Engineer

HTTP

⇄ Protocolo de solicitudes y respuestas en que se basa la web

Qué es

HTTP, Hypertext Transfer Protocol, es el conjunto de reglas que siguen un navegador, una aplicación o un script para pedir algo a un servidor y recibir una respuesta; cuando la conversación ocurre dentro de una conexión TLS cifrada, el mismo protocolo se llama HTTPS. Cada intercambio tiene dos partes. El cliente envía una solicitud: un método como `GET` o `POST`, el destino que quiere, una lista de campos de cabecera y, a veces, un cuerpo. El servidor responde con un código de estado de tres dígitos, sus propios campos y, normalmente, un cuerpo. Cargar una página, obtener una imagen y llamar de un programa a otro repiten el mismo intercambio.

```
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
```

La especificación lo define como un protocolo sin estado. Cada solicitud debe llevar todo lo que necesita el servidor para entenderla y el protocolo no guarda memoria de un intercambio al siguiente. Los inicios de sesión, los carritos y las sesiones se construyen encima, normalmente con cookies: un campo de cabecera que el servidor fija una vez y el cliente devuelve en cada solicitud posterior.

El vocabulario es pequeño y preciso. `GET` y `HEAD` son seguros: piden una representación y se espera que no cambien nada. `PUT` y `DELETE` son idempotentes: enviar dos veces lo mismo deja al servidor como si se hubiera enviado una. `POST` no promete ninguna de las dos cosas. Los códigos de estado

se agrupan por su primer dígito: 2xx para éxito, 3xx para redirección, 4xx para un problema en la solicitud y 5xx para uno en el servidor; así, incluso ante un código desconocido, el cliente sabe aproximadamente qué ocurrió.

El significado se ha mantenido aunque el formato en la red se ha sustituido dos veces. HTTP/1.1 escribe mensajes como líneas de texto sobre una conexión TCP y cada conexión devuelve sus respuestas de una en una, en orden. HTTP/2 conserva métodos y campos, pero los organiza en tramas binarias e intercala muchas solicitudes en una conexión. HTTP/3 transporta la misma semántica sobre QUIC, un transporte basado en UDP que evita el bloqueo de cabecera de línea de TCP en la capa de transporte entre flujos independientes. Desde 2022, la IETF publica la semántica en un documento y el formato de cada versión en otro: por eso «qué significa 404» y «cómo se disponen los bytes» tienen ahora respuestas separadas.

HTTP frente a HTTPS no es, por tanto, una competición entre dos protocolos: HTTPS es HTTP con TLS por debajo, que cifra el intercambio y acredita el nombre del servidor. WebSocket empieza como una solicitud HTTP y después convierte la conexión en un canal bidireccional; gRPC es una convención para llamar a funciones remotas que funciona sobre HTTP/2.

Utilice el concepto en una decisión

Aplique «¿Qué es HTTP?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Wi-Fi, cortafuegos y acceso remoto configurados para trabajar desde cualquier lugar y permitir el acceso solo a las personas autorizadas.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Administración de 30 VPS Ubuntu Linux — 30 servidores Linux administrados con recuperación planificada

Administración de 30 VPS Ubuntu Linux Póngase en contacto →

Por qué importa

Casi todo sistema que habla con otro por Internet ya usa HTTP, así que la decisión importante rara vez es si usarlo. Es si aprovechar lo que define o tratarlo como una tubería y reconstruir el resto encima.

Usarlo plenamente compensa porque una larga cadena de software ya entiende el vocabulario. Una biblioteca cliente puede reintentar un `GET` que superó su tiempo de espera; no debe reintentar un `POST`, y tampoco debe hacerlo ningún proxy intermedio. Una caché puede conservar diez minutos una respuesta con

`max-age=600` y servirla al siguiente lector sin preguntar al servidor. La monitorización cuenta las respuestas 5xx como errores sin que se le indique. Una API que envía todo por `POST` a una dirección y comunica fallos dentro de una respuesta 200 renuncia a todo ello; cada cliente debe aprender entonces sus reglas privadas.

El modelo tiene límites. Solicitud y respuesta sirven a un cliente que sabe qué quiere; encajan mucho peor con un servidor que tiene novedades que enviar, por lo que las actualizaciones en vivo suelen viajar por WebSocket, eventos enviados por el servidor o sondeo.

HTTP en claro sobre la Internet pública es otra cuestión con respuesta breve. Los navegadores lo marcan como inseguro, cualquiera en la ruta puede leerlo y alterarlo y los certificados no cuestan nada: el tráfico público debe ir por HTTPS. Elegir entre HTTP/1.1, HTTP/2 y HTTP/3 es, en gran medida, un ajuste en el servidor o en la infraestructura perimetral que tiene delante; el código de la aplicación permanece igual porque el significado de cada solicitud no cambia.

Cómo se aplica en la práctica

El caso más claro de este sitio es una API trasladada a un framework construido sobre la propia forma de HTTP. La API HTTP pasó de Fiber a Huma v2, que funciona sobre Fiber: cada endpoint se declara con estructuras de entrada y salida y la validación nace de los tipos, no de comprobaciones escritas a mano. Las mismas declaraciones producen una descripción OpenAPI y un control de CI compara las rutas que registra el servidor con las que anuncia la descripción. Con 649 rutas y 760 operaciones, la paridad es del 100 % y se mantiene porque una ruta sin describir hace fallar el proceso de construcción. Exactamente dos endpoints permanecieron en Fiber directo: los de WebSocket, donde no encaja el modelo de solicitud y respuesta de Huma.

El propio sitio usa negociación de contenido. Cada página se publica en HTML y Markdown; una forma de obtener Markdown es enviar una solicitud con `Accept: text/markdown` y dejar elegir al servidor. La dirección onion de esa página es `http://` deliberadamente: como codifica la clave pública del servicio, Tor ya ha autenticado y cifrado el circuito cuando viaja la primera solicitud HTTP.

HTTP también aparece como herramienta en el código de infraestructura. Al aprovisionar un segundo servidor, los cortafuegos de ambos proveedores se escribieron directamente contra sus API REST mediante una tarea HTTP genérica

en lugar de una colección del proveedor. Eso eliminó una dependencia y su divergencia de versiones, a cambio de escribir a mano la forma de cada solicitud. La página señala otro coste: la tarea HTTP genérica no declara compatibilidad con el modo de comprobación, por lo que una ejecución de prueba del play que crea la máquina omite todas sus tareas y el ensayo se ejecuta contra el play del cortafuegos.

Dónde surgen los problemas

La mayoría de los errores HTTP nacen de usar una palabra del vocabulario con un significado distinto del que entiende el resto de la cadena.

Un error que llega como éxito

Una API responde 200 y pone `"error": "card declined"` en el cuerpo. Las cachés pueden almacenarlo, los paneles cuentan un éxito y la lógica de reintento del cliente nunca se activa. Envíe el estado adecuado, 4xx o 5xx, y describa el problema en el cuerpo; el RFC 9457 define un formato JSON servido como `application/problem+json` .

Un enlace que elimina cosas

Una solicitud `GET` cambia el estado, por ejemplo un enlace para darse de baja o eliminar algo en un correo. Rastreadores, vistas previas de enlaces en aplicaciones de chat, analizadores de seguridad y precarga del navegador siguen enlaces por sí solos y cualquiera puede activarlo. Todo lo que cambia datos debe ir tras `POST` , `PUT` o `DELETE` , con una página de confirmación si el enlace debe existir.

Un pago cobrado dos veces

El cliente envía un `POST` , la conexión cae antes de llegar la respuesta y el cliente vuelve a intentarlo. El servidor ya había procesado el primero. HTTP no permite al cliente distinguir esos casos, así que debe hacerlo la API: aceptar una clave de idempotencia en una cabecera y devolver el resultado almacenado al repetir la clave, o dejar que el cliente elija el identificador del recurso nuevo y lo cree con `PUT` .

Un formulario que se convierte en GET

Un servidor responde a un `POST` con 301 o 302 para enviarlo a otro lugar. Por razones históricas, la especificación permite a los agentes de usuario cambiar el método a `GET` con esos dos códigos, y muchos lo hacen, de modo que se

descarta el cuerpo por el camino. Cuando deba conservarse el método durante la redirección, use 307 o 308.

La variante equivocada de una caché

Una URL sirve HTML o Markdown, inglés o danés, según los campos de cabecera de la solicitud. Una caché sin instrucciones conserva la primera variante que ve y la entrega al siguiente lector. La respuesta debe nombrar los campos de los que depende, por ejemplo `Vary: Accept`, para que la caché guarde una copia por variante y no por URL.

Lecturas adicionales

- RFC 9110: semántica HTTP — Métodos, códigos de estado y campos de cabecera definidos una sola vez para todas las versiones.
- RFC 9111: caché HTTP — Cuándo puede reutilizarse una respuesta almacenada y qué le indica `Cache-Control` a una caché.
- RFC 9112: HTTP/1.1 — Formato de texto en la red, gestión de conexiones y delimitación de mensajes.
- RFC 9113: HTTP/2 — Tramas binarias y muchos flujos sobre una conexión.
- RFC 9114: HTTP/3 — La misma semántica trasladada a flujos QUIC.
- RFC 9457: detalles de problemas para API HTTP — Un cuerpo JSON estándar para describir qué falló.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es HTTP?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/http/