

LES CONCEPTS HTTP FONDAMENTAUX

- Introduction au HTTP, le protocole de transfert hypertexte
- Requête, réponse et session HTTP

Introduction au HTTP, le protocole de transfert hypertexte

Le protocole de transfert hypertexte ou HTTP (HyperText Transfer Protocol) est, avec le langage HTML et les adresses web l'une des trois fondations qui permettent au World Wide Web tel qu'on le connaît aujourd'hui d'exister.

HTTP (HyperText Transfer Protocol ou Protocole de Transfert Hypertexte en français) est un protocole de communication permettant de récupérer et d'envoyer des ressources hypermédia. Au départ, HTTP a été créé pour permettre de transfert de documents HTML uniquement puis ses usages se sont rapidement étendus pour permettre de transférer d'autres types de ressources comme des images, des vidéos ou d'autres types de fichiers.

Un protocole est tout simplement un ensemble de règles. Dans un contexte réseau, un protocole est un ensemble de règles qui régissent les échanges de données.

Pourquoi apprendre comment fonctionne HTTP ?

Comprendre comment fonctionne HTTP n'est pas obligatoire au sens strict du terme pour développer des pages web. Cependant, cela vous permettra d'avoir une bien meilleure compréhension du fonctionnement du web en général et de « ce qu'il se passe en arrière-plan » lors d'un échange web et cela vous aidera également à mieux comprendre certaines erreurs qui peuvent survenir sur un site et comment les résoudre.

De la même manière, il est intéressant de comprendre le modèle réseau complet qui permet aux informations d'aller d'un point d'origine à un point d'arrivée dans lequel s'insère HTTP. Cela vous permettra d'avoir une bonne vision d'ensemble du fonctionnement de l'échange de données entre deux machines.

HTTP, un protocole de transfert client – serveur

Le protocole HTTP est basé sur un modèle client-serveur. Dans les modèles de communication client-serveur, le client est toujours celui qui demande à accéder (demande de récupérer) une ressource tandis que le serveur est toujours celui qui va répondre à la demande (qui va « servir » la ressource).

Dans la majorité des cas, le client va être un navigateur. Par exemple, lorsqu'un visiteur cherche à accéder à l'une de nos pages web, il va taper une URL dans son navigateur. Le navigateur va alors demander au serveur sur lequel est stockée la page de la lui renvoyer. La demande du client est appelée une requête HTTP.

Un serveur est un ordinateur très puissant donc l'unique rôle est de stocker des ressources et de les servir aux clients lorsque ceux-ci les demandent. Pour se faire, un serveur doit être toujours allumé et connecté à d'autres serveurs de manière à pouvoir être accessible.

Notez ici qu'une ressource peut être divisée en plusieurs parties et que chaque partie peut être stockée sur une machine différente. Au contraire, une même machine peut stocker des ressources appartenant à différents sites. On utilisera généralement le terme « le serveur » de manière indiscriminée pour désigner une machine à part entière, une partie d'une machine ou plusieurs machines à partir du moment où leur rôle est de stocker et de servir des ressources.

Les requêtes et les réponses HTTP (qu'on appelle également messages HTTP) sont au cœur du protocole de transfert hypertexte. Nous étudierons en détail comment elle sont créées et de quoi elles sont composées dans la suite de ce cours.

HTTP, un protocole sans état et évolutif

HTTP est l'une des trois inventions majeures qui ont permis la création du web. HTTP est donc présent depuis la création du web et a su évoluer jusqu'à aujourd'hui pour répondre à l'évolution et à la diversification des usages du web et pour faire en sorte qu'ils soient possibles.

Au début du web, les pages n'étaient composées que de HTML et ne faisaient pas appel à des ressources externes. Le client n'avait donc qu'à effectuer une requête au serveur afin que celui-ci lui renvoie le document HTML.

Aujourd'hui, les pages web sont composées de différentes ressources : une structure HTML, des ressources CSS, JavaScript, des images, vidéos, etc.

Les usages se sont également diversifiés : au départ du web, la seule opération possible en tant qu'utilisateur était la récupération et la visualisation d'une ressource. Aujourd'hui, on peut également envoyer ou modifier des informations (via des formulaires HTML notamment).

Tout cela est possible car HTTP a su évoluer et ajouter de nouveaux composants / fonctionnalités pour fournir davantage de possibilité d'échanges. Toutefois, comme les usages et les capacités du web étaient au départ très limités, les règles composant HTTP n'ont pas été formulées avec une quelconque idée de performance ou d'optimisation et cela représente le défi principal de HTTP.

Par exemple, deux des principes de base du HTTP sont qu'une requête ne peut récupérer qu'une seule ressource à la fois / n'effectuer qu'une action à la fois et que HTTP est un protocole sans état.

Lorsqu'on dit que HTTP est un protocole sans état, cela signifie que HTTP n'a pas besoin que le serveur conserve des informations sur un client entre deux requêtes. Autrement dit, chaque nouvelle requête peut agir de manière totalement indépendante et n'a pas de lien à priori avec les requêtes précédentes ou suivantes.

Les pages web actuelles sont composées de nombreuses ressources et vont donc demander au client d'effectuer plusieurs requêtes. Généralement, le client va demander qu'on lui renvoie le document HTML demandé, puis va commencer à le lire et va ensuite effectuer de nouvelles requêtes au serveur à chaque fois qu'il va tomber sur une autre ressource incluse dans le document.

Chaque nouveau jeu de requête-réponse implique un aller-retour des données, ce qui prend du temps et entraîne donc de la latence (un temps d'attente) pour l'utilisateur. Des techniques comme les connexions parallèles, persistantes ou en pipelines ont été progressivement formulées et implémentées pour résoudre ces problèmes de performance. Nous étudierons évidemment ces techniques en détail dans la suite du cours.

Pour le moment, contentez-vous de retenir que l'un des enjeux majeurs du HTTP depuis quelques années est un enjeu lié à l'optimisation des performances pour « rendre le web plus rapide ».

HTTP et les modèles TCP/IP et OSI

HTTP est un protocole qui définit la forme de la communication entre deux machines, c'est-à-dire la syntaxe des messages envoyés afin que les deux machines puissent effectivement communiquer.

Cependant, HTTP n'indique pas comment les messages doivent transiter pour partir d'une machine A et arriver à une machine B.

En fait, HTTP est un protocole qui ne sert à définir qu'une partie de la communication entre deux machines et qui s'insère dans un modèle réseau beaucoup plus large.

Aujourd'hui, il existe deux modèles réseaux de référence : les modèles modèle OSI (Open Systems Interconnection) et TCP/IP. Ces deux modèles permettent de fournir un cadre global pour la communication entre deux machines.

La chose importante à retenir de ces deux modèles pour le moment est qu'ils définissent tous les deux différentes « couches » de communication. Grosso-modo, ces deux modèles découpent la communication entre deux machines en différentes couches qui vont chacune définir les règles d'un point précis de la communication.

Chaque couche va contenir ses propres protocoles et les protocoles d'un modèle réseau ne vont pouvoir appartenir qu'à une couche précise. L'idée principale derrière de genre de modèle est de pouvoir faire évoluer les différents protocoles de manière indépendante (sans avoir à modifier l'ensemble du processus de communication) et de nous permettre d'implémenter tel ou tel protocole de telle couche.

Le modèle OSI par exemple est un modèle théorique composé de 7 couches différentes : application, présentation, session, transport, réseau, liaison et physique. Les protocoles du modèle OSI vont pouvoir appartenir à l'une ou l'autre de ces couches.

L'inconvénient du modèle OSI est qu'il reste difficile à appliquer de manière pratique pour le web car certains protocoles vont « logiquement » couvrir plusieurs couches.

Pour cela, un autre modèle a été formulé : le modèle TCP/IP, encore appelé « modèle Internet ». Ce modèle, composé uniquement de 4 couches nommées couches application, transport, internet et accès réseau traduit moins fidèlement la réalité des échanges entre deux machines mais est beaucoup plus pratique à mettre en place. C'est le modèle de référence utilisé pour les échanges sur le web.

Nous allons étudier chacun de ces deux modèles en détail dans la suite du cours. Pour le moment, vous pouvez retenir qu'HTTP est un protocole de la couche application dans chacun de ces deux modèles.

HTTP et HTTPS

HTTP est un protocole de transfert de la couche application des modèles OSI et TCP/IP. Ce protocole permet in fine l'envoi et la récupération ressources hypermedias.

Il y a quelques années, la plupart des utilisateurs du web ne faisaient pas confiance aux sites et n'osaient pas acheter sur le web car ils devaient transmettre des informations sensibles comme des numéros de carte bleu.

Netscape a alors inventé un protocole de sécurisation des échanges de données appelées SSL pour Secure Sockets Layer.

L'IETF (groupe chargé de la standardisation de certains protocoles et notamment chargé de l'évolution de HTTP) a ensuite continué son développement en le renommant TLS (Transport Layer Security).

TLS est aujourd'hui la norme dans le chiffrement / cryptage des échanges de données sur le net. HTTPS est simplement une variante de HTTP qui implémente ce protocole de sécurisation.

Pour information, un serveur HTTP utilise le port 80 par défaut. Un serveur supportant HTTPS utilisera le port 443 par défaut. Un « port » permet de distinguer différents interlocuteurs (différents programmes ou applications).

Requête, réponse et session HTTP

HTTP est un protocole qui décrit la façon dont des données doivent être transférées à travers des réseaux.

Le transfert de données se passe en deux temps : tout d'abord, un client (généralement un navigateur) effectue une requête HTTP (une demande) qui va être transmise à un serveur. Le serveur répond ensuite en renvoyant une réponse HTTP.

Notez bien que le HTTP n'a pour but que de définir la forme de ces messages et n'a pas pour objet d'indiquer les moyens de transfert des messages en soi. Cette tâche est laissée à d'autres protocoles qui fonctionnent avec HTTP mais à d'autres niveaux (d'autres couches) des modèles TCP/IP ou OSI.

Structure générale d'une requête et d'une réponse HTTP

Les requêtes et réponses HTTP partagent une structure similaire. Ces messages sont composés de :

- Une ligne de départ qui permet de décrire la requête et la façon dont elle doit être traitée ou son état de traitement ;
- Un ensemble (facultatif) d'en-têtes HTTP ;
- Une ligne vierge ;
- Un corps de message (facultatif) qui contient des données associées au message HTTP.

Structure détaillée d'une requête HTTP

Une requête HTTP est composée d'une ligne initiale suivie éventuellement d'en-têtes HTTP suivis d'une ligne vierge suivie éventuellement d'un corps de requête.

Exemple requete reponse HTTP

La première ligne d'une requête HTTP permet de préciser la requête HTTP. Celle-ci est composée de :

- La méthode de requête utilisée qui sert à indiquer le type de requête effectuée : simple récupération de ressources, envoi de données sur le serveur, etc. ;
- La cible de la requête (si applicable) qui va généralement prendre la forme d'une URL ou d'un chemin absolu ;

- La version HTTP utilisée pour la requête (qui sert également à indiquer la version attendue pour la réponse).

Sous cette ligne de départ, un ou plusieurs en-têtes HTTP peuvent être ajoutés. Ces en-têtes permettent de préciser la requête ou d'ajouter des informations de contexte. Les en-têtes vont par exemple nous servir à indiquer la langue préférée pour une ressource demandée, à indiquer comment la connexion avec le serveur doit être établie, le type et le poids du corps de requête, etc.

Il existe de très nombreux en-têtes HTTP différents et nous allons étudier les plus communs dans la suite de ce cours. Dans un message HTTP, chaque en-tête doit être décrit sur une ligne.

Après les lignes d'en-tête, une requête HTTP doit obligatoirement comporter une ligne vierge puis peut enfin comporter un corps de requête. Le corps de requête permet de transmettre des données au serveur et toutes les requêtes n'en ont pas.

Par exemple, la plupart des requêtes `GET` (permettant de récupérer des ressources) n'ont pas besoin d'ajouter un corps. En revanche, des requêtes comme `POST` qui servent à envoyer des informations au serveur vont transmettre ces informations dans le corps de la requête.

Note : depuis HTTP/2, les données sont encodées en binaire et ne sont donc plus lisibles pour un humain.

Types de requêtes et liste des méthodes HTTP

A l'origine, lors de la création de HTTP, le web se limitait à un ensemble de pages composées uniquement de HTML et la seule opération possible pour un client était de récupérer ces documents pour que le visiteur puisse les consulter.

Par la suite, la liste des opérations possibles pour le visiteur s'est étoffée, notamment grâce aux formulaires HTML qui permettent à un visiteur d'envoyer ou de modifier des informations sur le serveur.

Chaque type d'opération correspond à un type de requête HTTP. Les différents types de requêtes sont eux-mêmes définis par l'utilisation de méthodes HTTP différentes.

Ces méthodes HTTP permettent d'indiquer au serveur quelle opération le client souhaite effectuer. La méthode `GET` par exemple permet d'indiquer qu'on souhaite récupérer une ressource, tandis que `POST` est utilisée pour transmettre des données en vue d'un traitement à une ressource, `DELETE` est utilisée pour indiquer qu'on souhaite supprimer une ressource, etc.

HTTP est un protocole très évolutif et de nouvelles méthodes pourront être ajoutées dans le futur sans limitation.

Les méthodes disponibles à l'heure actuelle sont les suivantes :

- `GET` permet de demander une ressource sans la modifier ;
- `POST` permet de transmettre des données dans le but de manipuler une ressource ;
- `PUT` permet de remplacer ou d'ajouter une ressource sur le serveur ;
- `DELETE` permet de supprimer une ressource du serveur ;
- `HEAD` permet de demander des informations sur la ressource sans demander la ressource elle-même ;

- **PATCH** permet de modifier partiellement une ressource ;
- **OPTIONS** permet d'obtenir les options de communication d'une ressource ou du serveur ;
- **CONNECT** permet d'utiliser un proxy comme un tunnel de communication ;
- **TRACE** permet de tester et d'effectuer un diagnostic de la connexion et demandant au serveur de retourner la requête reçue.

Voici également un tableau avec les caractéristiques inhérentes à chaque méthode. Ce tableau pourra vous être utile plus tard.

Caractéristique des méthodes HTTP

Source : developer.mozilla.org

Structure détaillée d'une réponse HTTP

Les réponses HTTP sont également composées d'une ligne de départ (qu'on appelle également ligne de statut), d'éventuels en-têtes HTTP, d'une ligne vierge et d'un éventuel corps de réponse.

Exemple requete reponse HTTP

La ligne de statut est composée de la version du protocole, d'un code de statut HTTP indiquant l'état de complétion de la requête, son succès ou son échec et d'un court texte descriptif lié au code de statut.

Les en-têtes HTTP permettent à nouveau d'ajouter des informations de contexte et de préciser la réponse comme la date d'envoi de la réponse, des informations relatives au serveur, des informations relatives à la connexion ou encore la mise en place de cookies.

Finalement, la dernière partie d'une réponse HTTP est le corps et là non plus toutes les réponses n'en ont pas. Le corps d'une réponse HTTP va permettre par exemple de transmettre la ressource demandée par la requête HTTP.

Liste des codes de statut des réponses HTTP

Les codes de statut HTTP permettent de fournir des informations quant à la complétion (succès) ou la non-complétion (échec) d'une requête. Les statut code HTTP sont très utiles pour le débogage dans le cas où la requête n'aurait pas abouti et peuvent également être utilisés pour économiser des ressources.

Les codes de statut HTTP se décomposent en 5 grandes familles :

- Un code 1xx indique une réponse provisoire (non implémenté avec HTTP/1.0) ;
- Un code 2xx (200, 201, 202, 204) indique que la requête a été traitée avec succès ;
- Un code 3xx(300, 301, 302, 304) indique que la requête doit être redirigée ;
- Un code 4xx (400, 401, 403, 404) indique une erreur côté client ;
- Un code 5xx (500, 501, 502, 503) indique une erreur côté serveur.

Les codes de statut à connaître et à comprendre sont les suivants :

200 OK

Le code de statut 200 (OK) indique que la requête a été traitée avec succès. La réponse du serveur est donc envoyée sans problème.

301 Moved Permanently

Le statut code HTTP 301 (Moved Permanently) indique que la ressource demandée n'est plus disponible à l'URL indiquée dans la requête mais a été déplacée vers une autre URL (qui est alors indiquée dans un en-tête `Location`).

Lorsque le client reçoit une réponse avec un code 301, il va généralement effectuer une nouvelle requête avec la nouvelle URL fournie pour accéder à la ressource initialement demandée.

En tant que développeur, vous serez amené à paramétrer des redirections lors de changements d'URL. En effet, utiliser des redirections 301 permet notamment de conserver les liens gagnés sur l'URL d'origine. Nous reviendrons sur les redirections plus tard dans ce cours.

302 Found et 307

Les codes de statut HTTP 302 (Found) et 307 (Temporary Redirect) servent tous les deux à indiquer une redirection temporaire, c'est-à-dire une redirection qui n'a pas vocation à perdurer dans le temps. On va utiliser les redirections temporaires lors de maintenances notamment.

304 Not Modified

Le code de statut 304 (Not Modified) permet à un serveur d'indiquer à un client que la ressource demandée n'a pas été modifiée depuis la dernière visite et que celui-ci peut donc utiliser une version placée dans son cache local plutôt que de télécharger à nouveau la ressource à partir du serveur.

Cela permet de réduire la latence et permet donc à l'utilisateur d'obtenir la ressource demandée plus rapidement.

403 Forbidden

Le code de statut 403 (Forbidden) indique que le serveur a bien compris la requête mais refuse de la compléter. Ce statut peut être renvoyé suite à un certain nombre de tentative de connexion avec des mauvais identifiants par exemple.

404 Not Found

Le code de statut 404 (Not Found) indique que le serveur n'a pas trouvé la ressource demandée et ne peut donc pas compléter la requête.

500 Internal Server Error

Le code de statut 500 (Internal Server Error) indique que le serveur a rencontré une condition inattendue qui l'a empêché de répondre à la demande. Le problème n'est cette fois-ci plus situé côté client mais bien côté serveur.

Comment créer une requête ou une réponse HTTP en pratique ou « à quoi ça sert tout ça » ?

Ce point est important et devrait vous aider à y voir plus clair pour la suite de ce cours : en pratique, vous n'aurez en tant que développeur ou webmaster que rarement l'occasion de créer un message HTTP.

En effet, les requêtes HTTP sont formulées par le client (c'est-à-dire dans la majorité des cas le navigateur de vos visiteurs) et vous n'avez donc quasiment aucun contrôle dessus.

D'un autre côté, les réponses HTTP sont quant-à-elles créées par le serveur et vont en grande partie dépendre de la configuration de celui-ci. En fonction de votre hébergeur et de votre formule d'hébergement, vous allez avoir plus ou moins de contrôle sur la configuration de votre espace serveur.

L'intérêt principal d'apprendre le fonctionnement d'HTTP et de comprendre comment fonctionne le transfert de données dans son ensemble n'est pas de pouvoir directement intervenir dessus.

On apprend tout cela car ça aide grandement à comprendre comment fonctionne un site web, comment et pourquoi optimiser la vitesse de celui-ci et car cela aide à mieux comprendre les erreurs qui peuvent survenir pour pouvoir les régler plus rapidement et efficacement.

Ceci étant dit, nous allons apprendre dans ce cours à configurer correctement quelques en-têtes de réponse HTTP pour servir les objectifs décrits ci-dessus, et notamment les en-têtes liés aux redirections et au cache.

Déroulement d'une session HTTP

Le cycle requête-réponse HTTP s'insère dans un ensemble plus complet qu'on appelle « session HTTP ».

Une « session HTTP » correspond à l'ensemble des échanges qui se passent entre l'établissement d'une connexion et jusqu'à la déconnexion par le serveur.

Une session HTTP peut ainsi être classiquement divisée en 3 phases distinctes :

1. L'établissement de la connexion par le client ;
2. L'envoi d'une requête HTTP ;
3. L'envoi d'une réponse HTTP.

Pour pouvoir envoyer des données d'un point A (client) à un point B (serveur) et inversement, il faut avant tout établir une connexion entre ces deux entités. Cette connexion permet notamment aux deux machines de se mettre d'accord sur certaines modalités du transfert.

Notez bien ici que la phase de connexion n'est pas effectuée par HTTP puisque cela se trouve en dehors du champ d'influence de ce protocole. Pour être tout à fait précis, la connexion est gérée par les protocoles de la couche transport du modèle TCP/IP (ou OSI) et donc par le protocole TCP dans la majorité des cas pour HTTP jusqu'à sa version HTTP/2 (qui est la version de référence aujourd'hui).

Nous détaillerons les fonctions et enjeux de chaque couche des modèles TCP/IP et OSI et étudierons en détail les protocoles à connaître comme les protocoles TCP, IP, etc. dans la suite de ce cours. Pour l'instant, contentez-vous de retenir que l'établissement de la connexion ne dépend pas de HTTP et s'effectue à un autre niveau et à l'aide d'un autre protocole.

Dans les versions de HTTP précédant HTTP/1.1, le serveur fermait automatiquement la connexion dès l'envoi de la réponse, ce qui signifie qu'il fallait lancer une nouvelle connexion pour chaque nouvelle requête. Cette façon de procéder était bien évidemment très inefficace et créait de la latence.

Ce n'est plus le cas aujourd'hui : des procédés (que nous étudierons en détail plus tard) comme les connexions persistantes ou le multiplexing permettent de garder une connexion active et d'envoyer plusieurs requêtes d'affilée durant la même connexion.