Engineer

WebSocket

↔ un protocole de connexion bidirectionnelle persistante

Ce que c’est

WebSocket est un protocole de connexion sur laquelle les deux parties peuvent écrire à tout moment. Une négociation HTTP l’établit, et elle reste ouverte jusqu’à ce qu’un participant la ferme ou qu’elle tombe en panne. L’IETF l’a publié dans la RFC 6455 en décembre 2011 ; la partie qu’un navigateur expose aux scripts, l’objet `WebSocket`, est définie par le WebSockets Standard du WHATWG. Dans le code et la documentation, le nom s’écrit aussi souvent « websocket » ou « web socket », et le pluriel WebSockets désigne généralement la technologie dans son ensemble.

Dans la négociation classique en HTTP/1.1, la connexion commence par un simple `GET`. Le client ajoute `Upgrade: websocket` et `Connection: Upgrade`, un `Sec-WebSocket-Version` de 13 et un `Sec-WebSocket-Key`, valeur aléatoire de 16 octets encodée en base64. Un serveur qui accepte répond par le statut 101 Switching Protocols et un en-tête `Sec-WebSocket-Accept`. Cette valeur prouve qu’il a lu cette négociation précise plutôt que de rejouer une ancienne réponse : c’est la clé du client à laquelle un GUID fixe a été ajouté, le tout haché avec SHA-1 puis encodé en base64. À partir de là, HTTP n’intervient plus sur cette connexion et les deux parties échangent des trames. HTTP/2 et HTTP/3 transportent plutôt WebSocket sur un seul flux au moyen de CONNECT étendu, comme le précisent les RFC 8441 et 9220.

Une trame est petite. Son en-tête occupe d’abord deux octets, auxquels s’ajoutent deux ou huit octets lorsque la charge utile dépasse 125 octets, puis quatre autres dans les trames du client, qui portent une clé de masquage. Chaque trame porte un code d’opération : texte, données binaires ou continuation pour un message réparti sur plusieurs trames ; trois autres codes désignent les trames de contrôle de fermeture, ping et pong. Un message texte complet doit être du UTF-8 valide ; un fragment isolé peut couper une séquence UTF-8. Chaque trame envoyée par un client est masquée avec une nouvelle clé

aléatoire de 32 bits, tandis qu'un serveur ne masque jamais. Le masquage n'est pas du chiffrement. Il sert à empêcher un script hostile de choisir les octets exacts qui traversent le réseau pour empoisonner un proxy de mise en cache sur le trajet.

Les adresses utilisent les schémas `ws://` et `wss://`, avec les ports 80 et 443 par défaut, et `wss` fait passer tout l'échange dans TLS. Ce sont les ports qu'HTTP utilise déjà, délibérément : pour les pare-feu et les proxys traversés, la négociation ressemble à n'importe quelle autre requête web.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que WebSocket ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. La partie invisible pour les utilisateurs : où résident les données, quelles règles s'appliquent et comment les autres systèmes peuvent y accéder.

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. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c'est important

Avec des réponses HTTP courtes, le serveur ne parle que lorsqu'on l'interroge. Beaucoup de produits ont besoin de l'inverse : un message de chat, un prix qui vient de changer, un curseur dans un document partagé, une tâche de fond qui s'est terminée. Sans canal sur lequel le serveur peut écrire, le client doit interroger périodiquement le serveur. Il le fait toutes les quelques secondes ; la plupart des réponses disent seulement que rien n'a changé, chacune porte un ensemble complet d'en-têtes, et une information nouvelle peut attendre tout un intervalle avant d'être vue. Une connexion WebSocket ouverte supprime ces deux coûts. Le serveur peut envoyer dès qu'un événement survient, sans attendre la prochaine interrogation. Chaque message ne demande que quelques octets d'encadrement au lieu d'un ensemble complet d'en-têtes HTTP.

Le prix à payer est l'état à conserver. Une requête HTTP peut arriver sur n'importe quel serveur derrière un répartiteur de charge, tandis qu'une connexion WebSocket appartient à un processus pendant toute sa durée de vie. Ce processus réserve de la mémoire pour chaque connexion ouverte ; un déploiement ou un redémarrage les coupe toutes à la fois, et chaque proxy et répartiteur de charge du trajet doit prendre en charge la négociation et

supporter une connexion qui peut rester silencieuse plusieurs minutes. Exploiter plusieurs serveurs soulève une question qu'HTTP ordinaire ne pose jamais : lorsqu'un événement survient sur le serveur A, comment atteint-il un client dont la connexion réside sur le serveur B ? La réponse habituelle est un canal partagé de publication et d'abonnement qu'écoute chaque serveur, soit un système de plus à exploiter.

C'est donc un choix, pas une solution à adopter par défaut. Lorsque seul le serveur a des nouvelles à envoyer, les server-sent events, définis dans le standard HTML, poussent des événements dans une réponse HTTP ordinaire de type `text/event-stream`, et le navigateur se reconnecte tout seul si la réponse s'interrompt. Les mécanismes supplémentaires de WebSocket se justifient quand le client a autant à dire que le serveur, comme dans le chat, les jeux et l'édition collaborative.

À quoi cela ressemble en pratique

L'échange ci-dessous ouvre une connexion de chat. La clé est la valeur d'exemple de la RFC 6455 elle-même, et la valeur d'acceptation est celle que le standard donne, que tout serveur correct doit renvoyer pour cette clé.

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Origin: https://example.com

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Après la ligne vide qui termine la réponse du serveur, plus rien sur la connexion n'est du HTTP. Un navigateur effectue toute la négociation dans un seul appel au constructeur et confie la suite à des écouteurs d'événements :

```
const socket = new WebSocket("wss://example.com/chat");
socket.addEventListener("open", () => socket.send("hello"));
socket.addEventListener("message", (event) => console.log(event.data));
```

```
socket.addEventListener("close", (event) => {
  console.log(event.code, event.reason, event.wasClean);
});
```

L'événement de fermeture indique comment la connexion s'est terminée. Le code 1000 signale une fermeture normale, et 1001 indique qu'un participant s'en va, par exemple un serveur qui s'arrête ou un navigateur qui quitte la page. Le code 1006 signifie que la connexion s'est interrompue sans aucune trame de fermeture. Aucun point de connexion n'est autorisé à l'envoyer ; il existe uniquement pour que le programme qui utilise la connexion puisse distinguer une fin anormale d'une fermeture propre.

La distinction entre requête-réponse et socket apparaît clairement dans un projet du portfolio de ce site. Lors du passage d'une API à Huma v2, toutes les routes ont adopté le modèle requête-réponse de Huma, sauf exactement deux : les points d'accès WebSocket sont restés sur Fiber directement, car le code y a réellement besoin du socket lui-même et le modèle de Huma ne convient pas. Un framework conçu autour de la forme d'HTTP décrit et valide bien les requêtes et les réponses, tandis qu'une connexion bidirectionnelle durable relève simplement d'un autre fonctionnement.

Où les problèmes surviennent

Une page d'un autre site ouvre une connexion avec les cookies de vos visiteurs. Le navigateur envoie la requête de négociation à l'hôte du serveur et peut y joindre ses cookies, sous réserve de sa politique relative aux cookies. Si cette politique autorise des requêtes authentifiées entre sites, un serveur qui authentifie par cookie et ignore l'en-tête `Origin` peut accepter une connexion ouverte par une page d'un autre site sous l'identité du visiteur. La RFC 6455 indique qu'un serveur destiné uniquement à certains sites devrait comparer `Origin` aux origines attendues et refuser les autres avec 403 Forbidden. Faites-le pendant la négociation, avant d'accepter la connexion.

Le jeton d'accès se retrouve dans les journaux. Le constructeur `WebSocket` du navigateur accepte une URL et une liste facultative de sous-protocoles, rien d'autre : un script ne peut donc pas ajouter d'en-tête `Authorization` . Le raccourci consiste à mettre un jeton dans l'URL, où les proxys inverses et les journaux d'accès le consignent. La RFC laisse le choix de l'authentification ouvert et cite les cookies, l'authentification HTTP et l'authentification TLS parmi les possibilités. Un

cookie accompagné du contrôle `Origin` ci-dessus fonctionne, tout comme un ticket à usage unique et à courte durée de vie que la page obtient par un appel HTTP authentifié puis utilise pour la négociation.

Un redémarrage transforme tous les clients en déferlante. Un déploiement brutal ou un redémarrage peut fermer toutes les connexions à la fois. Les clients qui ne reçoivent aucune trame de fermeture signalent le code 1006 ; s'ils se reconnectent tous immédiatement et persistent, le serveur revient face à ce que la RFC compare à une attaque par déni de service. La RFC 6455 demande aux clients d'attendre un délai aléatoire avant le premier essai, suggère une durée comprise entre 0 et 5 secondes, puis d'attendre davantage après chaque échec, par exemple avec un délai exponentiel binaire tronqué. Un serveur qui ferme ses connexions avec 1001 avant de s'arrêter indique au moins aux clients que l'interruption était prévue.

```
function reconnectDelay(attempt) {
  if (attempt === 0) return Math.random() * 5000;
  const ceiling = Math.min(60_000, 1000 * 2 ** attempt);
  return Math.random() * ceiling;
}
```

Les connexions meurent sans que personne le remarque. Une connexion sans trafic ne donne à aucune partie le signe que le trajet existe encore, et un correspondant qui disparaît sans trame de fermeture ressemble exactement à un correspondant silencieux jusqu'à l'échec de la prochaine écriture. Ping et pong servent à cela : une extrémité doit répondre à un ping par un pong, sauf si elle a déjà reçu une trame de fermeture. Si plusieurs pings restent sans réponse, elle peut ne répondre qu'au dernier. La RFC précise qu'un ping peut servir à maintenir la connexion ou à vérifier que l'autre partie répond encore. L'API du navigateur ne permet pas d'envoyer de ping : le minuteur doit donc être géré côté serveur. Envoyez un ping à chaque connexion à intervalle fixe et fermez celles qui ne répondent pas par un pong.

Un seul message dévore la mémoire du serveur. Le champ de longueur de l'en-tête d'une trame permet des charges utiles bien supérieures à la mémoire d'un serveur, et la RFC avertit qu'un correspondant hostile peut envoyer une trame énorme ou une suite infinie de petits fragments. Les implémentations doivent se protéger : fixez une taille maximale de message dans la bibliothèque serveur et fermez avec le code 1009, message too big, lorsqu'un correspondant la dépasse. Une accumulation similaire se produit dans l'autre sens. Dans le navigateur,

`bufferedAmount` indique combien d'octets mis en attente par `send()` attendent encore de partir sur le réseau, et l'exemple du WHATWG n'envoie la mise à jour suivante que lorsque ce nombre est nul. Sur le serveur, un client qui lit lentement remplit de la même manière le tampon de sortie. Chaque connexion a donc besoin d'une limite pour ce tampon et d'une règle indiquant ce qui se passe lorsqu'elle est atteinte.

Pour aller plus loin

- RFC 6455: The WebSocket Protocol — la négociation, le découpage en trames, le masquage et les codes de fermeture, avec les raisons de sécurité de chacun.
- WebSockets Standard — l'API du navigateur : le constructeur, `send()`, `close()`, `bufferedAmount` et l'événement de fermeture.
- RFC 9110: HTTP Semantics — le champ d'en-tête `Upgrade` et le statut 101 Switching Protocols sur lesquels repose la négociation.
- Server-sent events — l'alternative unidirectionnelle du standard HTML, avec reconnexion intégrée.
- RFC 8441: Bootstrapping WebSockets with HTTP/2 — CONNECT étendu transporte WebSocket sur un flux HTTP/2 multiplexé.
- RFC 9220: Bootstrapping WebSockets with HTTP/3 — le mécanisme correspondant de CONNECT étendu pour HTTP/3.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que WebSocket ? » 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/websocket/