zu einem GET

Ein Server beantwortet ein `POST` mit 301 oder 302, um es woandershin zu leiten. Aus historischen Gründen erlaubt die Spezifikation User Agents bei diesen beiden Codes, die Methode zu `GET` zu ändern. Viele tun das, sodass der Body unterwegs verloren geht. Muss die Methode die Weiterleitung überstehen, verwenden Sie 307 oder 308.

Die falsche Variante aus einem Cache

Eine URL liefert je nach Anfrageheadern HTML oder Markdown, Englisch oder Dänisch. Ein nicht darüber informierter Cache speichert die zuerst gesehene Variante und reicht sie an den nächsten Leser weiter. Die Antwort muss die Felder nennen, von denen sie abhängt, etwa `Vary: Accept`. Dann hält der Cache eine Kopie pro Variante statt eine pro URL.

Weiterführende Lektüre

- RFC 9110: HTTP-Semantik — Methoden, Statuscodes und Headerfelder, einmal für alle Versionen definiert.
- RFC 9111: HTTP-Caching — Wann eine gespeicherte Antwort wiederverwendet werden darf und was `Cache-Control` einem Cache vorgibt.
- RFC 9112: HTTP/1.1 — Textbasiertes Übertragungsformat, Verbindungsverwaltung und Nachrichtenrahmung.
- RFC 9113: HTTP/2 — Binäre Frames und viele Streams auf einer Verbindung.
- RFC 9114: HTTP/3 — Dieselbe Semantik, abgebildet auf QUIC-Streams.
- RFC 9457: Problemdetails für HTTP-APIs — Ein standardisierter JSON-Body zur Beschreibung dessen, was schiefging.



Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist HTTP?“ 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/http/