Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP)	
Familia	Familia de protocolos de Internet
Función	Transferencia de hipertexto
Última versión	2.0
Puertos	80/TCP
Ubicación en la pila de protocolos	
Aplicación	**HTTP**
Transporte	TCP
Red	IP
Estándares	
RFC 1945 (HTTP/1.0, 1996)	
RFC 2616 (HTTP/1.1, 1999)	
RFC 2774 (HTTP/1.2, 2000)	
RFC 7540 (HTTP/2, 2015)	
[editar datos en Wikidata]	

Hypertext Transfer Protocol o **HTTP** (en español *protocolo de transferencia de hipertexto*) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web. HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force, colaboración que culminó en 1999 con la publicación de una serie de RFC, el másIMPORTANTE de ellos es el RFC 2616 que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de sesión, y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

Versiones

HTTP ha pasado por múltiples versiones del protocolo, muchas de las cuales son compatibles con las anteriores. El RFC 2145 describe el uso de los números de versión de HTTP. El cliente le dice al servidor al principio de la petición la versión que usa, y el servidor usa la misma o una anterior en su respuesta.

0.9 (lanzada en 1991)

Obsoleta. Soporta sólo un comando, GET, y además no especifica el número de versión HTTP. No soporta cabeceras. Como esta versión no soporta POST, el cliente no puede enviarle mucha información al servidor.

HTTP/1.0 (mayo de 1996)

Esta es la primera revisión del protocolo que especifica su versión en las comunicaciones, y todavía se usa ampliamente, sobre todo en servidores proxy. Permite los métodos de petición GET, HEAD y POST.

HTTP/1.1 (junio de 1999)[1] [2]

Versión más usada actualmente; Las conexiones persistentes están activadas por defecto y funcionan bien con los proxies. También permite al cliente enviar múltiples peticiones a la vez por la misma conexión (pipelining) lo que hace posible eliminar el tiempo de Round-Trip delay por cada petición.

HTTP/1.2 (febrero de 2000)

Los primeros borradores de 1995 del documento *PEP — an Extension Mechanism for HTTP* (el cuál propone el Protocolo de Extensión de Protocolo, abreviado PEP) los hizo el World Wide Web Consortium y se envió al Internet Engineering Task Force. El PEP inicialmente estaba destinado a convertirse en un rango distintivo de HTTP/1.2.[3] En borradores posteriores, sin embargo, se eliminó la referencia a HTTP/1.2. El RFC 2774 (experimental), *HTTP Extension Framework*, incluye en gran medida a PEP. Se publicó en febrero de 2000.

HTTP/2 (mayo de 2015)

En el año 2012 aparecen los primeros borradores de la nueva versión de HTTP (HTTP/2). Esta nueva versión no modifica la semántica de aplicación de http (todos los conceptos básicos continúan sin cambios). Sus mejoras se enfocan en como se empaquetan los datos y en el transporte. Por ejemplo, añade el uso de una única conexión, la compresión de cabeceras o el servicio 'server push'.

Descripción

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. El cliente (se le suele llamar "agente de usuario", en inglés *user agent*) realiza una petición enviando un mensaje, con cierto formato al servidor, El servidor (se le suele llamar un servidor web) le envía un mensaje de respuesta. Ejemplos de cliente son los navegadores web y las arañas web (también conocidas por su término inglés, *webcrawlers*).

Mensajes

Los mensajes HTTP son en texto plano lo que lo hace más legible y fácil de depurar. Esto tiene el inconveniente de hacer los mensajes más largos.

Los mensajes tienen la siguiente estructura:

- Línea inicial (termina con retorno de carro y un salto de línea) con
  - Para las peticiones: la acción requerida por el servidor (método de petición) seguido de la URL del recurso y la versión HTTP que soporta el cliente.
  - Para respuestas: La versión del HTTP usado seguido del código de respuesta (que indica que ha pasado con la

petición seguido de la URL del recurso) y de la frase asociada a dicho retorno.

- Las cabeceras del mensaje que terminan con una línea en blanco. Son metadatos. Estas cabeceras le dan gran flexibilidad al protocolo.
- Cuerpo del mensaje. Es opcional. Su presencia depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL. Típicamente tiene los datos que se intercambian cliente y servidor Por ejemplo para una petición podría contener ciertos datos que se quieren enviar al servidor para que los procese. Para una respuesta podría incluir los datos que el cliente ha solicitado.

Métodos de petición

Un pedido HTTP usando telnet. El pedido (*request*), cabeceras de respuesta (*response headers*) y el cuerpo de la respuesta (*response body*) están resaltados.

HTTP define una serie predefinida de métodos de petición (algunas veces referido como "verbos") que pueden utilizarse. El protocolo tiene flexibilidad para ir añadiendo nuevos métodos y para así añadir nuevas funcionalidades. El número de métodos de petición se han ido aumentando según se avanzaban en las versiones[4] . Cada método indica la acción que desea que se efectúe sobre el recurso identificado. Lo que este recurso representa depende de la aplicación del servidor. Por ejemplo el recurso puede corresponderse con un archivo que reside en el servidor.

HEAD

RFC 2616. Pide una respuesta idéntica a la que correspondería a una petición GET, pero en la petición no se devuelve el cuerpo. Esto es útil para poder recuperar los metadatos de los encabezados de respuesta, sin tener que transportar todo el contenido.

GET

RFC 2616. Pide una representación del recurso especificado. Por seguridad **no debería** ser usado por aplicaciones que causen efectos ya que transmite información a través de la URI agregando parámetros a la URL. La petición puede ser simple, es decir en una línea o compuesta de la manera que muestra el ejemplo. Ejemplo:

`GET /images/logo.png HTTP/1.1` obtiene un recurso llamado logo.png

Ejemplo con parámetros:

`/index.php?page=main&lang=es`

POST

RFC 2616. Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.

PUT

RFC 2616. Sube, carga o realiza un upload de un recurso especificado (archivo), es el camino más eficiente para subir archivos a un servidor, esto es porque en POST utiliza un **mensaje multiparte** y el mensaje es decodificado por el servidor. En contraste, el método PUT te permite escribir un archivo en una conexión socket establecida con el servidor. La desventaja del método PUT es que los servidores de hosting compartido no lo tienen habilitado. Ejemplo:

```
PUT /path/filename.html HTTP/1.1
```

DELETE

RFC 2616. Borra el recurso especificado.

TRACE

RFC 2616. Este método solicita al servidor que en la respuesta meta todos los datos que reciba en el mensaje de petición. Se utiliza con fines de depuración y diagnóstico ya que el cliente puede ver lo que llega al servidor y de esta forma ver lo que añaden al mensaje los servidores intermedios

OPTIONS

RFC 2616. Devuelve los métodos HTTP que el servidor soporta para un URL específico. Esto puede ser utilizado para comprobar la funcionalidad de un servidor web mediante petición en lugar de un recurso específico.

CONNECT

RFC 2616. Se utiliza para saber si se tiene acceso a un host, no necesariamente la petición llega al servidor, este método se utiliza principalmente para saber si un proxy nos da acceso a un host bajo condiciones especiales, como por ejemplo "corrientes" de datos bidireccionales encriptadas (como lo requiere SSL).

PATCH

[2]

SEARCH

[3]

COPY

RFC 2518

LOCK

RFC 2323

UNLOCK

RFC 2518

MOVE

RFC 2518

MKCOL

RFC 2518

PROPFIND

RFC 2518

PROPPATCH

RFC 2518

MERGE

RFC 3253

UPDATE

RFC 3253

LABEL

RFC 3253

Códigos de respuesta

Artículo principal: Anexo:Códigos de estado HTTP

El código de respuesta o retorno es un número que indica que ha pasado con la petición. El resto del contenido de la respuesta dependerá del valor de este código. El sistema es flexible y de hecho la lista de códigos ha ido aumentando para así adaptarse a los cambios e identificar nuevas situaciones. Cada código tiene un significado concreto. Sin embargo el número de los códigos están elegidos de tal forma que según si pertenece a una centena u otra se pueda identificar el tipo de respuesta que ha dado el servidor:

- Códigos con formato 1xx: Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.
- Códigos con formato 2xx: Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- Códigos con formato 3xx: Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- Códigos con formato 4xx: Errores causados por el cliente. Indica que ha habido un error en el procesado de la petición a causa de que el cliente ha hecho algo mal.
- Códigos con formato 5xx: Errores causados por el servidor. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor.

Cabeceras

Artículo principal: Cabeceras HTTP

Son los metadatos que se envían en las peticiones o respuesta HTTP para proporcionar información esencial sobre la transacción en curso. Cada cabecera es especificada por un nombre de cabecera seguido por dos puntos, un espacio en blanco y el valor de dicha cabecera seguida por un retorno de carro seguido por un salto de línea. Se usa una línea en blanco para indicar el final de las cabeceras. Si no hay cabeceras la línea en blanco debe permanecer.

Las cabeceras le dan gran flexibilidad al protocolo permitiendo añadir nuevas funcionalidades sin tener que cambiar la base. Por eso según han ido sucediendo las versiones de HTTP se han ido añadiendo más y más cabeceras permitidas.

Las cabeceras pueden tener metadatos que tienen que ser procesados por el cliente (ej. en respuesta a petición se puede indicar el tipo del contenido que contiene), por el servidor (ej. tipos de representaciones aceptables por el cliente del contenido que pide) o por los intermediarios (ej. como gestionar el cacheo por parte de los proxys)

Dependiendo del tipo de mensaje en el que puede ir una cabecera las podemos clasificar en cabeceras de petición, cabeceras de respuesta y cabeceras que pueden ir tanto en un petición como en una respuesta.

Podemos clasificar las cabeceras según su función. Por ejemplo:

- Cabeceras que indican las capacidades aceptadas por el que envía el mensaje: **Accept** (indica el MIME aceptado), **Accept-Charset** (indica el código de caracteres aceptado), **Accept-Encoding** (indica el método de compresión aceptado), **Accept-Language** (indica el idioma aceptado), **User-Agent** (para describir al cliente), **Server** (indica el tipo de servidor), **Allow** (métodos permitidos para el recurso)
- Cabeceras que describen el contenido: **Content-Type** (indica el MIME del contenido), **Content-Length** (longitud del mensaje), **Content-Range**, **Content-Encoding**, **Content-Language**, **Content-Location**.
- Cabeceras que hacen referencias a URIs: **Location** (indica donde está el contenido), **Referer** (Indica el origen de la petición).
- Cabeceras que permiten ahorrar transmisiones: **Date** (fecha de creación), **If-Modified-Since**, **If-Unmodified-Since**, **If-Match**, **If-None-Match**, **If-Range**, **Expires**, **Last-Modified**, **Cache-Control**, **Via**, **Pragma**, **Etag**, **Age**, **Retry-After**.
- Cabeceras para control de cookies: **Set-Cookie**, **Cookie**
- Cabeceras para autentificación: **Authorization**, **WW-Authenticate**
- Cabeceras para describir la comunicación: **Host** (indica máquina destino del mensaje), **Connection** (indica como establecer la conexión)
- Otras: **Range** (para descargar sólo partes del recurso), **Max-Forward** (límite de cabeceras añadidas en TRACE).

Ejemplo de diálogo HTTP

Para obtener un recurso con el URL http://www.example.com/index.html

1. Se abre una conexión en el puerto 80 del host www.example.com en. El puerto 80 es el puerto predefinido para HTTP. Si se quisiera utilizar el puerto XXXX habría que codificarlo en la URL de la forma http://www.example.com:XXXX/index.html
2. Se envía un mensaje en el estilo siguiente:

```
 GET /index.html HTTP/1.1
 Host: www.example.com
 User-Agent: nombre-cliente
 Referer: www.google.com
 User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:45.0) Gecko/20100101 Firefox/45.0
 Connection: keep-alive
 [Línea en blanco]
```

La respuesta del servidor está formada por encabezados seguidos del recurso solicitado, en el caso de una página web:

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221

<html lang="eo">
<head>
<meta charset="utf-8">
<title>Título del sitio</title>
</head>
<body>
<h1>Página principal de tuHost</h1>
(Contenido)
  .
  .
  .
</body>
</html>
```