Engineer

HTTP

⇄ Protocole de requêtes-réponses à la base du Web

Ce que c’est

HTTP, Hypertext Transfer Protocol, est l’ensemble de règles qu’un navigateur, une application ou un script suit pour demander quelque chose à un serveur et recevoir une réponse. Quand cet échange passe dans une connexion TLS chiffrée, le même protocole prend le nom de HTTPS. Chaque échange a deux parties. Le client envoie une requête : méthode telle que `GET` ou `POST`, cible voulue, liste de champs d’en-tête et parfois corps. Le serveur répond avec un code de statut à trois chiffres, ses propres en-têtes et généralement un corps. Charger une page, récupérer une image ou appeler un programme depuis un autre répète le même échange.

```
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 spécification le dit sans état. Chaque requête doit fournir tout ce dont le serveur a besoin pour la comprendre, et le protocole ne garde aucun souvenir d’un échange à l’autre. Connexions utilisateur, paniers et sessions sont construits au-dessus, généralement avec des cookies : un champ d’en-tête fixé une fois par le serveur et renvoyé par le client à chaque requête suivante.

Le vocabulaire est réduit et précis. `GET` et `HEAD` sont sûrs : ils demandent une représentation et ne sont pas censés modifier quoi que ce soit. `PUT` et `DELETE` sont idempotents : envoyer deux fois la même requête laisse le serveur dans l’état d’un seul envoi. `POST` ne promet ni l’un ni l’autre. Les codes sont

regroupés par premier chiffre : 2xx pour le succès, 3xx pour une redirection, 4xx pour un problème de requête et 5xx pour un problème serveur. Même devant un code inconnu, le client sait donc approximativement ce qui s'est passé.

Ce sens est resté stable alors que le format réseau a été remplacé deux fois. HTTP/1.1 écrit des messages en lignes de texte sur une connexion TCP, qui rend ses réponses une par une, dans l'ordre. HTTP/2 garde méthodes et champs, mais les place dans des trames binaires et entrelace plusieurs requêtes sur une connexion. HTTP/3 transporte la même sémantique sur QUIC, transport fondé sur UDP qui évite le blocage en tête de ligne de TCP au niveau du transport entre flux indépendants. Depuis 2022, l'IETF publie la sémantique dans un document et le format réseau de chaque version dans un autre : « que signifie 404 » et « comment les octets sont-ils disposés » ont donc des réponses séparées.

HTTP et HTTPS ne sont donc pas deux protocoles concurrents : HTTPS est HTTP sur TLS, qui chiffre l'échange et atteste le nom du serveur. WebSocket commence par une requête HTTP puis transforme la connexion en canal bidirectionnel. gRPC est une convention d'appels de fonctions distantes qui fonctionne sur HTTP/2.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que HTTP ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Du Wi-Fi, des pare-feu et des accès à distance configurés pour travailler de partout, en laissant entrer uniquement les personnes autorisées.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 30 instances VPS Ubuntu Linux administrées — 30 serveurs Linux gérés, avec récupération prévue

30 instances VPS Ubuntu Linux administrées Contactez-nous →

Pourquoi c'est important

Presque tout système échangeant avec un autre sur Internet parle déjà HTTP ; la décision importante est rarement de l'utiliser ou non. C'est d'employer ce qu'il définit ou de le traiter comme un tuyau et de reconstruire le reste au-dessus.

L'utiliser pleinement paie, car une longue chaîne de logiciels comprend déjà son vocabulaire. Une bibliothèque cliente peut réessayer un `GET` dont le délai d'attente a été dépassé ; elle ne doit pas réessayer un `POST`, pas plus qu'un proxy intermédiaire. Un cache peut garder dix minutes une réponse marquée `max-age=600` et la servir au lecteur suivant sans interroger le serveur. La

supervision compte les réponses 5xx comme erreurs sans qu'on le lui demande. Une API dont tous les appels se font par `POST` à une adresse unique et qui signale l'échec dans une réponse 200 renonce à cela ; chaque client doit alors apprendre ses règles privées.

Le modèle a ses limites. Requête et réponse conviennent au client qui sait ce qu'il veut ; beaucoup moins au serveur qui a des nouvelles à pousser. Les mises à jour en direct passent donc généralement par WebSocket, les événements envoyés par le serveur ou le sondage.

HTTP en clair sur Internet public est une autre question, à réponse courte. Les navigateurs le disent non sécurisé, toute personne sur le trajet peut lire et modifier les échanges, et les certificats sont gratuits : le trafic public doit donc passer par HTTPS. Choisir entre HTTP/1.1, HTTP/2 et HTTP/3 relève surtout d'un réglage du serveur ou de l'infrastructure en amont ; le code applicatif ne change pas, car le sens des requêtes reste identique.

À quoi cela ressemble en pratique

Le cas le plus net de ce site est une API déplacée vers un framework construit autour de la forme propre à HTTP. L'API HTTP est passée de Fiber à Huma v2, qui fonctionne au-dessus de Fiber : chaque point de terminaison est déclaré avec des structures d'entrée et de sortie, et la validation provient des types plutôt que de contrôles manuels. Ces déclarations produisent une description OpenAPI ; un contrôle CI compare les routes enregistrées par le serveur à celles annoncées par la description. Avec 649 chemins et 760 opérations, la parité est de 100 % et se maintient parce qu'une route non décrite fait échouer le processus de construction. Exactement deux points de terminaison sont restés directement sur Fiber : ceux de WebSocket, auxquels le modèle requête-réponse de Huma ne convient pas.

Le site lui-même utilise la négociation de contenu. Chaque page est publiée en HTML et en Markdown ; on peut obtenir le Markdown en envoyant une requête avec `Accept: text/markdown` et en laissant le serveur choisir. L'adresse onion de cette page est volontairement en `http://` : elle encode la clé publique du service, et Tor a donc déjà authentifié et chiffré le circuit au passage de la première requête HTTP.

HTTP sert aussi d'outil dans le code d'infrastructure. Lors du provisionnement d'un second serveur, les pare-feu des deux fournisseurs ont été configurés directement via leurs API REST, par une tâche HTTP générique plutôt qu'une collection du fournisseur. Cela a supprimé une dépendance et sa dérive de versions, au prix de l'écriture manuelle de chaque forme de requête. La page nomme un autre coût : la tâche générique ne déclare aucune prise en charge du mode de vérification. Une simulation du play qui crée la machine saute donc toutes ses tâches ; l'essai à blanc est effectué sur le play du pare-feu à la place.

Où les problèmes surviennent

La plupart des défauts HTTP viennent d'un mot du vocabulaire employé dans un autre sens que celui compris par le reste de la chaîne.

Une erreur qui arrive comme un succès

Une API répond 200 et place `"error": "card declined"` dans le corps. Les caches peuvent la stocker, les tableaux de bord comptent un succès et la logique de nouvelle tentative ne se déclenche jamais. Envoyez le code adapté, 4xx ou 5xx, et décrivez le problème dans le corps ; le RFC 9457 définit un format JSON servi comme `application/problem+json` .

Un lien qui supprime des données

Une requête `GET` change l'état, par exemple un lien de désabonnement ou de suppression dans un courriel. Robots, aperçus de liens des messageries, scanners de sécurité et préchargement des navigateurs suivent tous les liens de leur propre initiative et peuvent chacun déclencher l'action. Toute modification de données doit passer par `POST` , `PUT` ou `DELETE` , avec une page de confirmation si le lien doit exister.

Un paiement prélevé deux fois

Le client envoie un `POST` , la connexion se coupe avant la réponse et le client réessaie. Le serveur avait déjà traité le premier. HTTP ne permet pas au client de distinguer ces cas : l'API doit donc le faire. Elle peut accepter une clé d'idempotence en en-tête et rendre le résultat stocké si la clé revient, ou laisser le client choisir l'identifiant de la nouvelle ressource et la créer avec `PUT` .

Un formulaire qui devient un GET

Un serveur répond à un `POST` par 301 ou 302 pour l'envoyer ailleurs. Pour des raisons historiques, la spécification permet aux agents utilisateurs de passer la méthode à `GET` sur ces deux codes ; beaucoup le font, et le corps est abandonné en route. Si la méthode doit survivre à la redirection, utilisez 307 ou 308.

La mauvaise variante fournie par un cache

Une URL sert HTML ou Markdown, anglais ou danois, selon les en-têtes de requête. Un cache non informé stocke la première variante reçue et la rend au lecteur suivant. La réponse doit nommer les champs dont elle dépend, par exemple `Vary: Accept`, pour que le cache garde une copie par variante et non par URL.

Pour aller plus loin

- RFC 9110 : sémantique HTTP — Méthodes, codes de statut et champs d'en-tête définis une fois pour toutes les versions.
- RFC 9111 : cache HTTP — Réutilisation d'une réponse stockée et indications de `Cache-Control` au cache.
- RFC 9112 : HTTP/1.1 — Format réseau textuel, gestion des connexions et délimitation des messages.
- RFC 9113 : HTTP/2 — Trames binaires et flux multiples sur une connexion.
- RFC 9114 : HTTP/3 — Même sémantique appliquée aux flux QUIC.
- RFC 9457 : détails des problèmes pour les API HTTP — Corps JSON standard décrivant ce qui s'est mal passé.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que HTTP ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/http/