**UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE**

**SAN FRANCISCO XAVIER DE CHUQUISACA**



**FACULTAD DE TECNOLOGÍA**

**CARRERA DE INGENIERÍA EN CIENCIAS DE LA COMPUTACION**

**Tarea 1 Protocolo HTTP 1 y 2**

**Sis-256 Tecnologías y Desarrollo Web**

**Alumno: Igor Santiago Navarro Balanza**

Sucre, agosto de 2019

**Tabla de Contenido**

1. Protocolo HTTP 1………………………………………………………………………………………………………………………3

2. Protocolo HTTP 2………………………………………………………………………………………………………………………3

3. Estructura…………………………………………………………………………………………………………………………………3

4. Contenido de petición………………………………………………………………………………………………………………4

5. Contenido de respuesta……………………………………………………………………………………………………………4

6. Versiones…………………………………………………………………………………………………………………………………10

7. Novedades HTTP 2…………………………………………………………………………………………………………………..13

8. Bibliografía………………………………………………………………………………………………………………………………14

**1. Protocolo HTTP 1**

El Protocolo de Transferencia de HiperTexto (Hypertext Transfer Protocol) es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP. La especificación completa del protocolo HTTP 1/0 está recogida en el RFC 1945. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como el World Wide Web.

Desde el punto de vista de las comunicaciones, está soportado sobre los servicios de conexión TCP/IP, y funciona de la misma forma que el resto de los servicios comunes de los entornos UNIX: un proceso servidor escucha en un puerto de comunicaciones TCP (por defecto, el 80), y espera las solicitudes de conexión de los clientes Web. Una vez que se establece la conexión, el protocolo TCP se encarga de mantener la comunicación y garantizar un intercambio de datos libre de errores.

HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan; cada objeto Web (documento HTML, fichero multimedia o aplicación CGI) es conocido por su URL.

**2. Protocolo HTTP 2**

HTTP/2 es un protocolo binario que conserva la misma semántica que el protocolo HTTP1.X lo que significa que todos los verbos, cabeceras, etc. siguen funcionando sin cambios. De hecho, HTTP/2 busca resolver los defectos que tiene la comunicación a través TCP (la capa de transporte dentro del protocolo HTTP).  
Muchos consideran a HTTP/2 el reemplazo del protocolo [SPDY](https://es.wikipedia.org/wiki/SPDY) que desarrollo Google para mejorar el rendimiento de sus servicios en su navegador Chrome, de hecho el protocolo HTTP/2 está basado en algunas de las ideas del protocolo SPDY, el cual actualmente se considera obsoleto pues se ha apostado completamente por el protocolo HTTP/2.

**Ventajas**

Su navegador seguirá enviando peticiones a un servidor y obtendrá respuestas con lo necesario para para renderizar la página web como se debe, pero algunos matices por detrás van a cambiar.

HTTP/2 trae nuevas características tales como multiplexed streams, server push, la compresión de HEADERS y el formato binario.

**3. Estructura**

Fue diseñado para transmitir HTML (HyperText Markup Language) pero hoy en día se utiliza para transmitir todo tipo de documentos (imágenes, audio, video, PDF, etc.) y para crear aplicaciones Web.

Un protocolo es un documento que define las reglas y la estructura de los mensajes que se van a intercambiar entre máquinas.

Además de HTTP, ejemplos de protocolos incluyen SMTP (Simple Mail Transfer Protocol) para transmitir mensajes de correo electrónico y FTP (File Transfer Protocolo) para transmitir archivos.

HTTP es un protocolo cliente-servidor, lo que significa que el cliente envía una petición al servidor y espera un mensaje de respuesta del servidor. Es un protocolo sin estado, lo que significa que el servidor no guarda información del cliente, cada petición es independiente de las demás.

Un mensaje HTTP (no importa si es de petición o respuesta) se compone de 3 partes:

* La primera línea (que es diferente para la petición y la respuesta).
* Los encabezados.
* El cuerpo (opcional)

**4. Contenido de una petición (request)**

La primera línea de un mensaje de petición empieza con un verbo (también se le conoce como método). Los verbos definen la acción que se quiere realizar sobre el recurso. Los verbos más comunes son:

* GET: se utiliza para solicitar un recurso.
* POST: se utiliza para publicar un recurso.
* PUT: se utiliza para reemplazar un recurso.
* DELETE: se utiliza para eliminar un recurso.

Existen otros, pero estos son los más comunes.

Cuando ingresas a una página desde un navegador, por debajo el navegador envía un mensaje GET, lo mismo cuando oprimes un vínculo a otra página.

**5. Contenido de respuesta (response)**

La primera línea de un mensaje de respuesta tiene un código de 3 dígitos que le indica al cliente cómo interpretar la respuesta.

Los códigos de respuesta se dividen en cinco categorías dependiendo del dígito con el que inician:

* 1XX: Información
* 2XX: Éxito
* 3XX: Redirección
* 4XX: Error en el cliente
* 5XX: Error en el servidor

**Respuestas informativas**

[**100 Continue**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/100)

* Esta respuesta provisional indica que todo hasta ahora está bien y que el cliente debe continuar con la solicitud o ignorarla si ya está terminada.

[**101 Switching Protocol**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/101)

* Este código se envía en respuesta a un encabezado de solicitud [Upgrade](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Upgrade" \o "La documentación acerca de este tema no ha sido escrita todavía . ¡Por favor  considera contribuir !) por el cliente e indica que el servidor acepta el cambio de protocolo propuesto por el agente de usuario.

[**102 Processing**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/102)**([WebDAV](https://developer.mozilla.org/es/docs/Glossary/WebDAV" \o "La definición de ese término (WebDAV) no se ha escrito todavía; por favor considera contribuir escribiendola!))**

* Este código indica que el servidor ha recibido la solicitud y aún se encuentra procesandola, por lo que no hay respuesta disponible.

**Respuestas satisfactorias**

[**200 OK**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/200)

* La solicitud ha tenido éxito. El significado de un éxito varía dependiendo del método HTTP:  
  GET: El recurso se ha obtenido y se transmite en el cuerpo del mensaje.  
  HEAD: Los encabezados de entidad están en el cuerpo del mensaje.  
  PUT o POST: El recurso que describe el resultado de la acción se transmite en el cuerpo del mensaje.  
  TRACE: El cuerpo del mensaje contiene el mensaje de solicitud recibido por el servidor.

[**201 Created**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/201)

* La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello. Ésta es típicamente la respuesta enviada después de una petición PUT.

[**202 Accepted**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/202)

* La solicitud se ha recibido, pero aún no se ha actuado. Es una petición "Sin compromiso", lo que significa que no hay manera en HTTP que permita enviar una respuesta asíncrona que indique el resultado del procesamiento de la solicitud. Está pensado para los casos en que otro proceso o servidor maneja la solicitud, o para el procesamiento por lotes.

[**203 Non-Authoritative Information**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/203)

* La petición se ha completado con éxito, pero su contenido no se ha obtenido de la fuente originalmente solicitada, sino que se recoge de una copia local o de un tercero. Excepto esta condición, se debe preferir una respuesta de 200 OK en lugar de esta respuesta.

[**204 No Content**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/204)

* La petición se ha completado con éxito pero su respuesta no tiene ningún contenido, aunque los encabezados pueden ser útiles. El agente de usuario puede actualizar sus encabezados en caché para este recurso con los nuevos valores.

[**205 Reset Content**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/205)

* La petición se ha completado con éxito, pero su respuesta no tiene contenidos y además, el agente de usuario tiene que inicializar la página desde la que se realizó la petición, este código es útil por ejemplo para páginas con formularios cuyo contenido debe borrarse después de que el usuario lo envíe.

[**206 Partial Content**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/206)

* La petición servirá parcialmente el contenido solicitado. Esta característica es utilizada por herramientas de descarga como wget para continuar la transferencia de descargas anteriormente interrumpidas, o para dividir una descarga y procesar las partes simultáneamente.

[**207 Multi-Status**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/207)**([WebDAV](https://developer.mozilla.org/es/docs/Glossary/WebDAV" \o "La definición de ese término (WebDAV) no se ha escrito todavía; por favor considera contribuir escribiendola!))**

* Una respuesta Multi-Estado transmite información sobre varios recursos en situaciones en las que varios códigos de estado podrían ser apropiados. El cuerpo de la petición es un mensaje XML.

[**208 Multi-Status**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/208)**([WebDAV](https://developer.mozilla.org/es/docs/Glossary/WebDAV" \o "La definición de ese término (WebDAV) no se ha escrito todavía; por favor considera contribuir escribiendola!))**

* El listado de elementos DAV ya se notificó previamente, por lo que no se van a volver a listar.

[**226 IM Used**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/226)**(**[**HTTP Delta encoding**](https://tools.ietf.org/html/rfc3229)**)**

* El servidor ha cumplido una petición GET para el recurso y la respuesta es una representación del resultado de una o más manipulaciones de instancia aplicadas a la instancia actual.

**Redirecciones**

[**300 Multiple Choice**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/300)

* Esta solicitud tiene más de una posible respuesta. User-Agent o el usuario debe escoger uno de ellos. No hay forma estandarizado de seleccionar una de las respuestas.

[**301 Moved Permanently**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/301)

* Este código de respuesta significa que la URI  del recurso solicitado ha sido cambiado. Probablemente una nueva URI sea devuelta en la respuesta.

[**302 Found**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/302)

* Este código de respuesta significa que el recurso de la URI solicitada ha sido cambiado temporalmente. Nuevos cambios en la URI serán agregados en el futuro. Por lo tanto, la misma URI debe ser usada por el cliente en futuras solicitudes.

[**303 See Other**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/303)

* El servidor envía esta respuesta para dirigir al cliente a un nuevo recurso solicitado a otra dirección usando una petición GET.

[**304 Not Modified**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/304)

* Esta es usada para propositos de "caché". Le indica al cliente que la respuesta no ha sido modificada. Entonces, el cliente puede continuar usando la misma versión almacenada en su caché.

**305 Use Proxy**

* Fue definida en una versión previa de la especificación del protocolo HTTP para indicar que una respuesta solicitada debe ser accedida desde un proxy. Ha quedado obsoleta debido a preocupaciones de seguridad correspondientes a la configuración de un proxy.

**306 unused**

* Este código de respuesta ya no es usado más. Actualmente se encuentra reservado. Fue usado en previas versiones de la especificación HTTP1.1.

[**307 Temporary Redirect**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/307)

* El servidor envía esta respuesta para dirigir al cliente a obtener el recurso solicitado a otra URI con el mismo método que se usó la petición anterior. Tiene la misma semántica que el código de respuesta HTTP 302 Found, con la excepción de que el agente usuario *no debe* cambiar el método HTTP usado: si un POST fue usado en la primera petición, otro POST debe ser usado en la segunda petición.

[**308 Permanent Redirect**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/308)

* Significa que el recurso ahora se encuentra permanentemente en otra URI, especificada por la respuesta de encabezado HTTP Location:. Tiene la misma semántica que el código de respuesta HTTP 301 Moved Permanently, con la excepción de que el agente usuario *no debe* cambiar el método HTTP usado: si un POST fue usado en la primera petición, otro POST debe ser usado en la segunda petición.

**Errores de cliente**

[**400 Bad Request**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/400)

* Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.

[**401 Unauthorized**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/401)

* Es necesario autenticar para obtener la respuesta solicitada. Esta es similar a 403, pero en este caso, autenticación es posible.

**402 Payment Required**

* Este código de respuesta está reservado para futuros usos. El objetivo inicial de crear este código fue para ser utilizado en sistemas digitales de pagos. Sin embargo, no está siendo usado actualmente.

[**403 Forbidden**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/403)

* El cliente no posee los permisos necesarios para cierto contenido, por lo que el servidor está rechazando otorgar una respuesta apropiada.

[**404 Not Found**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/404)

* El servidor no pudo encontrar el contenido solicitado. Este código de respuesta es uno de los más famosos dada su alta ocurrencia en la web.

[**405 Method Not Allowed**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/405)

* El método solicitado es conocido por el servidor pero ha sido deshabilitado y no puede ser utilizado. Los dos métodos obligatorios, GET y HEAD, nunca deben ser deshabilitados y no debiesen retornar este código de error.

[**406 Not Acceptable**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/406)

* Esta respuesta es enviada cuando el servidor, despues de aplicar una [negociación de contenido servidor-impulsado](https://developer.mozilla.org/en-US/docs/HTTP/Content_negotiation#Server-driven_negotiation), no encuentra ningún contenido seguido por la criteria dada por el usuario.

[**407 Proxy Authentication Required**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/407)

* Esto es similar al código 401, pero la autenticación debe estar hecha a partir de un proxy.

[**408 Request Timeout**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/408)

* Esta respuesta es enviada en una conexión inactiva en algunos servidores, incluso sin alguna petición previa por el cliente. Significa que el servidor quiere desconectar esta conexión sin usar. Esta respuesta es muy usada desde algunos navegadores, como Chrome, Firefox 27+, o IE9, usa mecanismos de pre-conexión HTTP para acelerar la navegación. También hay que tener cuenta que algunos servidores simplemente desconectan la conexión sin enviar este mensaje.

[**409 Conflict**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/409)

* Esta respuesta puede ser enviada cuando una petición tiene conflicto con el estado actual del servidor.

[**410 Gone**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/410)

* Esta respuesta puede ser enviada cuando el contenido solicitado ha sido borrado del servidor.

[**411 Length Required**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/411)

* El servidor rechaza la petición porque el campo de encabezado Content-Length no esta definido y el servidor lo requiere.

[**412 Precondition Failed**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/412)

* El cliente ha indicado pre-condiciones en sus encabezados la cual el servidor no cumple.

[**413 Payload Too Large**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/413)

* La entidad de petición es más larga que los limites definidos por el servidor; el servidor puede cerrar la conexión o retornar un campo de encabezado Retry-After.

[**414 URI Too Long**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/414)

* La URI solicitada por el cliente es más larga que el servidor está dispuesto a interpretar.

[**415 Unsupported Media Type**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/415)

* El formato multimedia de los datos solicitados no está soportada por el servidor, por lo cual el servidor rechaza la solicitud.

[**416 Requested Range Not Satisfiable**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/416)

* El rango especificado por el campo de encabezado Range en la solicitud no cumple; es posible que el rango está fuera del tamaño de los datos objetivo del URI.

[**417 Expectation Failed**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/417)

* Significa que la expectativa indicada por el campo de encabezado Expect solicitada no puede ser cumplida por el servidor.

[**418 I'm a teapot**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/418)

* El servidor se reúsa a intentar hacer café con una tetera.

[**421 Misdirected Request**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/421)

* La petición fue dirigida a un servidor que no es capaz de producir una respuesta. Esto puede ser enviado por un servidor que no esta configurado para producir respuestas por la combinación del esquema y la autoridad que estan incluidos en la URI solicitada

[**422 Unprocessable Entity**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/422)**([WebDAV](https://developer.mozilla.org/es/docs/Glossary/WebDAV" \o "La definición de ese término (WebDAV) no se ha escrito todavía; por favor considera contribuir escribiendola!))**

* La petición estaba bien formada pero no se pudo seguir debido a errores de semántica.

[**423 Locked**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/423)**([WebDAV](https://developer.mozilla.org/es/docs/Glossary/WebDAV" \o "La definición de ese término (WebDAV) no se ha escrito todavía; por favor considera contribuir escribiendola!))**

* El recurso que está siendo accedido está bloqueado.

[**424 Failed Dependency**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/424)**([WebDAV](https://developer.mozilla.org/es/docs/Glossary/WebDAV" \o "La definición de ese término (WebDAV) no se ha escrito todavía; por favor considera contribuir escribiendola!))**

* La petición falló debido a una falla de una petición previa.

[**426 Upgrade Required**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/426)

* El servidor se reúsa a aplicar la solicitud usando el protocolo actual pero puede estar dispuesto a hacerlo después que el cliente se actualize a un protocolo diferente. El servidor envía un encabezado Upgrade en una respuesta para indicar los protocolos requeridos.

[**428 Precondition Required**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/428)

* El servidor origen requiere que la solicitud sea condicional. Tiene la intención de prevenir problemas de 'actualización perdida', donde un cliente OBTIENE un estado del recurso, lo modifica, y lo PONE devuelta al servidor, cuando mientras un tercero ha modificado el estado del servidor, llevando a un conflicto.

[**429 Too Many Requests**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/429)

* El usuario ha enviado demasiadas solicitudes en un periodo de tiempo dado.

[**431 Request Header Fields Too Large**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/431)

* El servidor no está dispuesto a procesar la solicitud porque los campos de encabezado son demasiado largos. La solicitud PUEDE volver a subirse después de reducir el tamaño de los campos de encabezado solicitados.

[**451 Unavailable For Legal Reasons**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/451)

* El usuario solicita un recurso ilegal, como alguna página web censurada por algún gobierno.

**Errores de servidor**

[**500 Internal Server Error**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/500)

* El servidor ha encontrado una situación que no sabe cómo manejarla.

[**501 Not Implemented**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/501)

* El método solicitado no está soportado por el servidor y no puede ser manejada. Los únicos métodos que los servidores requieren soporte (y por lo tanto no deben retornar este código) son GET y HEAD.

[**502 Bad Gateway**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/502)

* Esta respuesta de error significa que el servidor, mientras trabaja como una puerta de enlace para obtener una respuesta necesaria para manejar la petición, obtuvo una respuesta inválida.

[**503 Service Unavailable**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/503)

* El servidor no está listo para manejar la petición. Causas comunes puede ser que el servidor está caído por mantenimiento o está sobrecargado. Hay que tomar en cuenta que, junto con esta respuesta, una página usuario-amigable explicando el problema debe ser enviada. Estas respuestas deben ser usadas para condiciones temporales y el encabezado HTTP Retry-After: debería, si es posible, contener el tiempo estimado antes de la recuperación del servicio. El webmaster debe también cuidar los encabezados relacionados al caché que son enviados junto a esta respuesta, ya que estas respuestas de condición temporal deben usualmente no estar en el caché.

[**504 Gateway Timeout**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/504)

* Esta respuesta de error es dada cuando el servidor está actuando como una puerta de enlace y no puede obtener una respuesta a tiempo.

[**505 HTTP Version Not Supported**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/505)

* La versión de HTTP usada en la petición no está soportada por el servidor.

[**506 Variant Also Negotiates**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/506)

* El servidor tiene un error de configuración interna: negociación de contenido transparente para la petición resulta en una referencia circular.

[**507 Insufficient Storage**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/507)

* El servidor tiene un error de configuración interna: la variable de recurso escogida esta configurada para acoplar la negociación de contenido transparente misma, y no es por lo tanto un punto final adecuado para el proceso de negociación.

[**508 Loop Detected**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/508)**([WebDAV](https://developer.mozilla.org/es/docs/Glossary/WebDAV" \o "La definición de ese término (WebDAV) no se ha escrito todavía; por favor considera contribuir escribiendola!))**

* El servidor detectó un ciclo infinito mientras procesaba la solicitud.

[**510 Not Extended**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/510)

* Extensiones adicionales para la solicitud son requeridas para que el servidor las cumpla.

[**511 Network Authentication Required**](https://developer.mozilla.org/es/docs/Web/HTTP/Status/511)

* El código de estado 511 indica que el cliente necesita auntenticar para ganar acceso a la red.

**6. Versiones**

HTTP es el protocolo en el que se basa la Web. Fue inventado por Tim Berners-Lee entre los años 1989-1991, HTTP ha visto muchos cambios, manteniendo la mayor parte de su simplicidad y desarrollando su flexibilidad. HTTP ha evolucionado, desde un protocolo destinado al intercambio de archivos en un entorno de un laboratorio semi-seguro, al actual laberinto de Internet, sirviendo ahora para el intercambio de imágenes, vídeos en alta resolución y en 3D.

**Invención de la World Wide Web**[**Sección**](https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#Invenci%C3%B3n_de_la_World_Wide_Web)

En 1989, mientras trabajaba en el CERN, Tim Berners-Lee escribió una propuesta para desarrollar un sistema de hipertexto sobre Internet. Inicialmente lo llamó: '*Mesh'*(malla, en inglés), y posteriormente se renombró como*World Wide Web*(red mundial), durante su implementación en 1990. Desarrollado sobre los protocolos existentes TCP e IP, está basado en cuatro bloques:

* Un formato de texto para representar documentos de hiper-texto: [HyperText Markup Language](https://developer.mozilla.org/en-US/docs/Web/HTML) (HTML).
* Un protocolo sencillo para el intercambio de esos documentos, del inglés: *HypertText Transfer Protocol*(HTTP) : protocolo de transferencia de hiper-texto.
* Un cliente que muestre (e incluso pueda editar) esos documentos. El primer navegador Web, llamado: *WorldWideWeb*.
* Un servidor para dar acceso a los documentos, una versión temprana: *httpd (http daemon)*

Estos cuatro bloques fundamentales se finalizaron para finales de 1990, y los primeros servidores estaban ya funcionando fuera del CERN a principios del 1991. El 6 de Agosto de 1991, el [post](https://groups.google.com/forum/#!msg/alt.hypertext/eCTkkOoWTAY/urNMgHnS2gYJ) de Tim Berners-Lee, se considera actualmente como el inicio oficial de la Web como proyecto público.

La versión del protocolo HTTP usada en aquel momento, era realmente muy sencilla, posteriormente pasó a HTTP/0.9, referido algunas veces, como el protocolo de una sola línea.

**HTTP/0.9 – El protocolo de una sola línea**

La versión inicial de HTTP, no tenía número de versión; aunque posteriormente se la denominó como 0.9 para distinguirla de las versiones siguientes. HTTP/0.9 es un protocolo extremadamente sencillo: una petición consiste simplemente en una única linea, que comienza por el único método posible [GET](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/GET), seguido por la dirección del recurso a pedir (no la URL, ya que tanto el protocolo, el servidor y el puerto, no son necesarios una vez ya se ha conectado al servidor).

Al contrario que sus posteriores evoluciones, el protocolo HTTP/0.9 no usa cabeceras HTTP, con lo cual únicamente es posible transmitir archivos HTML, y ningún otro tipo de archivos. Tampoco había información del estado ni códigos de error: en el caso un problema, el archivo HTML pedido, era devuelto con una descripción del problema dentro de él, para que una persona pudiera analizarlo.

La versión HTTP/0.9 era ciertamente limitada y tanto los navegadores como los servidores, pronto ampliaron el protocolo para que fuera más flexible.

* La versión del protocolo se envía con cada petición: HTTP/1.0 se añade a la línea de la petición GET.
* Se envía también un código de estado al comienzo de la respuesta, permitiendo así que el navegador pueda responder al éxito o fracaso de la petición realizada, y actuar en consecuencia (como actualizar el archivo o usar la caché local de algún modo).
* El concepto de cabeceras de HTTP, se presentó tanto para las peticiones como para las respuestas, permitiendo la trasmisión de meta-data y conformando un protocolo muy versátil y ampliable.
* Con el uso de las cabeceras de HTTP, se pudieron transmitir otros documentos además de HTML, mediante la cabecera [Content-Type](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Content-Type).

Estas innovaciones, no se desarrollaron de forma planeada, sino más bien con una aproximación de prueba y error, entre los años 1991 y 1995: un servidor y un navegador, añadían una nueva funcionalidad y se evaluaba su aceptación. Debido a esto, en ese periodo eran muy comunes los problemas de interoperatividad.  En noviembre de 1996, para poner fin a estos problemas se publicó un documento informativo que describía las prácticas adecuadas, [RFC 1945](https://tools.ietf.org/html/rfc1945). Esté documento es la definición del protocolo HTTP/1.0. Resulta curioso, que realmente no es un estándar oficial.

**HTTP/1.1 – El protocolo estándar.**[**Sección**](https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#HTTP1.1_%E2%80%93_El_protocolo_est%C3%A1ndar.)

En paralelo al uso, un poco desordenado, y las diversas implementaciones de HTTP/1.0, y desde el año 1995, un año antes de la publicación del documento del HTTP/1.0, un proceso de estandarización formal ya estaba en curso. La primera versión estandarizada de HTTP: el protocolo HTTP/1.1, se publicó en 1997, tan solo unos meses después del HTTP/1.0

HTTP/1.1 aclaró ambigüedades y añadió numerosas mejoras:

* Una conexión podía ser reutilizada, ahorrando así el tiempo de re-abrirla repetidas veces para mostrar los recursos empotrados dentro del documento original pedido.
* Enrutamiento ('Pipelining' en inglés) se añadió a la especificación, permitiendo realizar una segunda petición de datos, antes de que fuera respondida la primera, disminuyendo de este modo la latencia de la comunicación.
* Se permitió que las respuestas a peticiones podían ser divididas en sub-partes.
* Se añadieron controles adicionales a los mecanismos de gestión de la cache.
* La negociación de contenido, incluyendo el lenguaje, el tipo de codificación, o tipos, se añadieron a la especificación, permitiendo que servidor y cliente, acordasen el contenido más adecuado a intercambiarse.
* Gracias a la cabecera, [Host](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Host), pudo ser posible alojar varios dominios en la misma dirección IP.

**HTTP/2 – Un protocolo para un mayor rendimiento**

A lo largo de los años, las páginas Web han llegado a ser mucho más complejas, incluso llegando a poder considerarse como aplicaciones por derecho propio. La cantidad de contenido visual, el tamaño de los scripts, y los scripts que añaden interactividad ha aumentado mucho también. Muchos más datos son transmitidos bajo muchas más peticiones HTTP. Las conexiones HTTP/1.1 han de enviar las peticiones HTTP en el orden correcto. Teóricamente, sería posible usar varias conexiones en paralelo (normalmente entre 5 y 8), aumentando consecuentemente la complejidad del proceso. Por ejemplo, el HTTP 'pipelining' ha demostrado ser un lastre para el desarrollo Web.

En la primera mitad de la década de 2010, Google demostró un proceso alternativo para el intercambio de data entre clientes y servidores, implementando el protocolo experimental SPDY (pronunciado como en inglés *'speedy'*). Este atrajo mucho interés por los desarrolladores de tanto los navegadores como los servidores. Definiendo una mejora en los tiempos de respuesta, y resolviendo el problema de datos duplicados transmitidos. SPDY sirvió como base para el desarrollo del protocolo HTTP/2.

El protocolo HTTP/2, tiene notables diferencias fundamentales respecto a la versión anterior HTTP/1.1

* Es un protocolo binario, en contraposición a estar formado por cadenas de texto, tal y como estaban basados sus protocolos anteriores. Así pues, no se puede leer directamente, ni crear manualmente A pesar de este inconveniente, gracias a este cambio es posible utilizar en él técnicas de optimización.
* Es un protocolo multiplexado. Peticiones paralelas pueden hacerse sobre la misma conexión, no está sujeto pues a mantener el orden de los mensajes, ni otras restricciones que tenían los protocolos anteriores HTTP/1.x
* Comprime las cabeceras, ya que estas, normalmente son similares en un grupo de peticiones. Esto elimina la duplicación y retardo en los datos a transmitir.
* Esto permite al servidor almacenar datos en la caché del cliente, previamente a que estos sean pedidos, mediante un mecanismo denominado '*server push*'.

Estandarizado de manera oficial en mayo de 2015, HTTP/2 ha conseguido muchos éxitos. En Julio de 2016, un 8.7% de todos los sitios Web[[1]](https://w3techs.com/technologies/details/ce-http2/all/all) estaban usándolo ya, representando más del 68% de todo su tráfico[[2]](https://www.keycdn.com/blog/http2-statistics/). Los sitios Web con mucho tráfico, fueron aquellos que lo adoptaron más rápidamente, ahorrando considerablemente las sobrecargas en la transferencia de datos, ... y en sus presupuestos.

Esta rápida adopción era esperada, ya que el uso de HTTP/2, no requiere de una adaptación de los sitios Web y aplicaciones: el uso de HTTP/1.1 o HTTP/2 es transparente para ellos. El uso de un servidor actual, comunicándose con un navegador actualizado, es suficiente para permitir su uso: únicamente en casos particulares fue necesario impulsar su utilización; y según se actualizan servidores y navegadores antiguos, su utilización aumenta, sin que requiera un mayor esfuerzo de los desarrolladores Web.

**Post-evolución del HTTP/2**[**Sección**](https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#Post-evoluci%C3%B3n_del_HTTP2)

Con la publicación de la versión del protocolo HTTP/2, esté no ha dejado de evolucionar. Como con el HTTP/1.x, anteriormente, la extensibilidad del HTTP se sigue usando para añadir nuevas funcionalidades. Podemos enumerar algunas de estas nuevas características que se desarrollaron en el año 2016:

* Soporte de la cabecera [Alt-Svc](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Alt-Svc), la cual permite disociar la identificación de una ubicación, con respecto a un recurso pedido, permitiendo el uso más inteligente de los mecanismos de cacheo de memoria de los [CDN](https://developer.mozilla.org/es/docs/Glossary/CDN).
* La introducción de la cabecera [Client-Hints](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Client-Hints), que permite al navegador, o cliente, comunicar proactivamente al servidor, sus necesidades o restricciones de hardware.
* La introducción de prefijos de seguridad en la cabecera [Cookie](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Cookie), esto ayuda a garantizar que una cookie, no ha sido alterada.

Esta evolución del HTTP demuestra su capacidad de ampliación y simplicidad, permitiendo así de forma deliberada su uso para muchas aplicaciones y favoreciendo el uso de este protocolo. El entorno en el que el HTTP se usa hoy en día es muy distinto al que había a principios de la década de 1990. El desarrollo original de HTTP, ha demostrado ser una obra maestra, permitiendo a la Web evolucionar a lo largo de un cuarto de siglo, sin la necesidad de un 'amotinamiento'. Corrigiendo errores, y manteniendo la flexibilidad y extensibilidad que han hecho al HTTP un éxito, la adopción del HTTP/2 tiene un brillante futuro.

**7. Novedades HTTP 2**

La [gran mejora que introduce el HTTP/2 es la velocidad](https://www.xataka.com/servicios/http-2-asi-va-a-mejorar-la-velocidad-de-tu-navegacion-sin-que-tu-tengas-que-hacer-nada), que se logra mediante diversas novedades. Todo esto repercutirá [optimizando el tiempo de carga de una página](https://www.escueladeinternet.com/8-maneras-avanzadas-mejorar-velocidad-carga-web/) sin tener que hacer nada por nuestra parte.

* Estructura binaria.  
  La estructura de cabeceras y datos en texto plano se sustituye por frames binarios que encapsulan la información. El uso de este tipo de estructura permite habilitar la compresión y la reutilización de una misma conexión TCP, lo que comporta una optimización de los recursos de red, el tiempo de carga y la latencia.
* Proactividad (Server push)  
  Los servidores pueden enviar respuestas por iniciativa propia, adelantándose a la petición del cliente. Por ejemplo, si hasta ahora se tenía que cargar todo el HTML de una página web antes de cargar el contenido en sí (CSS, fotos…), con HTTP/2 el servidor puede enviar elementos que componen la página como CSS, Javascript u otros de manera que cuando el cliente recibe el documento ya dispone de los recursos que éste precisa. El cliente además puede guardar en cache estos elementos para reusarlos en otros documentos y así reducir la carga de red.
* Multiplexado  
  Dado que en HTTP/1.1 resulta mucho más eficiente recuperar una imagen grande que hacer muchas peticiones de imágenes pequeñas, se nos ha venido recomendado condensar todos los iconos en un solo [fichero sprite](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/CSS_Image_Sprites) (que los devuelve todos en una sola petición HTTP). No obstante, esto comporta que en ocasiones el usuario reciba un archivo mucho más grande de lo necesario. En cambio, el protocolo HTTP/2 permite atender varios pedidos a la vez, realizando peticiones paralelas mediante una única conexión TCP, lo que se conoce como multiplexado.Por otro lado, el [sharding](http://brigomp.blogspot.com.es/2009/08/sharding-si-o-no.html)de dominios y la concatenación ya no son necesarios. Puesto que con el protocolo HTTP/2 se pueden solicitar tantos recursos como sean necesarios —a diferencia del HTTP/1.1, que limita el número de conexiones abiertas—, no hace falta repartir los recursos de gran tamaño en varios dominios. Asimismo, muchos desarrolladores concatenan todos los pequeños ficheros CSS y JavaScript de la página web con la intención de limitar las peticiones HTTP. Con HTTP/2, es preferible organizar los recursos en función de las páginas en las que se emplearán o teniendo en cuenta la frecuencia de cambio. Así se consigue enviar únicamente el código que el usuario necesita.

**8. Bibliografía**

HTTP 1 <http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/http.html>

HTTP 2 <https://somostechies.com/que-es-http2/>

Estructura <https://blog.makeitreal.camp/el-protocolo-http/>

Contenido de petición <https://blog.makeitreal.camp/el-protocolo-http/>

Contenido de respuesta <https://blog.makeitreal.camp/el-protocolo-http/> y <https://developer.mozilla.org/es/docs/Web/HTTP/Status>

Versiones <https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP>

Novedades HTTP 2 <https://www.escueladeinternet.com/protocolo-http2-novedades-empezar-usarlo/>