Engineer

HTTP

⇄ forespørselen og svaret weben er bygd på

Hva det er

HTTP, Hypertext Transfer Protocol, er regelsettet en nettleser, en app eller et skript følger for å be en server om noe og få et svar tilbake, og når samtalen går inne i en kryptert TLS-forbindelse, heter den samme protokollen HTTPS. Hver utveksling har to halvdeler. Klienten sender en forespørsel: en metode som `GET` eller `POST`, målet den vil ha, en liste med headerfelt og noen ganger en body. Serveren svarer med en statuskode på tre sifre, egne headerfelt og som regel en body. Å laste en side, hente et bilde og la ett program kalle et annet er den samme utvekslingen, gjentatt.

```
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
```

Spesifikasjonen kaller protokollen tilstandsløs. Hver forespørsel må ha med alt serveren trenger for å forstå den, og protokollen husker ingenting fra én utveksling til den neste. Innlogging, handlekurver og økter bygges oppå, som regel med informasjonskapsler: et headerfelt serveren setter én gang, og som klienten sender tilbake ved hver senere forespørsel.

Vokabularet er lite og presist. `GET` og `HEAD` er trygge, det vil si at de ber om en representasjon og forventes å endre ingenting. `PUT` og `DELETE` er idempotente: sender du den samme to ganger, står serveren der den ville stått etter én. `POST` lover ingen av delene. Statuskodene grupperes etter første

siffer, 2xx for suksess, 3xx for omdirigering, 4xx for et problem med forespørselen og 5xx for et problem på serveren, så en klient som møter en ukjent kode, vet likevel omtrent hva som skjedde.

Den betydningen har ligget fast mens overføringsformatet er byttet ut to ganger. HTTP/1.1 skriver meldinger som tekstlinjer over en TCP-forbindelse, og hver forbindelse leverer svarene sine ett om gangen, i rekkefølge. HTTP/2 beholder de samme metodene og feltene, men pakker dem i binære rammer og fletter mange forespørsler inn i hverandre på én forbindelse. HTTP/3 formidler den samme semantikken over QUIC, en transport bygd på UDP, som unngår TCPs blokkering fremst i køen på transportlaget mellom uavhengige strømmer. Siden 2022 har IETF publisert semantikken i ett dokument og hver versjons overføringsformat i et annet, og derfor har «hva betyr 404?» og «hvordan er byteføLgen bygd opp?» nå hvert sitt svar.

HTTP mot HTTPS er derfor ingen kamp mellom to protokoller: HTTPS er HTTP med TLS under, som krypterer utvekslingen og beviser serverens navn. WebSocket begynner livet som en HTTP-forespørsel og oppgraderer så forbindelsen til en toveis kanal, og gRPC, en konvensjon for å kalle funksjoner på en annen maskin, kjører over HTTP/2.

Bruk begrepet i en beslutning

Bruk «Hva er HTTP?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Wi-Fi, brannmurer og fjerntilgang satt opp slik at de ansatte kan jobbe hvor som helst, og bare de rette personene slipper inn.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Administrerte 30 Ubuntu Linux-VPS-instanser — 30 Linux-servere administrert, med gjenoppretting planlagt på forhånd

Administrerte 30 Ubuntu Linux-VPS-instanser Ta kontakt →

Hvorfor det er viktig

Nesten hvert system som snakker med et annet over internett, snakker allerede HTTP, så avgjørelsen som betyr noe, er sjelden om du skal bruke det. Den er om du skal bruke det protokollen definerer, eller behandle den som et rør og bygge resten på nytt oppå.

Full bruk lønner seg fordi en lang kjede av programvare allerede leser vokabularet. Et klientbibliotek kan prøve en `GET` som fikk tidsavbrudd, på nytt; en `POST` må det la være, og det samme gjelder enhver proxy på veien. En cache

kan beholde et svar merket `max-age=600` i ti minutter og servere det til neste leser uten å spørre serveren. Overvåking teller 5xx-svar som feil uten å bli bedt om det. Et API som sender hvert kall som en `POST` til én adresse og melder feil inne i et 200-svar, gir fra seg alt dette, og hver klient må da lære seg de private reglene til akkurat det API-et.

Modellen har sine grenser. Spørsmål og svar passer en klient som vet hva den vil ha; de passer langt dårligere en server som har nyheter å dytte ut, og derfor går sanntidsoppdateringer som regel over WebSocket, server-sent events eller polling.

Ukryptert HTTP på det åpne internettet er et annet spørsmål med et kort svar. Nettlesere merker det som usikkert, alle på veien kan lese og endre det, og sertifikater koster ingenting, så offentlig trafikk hører hjemme på HTTPS. Valget mellom HTTP/1.1, HTTP/2 og HTTP/3 er stort sett en innstilling på serveren eller på kanten foran den, og applikasjonskoden forblir den samme, fordi betydningen av hver forespørsel er den samme.

Slik ser det ut i praksis

Det tydeligste tilfellet på dette nettstedet er et API som ble flyttet over på et rammeverk bygd rundt HTTPs egen form. HTTP-API-et ble flyttet fra Fiber til Huma v2, som kjører oppå Fiber, slik at hvert endepunkt deklareres med struct-typer for inndata og utdata, og valideringen kommer fra typene i stedet for fra kontroller skrevet for hånd. En OpenAPI-beskrivelse faller ut av de samme deklarasjonene, og en kontroll i CI sammenligner rutene serveren registrerer, med dem beskrivelsen annonserer. Ved 649 stier og 760 operasjoner er samsvaret 100 %, og det holdes der fordi en rute ingen har beskrevet, får bygget til å feile. Nøyaktig to endepunkter ble igjen på ren Fiber: WebSocket-endepunktene, der forespørsel-og-svar-modellen til Huma ikke passer.

Nettstedet selv bruker innholdsforhandling. Hver side publiseres som HTML og som Markdown, og én måte å få Markdown på er å sende en forespørsel med `Accept: text/markdown` og la serveren velge. Den samme siden forklarer hvorfor onion-adressen er vanlig `http://` med vilje: adressen inneholder tjenestens offentlige nøkkel i kodet form, så kretsen er autentisert og kryptert før noe HTTP blir sagt.

HTTP dukker også opp som verktøy i infrastrukturkode. Da en server nummer to ble klargjort, ble begge leverandørenes brannmurer skrevet direkte mot REST-API-ene deres gjennom en generisk HTTP-oppgave i stedet for en collection fra leverandøren. Det fjernet en avhengighet og versjonsdriften dens, på bekostning av at formen på hver forespørsel må skrives for hånd. Det har en kostnad til, som siden selv navngir: den generiske HTTP-oppgaven deklarerer ingen støtte for check mode, så en tørrkjøring av playet som oppretter maskinen, hopper over hver oppgave i det, og øvingen kjøres i stedet mot brannmur-playet.

Hvor det går galt

De fleste HTTP-feil kommer av at et ord i vokabularet brukes om noe annet enn det resten av kjeden hører.

En feil som kommer fram som en suksess

Et API svarer 200 og legger `"error": "card declined"` i bodyen. Cacher kan lagre svaret, dashbord teller det som en suksess, og logikken for nye forsøk i klienten slår aldri inn. Send statuskoden som passer, 4xx eller 5xx, og beskriv problemet i bodyen; RFC 9457 definerer en JSON-form for det, servert som `application/problem+json`.

En lenke som sletter ting

En `GET`-forespørsel endrer tilstand, for eksempel en lenke for å melde seg av eller slette noe i en e-post. Søkeroboter, lenkeforhåndsvisninger i chatapper, sikkerhetsskannere og forhåndshenting i nettleseren følger alle lenker på egen hånd, og hver av dem kan utløse den. Alt som endrer data, hører hjemme bak `POST`, `PUT` eller `DELETE`, med en bekreftelsesside hvis lenken må finnes.

En betaling som trekkes to ganger

Klienten sender en `POST`, forbindelsen brytes før svaret kommer, og klienten prøver igjen. Serveren hadde allerede behandlet den første. HTTP gir klienten ingen måte å skille de to tilfellene fra hverandre på, så API-et må gjøre det: ta imot en idempotensnøkkel i en header og returnere det lagrede resultatet for en gjentatt nøkkel, eller la klienten velge identifikatoren til den nye ressursen og opprette den med `PUT`.

Et skjema som blir til en GET

En server svarer på en `POST` med 301 eller 302 for å sende den videre. Av historiske grunner lar spesifikasjonen klienter bytte metoden til `GET` på de to kodene, og mange gjør det, så bodyen faller bort på veien. Når metoden må overleve omdirigeringen, bruk 307 eller 308.

Cachen leverer en annen variant enn den som ble bedt om

Én URL serverer HTML eller Markdown, engelsk eller dansk, avhengig av headerfeltene i forespørselen. En cache som ikke har fått beskjed om noe, lagrer den varianten den så først, og gir den til neste leser. Svaret må navngi feltene det avhenger av, for eksempel `Vary: Accept`, slik at en cache beholder én kopi per variant i stedet for én per URL.

Les mer

- RFC 9110: HTTP Semantics — metoder, statuskoder og headerfelt, definert én gang for alle versjoner.
- RFC 9111: HTTP Caching — når et lagret svar kan brukes på nytt, og hva `Cache-Control` forteller en cache.
- RFC 9112: HTTP/1.1 — tekstformatet på ledningen, håndteringen av forbindelser og hvordan meldinger avgrenses.
- RFC 9113: HTTP/2 — binære rammer og mange strømmer over én forbindelse.
- RFC 9114: HTTP/3 — den samme semantikken lagt over på strømmer i QUIC.
- RFC 9457: Problem Details for HTTP APIs — en standardisert JSON-body for å beskrive hva som gikk galt.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er HTTP?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/http/