Engineer

HTTP

⇄ den begäran och det svar som webben bygger på

Vad det är

HTTP, Hypertext Transfer Protocol, är de regler som en webbläsare, en app eller ett skript följer när den ber en server om något och tar emot ett svar. Går samtalet inuti en krypterad TLS-anslutning heter samma protokoll HTTPS. Varje utbyte har två halvor. Klienten skickar en begäran: en metod som `GET` eller `POST`, det mål den vill åt, en lista med huvudfält och ibland en kropp. Servern svarar med en statuskod på tre siffror, egna huvudfält och oftast en kropp. Att ladda en sida, hämta en bild och låta ett program anropa ett annat är alla samma utbyte, upprepat.

```
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
```

Specifikationen kallar protokollet tillståndslöst. Varje begäran måste bära allt som servern behöver för att förstå den, och protokollet minns ingenting från ett utbyte till nästa. Inloggningar, varukorgar och sessioner byggs ovanpå, oftast med kakor: ett huvudfält som servern sätter en gång och som klienten skickar tillbaka i varje senare begäran.

Ordförrådet är litet och exakt. `GET` och `HEAD` är säkra, vilket betyder att de ber om en representation och förväntas lämna allt orört. `PUT` och `DELETE` är idempotenta: att skicka samma begäran två gånger lämnar servern där en enda gång hade lämnat den. `POST` lovar ingetdera. Statuskoderna grupperas efter

första siffran, 2xx för framgång, 3xx för omdirigering, 4xx för ett problem med begäran och 5xx för ett problem på servern, så en klient som möter en okänd kod vet ändå ungefär vad som hände.

Den betydelsen har legat fast medan överföringsformatet har bytts ut två gånger. HTTP/1.1 skriver meddelanden som textrader över en TCP-anslutning, och varje anslutning lämnar sina svar ett i taget, i ordning. HTTP/2 behåller samma metoder och fält men ramar in dem binärt och flätar ihop många begäranden på en anslutning. HTTP/3 förmedlar samma semantik över QUIC, en transport byggd på UDP, som undviker TCP:s blockering längst fram i kön på transportlagret mellan oberoende strömmar. Sedan 2022 publicerar IETF semantiken i ett dokument och varje versions överföringsformat i ett annat, och därför har frågan om vad 404 betyder och frågan om hur byteföljden är uppbyggd numera var sitt svar.

HTTP mot HTTPS är alltså ingen tävling mellan två protokoll: HTTPS är HTTP med TLS under, som krypterar utbytet och bevisar serverns namn. WebSocket börjar som en HTTP-begäran och uppgraderar sedan anslutningen till en kanal åt båda hållen, och gRPC är en konvention för att anropa fjärrfunktioner som körs över HTTP/2.

Använd begreppet i ett beslut

Använd ”Vad är HTTP?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Wi-Fi, brandväggar och fjärråtkomst uppsatta så att personalen kan arbeta varifrån som helst och bara rätt personer kommer in.

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. Förvaltade 30 VPS-instanser med Ubuntu Linux — 30 förvaltade Linux-servrar, med en plan för återställning

Förvaltade 30 VPS-instanser med Ubuntu Linux Hör av er →

Varför det spelar roll

Nästan varje system som talar med ett annat över internet talar redan HTTP, så det beslut som räknas handlar sällan om huruvida ni ska använda det. Det handlar om ni använder det som HTTP definierar, eller behandlar det som ett rör och bygger upp resten på nytt ovanpå.

Att använda protokollet fullt ut lönar sig, eftersom en lång kedja av mjukvara redan läser ordförrådet. Ett klientbibliotek får försöka igen med en `GET` som tog för lång tid; en `POST` måste det låta bli, och detsamma gäller varje proxy på

vägen. En cache kan spara ett svar märkt `max-age=600` i tio minuter och lämna det till nästa läsare utan att fråga servern. Övervakning räknar 5xx-svar som fel utan att någon säger åt den. Ett API som skickar varje anrop som en `POST` till en enda adress och rapporterar fel inuti ett 200-svar avstår från allt det, och varje klient måste då lära sig just det API:ets privata regler.

Modellen har sina gränser. Begäran och svar passar en klient som vet vad den vill ha; de passar betydligt sämre en server som har nyheter att skicka ut, och därför färdas liveuppdateringar oftast över WebSocket, server-sent events eller upprepad avfrågning.

Okrypterad HTTP på det publika internet är en annan fråga med ett kort svar. Webbläsare märker den som osäker, vem som helst på vägen kan läsa och ändra den, och certifikat kostar ingenting, så publik trafik hör hemma på HTTPS. Valet mellan HTTP/1.1, HTTP/2 och HTTP/3 är mest en inställning på servern eller på kanten framför den, och applikationskoden förblir densamma eftersom varje begäran betyder samma sak.

Så ser det ut i praktiken

Det tydligaste fallet på den här webbplatsen är ett API som flyttades till ett ramverk byggt kring HTTP:s egen form. HTTP-API:t flyttades från Fiber till Huma v2, som ligger ovanpå Fiber. Varje slutpunkt deklareras med structar för indata och utdata, och valideringen följer av typerna i stället för av kontroller som skrivits för hand. En OpenAPI-beskrivning kommer på köpet ur samma deklarationer, och en kontroll i CI jämför de rutter som servern registrerar med dem som beskrivningen utlovar. Vid 649 sökvägar och 760 operationer är pariteten 100 %, och den ligger kvar där eftersom en rutt som saknar beskrivning stoppar bygget. Exakt två slutpunkter blev kvar på ren Fiber: de för WebSocket, där Humas modell med begäran och svar inte passar.

Webbplatsen själv använder innehållsförhandling. Varje sida publiceras både som HTML och som Markdown, och ett sätt att få Markdown är att skicka en begäran med `Accept: text/markdown` och låta servern välja. Samma sida förklarar varför onion-adressen med avsikt är vanlig `http://` : adressen innehåller tjänstens publika nyckel i kodad form, så kretsen är autentiserad och krypterad innan något HTTP alls talas.

HTTP dyker också upp som verktyg i infrastrukturkod. När en andra server provisionerades skrevs båda leverantörernas brandväggar direkt mot deras REST-API:er genom en generisk HTTP-task i stället för en collection från leverantören. Det tog bort ett beroende och dess versionsglidning, till priset av att varje begärans form skrivs för hand. Det har ytterligare en kostnad som sidan namnger: den generiska HTTP-tasken deklarerar inget stöd för check mode, så en torrkörning av den play som skapar maskinen hoppar över varje task i den, och provkörningen görs i stället mot brandväggens play.

Var det går fel

De flesta HTTP-buggar uppstår när ett ord i ordförrådet används för att betyda något annat än det som resten av kedjan hör.

Ett fel som kommer fram som en framgång

Ett API svarar 200 och lägger `"error": "card declined"` i kroppen. En cache kan spara svaret, instrumentpanelerna räknar det som en framgång, och klientens logik för nya försök startar aldrig. Skicka den statuskod som stämmer, 4xx eller 5xx, och beskriv problemet i kroppen; RFC 9457 definierar en JSON-form för det, serverad som `application/problem+json`.

En länk som raderar saker

En `GET`-begäran ändrar tillstånd, till exempel en länk för att avsluta en prenumeration eller radera något i ett mejl. Sökrobotar, länkförhandsvisningar i chattappar, säkerhetsskannrar och webbläsarens förhämtning följer alla länkar på egen hand, och var och en av dem kan utlösa den. Allt som ändrar data hör hemma bakom `POST`, `PUT` eller `DELETE`, med en sida för bekräftelse om länken måste finnas.

En betalning som dras två gånger

Klienten skickar en `POST`, anslutningen bryts innan svaret kommer fram, och klienten försöker igen. Servern hade redan behandlat den första. HTTP ger klienten inget sätt att skilja de två fallen åt, så API:et måste göra det: ta emot en idempotensnyckel i ett huvudfält och lämna tillbaka det sparade resultatet för en upprepad nyckel, eller låta klienten välja den nya resursens identifierare och skapa den med `PUT`.

Ett formulär som blir en GET

En server svarar på en `POST` med 301 eller 302 för att skicka den vidare. Av historiska skäl tillåter specifikationen klienter att byta metod till `GET` för de två koderna, och många gör det, så kroppen försvinner på vägen. När metoden måste överleva omdirigeringen är det 307 eller 308 som gäller.

Fel variant ur en cache

En URL serverar HTML eller Markdown, engelska eller danska, beroende på huvudfälten i begäran. En cache som inte har fått veta något sparar den variant den såg först och lämnar den till nästa läsare. Svaret måste namnge de fält det beror på, till exempel `Vary: Accept`, så att en cache håller en kopia per variant i stället för en per URL.

Läs mer

- RFC 9110: HTTP Semantics — metoder, statuskoder och huvudfält, definierade en gång för alla versioner.
- RFC 9111: HTTP Caching — när ett sparat svar får återanvändas och vad `Cache-Control` säger till en cache.
- RFC 9112: HTTP/1.1 — trådformatet i text, hanteringen av anslutningar och hur meddelanden avgränsas.
- RFC 9113: HTTP/2 — binära ramar och många strömmar över en anslutning.
- RFC 9114: HTTP/3 — samma semantik överförd till strömmar i QUIC.
- RFC 9457: Problem Details for HTTP APIs — en standardiserad JSON-kropp som beskriver vad som gick fel.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är HTTP?” 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/http/