

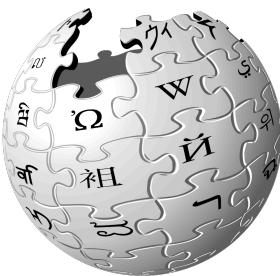
SERVICIOS WEB



INTRODUCCIÓN

Precursor: Memex

- **Vannevar Bush** (1890 - 1974), ingeniero y profesor del MIT, describió en **1945** la **primera aplicación hipertextual** en su artículo ***As we may think***, publicado en **The Atlantic**.
- En su artículo predijo muchos tipos de tecnología que se inventaría después: **hipertexto**, ordenadores personales, Internet, la www, reconocimiento de voz, enciclopedias online...



Wholly new forms of encyclopedias will appear, ready-made with a mesh of associative trails running through them, ready to be dropped into the memex and there amplified.



Precursor: Memex

- Acrónimo de **Memory Index** (o Memory Extender, según algunos autores).
- Es un dispositivo de base de datos que nunca fue construido.
- El dispositivo constaría de una mesa con un teclado y palancas, que permitiría consultar datos almacenados en **microfilms** que serían proyectados en pantallas translúcidas.
- En él se almacenarían **todo tipo de documentos**.
- El usuario además de poder **consultar documentos**, podría también **realizar anotaciones**.
- El sistema permitiría al usuario poder seguir **enlaces**: El usuario podría navegar por la información según su propio interés.



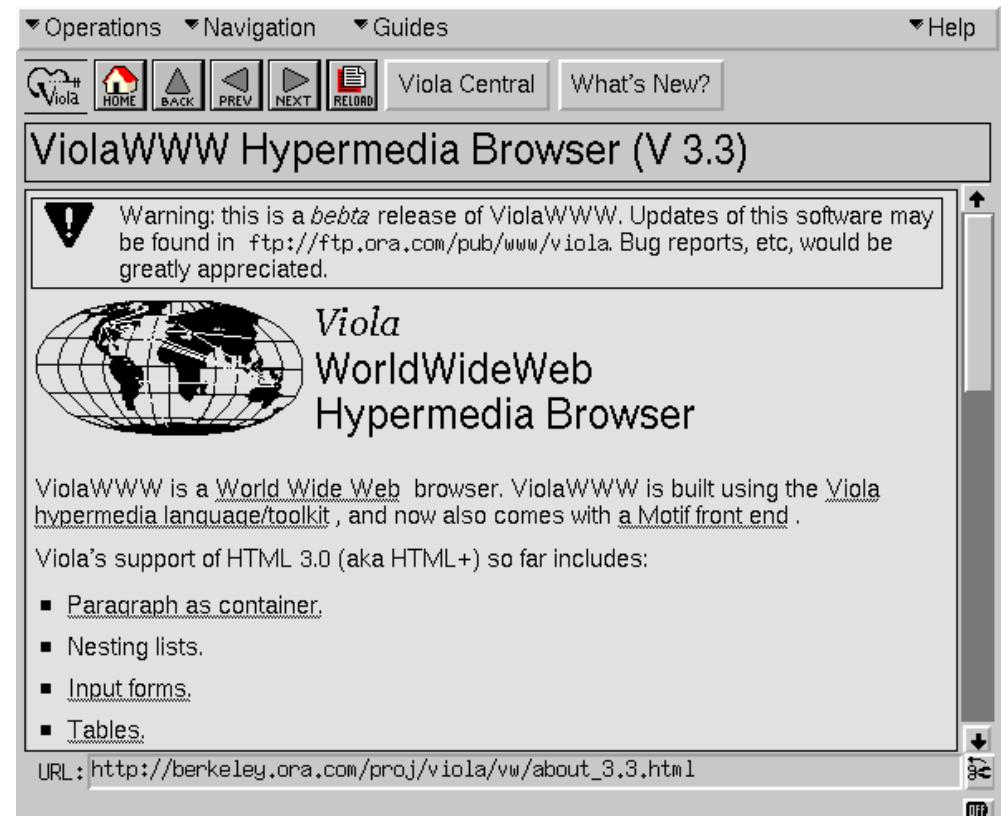
"La mente humana no funciona así" ("As We May Think"), sino por asociación. "Sujetando" un hecho o una idea, "la mente salta instantáneamente al dato siguiente, que le es sugerido por asociación de ideas, siguiendo alguna intrincada trama de caminos conformada por las células del cerebro"

Introducción

- WWW, “World Wide Web”: **Red Global Mundial**.
- En 1989 **Tim Berners Lee** y **Robert Caillou**, proponen la creación de la **WWW** mientras trabajaban en el **CERN** (Consejo Europeo para la Investigación Nuclear).
- La intención era crear un sistema más fácil de utilizar para **compartir textos de investigación entre científicos** y permitir al lector revisar las referencias de un artículo mientras lo iba leyendo.
- La **WWW** es un sistema global de documentos o páginas enlazadas entre sí mediante **hipervínculos** o **hiperenlaces**.
- El servicio web es el más utilizado por los usuarios de Internet y hacemos uso de él cada vez que abrimos una página en nuestro navegador.

Introducción

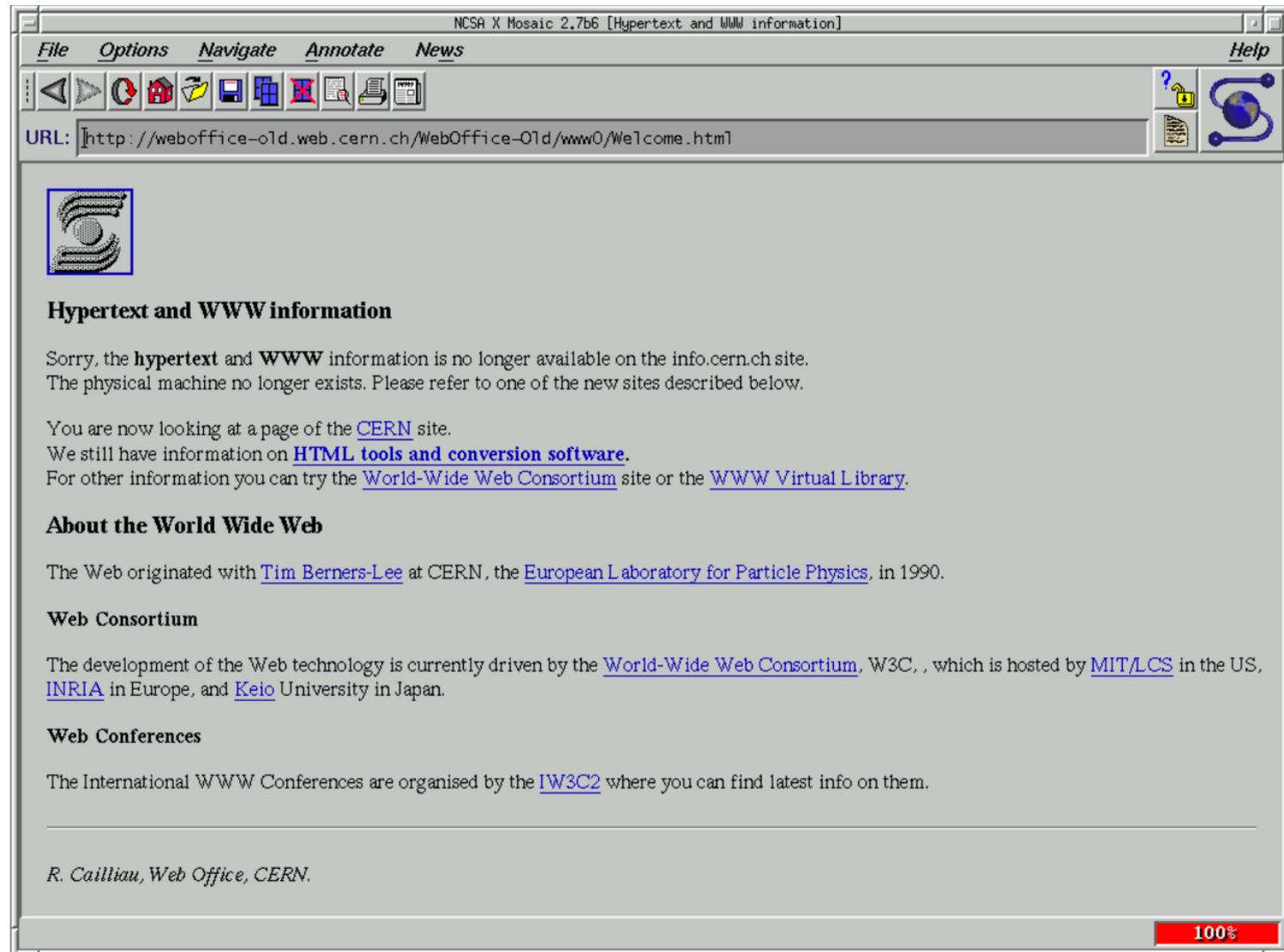
- Inicialmente el programa del CERN solo consideraba **texto**.
- **Pei-Yuan Wei** (1992) añadió la capacidad de **presentar gráficos** en su navegador ViolaWWW.
- **ViolaWWW** fue el primer navegador web gráfico, aunque solo soportaba gráficos en bitmap en blanco y negro y su funcionalidad se restringía a entornos **X-Window**.
- Incluía un **lenguaje de scripts** similar a Javascript, un modelo de objetos similar a **Document Object Model (DOM)** y **hojas estilo en cascada** similar a CSS.

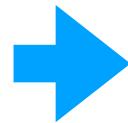
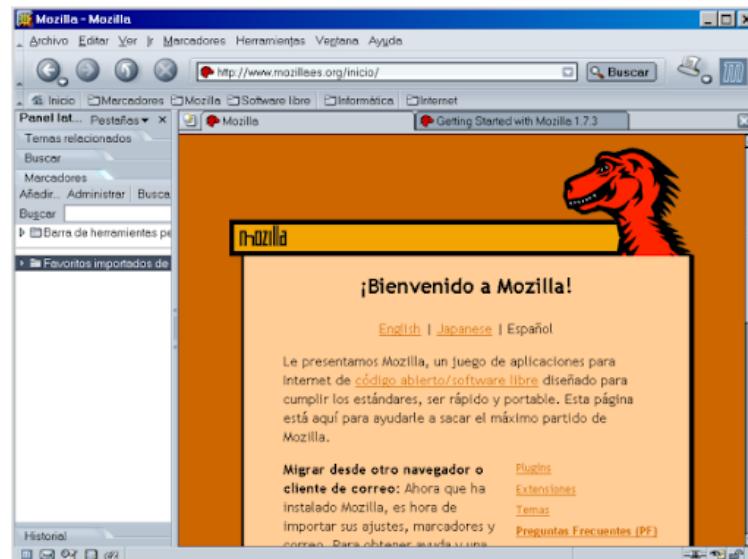
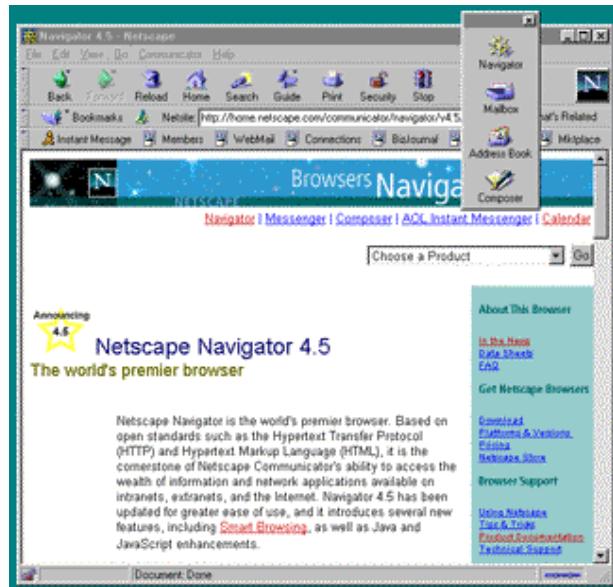


Marc Andreessen

- **Marc Andreessen** (junto a Eric Bina) desarrolló en 1993 un **navegador gráfico** llamado **Mosaic** que disparó la popularidad de la web.
- Creó el Mosaic a los 22 años mientras todavía era un estudiante y becario en el **NCSA** (National Center for Supercomputing Applications).
- Mosaic se distribuyó **gratuitamente** entre la comunidad científica y contribuyó a la rápida expansión de la **World Wide Web**.
- El Mosaic añadiría capacidades para reproducir contenido dinámico, música y animaciones.
- Marc Andreessen es inversor de **Digg** y uno de los diseñadores de **SSL** (Secure Sockets Layer).
- Junto a **Jim Clark**, fundó Mosaic Communications Corporation. La compañía cambió de nombre el 14 de noviembre de 1994 a Netscape Communications Corporation, y el Mosaic a **Netscape Navigator**.







Estándares HTTP

- En 1994, TimBL fundó la **World Wide Web Consortium** (W3C), una organización internacional que genera recomendaciones y estándares para la World Wide Web.
- El **protocolo HTTP** fue desarrollado por el **World Wide Web Consortium** y la **Internet Engineering Task Force** (IETF), que es una organización abierta de normalización que regula las propuestas y estándares conocidos como **RFC**.
- En 1999 publican varios RFC, entre ellos el RFC 2616 que especifica la versión 1.1 de HTTP.



www.w3.org

RFCs

HTTP Original	HTTP/0.9	1991
RFC 1945	HTTP/1.0	1996
RFC 2616	HTTP/1.1	1999
RFC 2774	HTTP/1.2	2000
RFC 7230 ← → RFC 7235	Revisión de HTTP/1.1	2014
RFC 7540	HTTP/2	2015
Borrador	HTTP/3	2018

A horizontal timeline showing the evolution of HTTP versions over time. The timeline is represented by a green bar divided into segments labeled with their respective versions: HTTP 0.9, HTTP 1.0, HTTP 1.1, HTTP 2.0, and HTTP 3.0. The bar spans from 1991 to 2021. The labels for HTTP 2.0 and HTTP 3.0 are in blue, while the others are in white.

1991 1996 2001 2006 2011 2016 2021

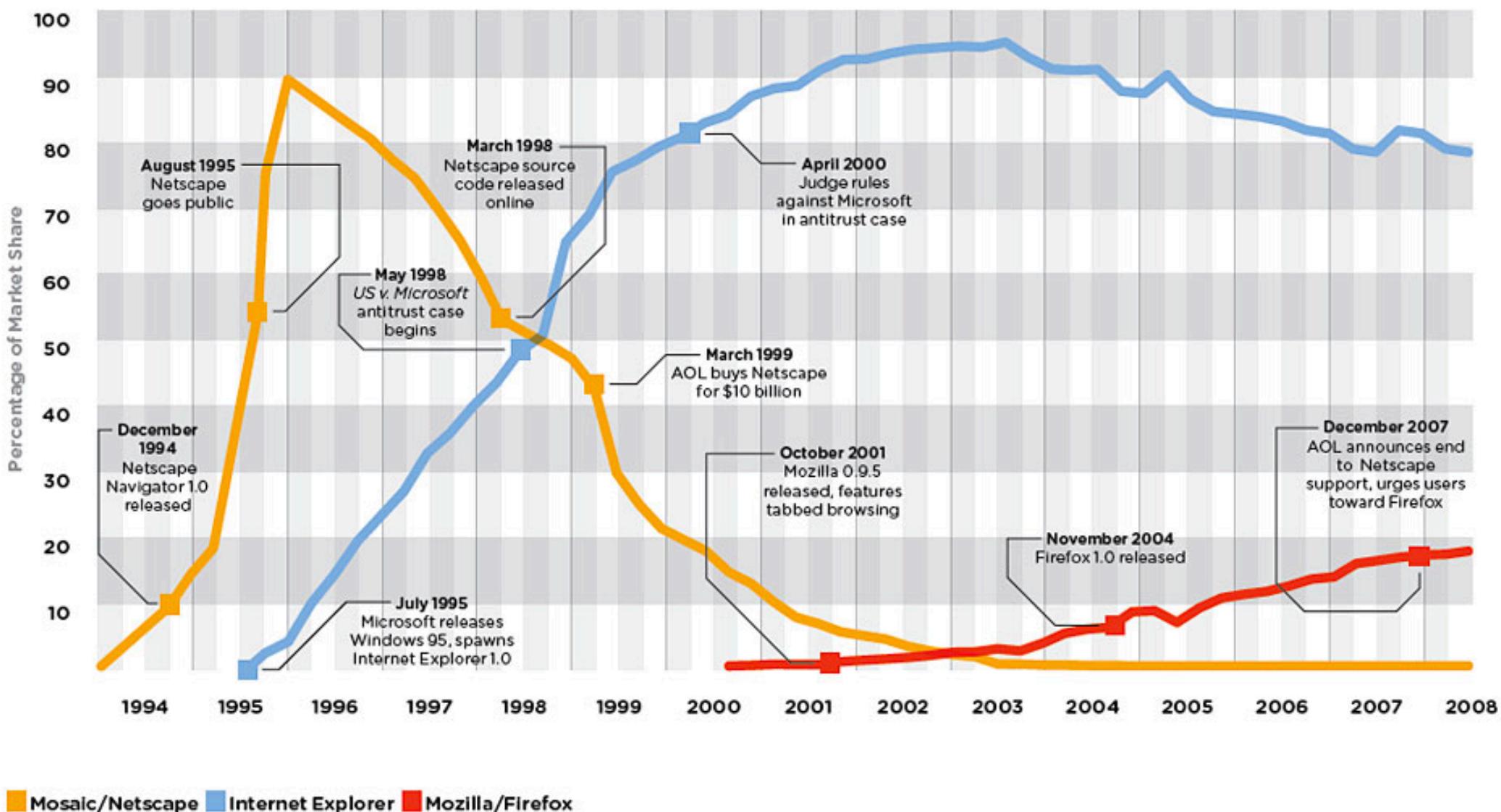
HTTP 0.9 HTTP 1.0 HTTP 1.1 HTTP 2.0 HTTP 3.0

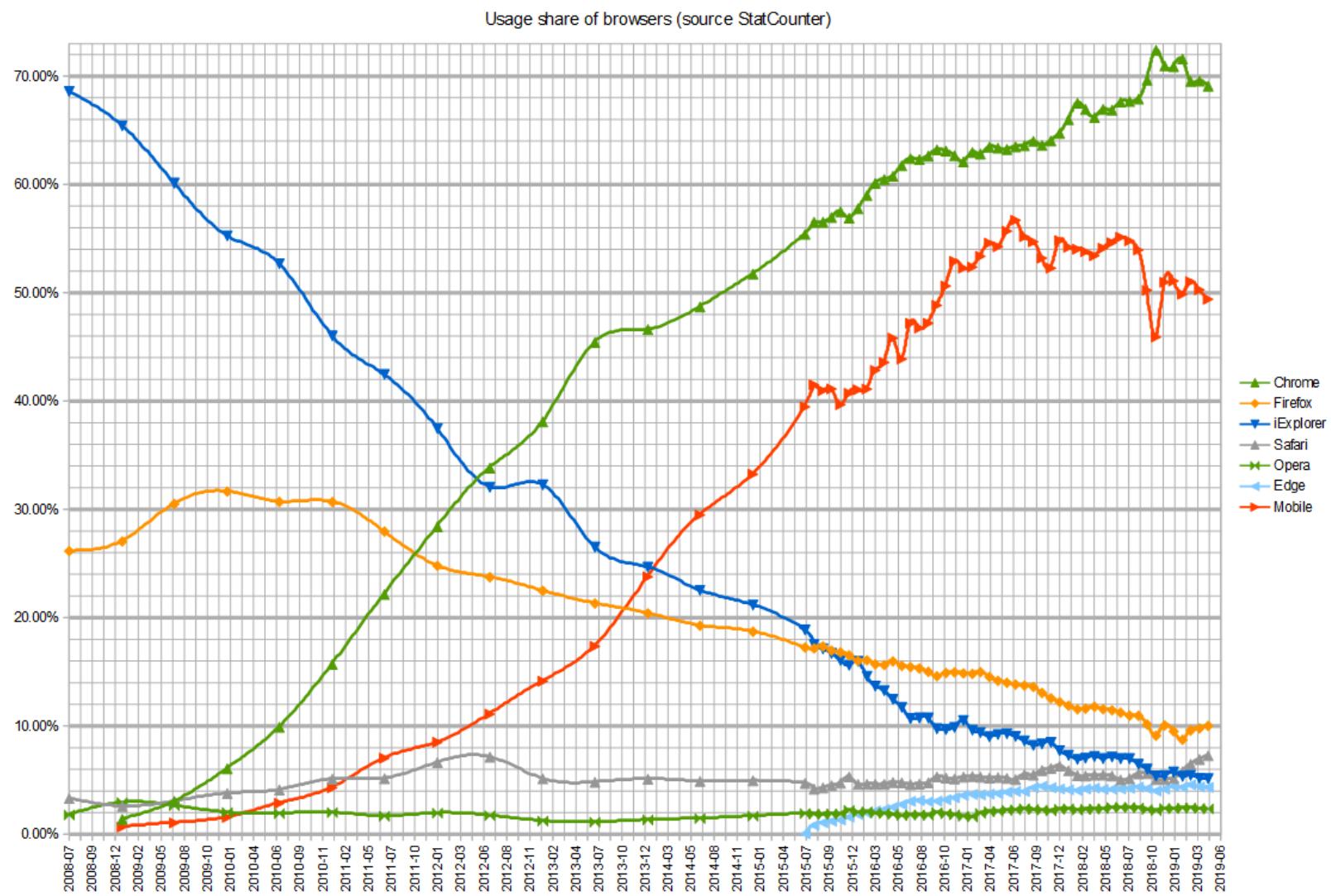
Guerras de los Navegadores

- La evolución de los navegadores siempre fue más rápida que la de los estándares.
 - Incompatibilidades.
 - Nuevas características y aparición de bugs.
 - Prácticas monopolísticas.
 - Necesidades de las empresas...
- **Internet Explorer vs Netscape**
- **Microsoft Edge vs Mozilla Firefox vs Google Chrome vs Opera vs Safari...**

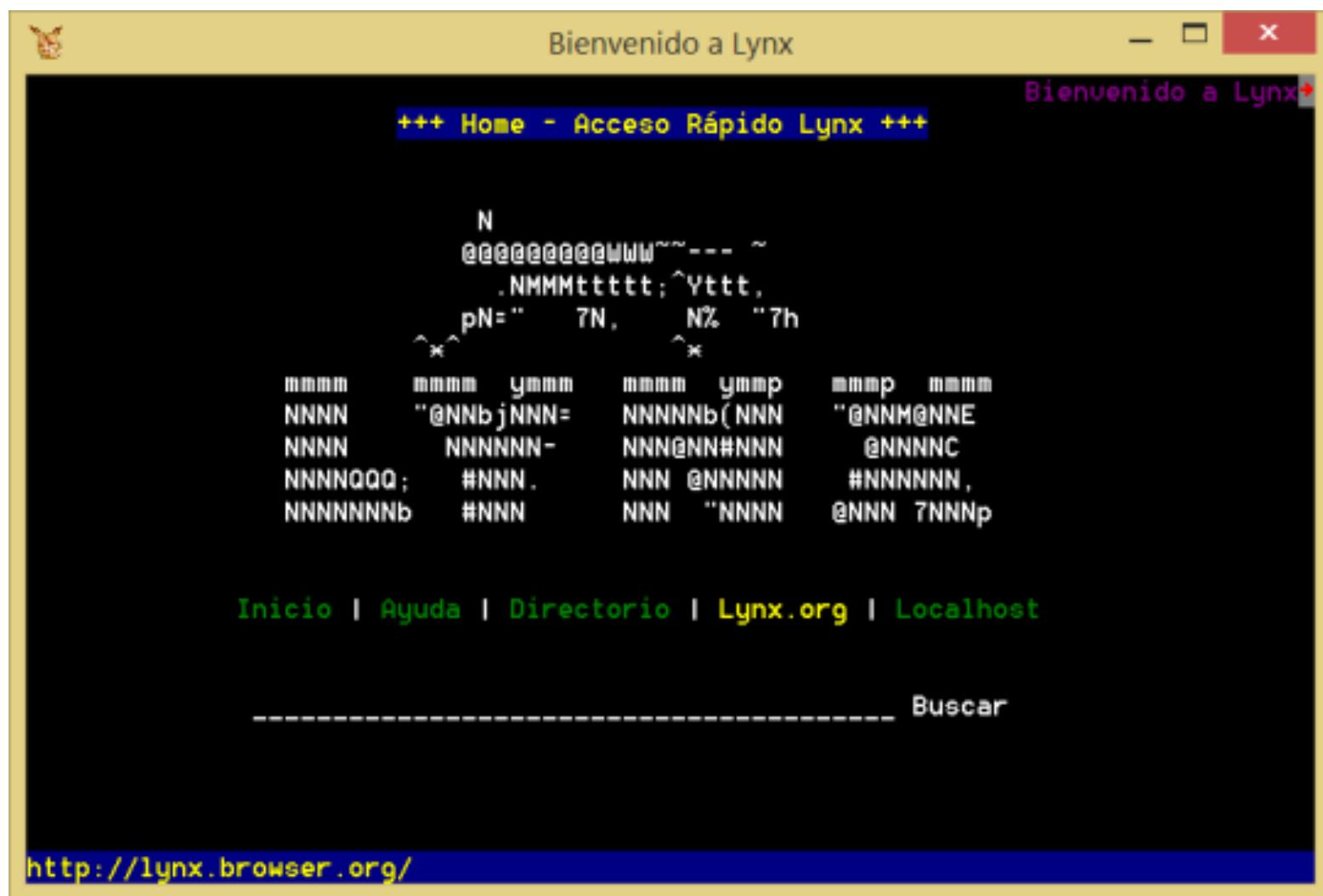


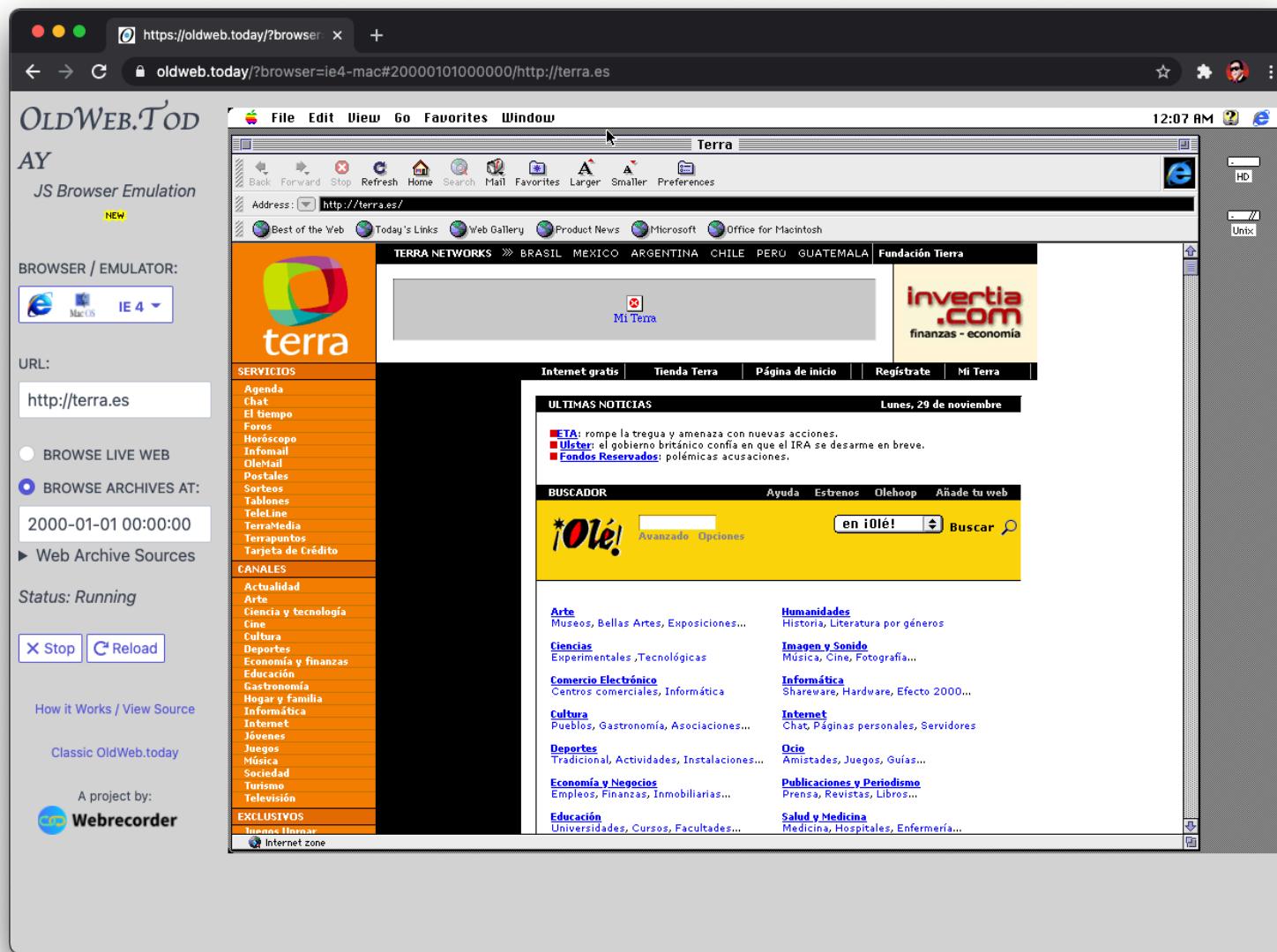
En octubre de 1997 fue lanzado Internet Explorer 4.0. La fiesta de lanzamiento en San Francisco contaba con un gigantesco logo de la letra "e". Los empleados de Netscape se encontraron en el patio delantero del edificio con el logo gigante junto a una leyenda: "del equipo de IE". Los empleados de Netscape le dieron inmediatamente la vuelta y colocaron una figura gigante de su mascota, el dragón Mozilla, sobre él, con la leyenda "Netscape 72, Microsoft 18" (representando las porciones del mercado).





https://en.wikipedia.org/wiki/Comparison_of_web_browsers





<https://oldweb.today/>

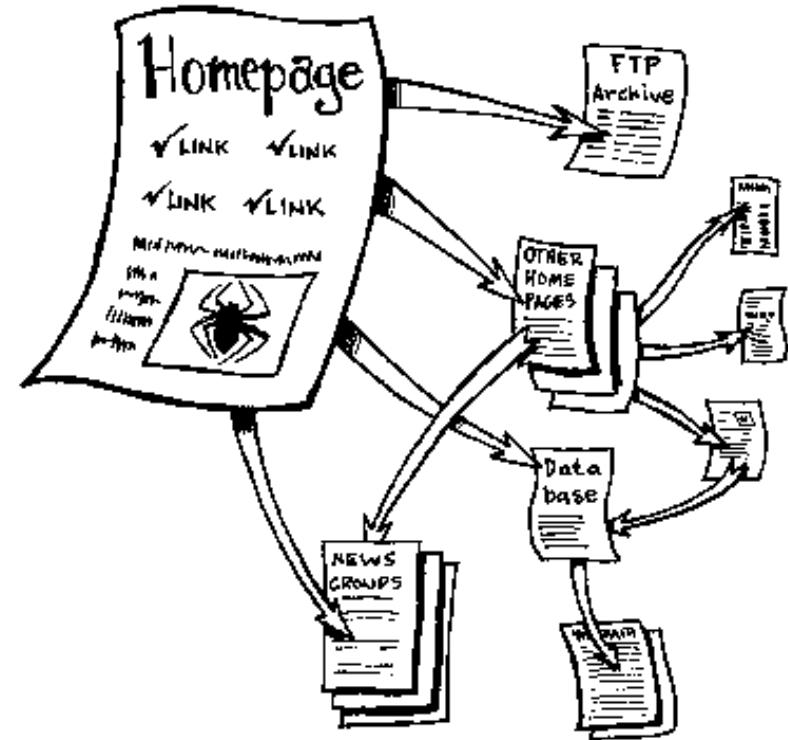
CONCEPTOS

Hipertexto

- El **hipertexto** es un **texto con enlaces**, que son referencias a otras partes del documento o a otro documento.
- Es un término que fue acuñado por **Ted Nelson** en **1965**, aunque sus orígenes, como ya vimos, se remontan a 1945.
- De esta forma **el documento no tiene que ser leído de forma secuencial**. Se puede saltar a puntos distintos y volver al punto original.
- La idea de los enlaces está presente en: índices, tablas de contenidos, pies de páginas, referencias bibliográficas....
- El hipertexto es **fundamental en el desarrollo de Internet**, ya que permite que **un documento** esté físicamente distribuido en **distintas máquinas** y esta idea da origen a la **web: una gran base de datos distribuida de documentos**.

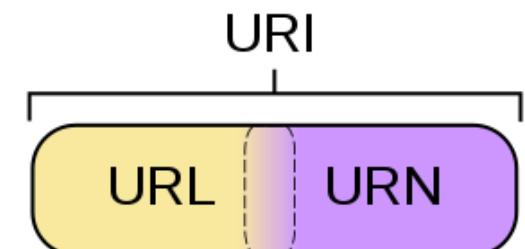
Hipertexto

- Las referencias entre las partes de un documento se establecen mediante **anclas** y **enlaces**.
- Un **ancla** es un fragmento de información dentro de un documento al que se le asocia un enlace.
- Un **enlace** es un apuntador a otro fragmento de información.
 - El enlace debe contener toda la información necesaria para acceder al fragmento enlazado: **nombre, ubicación y protocolo**.



URI, URL, URN

- **URI**: Identificador de recurso uniforme (Uniform Resource Identifier)
 - **URL**: Localizador de recurso uniforme (Uniform Resource Locator)
 - **URN**: Nombre de recurso uniforme (Uniform Resource Name)



- **URI**: Es una cadena de caracteres utilizada para identificar a un recurso en internet, por su **localización**, por su **nombre** o por **las dos cosas**.

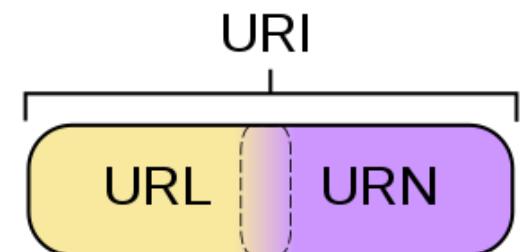
URL

URN

URL + URN

URI, URL, URN

- Imaginemos un nombre y una dirección:
 - John Doe → **URN**
 - Avda. da Coruña 5, 3º. Santiago. A Coruña. → **URL**
- **URI:** Nombre, la dirección o la dirección y el nombre.

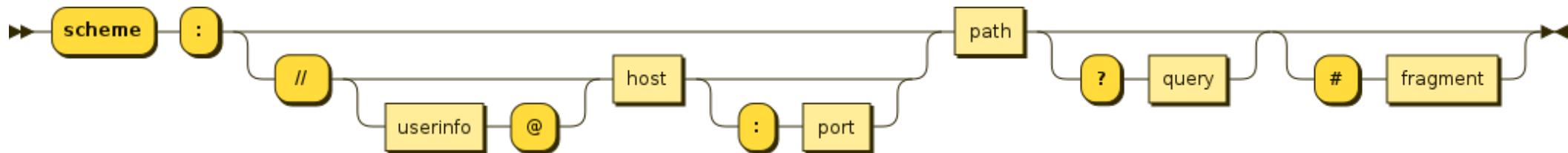


URI, URL, URN

URL:	<u>ftp://ftp.is.co.za/rfc/rfc1808.txt</u>
URL:	<u>http://www.ietf.org/rfc/rfc2396.txt</u>
URL:	<u>ldap://[2001:db8::7]/c=GB?objectClass?one</u>
URL:	<u>mailto:John.Doe@example.com</u>
URL:	<u>news:comp.infosystems.www.servers.unix</u>
URL:	<u>telnet://192.0.2.16:80/</u>
URN:	(not URL): <u>urn:oasis:names:specification:docbook:dtd:xml:4.1.2</u>
URN:	(not URL): <u>tel:+1-816-555-1212 (?)</u>
URN:	<u>urn:ietf:rfc:2648</u>
URN:	<u>urn:isbn:9780007525546</u>
URL:	<u>jdbc:datadirect:oracle://myserver:1521;sid=testdb</u>

<https://www.ietf.org/rfc/rfc3986.txt>

URI



URL

<foo://user:pass@www.host.com:54/search?type=red&name=john#fragment>

URN

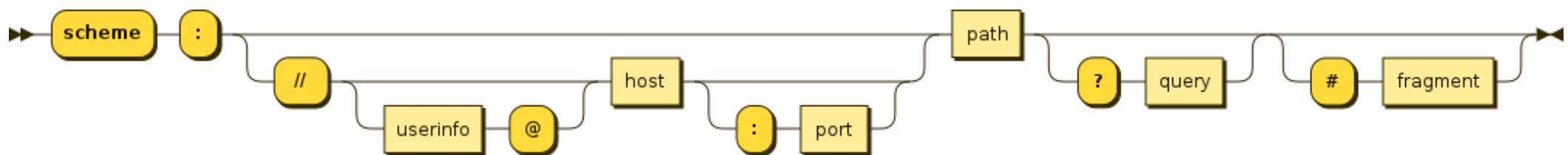
- Los conceptos no están del todo claros: Distintas fuentes indican información contradictoria.
- **La idea básica:**
 - Las **URL** especifican el **protocolo** y la **localización** del recurso.
 - Las **URN** identifican al recurso pero no indican cómo localizarlo o cómo acceder a él.
 - **URI** es el identificador más completo.

URL

- Localizador de recurso uniforme.
- Las **URL** fueron una **innovación en la historia de Internet**. Fueron usados por primera vez por **TimBL** en 1991 para permitir a los autores de documentos **establecer enlaces**.
- Formato general: **esquema://maquina/directorio/archivo** (RFC 1738)
- La URL se clasifica por su esquema que **indica el protocolo de red** que se usa para recuperar el recurso identificado.
 - Algunos ejemplos: **http, https, ftp, mailto, file, telnet, news...**
- Algunos esquemas se detallaron en el RFC 1630 en 1994, y se sustituyeron un año después por los más específicos: RFC 1738 y RFC 1808.
- La definición de la sintaxis general de los URL se comenta en la especificación URI: RFC 2396 (1998) y RFC 2732 (1999), ya obsoletos.

URL

- Muchos navegadores web no requieren que el usuario introduzca **http://** porque es el protocolo más común y lo sobreentienden.
- Las URL son **válidas fuera del contexto de la WWW**, por ejemplo:
 - `jdbc:datadirect:oracle://myserver:1521;sid=testdb`
- Las urls conforman a la sintaxis general de una URI genérica:
 - **URI = scheme:[//authority]path[?query][#fragment]**
- El componente **authority** se divide en tres subcomponentes:
 - **authority = [userinfo@]host[:port]**



URL Encoding (Codificación de la URL)

- Los navegadores solicitan páginas de servidores utilizando una URL.
- **URL encoding** convierte los caracteres en un formato que pueda ser transmitido en Internet.
- Las URLs solo se pueden enviar utilizando el juego de caracteres **ASCII**.
- Dado que las URL a menudo contienen caracteres fuera del conjunto ASCII, la URL debe convertirse a un **formato ASCII válido**.
- **URL encoding** reemplaza los caracteres ASCII inseguros con un % seguido de dos dígitos hexadecimales.
- Las URL **no pueden contener espacios**. La codificación de URL normalmente reemplaza un espacio con un signo + o con %20.
- En la página web https://www.w3schools.com/tags/ref_urlencode.asp se pueden hacer pruebas para ver cómo el navegador va a codificar la URL.

Web

- Una página web es un **documento** que puede contener texto, programas, enlaces, imágenes, hipervínculos y otros elementos, adaptada a la Word Wide Web (WWW) y que puede ser accedida por medio de un navegador Web.
- La información se encuentra generalmente en **formato HTML o XHTML**, y frecuentemente incluye otros recursos como scripts, hojas de estilo en cascada, imágenes, videos, etc.
- El acceso a la web se realiza utilizando el protocolo de transferencia de hipertexto (**HTTP**).
- Ideas básicas:
 - 1. Desaparición de la idea de un servidor único de información mantenido por un equipo. Cualquiera puede crear sus documentos y referenciar a otros documentos. **No hay una autoridad central**.
 - 2. La especificación de un mecanismo para localizar de forma única recursos distribuidos. Para ello se utiliza el **Uniform Resource Identifier URI**, junto con el protocolo, generalmente HTTP.
 - 3. La existencia de una interfaz de usuario que esconde los detalles de los formatos y protocolos utilizados para la transferencia de la información, simplificando el acceso a la misma (**navegadores**).

Web 1.0

- Es un término que se usa para describir la Web antes del impacto de la “**fiebre punto com**”. El momento en el que Internet dio un giro.
 - **Páginas estáticas** en lugar de dinámicas.
 - Uso de framesets o marcos.
 - Uso de botones gif promocionando navegadores webs u otros productos.
 - Formularios web enviados a través de correo electrónico.
 - No se podían realizar comentarios ni participar.
 - Las páginas web no eran dinámicas y si se actualizaban, se hacía manualmente en muchos casos.
 - Publicidad con banners y popups.



Web 1.0



GEO CITIES

Our communities are home to the most popular collection of **FREE HOME PAGES & E-MAIL** on the web. Please join or visit one of our 29 neighborhoods today.

Next Stop PLANET DIRECT LVOS FREELoader NEW! GeoStore DISCOVER A 3-D WORLD

GeoCities Yellowpages

TechWire just got more reporters...more news, and of course, it just got a whole lot better. You should come see what all the talk is about. <http://www.techweb.com> (or just click here).

TechWire [click here!](#)

GEO CITIES

AREA 51 PARIS HEARTLAND ATHENS TIMES SQUARE

YOUR HOME ON THE WEB

- [ENTER HERE](#)
- [INFORMATION](#)
- [NEIGHBORHOODS](#)
- [WHAT'S NEW](#)
- [WHAT'S COOL](#)
- [WHAT IS GEOCITIES?](#)

* [Free Home Pages & Free Member Email](#) [Advertiser Information](#)

Today's Cool Homestead [Audio Update](#) [GeoCities Daily Audio Update](#)

HotSprings 1837

So you hit the snooze bar ten times every morning. You might be lazy. But then again, you might have a sleep disorder. Find out here.

GeoCities News of the Day - 10/22/96

GEO+PLUS Experience GeoCities on a new level! Subscribe to GeoPlus

Introducing GeoPlus! Use up to 10MB for your site. Earn double GeoPoints. Post live news feeds on your home page. Get a bundle of free software from McAfee Associates. And more. [Start today](#) with our pre-launch subscription offer!

Is modern society getting you down? Come meet Grogg, the Internet's only Neanderthal advice columnist, this week on the [GeoCities Mainstage](#).

Business Upgrade Package or a Xerox Phaser 8400 Color Printer

World's fastest color printer under \$1,000. 8400 Printer \$999

Netscape What's New

Science Proves It: Drinking Causes... Teen Girls Beware: Popular Boys Do THIS Look What Was Found in Stone Age Cave Strange Space Object Mystery Deepens Did You See What Bush Dared to Wear? If You Get This E-Mail Scam, Delete It!

Lower your cost, not your expectations! Are You Paying Rates are at

pod Home | New | TriTeca | Work/Money | Politics/Community | Living/Travel | Planet T | Daily Scoop

AOL Members' Choice

It's Saturday, December 21, 1996

Build it and they will come. CLICK HERE!

TRIPOD tools for life

highlights

TriTeca Suck it up and learn HTML; follow Tripod's VP of geeks to Gates descend on Barnes & Noble; and build on the renovate

Page Slave Suck it up! Learn HTML (12/13)

Tripod's Yule Log Bask in the glow of our virtual log. (12/20)

living & travel Thrill & thriftseek in the Women's Room; witness the Velveet nominate dubious Web achievements; and control your flatline

Web 2.0

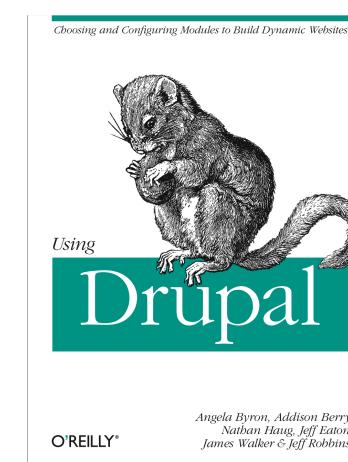
- La **web 2.0** es la **web social**.
- El término fue inventado por Darcy DiNucci en 1999, y popularizado por **Tim O'Reilly** y **Dale Dougherty**.
- Los sitios facilitan **compartir** información, la interoperabilidad y el diseño centrado en el usuario y en la colaboración.
- Servicios web, aplicaciones web, redes sociales, wikis...
- No se refiere a una actualización de las especificaciones técnicas de la web, se refiere a cambios acumulativos en la que desarrolladores de software y usuarios finales utilizan la web.
- Publicidad contextual.



Tim O'Reilly, fundador de O'Reilly Media e impulsor del software libre. Participa en el desarrollo de Perl.



Dale Dougherty, cofundador de O'Reilly Media. Fundador de la revista Make



Web 3.0

- Web 3.0 o la web semántica. Trata de añadir metadatos semánticos que enriquezcan la información.
- Una web no solo para navegadores, también para apps. **Multidispositivo**.
- Capaz de interconectar un mayor número de datos.
- El contenido y el conocimiento se relacionan de forma más eficiente.
- Uso de un lenguaje más natural.
- **Resultados personalizados** descartando información que nos será irrelevante.



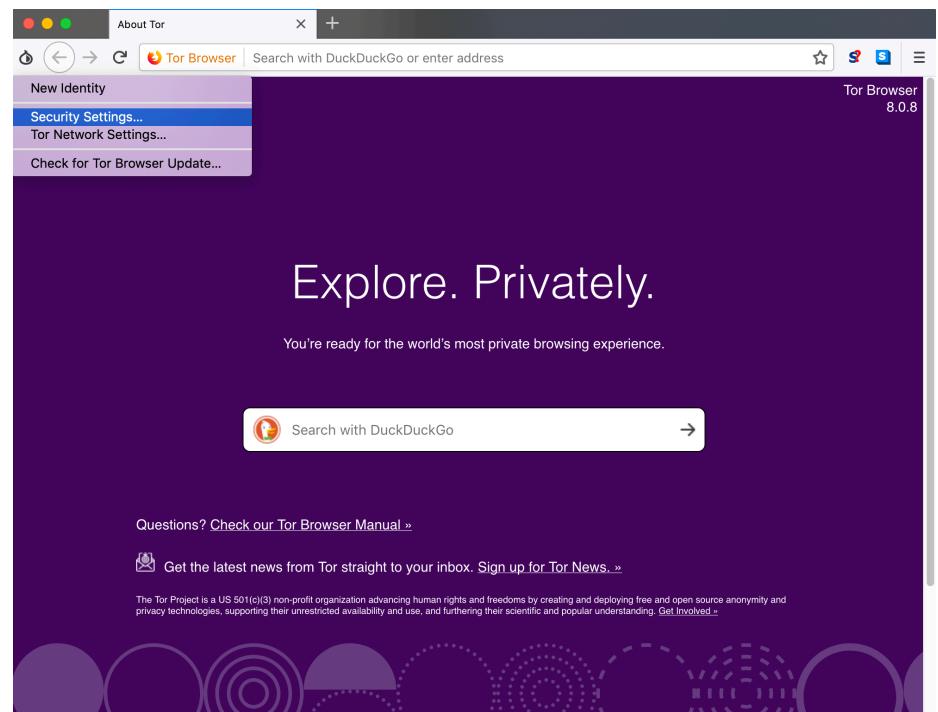
Web 4.0

- Un nivel de interacción más completo y personalizado: “Pide un taxi”, “Reserva mesa para dos”.
- **Información de contexto:** GPS, ritmo cardiaco que tu reloj inteligente registra, otros sensores...
- Comprensión del **lenguaje natural**. Asistentes de voz.
- Nuevo modelo de interacción con el usuario.
- Uso de los avances tecnológicos: **Deep Learning, Machine Learning, Redes Neuronales...**
- Pretende cerrar la **brecha digital**.
- Puede ser un paso a que nuestras rutinas se hagan por sí solas.



Dark Web

- La **Dark Web** es el contenido que se puede encontrar en las diferentes **Darknets**.
- Muchos de los **mitos de la Dark Web son falsos**:
 - <https://www.xataka.com/analisis/asesinos-a-sueldo-anonimato-total-y-habitaciones-rojas-son-ciertos-todos-los-mitos-de-la-deep-web>
- Dominio **.onion**
- Navegador **TOR**.
- La red **Tor** implementa una técnica llamada **Onion Routing**, para garantizar el anonimato y la privacidad de los datos.
 - Aunque uses HTTPS, las cabeceras de los paquetes van desencriptadas (si no, no se podrían leer).
 - **TOR es más segura y privada**. En principio el primer y último nodo serían ocultos, aunque con tiempo y dedicación, se podría averiguar quienes son.



Aplicación web

- La web surgió con la idea de compartir información, pero su tecnología y evolución ha permitido que surjan aplicaciones completamente funcionales que corren en el navegador.
- **Aplicación web:**
 - Son aplicaciones basadas en tecnologías web.
 - Se accede a ellas a través del navegador.
 - Google Docs, Gmail, webmin, etc.
 - **SaaS:** Software como servicio. Las compañías pueden cobrar suscripciones por el uso de sus apps.
 - **PWAs:** Progressive web apps. Adaptar una app web a una app móvil.



Aplicación web

- **Ventajas:**

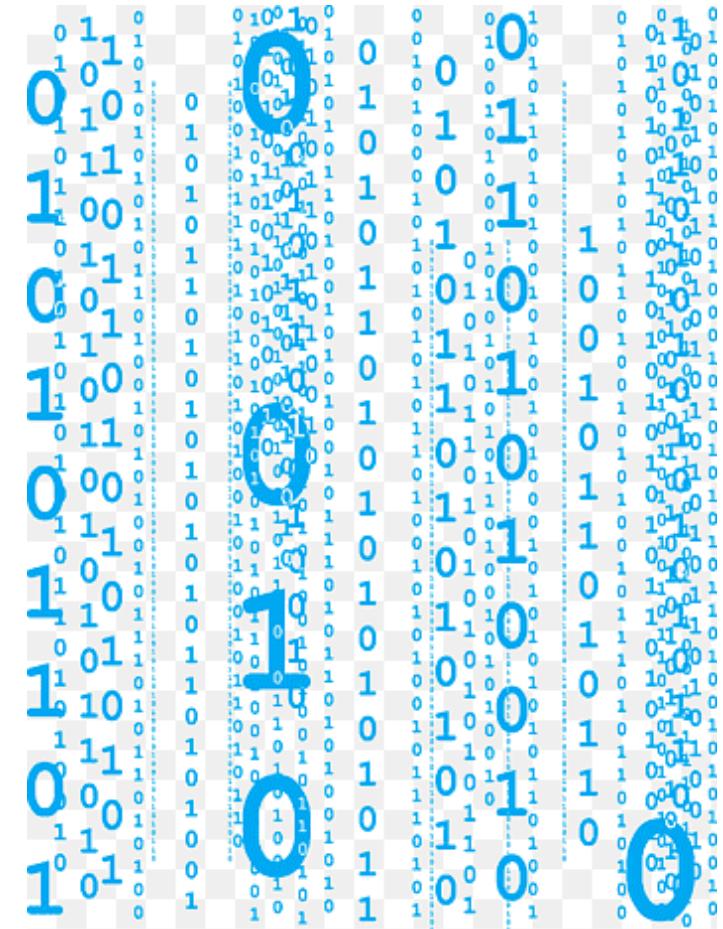
- No hace falta instalarlas.
- Se puede cambiar de equipo y seguir usándolas.
- Consumo de recursos bajo. Parte del procesos largos pueden ser llevados a cabo en la nube.
- Multiplataforma.
- Independiente del Sistema Operativo.
- Seguridad: Menos proclives a virus/malware.
- Actualizaciones inmediatas.
- Disponibilidad.
- Colaboración.

- **Inconvenientes:**

- Requieren de un navegador compatible y, a veces, de plugins.
- El usuario no tiene libertad para elegir la versión que quiere utilizar.
- El desarrollador puede rastrear cualquier actividad del usuario.
- Ofrecen menos funcionalidades que las aplicaciones nativas o de escritorio.
- Dependencia de un tercero, con todo lo que eso implica: no se puede modificar el programa, los datos quizás están en un servidor remoto, quizás no sea posible migrar a otro servicio, etc.
- Pueden ser más lentas: tiempos de transferencia, lenguajes interpretados, etc.

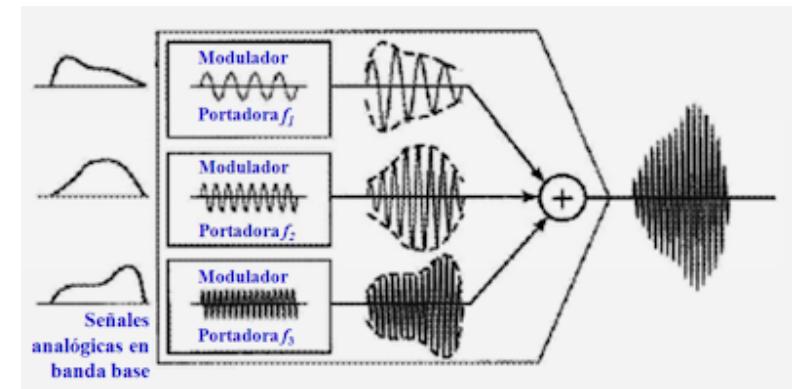
Stream

- Un stream es un **flujo de bytes** que contienen cierto tipo de información en formato binario que de alguna manera es intercambiada.
- Los streams aparecerán de HTTP/2 en adelante, puesto que se intercambiarán datos en binario y no en texto plano, como en las versiones anteriores.



Multiplexación

- Es la combinación de dos o más flujos de información en un mismo medio de transmisión.
- Básicamente consiste en **enviar dos o más flujos de datos** que pueden tener o no relación, por el mismo sitio.
- Sirve para aumentar la **eficiencia** en las transmisiones, minimizar la cantidad de físicas requeridas y maximizar el ancho de banda.
- Existen técnicas de multiplexación de **señales analógicas** y de multiplexación de **señales digitales**. Pero se escapan al ámbito de la asignatura.
- Simplemente hay que quedarse con que son técnicas que permiten mandar flujos de datos independientes por el mismo canal, para aprovechar mejor los recursos.



HTTP

- Protocolo de transporte de Hipertexto (**Hypertext Transfer Protocol**).
- Es un **protocolo** de la **capa de aplicación** del modelo **OSI** que se utiliza prácticamente para transmitir la mayoría de los archivos y datos de Internet, ya sean archivos HTML, imágenes, resultados de consultas u otras cosas.
- Es un **protocolo sin estado**. El servidor no guarda un historial con las consultas y acciones anteriores del cliente.
- ¡No se utiliza sólo para navegar! La mayoría de las **apps** utilizan el protocolo HTTP para **transmitir datos e intercambiar información con servidores**. Pero en lugar de páginas **HTML**, obtendrán archivos **XML**, **JSON**, **YAML** u otro tipo de formatos o recursos (imágenes, videos, documentos...)
 - Lista de **APIs públicas**: <https://github.com/public-apis/public-apis>

HTML

- **HTML** (HyperText Markup Language) Lenguaje de marcas de hipertexto.
- Es un estándar a cargo de la **W3C**.
- Elementos
 - Etiqueta de inicio (y quizás de fin)
 - Contenido.
 - Atributos.
- Pueden ser creados con editores de texto o con herramientas **WYSIWYG** como Dreamweaver.

Item	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0	HTML 5.0
Hyperlinks	x	x	x	x	x
Images	x	x	x	x	x
Lists	x	x	x	x	x
Active maps & images		x	x	x	x
Forms		x	x	x	x
Equations			x	x	x
Toolbars			x	x	x
Tables			x	x	x
Accessibility features				x	x
Object embedding				x	x
Style sheets				x	x
Scripting				x	x
Video and audio					x
Inline vector graphics					x
XML representation					x
Background threads					x
Browser storage					x
Drawing canvas					x

HTML

```
<html>
<head> <title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page </h1>
 <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's</b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by email. </p>
<hr>
<h2> Product information </h2>
<ul>
  <li> <a href="http://widget.com/products/big"> Big widgets </a> </li>
  <li> <a href="http://widget.com/products/little"> Little widgets </a> </li>
</ul>
<h2> Contact information </h2>
<ul>
  <li> By telephone: 1-800-WIDGETS </li>
  <li> By email: info@amalgamated-widget.com </li>
</ul>
</body>
</html>
```

Welcome to AWI's Home Page



We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope you will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by email.

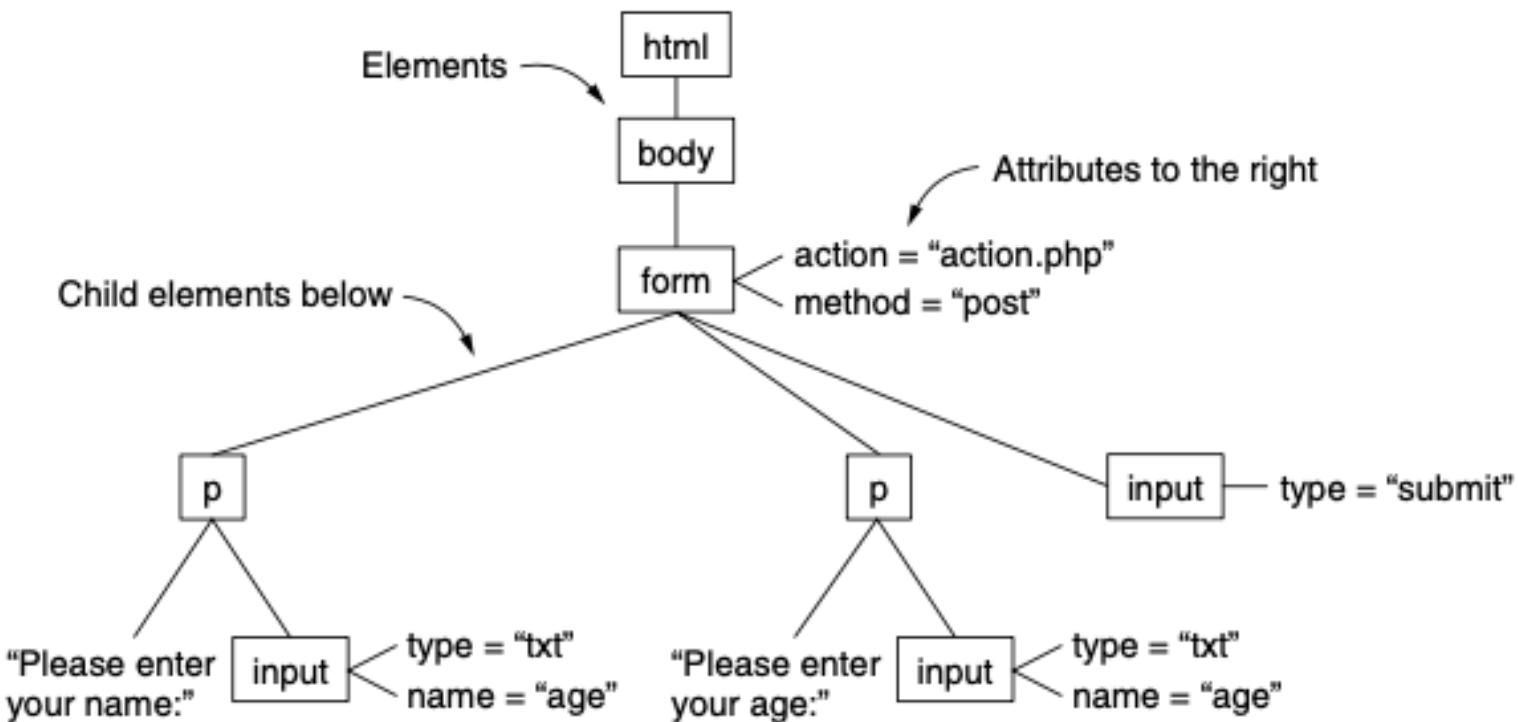
Product Information

- [Big widgets](http://widget.com/products/big)
- [Little widgets](http://widget.com/products/little)

Contact Information

- By telephone: 1-800-WIDGETS
- By email: info@amalgamated-widget.com

DOM (Document Object Model)



- Es una representación de una página HTML accesible para los programas con forma de **árbol** que muestra la estructura de los elementos HTML.

XML

- **Lenguaje de marcas extensible** (eXtensible Markup Language).
- Desarrollado por el **World Wide Web Consortium** (W3C).
- Proviene del lenguaje **SGML** y permite definir la gramática de lenguajes específicos para estructurar documentos grandes.
- MS Office usa XML para sus documentos Office Open XML .docx, .xlsx y .pptx.
- Muchos programas utilizan **XML** para **intercambiar información** y comunicarse con servidores.
- Por ejemplo, **MeteoGalicia** publica información meteorológica en XML y en otros formatos: https://servizos.meteogalicia.gal/rss/predicion/rssCortes.action?request_locale=gl

```
<?xml version="1.0"?>
<quiz>
  <qanda seq="1">
    <question>
      Who was the forty-second
      president of the U.S.A.?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
  </qanda>
  <!-- Note: We need to add
       more questions later.-->
</quiz>
```

XML

XHTML

- HTML desciende también de SGML. XHTML desciende de XML, y es **más estricto**.
 - <https://es.wikipedia.org/wiki/XHTML>
- El objetivo del W3C con XHTML es lograr una **web semántica**, donde la **información y la forma** de representarla estén claramente **separadas**.

```
<?xml version="1.0"
<!DOCTYPE html PUBLIC
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>XYZ</title>
</head>
<body>
<p>
    voluptatem accusantium doloremque
    totam rem aperiam eaque
</p>
</body>
</html>
```

XHTML

JSON

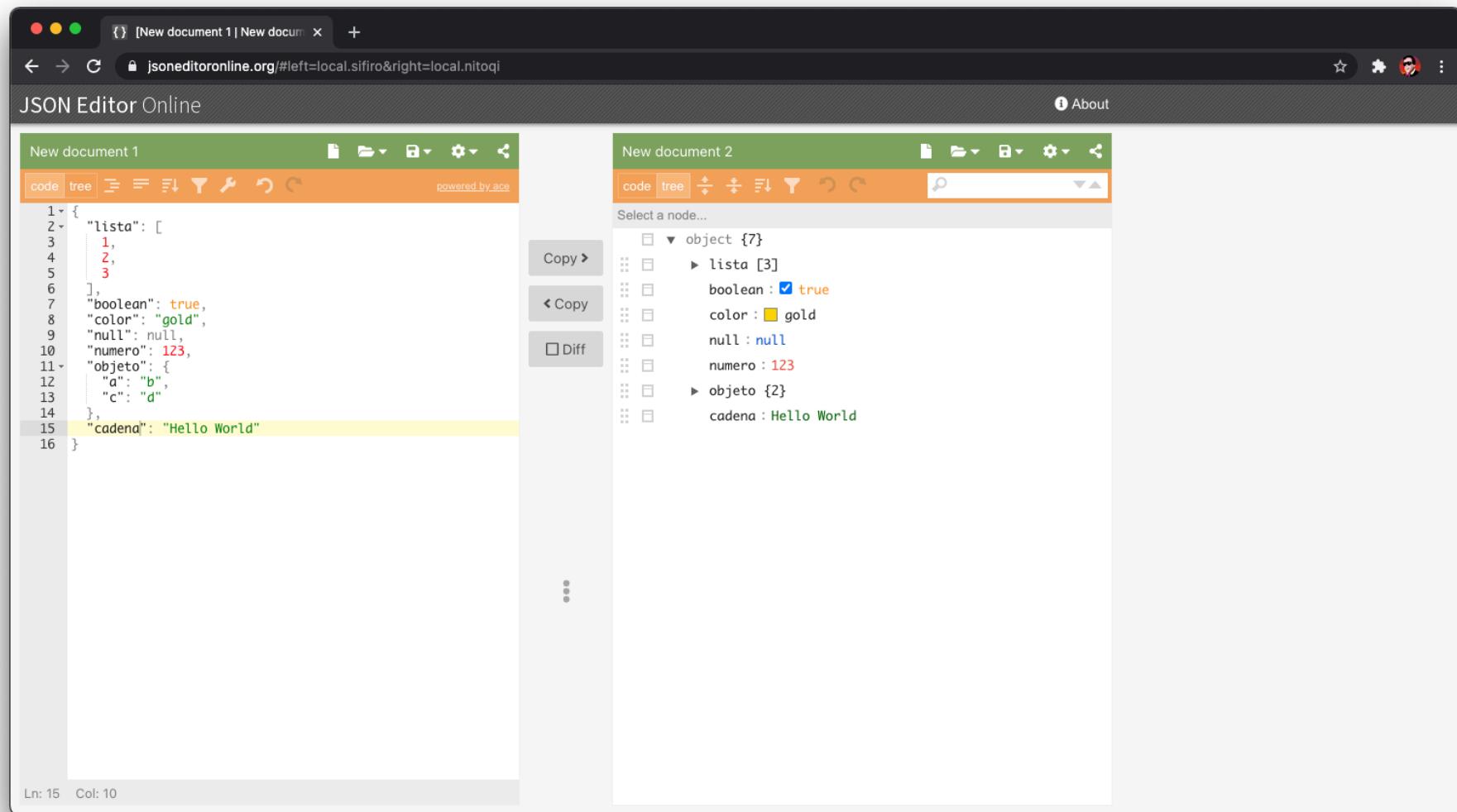
- **JavaScript Object Notation.**
- Es un formato de texto sencillo para el intercambio de datos.
- Debido a su **amplia adopción como alternativa a XML**, se considera desde el año **2019** un formato independiente de lenguaje.
- Es más fácil y rápido de parsear que XML, y existen librerías para casi todos los lenguajes.



JSON se lee Jason (/ˈdʒeɪsən/ «yéison»), como el malo de Viernes 13. No se lee “JOTASON”.

```
{  
  "departamento": 8,  
  "nombredepto": "Ventas",  
  "director": "Juan Rodríguez",  
  "empleados": [  
    {  
      "nombre": "Pedro",  
      "apellido": "Fernández"  
    }, {  
      "nombre": "Jacinto",  
      "apellido": "Benavente"  
    }  
  ]  
}
```

Jsoneditoronline.org

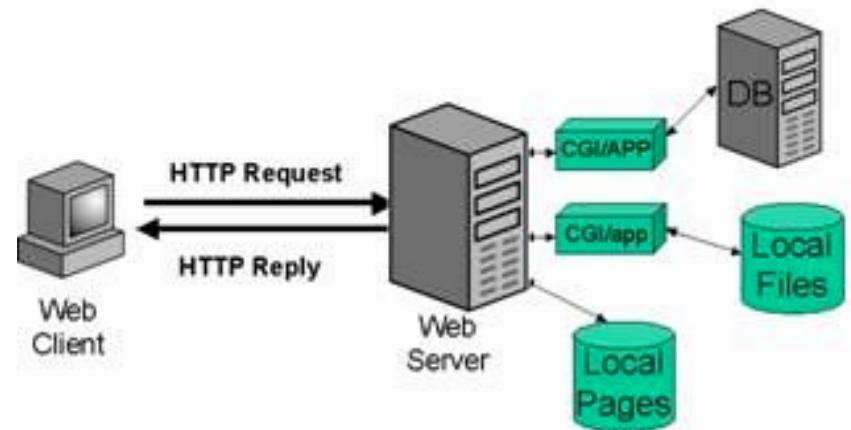


ARQUITECTURA, PETICIONES Y RESPUESTAS

HTTP: Cliente - Servidor

- Modelo Cliente - Servidor.
- **Un servidor web** es un programa que atiende y responde las peticiones de los navegadores usando el protocolo HTTP (o HTTPS).
 - Puerto **80** para **HTTP**
 - Puerto **443** para **HTTPS**

Web server Architecture



Visión general de HTTP

- El usuario especifica en el cliente el **URI** que desea consultar.
- El **DNS** proporciona la dirección IP y el navegador intentará establecer una **conexión TCP** en el puerto **80** y solicitará el recurso deseado.
- El servidor buscará el recurso deseado y si lo encuentra lo transferirá utilizando la misma conexión. Si no, devolverá un código de error.
- El cliente interpreta la respuesta y muestra el recurso o el error al usuario.
- La conexión siempre se libera al terminar la transmisión. HTTP es un protocolo que **no mantiene estado** (stateless) entre las distintas transacciones de un mismo cliente.

Mensajes HTTP

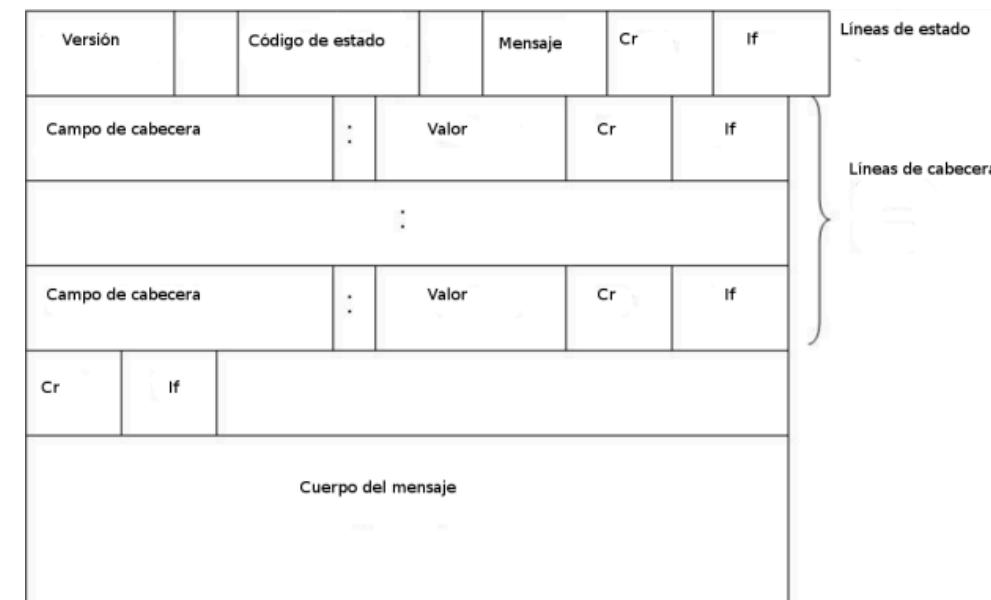
- Son los mensajes del protocolo HTTP con los que se comunican cliente y servidor.
- Los **mensajes HTTP** pueden ser: **Solicitudes** o **Respuestas**.
- Utilizan el formato genérico de emails (RFC-822)
- Consisten en:
 - **Una linea inicial.**
 - **Cero o más encabezados (headers / cabeceras).**
 - **Una línea en blanco.**
 - **Un cuerpo del mensaje** (opcional, por ejemplo archivo, datos de una consulta).

Mensajes HTTP: Ej. Respuesta

Status Line → HTTP/1.0 200 OK

Header Lines {
Connection: Close
Date: Mon, 20 Nov 2006 12:00:10 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon 22 Jul 2006
Content-Length: 6821
Content-Type: Text/html

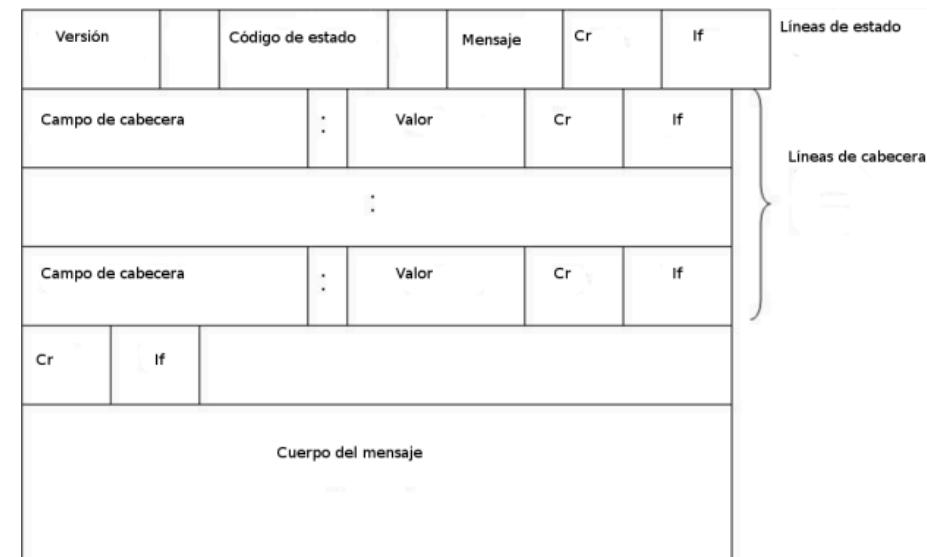
Data → Data, data, data, data.....
i.e.: requested file



- La línea inicial es distinta para solicitudes y respuestas.
- El cuerpo de mensaje es opcional, pero puede incluir contenidos de un archivo, de una consulta, datos binarios, etc.

Mensajes HTTP: Ej. Petición

```
GET /doc/test.html HTTP/1.1 → Request Line
Host: www.test101.com
Accept: image/gif, image/jpeg, */
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
                                         } Request Headers
                                         } Request Message Header
                                         } A blank line separates header & body
                                         } Request Message Body
bookId=12345&author=Tan+Ah+Teck
```



- La línea inicial es distinta para solicitudes y respuestas.
- El cuerpo de mensaje es opcional, pero puede incluir contenidos de un archivo, de una consulta, datos binarios, etc.

Mensajes HTTP: Petición y respuesta

- El cliente establece una conexión TCP al puerto 80 de www.ieee.org y envía la solicitud con el método GET.

GET /numero/uno.html HTTP/1.0

User-Agent: MiBrowser/2.0

[línea en blanco]

HTTP/1.0 200 OK

Date: Sat, 18 Nov 2000 15:18:02 GMT

- El servidor responde por la misma conexión:

Content-Type: text/html

Content-Length: 52

[línea en blanco]

<html><body>

<h1>Mi Archivo HTML</h1>

</body></html>

TIPOS MIME

Tipos MIME

- **MIME** (Multipurpose Internet Mail Extensions): Es una forma estandarizada de **indicar la naturaleza** y el **formato** de un documento, archivo o un surtido de bytes. (RFC 6838)
- Se crearon originalmente para poder enviar varios tipos de archivo a través de **correo electrónico**, pero se han llevado a HTTP para dar más funcionalidad al protocolo.
- La **IANA** (Internet Assigned Numbers Authority) es el organismo oficial de realizar un seguimiento de todos los tipos MIME oficiales.
 - <https://www.iana.org/assignments/media-types/media-types.xhtml>
- Los navegadores utilizan el tipo **MIME**, no la extensión del archivo para determinar **cómo procesar el contenido de una URL**.
- Es **importante** que los servidores web envíen el **tipo MIME correcto** en la respuesta utilizando la cabecera **Content-Type**.
- Si los navegadores reciben un tipo MIME erróneo es posible que interpreten mal el contenido de los archivos o no sepan cómo abrirllos.

Tipos MIME: Estructura

- La estructura más simple consiste en un **tipo** y un **subtipo**, concatenados con el carácter “/”:
 - **tipo/subtipo**
- El **tipo** representa la **categoría** de los datos y el **subtipo** identifica el **tipo exacto** de datos.
- Se pueden añadir **parámetros opcionales** para dar más detalles:
 - **tipo/subtipo;parametro=valor**
- Por ejemplo, para un tipo de texto plano, se puede añadir como parámetro su codificación:
 - **text/plain;charset=UTF-8**
- Los tipos MIME se pueden escribir con minúsculas y mayúsculas, pero **tradicionalmente se escriben en minúsculas**, con la excepción de los parámetros.

Tipos MIME: Tipos discretos

- Los **tipos discretos** indican la categoría del documento, que puede ser uno de los siguientes:

Tipo	Descripción	Ejemplo de subtipos típicos
text	Representa cualquier documento que contenga texto y es teóricamente legible por humanos	text/plain, text/html, text/css, text/javascript
image	Representa cualquier tipo de imagen. Los videos no están incluidos, aunque las imágenes animadas (como el gif animado) se describen con un tipo de imagen.	image/gif, image/png, image/jpeg, image/bmp, image/webp
audio	Representa cualquier tipo de archivos de audio	audio/midi, audio/mpeg, audio/webm, audio/ogg, audio/wav
video	Representa cualquier tipo de archivos de video	video/webm, video/ogg
application	Representa cualquier tipo de datos binarios.	application/octet-stream, application/pkcs12, application/vnd.mspowerpoint, application/xhtml+xml, application/xml, application/pdf

- Para documentos de texto sin subtipo específico se debe usar **text/plain**.
- Para documentos binarios sin subtipo específico o conocido, se debe usar **application/octet-stream**.

Tipos MIME: Tipos multiparte

- Los tipos **MIME multipartes** indican una categoría de documento que está dividido en distintas partes, a menudo con **diferentes tipos de MIME**.
- Es una forma de representar un **documento compuesto**. Con la excepción de:
 - **multipart/form-data**, que se utiliza con los formularios HTML y el método POST.
 - **multipart/byteranges** que se utiliza con el mensaje de estado de **Contenido Parcial 206** para enviar sólo un subconjunto del documento completo.
- HTTP no maneja documentos multiparte de una manera específica. El mensaje simplemente se transmite al **navegador** que propondrá una ventana “**Guardar como...**”, sin saber cómo mostrar el documento en linea.

Tipos MIME Importantes

- **application/octet-stream**
 - Es el valor **predeterminado para un archivo binario**.
 - Como significa “**archivo binario desconocido**”, los navegadores no lo ejecutan automáticamente y, en general, muestran una ventana de “Guardar como...”
- **text/plain**
 - Es el **valor predeterminado para los archivos de texto**, incluso si realmente es un archivo textual desconocido, los navegadores **asumen que pueden mostrarlo**.
 - Si los navegadores se descargan un archivo de texto sin formato text/plain de un elemento <link> que declara archivos **CSS**, no lo reconocerán como un archivo CSS válido si se le presenta como text/plain. Se debe usar el tipo MIME CSS **text/css**.
- **text/css**
 - Todos los archivos **CSS** que deben interpretarse como tales en una página web deben ser de tipo **text/css**.
 - Si el servidor no los envía con un tipo **MIME correcto**, los navegadores no los reconocerán y serán **ignorados en la carga de la página**.

Tipos MIME Importantes

- **text/html**
 - Todo el **contenido HTML** debe de ser servido con este tipo.
 - Los tipos alternativos para XHTML como application/xml+html son inútiles ya que HTML5 unificó estos formatos.
- Tipos de imagen:

Tipo MIME	Tipo de imagen
image/gif	Imágenes GIF (compresión sin pérdida, reemplazada por PNG)
image/jpeg	Imágenes JPEG
image/png	Imágenes PNG
image/svg+xml	Imágenes SVG (imágenes vectoriales)

- Existen más tipos como WebP (image/webp) o iconos ICO (image/x-icon).

Tipos MIME Importantes

- Al igual que **HTML no define un conjunto de tipos soportados** para imágenes, tampoco lo hace para los elementos **<audio>** y **<video>**, por lo que solo un conjunto pequeño de ellos puede ser utilizado en la web.
- Explican tanto los **códecs** como los **formatos** contenedor que se pueden utilizar.

Tipo MIME	Tipo de audio o video
audio/wave	Un archivo de audio en formato de contenedor WAVE. El códec de audio PCM (códec WAVE "1") a menudo es compatible, pero otros códecs tienen soporte más limitado (si lo hay).
audio/wav	Un archivo de audio en formato de contenedor WebM. Vorbis y Opus son los códecs de audio más comunes.
video/webm	Un archivo de video, posiblemente con audio, en el formato de contenedor de WebM. VP8 y VP9 son los códecs de video más comunes utilizados en él: Vorbis v Opus los códecs de audio más comunes.
audio/ogg	Un archivo de audio en el formato de contenedor OGG. Vorbis es el códec de audio más común utilizado en dicho contenedor.
video/ogg	Un archivo de video, posiblemente con audio, en el formato de contenedor OGG. Theora es el códec de video habitual utilizado en él; Vorbis es el códec de audio habitual.
application/ogg	Un archivo de audio o video usando el formato de contenedor OGG. Theora es el códec de video habitual utilizado en él; Vorbis es el códec de audio más común.

Tipos MIME Importantes

- **multipart/form-data**

- Se utiliza para enviar el **contenido de un formulario HTML completo** desde el navegador al servidor.
- Como formato de documento multiparte, consta de **diferentes partes**, delimitadas por un límite (una cadena que comienza con un doble guión ‘—’)
- **Cada parte** es una entidad en sí misma, con sus propios **encabezados HTTP, Content-Disposition y Content-Type** para los campos de carga de archivos.

```
Content-Type: multipart/form-data; boundary=aBoundaryString  
(other headers associated with the multipart document as a whole)  
  
--aBoundaryString  
Content-Disposition: form-data; name="myFile"; filename="img.jpg"  
Content-Type: image/jpeg  
  
(data)  
--aBoundaryString  
Content-Disposition: form-data; name="myField"  
  
(data)  
--aBoundaryString  
(more subparts)  
--aBoundaryString--
```

Este formulario en HTML...

```
<form action="http://localhost:8000/" method="post" enctype="multipart/form-data">
  <label>Name: <input name="myTextField" value="Test"></label>
  <label><input type="checkbox" name="myCheckBox"> Check</label>
  <label>Upload file: <input type="file" name="myFile" value="test.txt"></label>
  <button>Send the file</button>
</form>
```

...mandaría este mensaje:



```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-----8721656041911415653955004498
Content-Length: 465

-----8721656041911415653955004498
Content-Disposition: form-data; name="myTextField"

Test
-----8721656041911415653955004498
Content-Disposition: form-data; name="myCheckBox"

on
-----8721656041911415653955004498
Content-Disposition: form-data; name="myFile"; filename="test.txt"
Content-Type: text/plain

Simple file.
-----8721656041911415653955004498--
```

Tipos MIME Importantes

- **multipart/byteranges**

- Cuando se manda el código de estado **206 de Partial Content**, este tipo MIME indica que **el documento está compuesto de varias partes**, una para cada uno de los **rangos**.
- **Content-Type** utiliza “**boundary**” para separar las piezas.
- Cada parte tiene una cabecera **Content-Type** con el tipo y **Content-Range** con el rango que abarca ese tipo.

```
HTTP/1.1 206 Partial Content
Accept-Ranges: bytes
Content-Type: multipart/byteranges; boundary=3d6b6a416f9b5
Content-Length: 385

--3d6b6a416f9b5
Content-Type: text/html
Content-Range: bytes 100-200/1270

eta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta name="viewport" content
--3d6b6a416f9b5
Content-Type: text/html
Content-Range: bytes 300-400/1270

-color: #f0f0f2;
margin: 0;
padding: 0;
font-family: "Open Sans", "Helvetica
--3d6b6a416f9b5--
```

La importancia del MIME correcto

- Muchos servidores mandan recursos con el tipo **application/octet-stream**
- Por razones de seguridad, muchos navegadores no permiten fijar una acción por defecto para esos recursos, forzando al usuario a salvar el archivo al disco para ser usado.
- Con **archivos RAR** comprimidos: Lo ideal sería el tipo de los archivos originales, pero es imposible y además un .rar puede contener muchos tipos. En estos casos debe configurarse el servidor para enviar **application/x-rar-compressed**
- **Audio y vídeo:** sólo los recursos con un correcto MIME se reproducirán en las etiquetas **<video>** o **<audio>**.
- Con **archivos propietarios** a veces no es posible especificar un comportamiento por defecto.
- **Plugins:** Añadidos al navegador, como **FLASH**. Pueden hacer que determinados contenidos puedan ser visualizados en el navegador.
- **Helpers:** Son aplicaciones que el navegador puede **abrir predeterminadamente** tras descargarse un archivo de un tipo MIME conocido, como es el caso de un documento de PowerPoint **application/vnd.ms-powerpoint**

Alternativas a MIME

- Los tipos MIME no son las únicas formas de saber el tipo de documento.
- **En sistemas Windows se utilizan las extensiones** (.txt, .exe, .doc, .zip...). Pero otros sistemas pueden no considerar estos sufijos significativos (Linux o MacOS).
- A través de **números mágicos** (Magic Numbers): Algunos tipos de archivo se pueden identificar por su estructura de bytes.
 - Los **.gif** comienzan por 47 48 46 38 39 (GIF89)
 - Los **.png** comienzan con 89 50 4E 47 (.PNG)
 - No todos los tipos de archivo tienen números mágicos.

COOKIES

- Navegar la web consiste en realizar diferentes peticiones a servidores HTTP.
- **HTTP es un protocolo sin estado.** El servidor **no guarda un historial** de las peticiones que ha realizado el cliente. Le da una respuesta, y se olvida de él.
 - Este modelo está bien para consultar información de documentos públicos.
- Surge la **necesidad de identificar** al cliente:
 - Periódicos online con **suscripción**.
 - Tiendas online con **carrito de la compra**.
 - Portales como Yahoo! que permiten una **personalización** de una página inicial.
- Observar la **IP del cliente** no es una buena idea:
 - Puede **cambiar** con el tiempo.
 - El cliente puede comprar IP con **NAT**.
 - La IP podría **identificar al equipo**, pero no al usuario.



Cookies

- El mecanismo que utilizaron los servidores para **reconocer a los clientes** fueron las **Cookies** (Galletitas).
- Una cookie HTTP es una pequeña **pieza de datos** que **el servidor envía al navegador** del usuario.
- Se usan principalmente para tres propósitos:
 - **Gestión de sesiones**: Inicios de sesión, carritos de la compra, o cualquier cosa que el servidor deba recordar.
 - **Personalización**: Temas de usuario, preferencias y otras configuraciones.
 - **Rastreo**: Guardar y analizar el comportamiento del usuario. (Por este motivo han sido criticadas).
- Las cookies **se envían con cada solicitud**, por lo que pueden empeorar el rendimiento.
- En la actualidad hay alternativas como **Web Storage API** e **IndexedDB**.

Cookies



Algunos ejemplos de Cookies

Dominio	Path	Contenido	Caducidad	Segura (Secure)
portalnoticias.com	/	IDUsuario=297932	15-10-10 17:00	Sí
tiendaropa.com	/	Carrito=1-004;1-080; 2-1454	11-1-11 22:30	No
noticias.com	/	Preferencias=Tiempo: Santiago;Equipo:Celt a	31-12-20 23:59	NO

- Restricciones: Límite de 50 cookies por dominio, tamaño menor o igual a 4093 bytes.

Cookies

- Al recibir una solicitud HTTP, un servidor puede enviar un encabezado **Set-Cookie** con la respuesta.
- La cookie se envía con solicitudes hechas al mismo servidor dentro de un encabezado **Cookie**.
- Se puede especificar la fecha de caducidad, después de la cual, deja de enviarse la cookie.
- Se pueden establecer **restricciones a un dominio y ruta específicos**, para **limitar el lugar al que se envía una cookie**.

Respuesta servidor:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry

[page content]
```

Siguiente petición del cliente:

```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

Cookies de Sesión, Permanentes y Seguras

- **Cookies de sesión:** se eliminan cuando el cliente se cierra.
- Por eso no se especifica una directiva **Expires** o **Max-Age**. Sin embargo los navegadores pueden usar la **restauración de sesiones**, lo que hacen que **las cookies de sesión sean permanentes**, como si el navegador nunca se cerrase.
- **Cookies Permanentes:** En lugar de expirar cuando el cliente se cierra, exiran en una fecha específica (Expires) o tras un periodo de tiempo (Max-Age)

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;
```

NOTA: La fecha y hora son relativas al cliente, no al servidor.

- **Cookies Secure:** Una cookie segura sólo se envía al servidor con **una petición cifrada sobre el protocolo HTTPS**. Incluso con la directiva Secure, **no debería almacenarse nunca información sensible en Cookies**.
 - Los sitios inseguros **http no pueden establecer cookies con la directiva Secure** en navegadores modernos.

Las Cookies y la privacidad

- Las cookies pueden utilizarse para **web tracking**, por eso han sido controvertidas.
- Un servidor puede llevar la cuenta de las páginas que visita un usuario. Le fija una cookie Contador=1 y luego la va incrementando.
- **Plataformas de publicidad** pueden **seguir el comportamiento del cliente** en varias páginas webs de servidores distintos:
 - Imaginemos que una agencia de publicidad pone una imagen en varias páginas webs: <http://www.agenciapublicidad.es/imagendesequimiento.gif>
 - El navegador, al cargar una de esas webs, solicita también esa imagen referenciada en el código HTML de la web que visita.
 - El servidor de la agencia de publicidad le asigna una cookie para su dominio que lo identifique como usuario: **UserId=341232**
 - El cliente visita otra página que contenga también la imagen de la agencia de publicidad.
 - La agencia de publicidad ya sabe que el usuario **ha visitado dos páginas diferentes y elabora un perfil**. Incluso aunque el cliente no haya pinchado en la publicidad o en ninguna imagen.

<https://www.ionos.es/digitalguide/paginas-web/derecho-digital/la-ley-de-cookies-y-su-aplicacion-en-espana/>

Ataques XSS

- Ataques de secuencia de comandos en sitios cruzados o **Cross-site scripting (XSS)**
- Es una vulnerabilidad informática muy popular y típica de las aplicaciones web.
- Permiten que una tercera persona inyecte código de cliente (JavaScript, VBScript...) en las páginas de forma que los navegadores ejecuten ese código al visitarla.
- Se producen porque la aplicación web **no valida la entrada de los usuarios correctamente**, por ejemplo, en formularios.
- Sirven para **robar cookies y sesiones, modificar** el sitio web, realizar **peticiones con la sesión del usuario, redireccionar** usuarios a sitios dañinos (**phishing**), atacar al navegador, instalar **malware**, reescribir o manipular las extensiones del navegador, etc.
- Ejemplo: https://www.youtube.com/watch?v=wED_DyDCYXQ

Ataques XSS

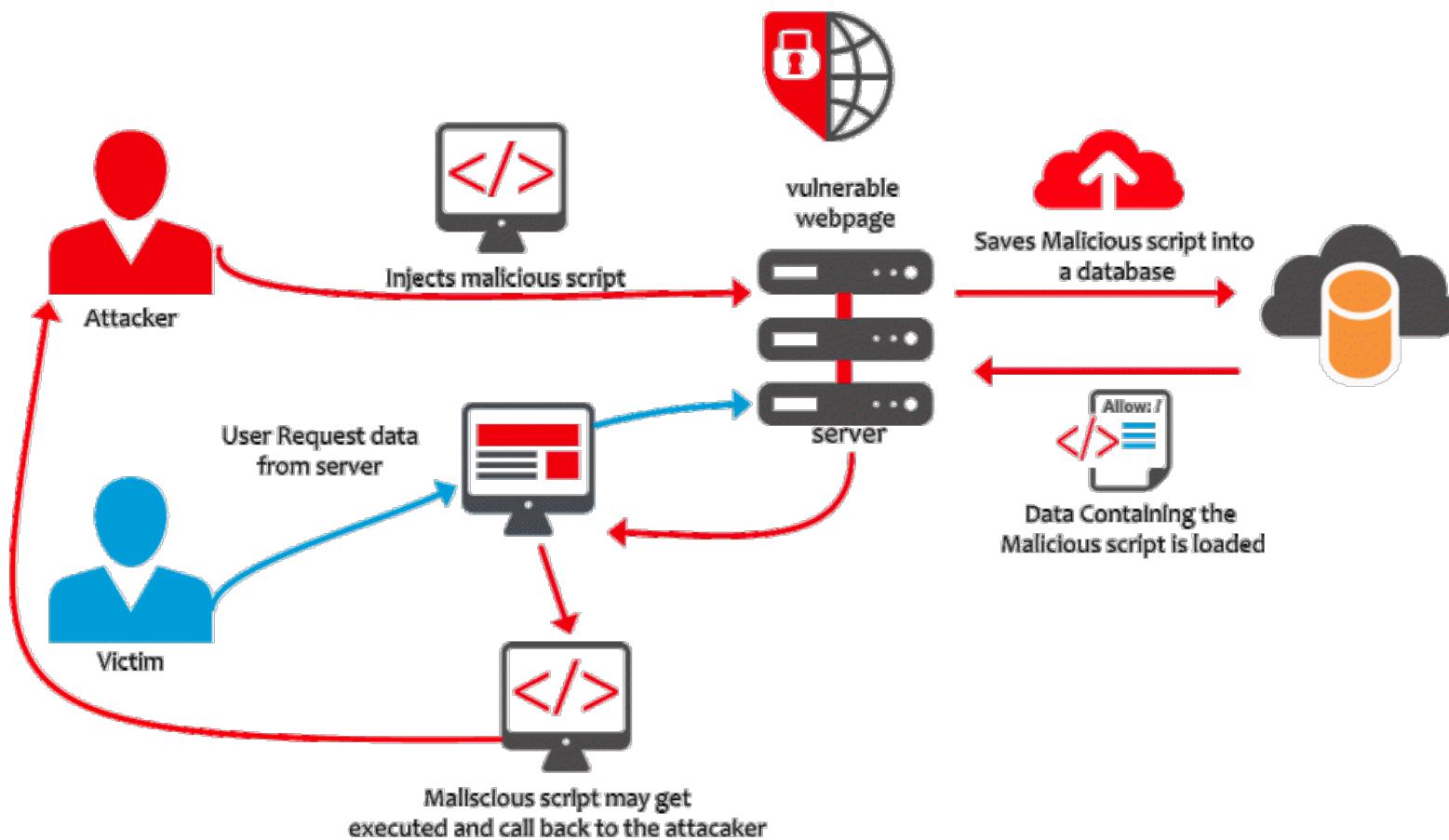
- **XXS Persistente:**

- Si el código insertado queda **almacenado en el servidor**, formando parte de un mensaje de un foro, el ataque se dice que es persistente.
- Cualquier usuario que entre a leer dicha contribución leerá el mensaje, pero no el código inyectado, que será ejecutado por su navegador.
- Por ejemplo puede robarle la sesión enviando todas sus cookies a un tercer servidor.

- **XSS Reflejado:**

- El código insertado no está almacenado en la web, **va dentro de un enlace** que se hace llegar de algún modo a la víctima para que pinche en él.
- El enlace puede ofuscarse para que no levante sospechas entre las víctimas.
- Este ataque sí puede ser dirigido contra un usuario en concreto, al que se quiere suplantar su acceso en el servidor.

Ataques XSS

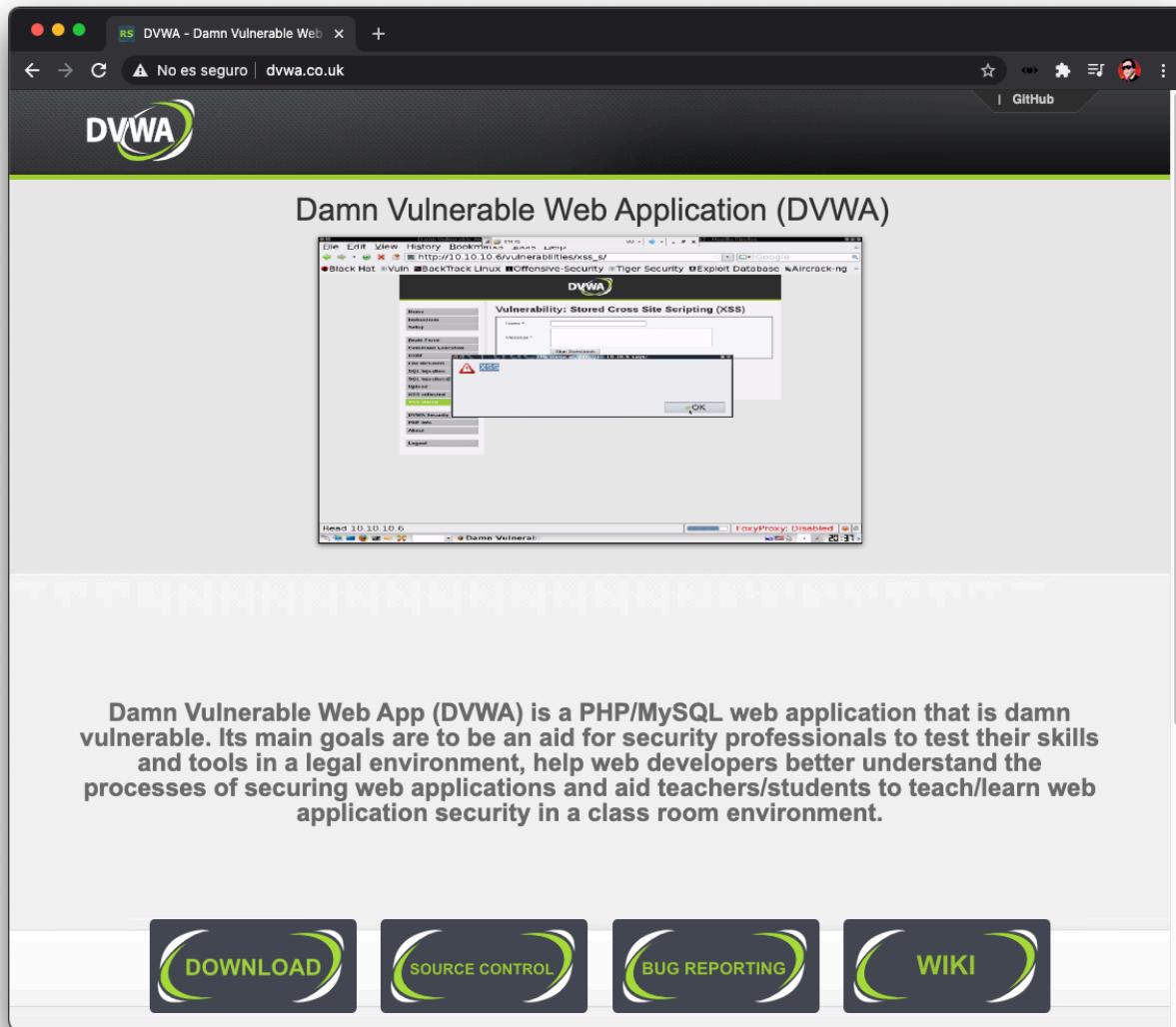


Cookies y los ataques XSS

- Para prevenir ataques de **cross-site scripting (XSS)**, las **cookies HttpOnly** son **inaccesibles desde la API de Javascript** (`Document.cookie`).
- Por ejemplo, **las cookies que persisten sesiones** del lado del servidor **no necesitan estar disponibles para Javascript**, por lo que debería establecerse el flag **HttpOnly**.

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly
```

- Otras maneras de defenderse:
 - El administrador debe **filtrar las entradas** de los usuarios.
 - No hacer **click** en enlaces sospechosos.
 - Etc.

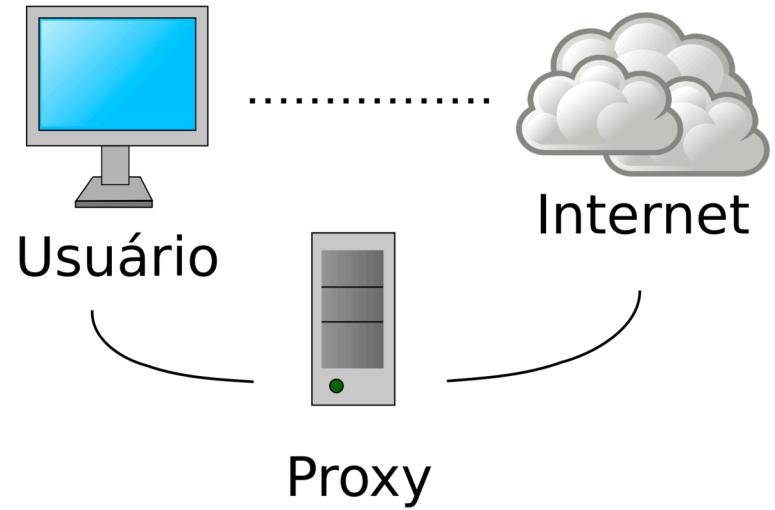


- **Damn Vulnerable Web Application (DVWA):** <http://www.dvwa.co.uk/>

PROXY

Proxy

- Un proxy es un equipo que hace de **intermediario entre las conexiones** de un cliente y de un servidor.
- En lugar de establecer una conexión directamente con un servidor, la conexión se puede establecer a través de un proxy: tu navegador se conecta al proxy y el proxy al servidor.
 - El **servidor** al que haces peticiones **no sabrá tu IP**.
 - Puedes saltarte **bloqueos de contenido regionales** o beneficiarte de ofertas disponibles en otra región.
 - Los proxy pueden bloquear cookies, scripts y otros objetos alojados en webs.
 - Permiten navegar de una forma más **privada y anónima** (si te fías del proxy, claro).
- Un **VPN cifra todo el tráfico** que pasa por él, un **proxy no** cifra el tráfico y sólo actúa para las conexiones donde lo actives.



Proxy

- Un proxy o servidor proxy en una red informática es un programa o dispositivo que hace de **intermediario en las peticiones de recursos** que realiza un cliente A a un servidor C.
- El **cliente A** realiza la petición a través del **equipo B (el proxy)**, que a su vez trasladará la petición al **servidor C**.
- Esta situación estratégica de punto intermedio permite ofrecer diversas funcionalidades:
 - **Control de acceso**
 - **Registro de tráfico**
 - **Restricción a determinados tipos de tráfico**
 - **Mejora de rendimiento**
 - **Anonimato de la comunicación**
 - **Caché web**
 - etc.

Proxy: Ventajas

- **Control:** Solamente el intermediario hace el **trabajo real**, por lo tanto se pueden **limitar y restringir los derechos de los usuarios** y dar permisos únicamente al servidor proxy.
- **Ahorro:** Solamente uno de los usuarios (proxy) ha de estar preparado para hacer el trabajo real. es el único que necesita los **recursos necesarios** para realizar esa funcionalidad. Podemos entender como recursos la **capacidad y lógica de la dirección de red externa (IP)**.
- **Velocidad:** Si varios clientes van a pedir el **mismo recurso**, el proxy puede hacer de **caché**.
- **Filtrado:** El proxy puede **negarse a responder** algunas peticiones si detecta que están prohibidas.
- **Modificación / Seguridad:** Un proxy puede **falsificar información o modificarla** siguiendo un algoritmo. Aunque según como se mire, también se puede considerar una desventaja. Puede **adaptar los contenidos a un dispositivo** (mejorando la visualización o ahorrando datos), **bloquear cookies**, **modificar cabeceras HTTP** o quitar **código JavaScript** que considere **peligroso**.
- **Anonimato / Privacidad:** Un proxy puede conectarse de forma anónima a un recurso externo **sin revelar la IP del cliente**.

Proxy: Inconvenientes

- **Anonimato:** Si todos los usuarios parecen ser uno solo, a lo mejor el recurso accedido no es capaz de diferenciarlos.
- **Carga:** Tiene que hacer el **trabajo de muchos usuarios**.
- **Intromisión:** Los **usuarios pueden no querer pasar por el proxy**, más si guarda caché o copias de los datos.
- **Irregularidad:** Al representar a más de un usuario, es posible que haya **escenarios donde aparezcan problemas**, en concreto en los que presuponen una comunicación directa entre 1 emisor y 1 receptor (como TCP/IP).

Proxy Caché

- Conserva el contenido solicitado por el usuario para **acelerar la respuesta a futuras peticiones** de la misma información **de la misma máquina o de otra**.
- Algunos usuarios pueden considerarlo una **intromisión** o violación de su intimidad.
- **Reduce el tráfico**, mejora la **velocidad**, permite **filtrar contenidos**, **esconde la identidad** de quien solicita el recurso, puede contener mecanismos de **seguridad**, etc.

Proxy transparente

- Muchas organizaciones usan los proxies para **reforzar las políticas de uso de la red**, proporcionar **seguridad** y servicios de **caché**.
- Un **proxy web no es transparente**: debe ser configurado manualmente. Por lo tanto **el usuario puede cambiar la configuración**.
- Un **proxy transparente** combina **un proxy con un cortafuegos**, de forma que **intercepta las conexiones y las desvía** sin necesidad de cambios en la configuración en el cliente y sin que el usuario conozca su existencia.

Proxy Inverso (Reverse Proxy)

- Es un **servidor proxy** situado en el **alojamiento de uno o más servidores web**.
- Todo el **tráfico** de internet con destino a esos servidores **es recibido por el proxy**.
 - **Seguridad:** Es una capa adicional de defensa.
 - **Cifrado / Aceleración SSL:** Puede estar equipado con hardware de aceleración SSL/TLS.
 - **Distribución de carga:** Puede repartir la carga entre varios servidores.
 - **Caché de contenido estático:** Puede descargar de trabajo a los servidores web sirviendo el contenido estático.

Proxy abierto

- Es un servidor proxy que **acepta peticiones desde cualquier ordenador** esté o no conectado a su red.
- Puede ofrecer las ventajas e inconvenientes típicos de los proxies, pero al estar “**abierto**” también se puede utilizar para **usos maliciosos** como **spam** o ciertos tipos de **ataques**.
- Muchos servidores, como los de **IRC** (un protocolo de chat) y de **correo**, **bloquean este tipo de proxys** utilizando listas negras (**blacklists**).

HTTP: CÓDIGOS DE ESTADO

Códigos de estado

- Son códigos de 3 dígitos definidos por el IANA, que se incluyen en las respuestas.

1XX **Códigos informativos.** Informan al navegador de las acciones que se van a realizar.

2XX **Códigos de éxito.** Indican que la petición del navegador se ha recibido, procesado y respondido correctamente.

3XX **Códigos de acción adicional** (normalmente, redirección). El navegador debe realizar alguna acción para que la petición se complete.

4XX **Códigos de error del cliente.** Indica que se ha producido un error y el responsable es el cliente.

5XX **Códigos de error del servidor.** Indica que se ha producido un error y que el responsable es el servidor.

- <https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml>

Códigos de estado 1XX

Informan al navegador de algunas acciones que se van a realizar:

- **100 (Continue)**, el navegador puede continuar realizando su petición (se utiliza para indicar que la primera parte de la petición del navegador se ha recibido correctamente).
- **101 (Switching Protocols)**, el servidor acepta el cambio de protocolo propuesto por el navegador (puede ser por ejemplo un cambio de HTTP 1.0 a HTTP 1.1).
- **102 (Processing (WebDAV))**, el servidor está procesando la petición del navegador pero todavía no ha terminado (esto evita que el navegador piense que la petición se ha perdido cuando no recibe ninguna respuesta).
- **103 (Checkpoint)**, se va a reanudar una petición POST o PUT que fue abortada previamente.

Códigos de estado 2XX

Indican que la petición del navegador se ha recibido, procesado y respondido correctamente:

- **200 (Ok)**, la petición del navegador se ha completado con éxito.
- **201 (Created)**, la petición del navegador se ha completado con éxito y como resultado, se ha creado un nuevo recurso (la respuesta incluye la URI de ese recurso).
- **202 (Accepted)**, la petición del navegador se ha aceptado y se está procesando en estos momentos, por lo que todavía no hay una respuesta (se utiliza por ejemplo cuando un proceso realiza una petición muy compleja a un servidor y no quiere estar horas esperando la respuesta).
- **203 (Non-Authoritative Information)**, la petición se ha completado con éxito, pero su contenido no se ha obtenido de la fuente originalmente solicitada sino de otro servidor.
- **204 (No Content)**, la petición se ha completado con éxito pero su respuesta no tiene ningún contenido (la respuesta sí que puede incluir información en sus cabeceras HTTP).
- **205 (Reset Content)**, la petición se ha completado con éxito, pero su respuesta no tiene contenidos y además, el navegador 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)**, La respuesta de esta petición sólo tiene parte de los contenidos, tal y como lo solicitó el propio navegador (se utiliza por ejemplo cuando se descarga un archivo muy grande en varias partes para acelerar la descarga).
- **207 (Multi-Status (WebDAV))**, la respuesta consiste en un archivo XML que contiene en su interior varias respuestas diferentes (el número depende de las peticiones realizadas previamente por el navegador).
- **208 (Already Reported (WebDAV))**, el listado de elementos DAV ya se notificó previamente, por lo que no se van a volver a listar.

Códigos de estado 3XX

- **300 (Multiple Choices)**, existe más de una variante para el recurso solicitado por el navegador (por ejemplo si la petición se corresponde con más de un archivo).
- **301 (Moved Permanently)**, el recurso solicitado por el navegador se encuentra en otro lugar y este cambio es permanente. El navegador es redirigido automáticamente a la nueva localización de ese recurso (este código es muy importante para tareas relacionadas con el SEO de los sitios web).
- **302 (Moved Temporarily)**, el recurso solicitado por el navegador se encuentra en otro lugar, aunque sólo por tiempo limitado. El navegador es redirigido automáticamente a la nueva localización de ese recurso.
- **303 (See Other)**, el recurso solicitado por el navegador se encuentra en otro lugar. El servidor no redirige automáticamente al navegador, pero le indica la nueva URI en la que se puede obtener el recurso.
- **304 (Not Modified)**, cuando el navegador pregunta si un recurso ha cambiado desde la última vez que se solicitó, el servidor responde con este código cuando el recurso no ha cambiado.
- **305 (Use Proxy)**, el recurso solicitado por el navegador debe obtenerse a través del *proxy* cuya dirección se indica en la cabecera *Location* de esta misma respuesta.
- **306 (Switch Proxy)**, este código se utilizaba en las versiones antiguas de HTTP pero ya no se usa (aunque está reservado para usos futuros).
- **307 (Temporary Redirect)**, el recurso solicitado por el navegador se puede obtener en otro lugar, pero sólo para esta petición. Las próximas peticiones pueden seguir utilizando la localización original del recurso.
- **308 (Permanent Redirect)**, el recurso solicitado por el navegador se encuentra en otro lugar y este cambio es permanente. A diferencia del código 301, no se permite cambiar el método HTTP para la nueva petición (así por ejemplo, si envías un formulario a un recurso que ha cambiado de lugar, todo seguirá funcionando bien).

Códigos de estado 4XX (400-408)

Indican que se ha producido un error cuyo responsable es el navegador:

- **400 (Bad Request)**, el servidor no es capaz de entender la petición del navegador porque su sintaxis no es correcta.
- **401 (Unauthorized)**, el recurso solicitado por el navegador requiere de autenticación. La respuesta incluye una cabecera de tipo `WWW-Authenticate` para que el navegador pueda iniciar el proceso de autenticación.
- **402 (Payment Required)**, este código está reservado para usos futuros.
- **403 (Forbidden)**, la petición del navegador es correcta, pero el servidor no puede responder con el recurso solicitado porque se ha denegado el acceso.
- **404 (Not Found)**, el servidor no puede encontrar el recurso solicitado por el navegador y no es posible determinar si esta ausencia es temporal o permanente.
- **405 (Method Not Allowed)**, el navegador ha utilizado un método (GET, POST, etc.) no permitido por el servidor para obtener ese recurso.
- **406 (Not Acceptable)**, el recurso solicitado tiene un formato que en teoría no es aceptable por el navegador, según los valores que ha indicado en la cabecera `Accept` de la petición.
- **407 (Proxy Authentication Required)**, es muy similar al código 401, pero en este caso, el navegador debe autenticarse primero con un proxy.
- **408 (Request Timeout)**, el navegador ha tardado demasiado tiempo en realizar su petición y el servidor ya no espera esa petición. No obstante, el navegador puede realizar nuevas peticiones cuando quiera.

Códigos de estado 4XX (409-416)

Indican que se ha producido un error cuyo responsable es el navegador:

- **409** (*Conflict*), la petición del navegador no se ha podido completar porque se ha producido un conflicto con el recurso solicitado. El caso más habitual es el de las peticiones de tipo PUT que intentan modificar un recurso que a su vez ya ha sido modificado por otro lado.
- **410** (*Gone*), no es posible encontrar el recurso solicitado por el navegador y esta ausencia se considera permanente. Si existe alguna posibilidad de que el recurso vuelva a estar disponible, se debe utilizar el código 404.
- **411** (*Length Required*), el servidor rechaza la petición del navegador porque no incluye la cabecera Content-Length adecuada.
- **412** (*Precondition Failed*), el servidor no es capaz de cumplir con algunas de las condiciones impuestas por el navegador en su petición.
- **413** (*Request Entity Too Large*), la petición del navegador es demasiado grande y por ese motivo el servidor no la procesa.
- **414** (*Request-URI Too Long*), la URI de la petición del navegador es demasiado grande y por ese motivo el servidor no la procesa (esta condición se produce en muy raras ocasiones y casi siempre porque el navegador envía como GET una petición que debería ser POST).
- **415** (*Unsupported Media Type*), la petición del navegador tiene un formato que no entiende el servidor y por eso no se procesa.
- **416** (*Requested Range Not Satisfiable*), el navegador ha solicitado una porción inexistente de un recurso. Este error se produce cuando el navegador descarga por partes un archivo muy grande y calcula mal el tamaño de algún trozo.

Códigos de estado 4XX (417-431)

Indican que se ha producido un error cuyo responsable es el navegador:

- **417** (*Expectation Failed*), la petición del navegador no se procesa porque el servidor no es capaz de cumplir con los requerimientos de la cabecera Expect de la petición.
- **422** (*Unprocessable Entity (WebDAV)*), la petición del navegador tiene el formato correcto, pero sus contenidos tienen algún error semántico que impide al servidor responder.
- **423** (*Locked (WebDAV)*), el recurso solicitado por el navegador no se puede entregar porque está bloqueado.
- **424** (*Failed Dependency (WebDAV)*), la petición del navegador ha fallado debido al error de alguna petición anterior (por ejemplo una petición con el método PROPPATCH).
- **426** (*Upgrade Required*), el navegador debe cambiar a un protocolo diferente para realizar las peticiones (por ejemplo TLS/1.0).
- **428** (*Precondition Required*), el servidor requiere que la petición del navegador sea condicional (este tipo de peticiones evitan los problemas producidos al modificar con PUT un recurso que ha sido modificado por otra parte).
- **429** (*Too Many Requests*), el navegador ha realizado demasiadas peticiones en un determinado período de tiempo (se utiliza sobre todo para forzar los límites de consumo de recursos de las APIs).
- **431** (*Request Header Fileds Too Large*), el servidor no puede procesar la petición porque una de las cabeceras de la petición es demasiado grande. Este error también se produce cuando la suma del tamaño de todas las peticiones es demasiado grande.

Códigos de estado 5XX (500-506)

Indican que se ha producido un error cuyo responsable es el servidor:

- **500 (*Internal Server Error*)**, la solicitud del navegador no se ha podido completar porque se ha producido un error inesperado en el servidor.
- **501 (*Not Implemented*)**, el servidor no soporta alguna funcionalidad necesaria para responder a la solicitud del navegador (como por ejemplo el método utilizado para la petición).
- **502 (*Bad Gateway*)**, el servidor está actuando de *proxy* o *gateway* y ha recibido una respuesta inválida del otro servidor, por lo que no puede responder adecuadamente a la petición del navegador.
- **503 (*Service Unavailable*)**, el servidor no puede responder a la petición del navegador porque está congestionado o está realizando tareas de mantenimiento.
- **504 (*Gateway Timeout*)**, el servidor está actuando de *proxy* o *gateway* y no ha recibido a tiempo una respuesta del otro servidor, por lo que no puede responder adecuadamente a la petición del navegador.
- **505 (*HTTP Version Not Supported*)**, el servidor no soporta o no quiere soportar la versión del protocolo HTTP utilizada en la petición del navegador.
- **506 (*Variant Also Negotiates*)**, el servidor ha detectado una referencia circular al procesar la parte de la negociación del contenido de la petición.

Códigos de estado 5XX (507-511)

Indican que se ha producido un error cuyo responsable es el servidor:

- **507** (*Insufficient Storage (WebDAV)*), el servidor no puede crear o modificar el recurso solicitado porque no hay suficiente espacio de almacenamiento libre.
- **508** (*Loop Detected (WebDAV)*), la petición no se puede procesar porque el servidor ha encontrado un bucle infinito al intentar procesarla.
- **510** (*Not Extended*), la petición del navegador debe añadir más extensiones para que el servidor pueda procesarla.
- **511** (*Network Authentication Required*), el navegador debe autenticarse para poder realizar peticiones (se utiliza por ejemplo con los portales cautivos que te obligan a autenticarte antes de empezar a navegar).

Códigos de estado más importantes

- **Código de estado 200 –OK:**
 - El código de respuesta 200 indica que la solicitud ha sido procesada correctamente.
 - Todos los datos solicitados fueron recibidos por el servidor y se trasmitirán al cliente.
 - Los usuarios de Internet no suelen encontrarse este código.
- **Código de estado 301 –Moved Permanently:**
 - El código 301 significa que los datos solicitados por el cliente ya no se encuentran bajo la misma dirección de Internet, sino que han sido desplazados de manera permanente.
 - Debido a que la ubicación actual del contenido solicitado está incluida en el informe de estado, el navegador web podrá redireccionar la petición a la nueva dirección. Así, el usuario es redireccionado y la antigua dirección pierde validez.
 - Este código suele pasar desapercibido ante los ojos de los usuarios, el único cambio visible es la redirección de la URL en la barra de direcciones.
- **Código de estado 302 – Moved Temporarily:**
 - A diferencia del 301, que hace referencia a una reubicación permanente, el código 302 informa que los datos solicitados están disponibles temporalmente en una dirección diferente. A
 - Aquí, la ubicación de la información es especificada, lo que genera una redirección automática. La antigua dirección sigue siendo válida.

Códigos de estado más importantes

- **Código de estado 403 – Forbidden:**
 - El código de estado HTTP 403 indica al cliente que los datos solicitados están protegidos y, por ende, se le ha denegado el acceso debido a la falta de autorización del cliente.
 - En estos casos, el usuario de Internet verá una página en formato HTML generada automáticamente que le informará sobre su falta de permisos para ejecutar la acción.
- **Código de estado 404 – Not Found:**
 - Cuando el servidor envía el código 404 como respuesta significa que no fue posible encontrar los datos de la página web solicitada en el servidor.
 - Esto sucede cuando la dirección ya no existe o los contenidos han sido trasladados sin especificar la nueva dirección. Cuando a un usuario le aparece el código de estado 404, deberá comprobar si ha introducido la URL de manera correcta en la barra de direcciones.
 - Aquellos enlaces que dirigen a páginas inexistentes suelen conocerse como “**enlaces muertos**”.

Códigos de estado más importantes

- **Código de estado 500 –Internal Server Error:**

- Este tipo de respuesta actúa como un código de estado colectivo para un error inesperado en el servidor.
- Si el servidor falla y está interrumpiendo la recuperación de los datos solicitados, se emitirá automáticamente el código de estado 500.
- Además de la respuesta al cliente, el servidor web generará un tipo de registro de error interno en donde se expliquen más detalles sobre el error.
- Este último debe ser analizado por propietarios de páginas web que estén interesados en hacer las respectivas reparaciones en el software del servidor.

- **Código de estado 503 –Service Unavailable:**

- Este código de respuesta indica que el servidor web destinado a proporcionar la información está sobrecargado.
- Es común que esta respuesta del servidor incluya, además, información sobre cuándo se podrá procesar nuevamente la primera solicitud.
- Aquí, los usuarios de Internet pueden asumir que el administrador está trabajando en el problema y que, por lo tanto, el servidor estará disponible de nuevo.

Códigos de estado más importantes

- **Código de estado 500 –Internal Server Error:**

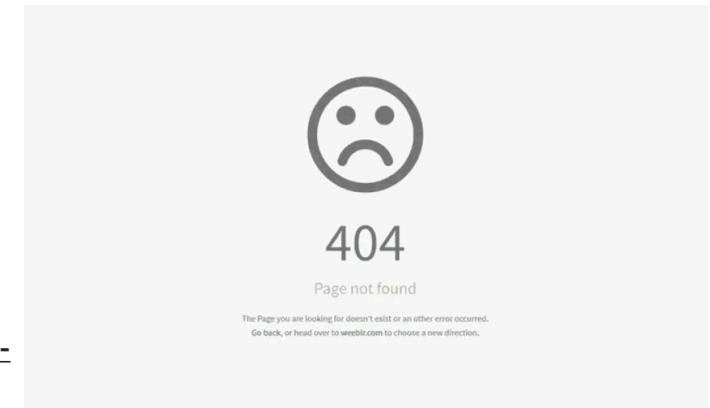
- Este tipo de respuesta actúa como un código de estado colectivo para un error inesperado en el servidor.
- Si el servidor falla y está interrumpiendo la recuperación de los datos solicitados, se emitirá automáticamente el código de estado 500.
- Además de la respuesta al cliente, el servidor web generará un tipo de registro de error interno en donde se expliquen más detalles sobre el error.
- Este último debe ser analizado por propietarios de páginas web que estén interesados en hacer las respectivas reparaciones en el software del servidor.

- **Código de estado 503 –Service Unavailable:**

- Este código de respuesta indica que el servidor web destinado a proporcionar la información está sobrecargado.
- Es común que esta respuesta del servidor incluya, además, información sobre cuándo se podrá procesar nuevamente la primera solicitud.
- Aquí, los usuarios de Internet pueden asumir que el administrador está trabajando en el problema y que, por lo tanto, el servidor estará disponible de nuevo.

Códigos de estado para un administrador

- Los administradores de webs deben de mejorar la experiencia de usuario **limitando el número de páginas de error HTML al mínimo.**
- Es especialmente importante en páginas de tiendas, con contenido dinámico.
- Un **error 404** puede hacerte **perder clientes**. Puedes también, convertirlo en una oportunidad:
 - <https://juannunezblasco.es/error-404-not-found/>
- Existen **herramientas para identificar errores:**
 - Plugins de **Wordpress** que localizan enlaces rotos.
 - **Webmaster Tools de Google**
 - Utilizar el archivo **.htaccess** para redirigir las páginas:
 - <https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/los-mejores-trucos-para-archivos-htaccess/> (permite definir directivas de configuración)
 - Etc.
- Juegan **un papel esencial en el SEO**: El posicionamiento de la página en Internet.
 - Se penalizan los enlaces rotos.
 - Un crawler o robot de un motor de búsqueda, registra las páginas 301 como inválidas.t



HTTP: MÉTODOS

Métodos HTTP

- HTTP define un conjunto de **métodos de petición** para **indicar la acción** que se desea realizar sobre un recurso determinado. Son también llamados verbos (**HTTP verbs**).
 - En **RFC 7231**, sección 4 se especifican GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS y TRACE.
 - En **RFC 5789**, sección 2: Se especifica el método PATCH.

Método	Descripción
GET	Lee una web
HEAD	Lee la cabecera de una web
POST	Añade a una web
PUT	Almacena una web
DELETE	Borra una web
TRACE	Hace eco de la petición
CONNECT	Conecta a través de un proxy
OPTIONS	Consulta los métodos de una web
PATCH	Realiza modificaciones parciales a una web

Donde se dice web, puede entenderse objeto en el caso general.

Métodos HTTP: GET y HEAD

- El método **GET** solicita una representación de un recurso especificado.
- Las peticiones que utilizan el método GET **sólo deben recuperar datos**.
 - **GET /images/logo.png HTTP/1.1**
- El método **HEAD** pide una respuesta idéntica a **GET** pero **sin el cuerpo de la respuesta**. Es útil para recuperar meta-information escrita en los encabezados de la respuesta sin tener que transportar todo el contenido.

Métodos HTTP: POST

- El método **POST** se utiliza para **enviar datos al servidor** (por ejemplo, para enviar un formulario).
- **Se diferencia de GET en:**
 - Hay un bloque de datos que se envía con la solicitud (en el cuerpo de la misma).
 - Hay normalmente headers que describen el cuerpo que se envía como Content-Type y Content-Length.
 - En caso de formularios: **Content-Type: application/x-www-form-urlencoded**
 - El URI que se solicita no es un recurso, normalmente es un script al que se le envían los datos.
 - La respuesta HTTP normalmente es generada de forma **dinámica**.
 - Causa a menudo un **cambio en el estado o efectos** secundarios en el servidor.

Métodos HTTP: POST

POST /directorio/script.cgi HTTP/1.0

User-Agent: TuBrowser/1.7

Content-Type: application/x-www-form-urlencoded

Content-Length: 26

nombre=Juan&Apellido=Perez

Métodos HTTP: POST vs GET

- Los navegadores pueden enviar **formularios** usando métodos GET y POST.
- **GET:**
 - Añade los datos en la **URL** en pares de **nombre y valor**.
 - La longitud de la URL está limitada a 3000 caracteres.
 - **Nunca** se usa GET para **mandar datos sensibles**, como claves, porque serán vistas en la URL.
 - GET es bueno para datos no seguros, como términos de búsqueda, porque se puede añadir a marcadores o compartir el enlace.
- **POST:**
 - Añade los datos del formulario **dentro del cuerpo de la petición HTTP**.
 - Los datos no se ven en la URL.
 - No tiene límite de tamaño.
 - Las peticiones POST no pueden añadirse a marcadores.

PETICIÓN GET

GET /registro.php?usuario=jose&clave=secreta HTTP/1.1
Host:www.dominio.com

PETICIÓN POST

POST /registro.php HTTP/1.1
Host: www.dominio.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

usuario=jose&clave=secreta

https://www.w3schools.com/tags/att_form_method.asp

Métodos HTTP: PUT

- El método PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- Sube o actualiza un recurso especificado en el cuerpo del método (body).
- PUT es igual que POST, excepto por su diferencia semántica. Mientras en POST el URI identifica el recurso que manejará los datos enviados (como un script), con PUT la URI el recurso que se creará o reemplazará con el contenido de body.
- La diferencia entre PUT y POST es que PUT es un método idempotente. Llamarlo una o dos veces de forma sucesiva tiene el mismo efecto (sin efectos secundarios), mientras que una sucesión de peticiones POST idénticas pueden tener efectos adicionales, como enviar una orden varias veces.

Métodos HTTP: PUT

PETICIÓN

```
PUT /nuevo.html HTTP/1.1
Host: ejemplo.com
Content-type: text/html
Content-length: 16

<p>Nuevo Archivo</p>
```



RESPUESTA

```
HTTP/1.1 201 Created
Content-Location: /nuevo.html
```

201 Created: El elemento destino no existe y la petición PUT lo crea de forma satisfactoria

RESPUESTA

```
HTTP/1.1 204 No Content
Content-Location: /existente.html
```

204 No content: El elemento existe actualmente y es modificado de forma satisfactoria.

<https://tools.ietf.org/html/rfc7231#section-4.3.4>

Métodos HTTP: DELETE

- El método **borra el recurso** especificado.

PETICIÓN

```
DELETE /file.html HTTP/1.1
```

RESPUESTA 200 OK

```
HTTP/1.1 200 OK
Date: Wed, 21 Oct 2015 07:28:00 GMT

<html>
  <body>
    <h1>File deleted.</h1>
  </body>
</html>
```

- Posibles respuestas de éxito:
 - **202 (Accepted)**: La acción probablemente tendrá éxito pero aún no se ha promulgado.
 - **204 (No Content)**: La acción se ha ejecutado y no se debe proporcionar más información.
 - **200 (OK)**: La acción se ha realizado y el mensaje de respuesta incluye una representación que describe el estado.

Métodos HTTP: CONNECT

- El método HTTP Connect inicia la comunicación en dos caminos con la fuente del recurso solicitado. Puede ser usado para **abrir una comunicación túnel**.
- El método CONNECT puede ser usado para acceder a sitios web que usan SSL (HTTPS). El cliente realiza la petición al **Servidor Proxy HTTP** para establecer una conexión tunel hacia un destino deseado. Entonces el servidor Proxy procede a realizar la conexión en nombre del cliente y una vez establecida la conexión el servidor Proxy envía los datos desde y hacia el cliente.
- Es un **método de salto entre servidores**.

Métodos HTTP: CONNECT

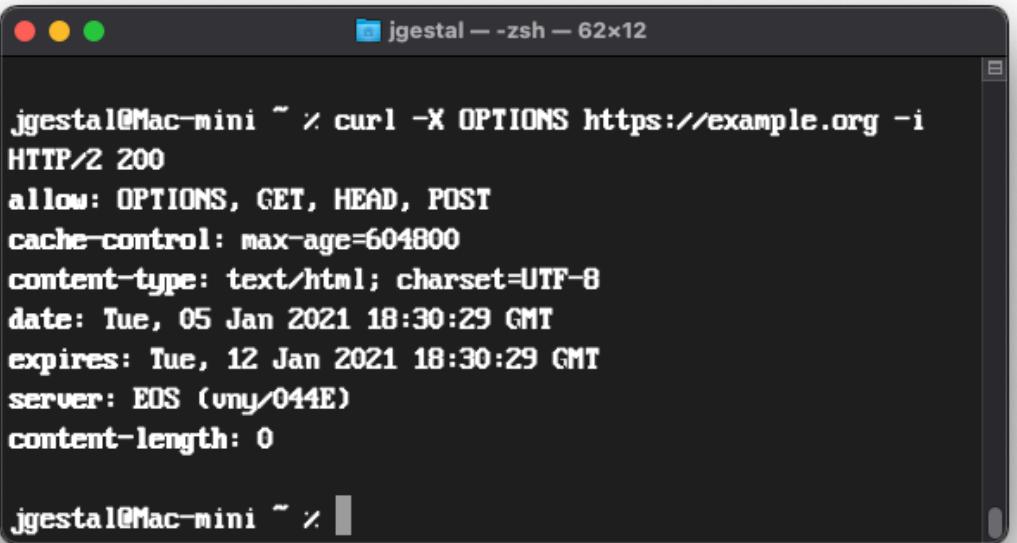
- Algunos servidores proxy pueden necesitar autorización para crear túneles. Para eso se envía el encabezado **Proxy-Authorization**.

PETICIÓN

```
CONNECT server.example.com:80 HTTP/1.1
Host: server.example.com:80
Proxy-Authorization: basic aGVsbG86d29ybGQ=
```

Métodos HTTP: OPTIONS

- El método **HTTP OPTIONS** solicita los comandos que se pueden utilizar para una URL o para un servidor (*).
 - **OPTIONS /index.html HTTP/1.1**
 - **OPTIONS * HTTP/1.1**



A screenshot of a macOS terminal window titled "jgestal -- zsh -- 62x12". The window displays the output of a curl command. The output shows the server's response headers for an OPTIONS request to https://example.org. The headers include:

```
jgestal@Mac-mini ~ % curl -X OPTIONS https://example.org -i
HTTP/2 200
allow: OPTIONS, GET, HEAD, POST
cache-control: max-age=604800
content-type: text/html; charset=UTF-8
date: Tue, 05 Jan 2021 18:30:29 GMT
expires: Tue, 12 Jan 2021 18:30:29 GMT
server: EOS (vny/044E)
content-length: 0
```

The terminal prompt "jgestal@Mac-mini ~ %" is visible at the bottom.

Métodos HTTP: TRACE

- El método HTTP **TRACE** solicita al servidor que envíe de vuelta, en un mensaje de respuesta, en la sección del cuerpo de entidad, los datos que reciba del mensaje de solicitud: **prueba de bucle**.
- Se utiliza con **fines de comprobación y diagnóstico**.

PETICIÓN

```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

RESPUESTA

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Connection: close
Content-Type: message/http
Content-Length: 39
```

```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

Métodos HTTP: TRACE

- El método HTTP **TRACE** solicita al servidor que envíe de vuelta, en un mensaje de respuesta, en la sección del cuerpo de entidad, los datos que reciba del mensaje de solicitud: **prueba de bucle**.
- Se utiliza con **fines de comprobación y diagnóstico**.

PETICIÓN

```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

RESPUESTA

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Connection: close
Content-Type: message/http
Content-Length: 39
```

```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

Métodos HTTP: PATCH

- El método HTTP **PATCH** aplica **modificaciones parciales** a un recurso.
- A diferencia de PUT, PATCH **no es idempotente**. Peticiones idénticas sucesivas pueden tener efectos diferentes. Sin embargo es posible emitir peticiones PATCH de tal forma que sí sean idempotentes.
- PATCH al igual que **POST** puede provocar **efectos secundarios** en otros recursos.
- Para averiguar si un servidor soporta PATCH, puede notificar su compatibilidad al añadirlo a la lista en el header: **Allow** o **Access-Control-Allow-Methods**.
- Otra indicación implícita es la presencia de la cabecera **Accept-Patch**.

PETICIÓN

```
PATCH /file.txt HTTP/1.1
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[description of changes]
```

RESPUESTA

```
HTTP/1.1 204 No Content
Content-Location: /file.txt
ETag: "e0023aa4f"
```

	La petición tiene cuerpo	La respuesta satisfactoria tiene cuerpo	Seguro (no altera datos en el servidor)	Idempotente	Cacheable	Permitido en formularios
GET	NO	SÍ	SÍ	SÍ	SÍ	SÍ
HEAD	NO	NO	SÍ	SÍ	SÍ	NO
POST	SÍ	SÍ	NO	NO	NO (salvo excepciones)	SÍ
DELETE	QUIZÁ	QUIZÁ	NO	SÍ	NO	NO
CONNECT	NO	SÍ	NO	NO	NO	NO
TRACE	NO	NO	SÍ	SÍ	NO	NO
PATCH	SÍ	SÍ	NO	NO	NO	NO

HTTP: CABECERAS

Cabeceras HTTP

```
1 Request:  
2 GET https://the-responsible.dev/  
3 Accept: text/html,application/xhtml+xml,application/xml  
4 Accept-Encoding: gzip, deflate, br  
5 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,de;q=0.7  
6 ...  
7  
8 Response:  
9 Connection: keep-alive  
10 Content-Type: text/html; charset=utf-8  
11 Date: Mon, 11 Mar 2019 12:59:38 GMT  
12 ...  
13 Response Body
```

- Las **cabeceras** (headers) **HTTP** permiten al cliente y al servidor enviar **información adicional** junto a una **petición o respuesta**.
- Una cabecera está compuesta por su nombre (no sensible a las mayúsculas) seguido de ':' y a continuación de su valor, sin saltos de línea. Los espacios en blanco a su izquierda son ignorados.

Cabeceras HTTP, un ejemplo

Request URI: <http://www.example.com>

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Age: 521648
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Fri, 06 Mar 2020 17:36:11 GMT
Etag: "3147526947+gzip"
Expires: Fri, 13 Mar 2020 17:36:11 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7EC9)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 648
```

- **HTTP/1.1** es la versión válida del protocolo.
- **200 OK** es el código de estado. Indica que el servidor ha recibido, entendido y aceptado la solicitud.
- Cabeceras:
 - **Content-Encoding** y **Content-Type** proporcionan información sobre el tipo de archivo.
 - **Age, Cache-Control, Expires, Vary** y **X-Cache** se refieren al caching del archivo.
 - **Etag** y **Last-Modified** se utilizan para el control del archivo entregado.
 - **Server** se refiere al software del servidor web.
 - **Content-Length** es el tamaño del archivo en bytes.

Cabeceras HTTP

- Las **cabeceras** (headers) **HTTP** permiten al cliente y al servidor enviar **información adicional** junto a una **petición o respuesta**.
- Una cabecera está compuesta por su **nombre** (no sensible a las mayúsculas) seguido de ':' y a continuación de su valor, sin saltos de línea. Los espacios en blanco a su izquierda son ignorados.
- Se pueden agregar cabeceras propietarias personalizadas usando el prefijo “**X-**”, pero **esta convención se encuentra desfasada** desde julio de 2012, debido a los inconvenientes causados cuando se estandarizaron campos no estándar en el **RFC 6648** “*Deprecating the “X-“ Prefix and Similar Constructs in Application Protocols*”.
- Otras cabeceras están **listadas en un registro del IANA** (Internet Assigned Numbers Authority)
 - <https://www.iana.org/assignments/message-headers/message-headers.xhtml>
- **IANA** también mantiene un **registro de propuestas para nuevas cabeceras HTTP**.

Cabeceras HTTP

- Las cabeceras pueden ser agrupadas de acuerdo a sus contextos:
 - **Cabecera general:** Cabeceras que se aplican tanto a las **peticiones como a las respuestas**, pero **sin relación con los datos** que finalmente se transmiten en el cuerpo.
 - **Cabecera de consulta:** Cabeceras que contienen **más información sobre el contenido** que va a obtenerse o sobre el cliente.
 - **Cabecera de respuesta:** Cabeceras que contienen **más información sobre el contenido**, como su origen o el servidor (nombre, versión, etc.)
 - **Cabecera de entidad:** Cabeceras que contienen más información sobre el **cuerpo de la entidad**, como el tamaño del contenido o su tipo MIME.

Cabeceras HTTP

- Las cabeceras también pueden clasificarse de acuerdo a **cómo se comportan frente a ellas los proxys**:
 - **Cabecera de extremo a extremo**: Cabeceras que deben ser enviadas al recipiente final del mensaje (servidor para una petición, cliente para una respuesta) sin modificar, y **las cachés deben guardarlas tal y como son recibidas**.
 - **Cabecera de paso**: Cabeceras que **sólo son significativas para conexiones de paso y no deben de ser retransmitidas por proxys o almacenarse en caché**: Connection, Keep-Alive, Proxy-Authenticate, Proxy-Authorization, TE, Trailer, Transfer-Encoding y Upgrade.
 - La cabecera general **Connection** solo puede usarse para este tipo de cabeceras.

Cabeceras comunes de solicitud (1)

Campo de cabecera	Significado	Ejemplo
Accept	Qué tipos de contenido puede procesar el cliente; si el campo está vacío, esos son todos los tipos de contenido	Accept: text/html, application/xml
Accept-Charset	Qué caracteres puede mostrar el cliente	Accept-Charset: utf-8
Accept-Encoding	Qué formatos comprimidos soporta el cliente	Accept-Encoding: gzip
Accept-Language	Preferencia de idioma	Accept-Language: de-DE
Authorization	Datos de autenticación (por ejemplo, para un inicio de sesión)	Basic WjbU7D25zTAIV2tZ7==
Cache-Control	Opciones de mecanismo de caching	Cache-Control: no-cache
Cookie	Cookies almacenadas para este servidor	Cookie: \$Version=1; Content=23
Content-Length	Longitud del cuerpo de la solicitud	Content-Length: 212
Content-Type	Tipo MIME del cuerpo; pertinente para las solicitudes POST v PUT	Content-Type: application/x_222-form-urlencoded
Date	Fecha y hora de la solicitud	Date: Mon, 9 March 2020 09:02:22 GMT
Expect	Formula una expectativa al servidor, generalmente la recepción de una solicitud grande	Expect: 100-continue (El servidor debe enviar el código 100 cuando esté preparado para recibir una solicitud)
Host	Nombre de dominio del servidor	Host: ejemplo.es

Cabeceras comunes de solicitud (2)

Campo de cabecera	Significado	Ejemplo
If-Match	Permite la ejecución condicional de una acción, dependiendo de la coincidencia de un código transmitido	If-Match: „ft678iujhnjio90'pöl“
If-Modified-Since	Permite enviar contenido solo si éste ha sido modificado desde el momento especificado	IF-Modified-Since: Mon 2 Mar 2020 1:00:00 GMT
If-None-Match	Como arriba, pero especificado a través de un ETag (etiqueta de entidad, ver abajo)	If-None-Match: „cxdrt5678iujhgbvb“
If-Range	Solicita solo la parte del contenido que fue cambiada o que falta en el cache del cliente	If-Range: Mon 2 Mar 2020 1:00:00 GMT
If-Unmodified-Since	IF-Modified-Since análogo	If-Modified-Since: Mon 2 Mar 2020 1:00:00 GMT
Max-Forwards	Define el número máximo de veces que la respuesta del servidor puede ser reenviada	Max-Forwards: 12
Proxy-Authorization	Se utiliza para autenticar al cliente en un servidor proxy	Proxy-Authorization: Basic WjbU7D25zTAIV2tZ7==
Range	Especifica una parte del contenido solicitado	Range: bytes=0-9999
Referrer	URL del recurso del que procede la solicitud (es decir, del que se hizo el enlace)	Referrer: https://ejemplo.es/index.html
TE	Codificación de transferencia aceptada	TE: gzip, deflate
User-Agent	User-Agent del cliente (es decir, el navegador)	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36

Cabeceras comunes de respuesta (1)

Campo de cabecera	Significado	Ejemplo
Date	Tiempo de la respuesta	Date: Mon 2 Mar 2020 1:00:00 GMT
ETag	Marca una versión específica del archivo	ETag: „vt6789oi8uztgvbn“
Expires	Cuándo el archivo debe ser considerado obsoleto	Expires: Tue 3 Mar 2020 1:00:00 GMT
Last-Modified	Hora de la última modificación del archivo	Last-Modified: Mon 2 Mar 2020 1:00:00 GMT
Location	Identifica el lugar al que se envió la solicitud	Location: https://www.ejemplo.es
Proxy-Authenticate	Dice si el cliente debe autenticarse al Proxy y cómo debe hacerlo	Proxy-Authenticate: Basic
Retry-After	A partir de cuándo el cliente debe reenviar la solicitud si el recurso no está disponible temporalmente (fecha o segundos)	Retry-After: 300
Server	Identificación del servidor	Server: Apache
Set-Cookie	Establece una cookie en el cliente	Set-Cookie: UserID=XY; Max-Age=3800; Version=1
Transfer-Encoding	Método de codificación	Transfer-Encoding: gzip
Vary	Indica qué campos de cabecera deben considerarse variables si se solicita un archivo del cache.	Vary: User-Agent (=el servidor tiene preparadas diferentes versiones de archivos dependiendo del User Agent)

Cabeceras comunes de respuesta (2)

Campo de cabecera	Significado	Ejemplo
Date	Tiempo de la respuesta	Date: Mon 2 Mar 2020 1:00:00 GMT
ETag	Marca una versión específica del archivo	ETag: „vt6789oi8uztgvbn“
Expires	Cuándo el archivo debe ser considerado obsoleto	Expires: Tue 3 Mar 2020 1:00:00 GMT
Last-Modified	Hora de la última modificación del archivo	Last-Modified: Mon 2 Mar 2020 1:00:00 GMT
Location	Identifica el lugar al que se envió la solicitud	Location: https://www.ejemplo.es
Proxy-Authenticate	Dice si el cliente debe autenticarse al Proxy y cómo debe hacerlo	Proxy-Authenticate: Basic
Retry-After	A partir de cuándo el cliente debe reenviar la solicitud si el recurso no está disponible temporalmente (fecha o segundos)	Retry-After: 300
Server	Identificación del servidor	Server: Apache
Set-Cookie	Establece una cookie en el cliente	Set-Cookie: UserID=XY; Max-Age=3800; Version=1
Transfer-Encoding	Método de codificación	Transfer-Encoding: gzip
Vary	Indica qué campos de cabecera deben considerarse variables si se solicita un archivo del cache.	Vary: User-Agent (=el servidor tiene preparadas diferentes versiones de archivos dependiendo del User Agent)

Chrome Header Analyzer

<https://chrome.google.com/webstore/detail/http-header-analyzer/paomhfdiccilfelmhlcdbahjpiiddgne>

The screenshot shows a browser window with the URL `pixfans.com` in the address bar. A sidebar on the right displays the results of the Chrome Header Analyzer extension. The sidebar has two main sections: "Request Headers" and "Response Headers".

Request Headers:

- GET / HTTP/1.1
- Upgrade-Insecure-Requests:** 1
- User-Agent:** Mozilla/5.0 (Macintosh; Intel Mac OS X 11_0_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
- Accept:** text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
- sec-ch-ua:** "Google Chrome";v="87", "Not;A Brand";v="99", "Chromium";v="87"
- sec-ch-ua-mobile:** ?0

Response Headers:

- HTTP/1.1 200 OK
- Date:** Tue, 05 Jan 2021 21:52:42 GMT
- Server:** Apache/2.4.25 (Debian)
- Vary:** Accept-Encoding
- Content-Encoding:** gzip
- Keep-Alive:** timeout=5, max=100
- Connection:** Keep-Alive
- Transfer-Encoding:** chunked
- Content-Type:** text/html; charset=UTF-8

At the bottom of the sidebar, there is a link labeled "CREDITS".

Listados de cabeceras HTTP

- **Mozilla**
 - <https://developer.mozilla.org/es/docs/Web/HTTP/Headers>
- **Wikipedia**
 - https://en.wikipedia.org/wiki/List_of_HTTP_header_fields
- **IANA**
 - <https://www.iana.org/assignments/message-headers/message-headers.xhtml>

HTTP: CACHÉ

Caché

- Mucha gente visita páginas que **ya ha visto antes**.
- Las páginas tienen **recursos** embebidos que quizá sean los **mismos entre páginas**: logo del título, hojas de estilo, foto en el pie de página, iconos de redes sociales...
- No tiene sentido volver a descargar ese contenido:
 - Se desperdiciarían recursos.
 - La navegación sería más lenta.
- Por eso el protocolo HTTP tiene un **mecanismo de caché** para ayudar a los clientes a identificar qué recursos se han descargado.
- Se utilizan dos mecanismos para solucionar este problema:
 - **1. Validación de página**
 - **2. GET condicional**

Caché: 1. Validación de página

- Se consulta la **caché** y si tiene una página que está fresca, no es necesario solicitarla de nuevo del servidor.
- Se utiliza la cabecera **Expires**, incluida con la petición original y la fecha y hora actual para hacer esa verificación.
- Pero **no todas las páginas vienen con una cabecera Expires** que diga cuándo tienen que ser solicitadas de nuevo.
- Hacer estas **predicciones es difícil**. El navegador puede emplear heurística: Si una página no se ha modificado en un año (cabecera **Last-Modified**), probablemente no cambie en la siguiente hora. Pero **nada lo garantiza**.
- Las **heurísticas** en general funcionan bien, pero hay que proceder con cautela.

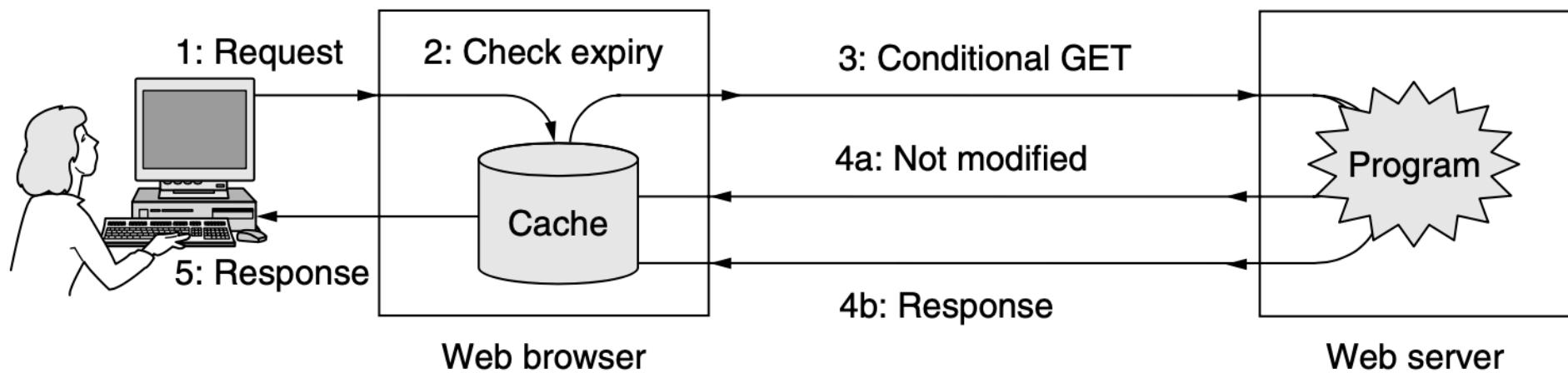
Caché: 2. Conditional GET

- El navegador le pregunta al servidor **si su copia local sigue siendo válida** y el servidor le envía una respuesta corta para confirmárselo. Si no, le envía **la respuesta completa**.
- **Se necesitan más cabeceras** para chequear si la copia sigue siendo válida.
 - **If-Modified-Since** se usa para preguntar **si la página ha cambiado desde entonces**.
 - También el servidor puede enviar una cabecera **ETag** que sea un hash de la página.
- El cliente puede validar sus copias usando **If-None-Match** con la lista de tags. El servidor responderá con la lista de tags de copias que pueden ser usadas.
 - Este método es usado cuando **no es conveniente o útil determinar la frescura de las páginas**.
 - Hay que tener en cuenta que un servidor puede **responder con diferente contenido para una misma URL** dependiendo de qué lenguajes o de qué tipo MIME son los preferidos por el navegador.

Caché: Cabecera Cache-Control

- Las dos estrategias pueden ser restringidas por la cabecera **Cache-Control**
 - Por ejemplo, no permitir caché: “**no-cache**”.
 - Se podría utilizar en una **página dinámica que será distinta la próxima vez que sea solicitada**.
 - Las páginas que **requieren autorización** tampoco son cacheadas.

HTTP Caching



Caché: Otros tipos

- Hay varios **niveles de caché** tanto por parte del cliente como por parte del servidor para **reducir el tráfico y servir los recursos cuanto antes**:
 - Los proxies pueden cachear: **proxy-caching**.
 - Los **ISP** usan proxy caching para reducir tráfico.
- Los recursos cacheados pueden ser costosos de almacenar si son grandes, como los vídeos.
- Los servidores web pueden **cachear una página dinámica** creando una **versión estática** de la misma mientras siga siendo válida. Como hace **Wordpress**, con el plugin **WP-Cache**, reduciendo el uso de CPU y mejorando el tiempo de respuesta.
 - Por ejemplo, en un blog, hasta que se publique una noticia nueva o se añada un comentario a la noticia, la copia estática seguirá siendo válida. No hace falta generarla de nuevo haciendo consultas a bases de datos.

HTTP: REDIRECCIONES

Redirecciones

- **Redirección URL** (URL redirection, URL forwarding) es una técnica que nos permite dar más de una URL a una página, un formulario o a un sitio o aplicación web.
- Esta técnica se conoce como redirección HTTP (**HTTP redirect**)
 - **Redirecciones temporales**: Consiste en redireccionar temporalmente las peticiones si el sitio está caído o en labores de mantenimiento.
 - **Redirecciones permanentes**: Consiste en preservar los actuales enlaces/bookmarks después de un cambio en las URL de las páginas. Crear páginas de progreso al acabar de subir un fichero, etc.
- Las redirecciones son lanzadas por el servidor enviando una respuesta de redirección (código de estado que empieza con un **3**) y una cabecera **Location** con la URL de la dirección a la que se redirige al usuario.
- Cuando los **navegadores** reciben una redirección, inmediatamente **cargan la nueva URL** recibida en la cabecera Location. Tiene un **impacto en el rendimiento** que rara vez es notada por los usuarios.

Redirecciones



Redirecciones permanentes

- Son redirecciones que **durarán para siempre**.
- La **URL original debe dejar de ser usada** y ser reemplazada por la nueva.
- Los crawlers, robots, lectores de RSS y otras aplicaciones deben actualizar la URL para acceder al recurso.

Code	Text	Method handling	Typical use case
301	Moved Permanently	Los métodos GET no se cambian. Otros métodos pueden cambiar. [1]	Reorganización de un sitio/app web.
308	Permanent Redirect	Method and body not changed.	Reorganización de un sitio/app web con operaciones que no son GET.

[1] La especificación nunca tuvo intención de permitir cambios de método, pero existen agentes de usuario que cambian su método. El código 308 se creó para eliminar la ambigüedad del comportamiento cuando se usan métodos que no son GET.

<https://stackoverflow.com/questions/42136829/whats-the-difference-between-http-301-and-308-status-codes>

Redirecciones temporales

- En ocasiones, el recurso solicitado no puede ser accedido en su localización, pero sí puede ser accedido en otro lugar.
- Los crawlers y los robots de buscadores **no actualizan la nueva URL temporal**.
- Las URLs temporales también se utilizan cuando se está actualizando, creando o borrando recursos de las páginas.

Code	Text	Method handling	Typical use case
302	Found	<code>GET</code> methods unchanged. Others may or may not be changed to <code>GET</code> . ^[2]	The Web page is temporarily unavailable for unforeseen reasons.
303	See Other	<code>GET</code> methods unchanged. Others <i>changed</i> to GET (body lost).	Used to redirect after a <code>PUT</code> or a <code>POST</code> , so that refreshing the result page doesn't re-trigger the operation.
307	Temporary Redirect	Method and body not changed	The Web page is temporarily unavailable for unforeseen reasons. Better than 302 when non-GET operations are available on the site.

Redirecciones especiales

- **304 (Not Modified)**: redirige a la caché local, porque la página no ha cambiado desde la última visita.
- **300 (Multiple Choice)**: Es una redirección manual. El cuerpo de la respuesta lo presenta el navegador como una página web con una lista de enlaces y **el usuario debe pinchar en uno de ellos**.

Code	Text	Typical use case
300	Multiple Choice	Not many: the choices are listed in an HTML page in the body. Machine-readable choices are encouraged to be sent as Link headers with <code>rel=alternate</code> .
304	Not Modified	Sent for revalidated conditional requests. Indicates that the cached response is still fresh and can be used.

Formas alternativas de redireccionamiento

- Las redirecciones HTTP no son la única forma de definir redireccionamientos:
 - **Redirecciones HTML:** Utilizando el elemento <meta>
 - **Redirecciones Javascript:** Utilizando el DOM.
- **Orden de precedencia:**
 - 1. Las redirecciones por código HTTP se ejecutan primero.
 - 2. Se ejecuta la redirección <meta> de HTML si no hubo redirecciones HTTP.
 - 3. Las redirecciones JavaScript se ejecutan al final, si el usuario tiene el JavaScript de su navegador activado.

Redirecciones HTML

- Las redirecciones HTTP son la mejor manera de crear redirecciones, pero en ocasiones, uno no tiene control sobre la configuración del servidor.
- Para estos casos:

```
<head>
  <meta http-equiv="Refresh" content="0; URL=https://example.com/">
</head>
```

- El atributo content define el **número de segundos que el navegador debe de esperar antes de hacer la redirección**. Lo normal es poner 0 salvo que se quiera mostrar algún tipo de mensaje.
- Obviamente este tipo de redirecciones funciona **solo cuando se solicitan archivos HTML** y no puede ser usado con imágenes o con cualquier otro tipo de contenido.

Redirecciones Javascript

- Las redirecciones JavaScript se ejecutan asignando la **URL** a la propiedad **window.location**.
- Al igual que las redirecciones HTML, no funcionan en todos los recursos y además, **el cliente debe tener activado JavaScript**.
- Por otra parte, este tipo de redirecciones ofrece mucha más flexibilidad, puesto que se puede **redirigir al usuario en base a ciertas condiciones**, o a una URL o a otra...

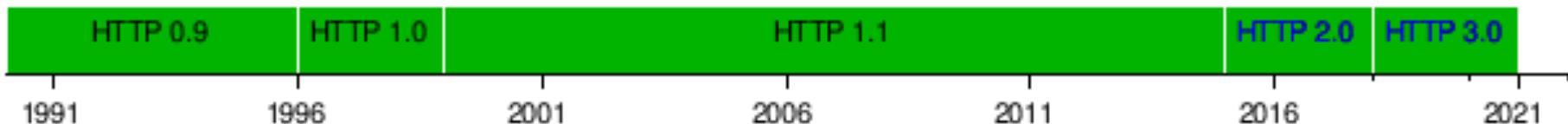
Redirecciones: Loops

- Los loops de redirección ocurren cuando las redirecciones adicionales siguen a la que ya se ha seguido.
- La mayoría de las veces se trata de un **problema del servidor**.
- Los navegadores detectarán los loops y mostrarán un mensaje de error al usuario.

HTTP: VERSIONES Y EVOLUCIÓN

Evolución del protocolo HTTP

- Inventado por **TimBL** en el CERN en los años 1989-1991, **HTTP** es el protocolo de la **WWW**.
- Los estándares son desarrollados por la **IETF** (Internet Engineering Task Force) y el **W3C** (World Wide Web Consortium).
- Se culminan con publicaciones de **RFC (Requests for Comments)**.
 - HTTP/0.9
 - HTTP/1.0
 - HTTP/1.1
 - HTTP/2.0
 - HTTP/3.0



HTTP/0.9 - El protocolo de una sola línea

- La versión inicial de HTTP **no tenía número de versión**, aunque posteriormente se la denominó **0.9** para distinguirla de las siguientes.
- Una petición consiste simplemente en una única línea, que comienza por el único método posible **GET**, seguido por la **dirección del recurso** a pedir (no la URL, puesto que el protocolo, servidor y puerto no son necesarios una vez se ha conectado con el servidor).
- **No usa cabeceras HTTP**, con lo cual solo es posible transmitir archivos **HTML**.
- No había **información del estado ni códigos de error**.
- En caso de un **problema**, el archivo **HTML** pedido era devuelto con la descripción del problema dentro de él.
- Esta versión es claramente **limitada**, y tanto navegadores como servidores pronto ampliaron el protocolo para hacerlo más flexible.

PETICIÓN:

```
GET /miPaginaWeb.html
```

RESPUESTA:

```
<HTML>
Una pagina web muy sencilla
</HTML>
```

HTTP/0.9 - El protocolo de una sola línea

```
$> telnet ashenlive.com 80
```

(Connection 1 Establishment – TCP Three-Way Handshake)
Connected to xxx.xxx.xxx.xxx

(Request)
GET /my-page.html

(Response in hypertext)
<HTML>
A very simple HTML page
</HTML>

(Connection 1 Closed – TCP Teardown)

HTTP/1.0 - Desarrollando extensibilidad

- La **versión** del protocolo se envía con cada petición método.
- Protocolo **amigable con los navegadores**: Se envía un **código de estado** al comienzo de la respuesta, permitiendo 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).
- Aparecen las **cabeceras HTTP** para las peticiones como para las respuestas, permitiendo la transmisión de **meta-data** y conformando un protocolo versátil y ampliable.
- Con el uso de cabeceras HTTP **se pueden transmitir otros documentos** mediante la cabecera **Content-Type** (scripts, hojas de estilo, archivos multimedia).
- Soporta: **GET, HEAD y POST**.
- Las **conexiones se terminan inmediatamente después de la respuesta**.
- Las innovaciones no se realizaron de forma planeada (fue **prueba y error**). Entre los años 1991 y 1995. Fueron comunes los **problemas de interoperatividad**. En Noviembre de 1996 se publicó un documento que describía las prácticas adecuadas RFC 1945, que define el protocolo **HTTP/1.0** aunque no es un estándar oficial.

<https://tools.ietf.org/html/rfc1945>

HTTP/1.0 - Desarrollando extensibilidad

PETICIÓN

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

RESPUESTA

```
200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
<HTML>
Una pagina web con una imagen
<IMG SRC="/miImagen.gif">
</HTML>
```

PETICIÓN

```
GET /myImagen.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

RESPUESTA

```
200 OK
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif
(image content)
```

HTTP/1.0 - Desarrollando extensibilidad

(Connection 1 Establishment – TCP Three-Way Handshake)
Connected to xxx.xxx.xxx.xxx

(Request)
GET /my-page.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

(Response)
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 137582
Expires: Thu, 01 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 1 May 1996 12:45:26 GMT
Server: Apache 0.84

<HTML>
A page with an image

</HTML>

(Connection 1 Closed – TCP Teardown)

(Connection 2 Establishment – TCP Three-Way Handshake)
Connected to xxx.xxx.xxx.xxx

(Request)
GET /myimage.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

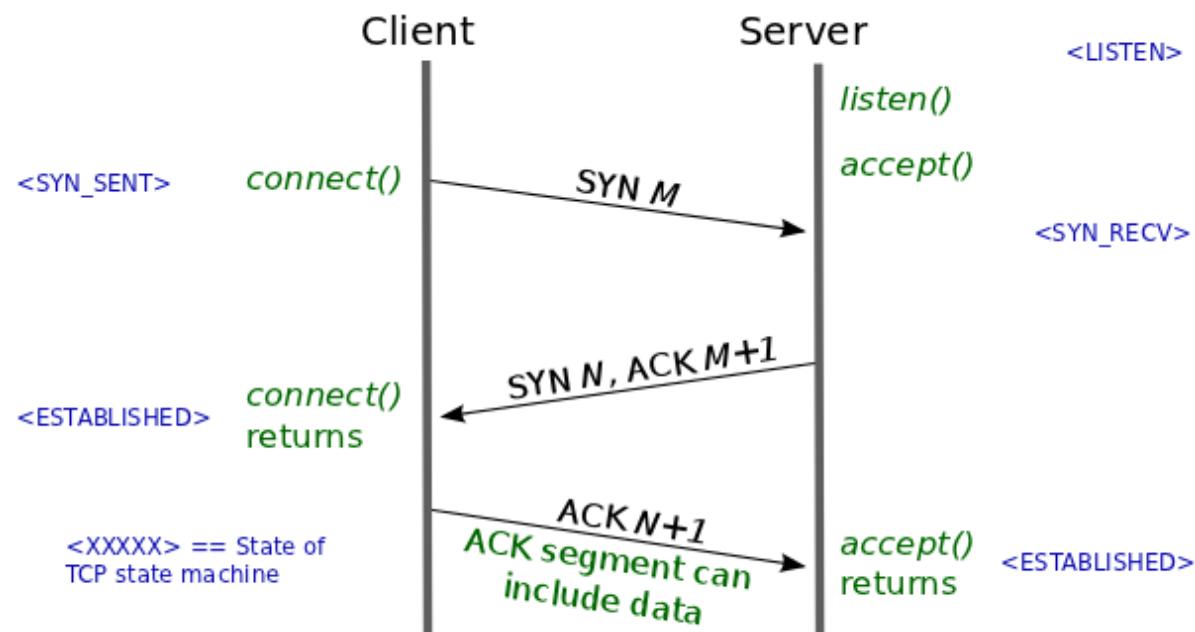
(Response)
HTTP/1.0 200 OK
Content-Type: text/gif
Content-Length: 137582
Expires: Thu, 01 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 1 May 1996 12:45:26 GMT
Server: Apache 0.84

[image content]

(Connection 2 Closed – TCP Teardown)

HTTP/0.9 y HTTP/1.0

- **PROBLEMA:** Crean una nueva conexión por cada petición y la cierran inmediatamente tras obtener la respuesta.
 - Cada vez que se establece una conexión TCP, el **handshake** (apretón de manos) de tres vueltas ocurre.
 - Es crucial reducir esas tres vueltas entre cliente y servidor, y **HTTP/1.1** solucionará el problema con **conexiones persistentes**.



Típico handshake de TCP

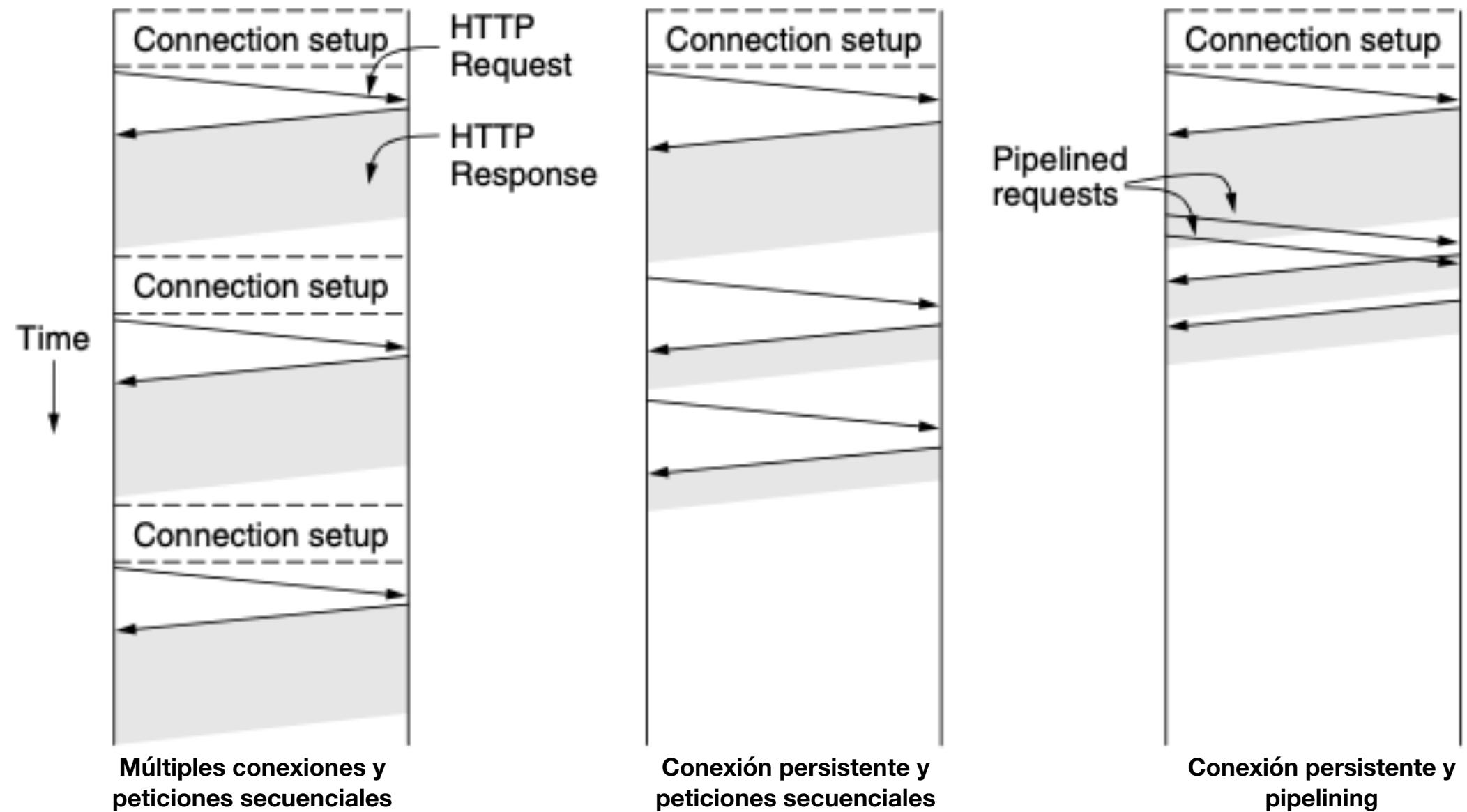
HTTP/1.1 - El protocolo estándar

- Un año antes de la versión 1.0, una estandarización formal ya estaba en curso. El protocolo **HTTP/1.1 se publicó tan solo unos meses después del HTTP/1.0,**
 - Una **conexión TCP** podía ser **reutilizada**, ahorrando el tiempo de re-abrirla repetidas veces.
 - Se añadió enrutamiento (**Pipelining**), permitiendo realizar **una segunda petición de datos por la misma conexión antes de que fuera respondida la primera**, disminuyendo la **latencia** de la comunicación.
 - Se permitió que las **respuestas** a peticiones fueran **subdivididas en partes**.
 - Se añadieron controles adicionales a los mecanismos de **gestión de la caché**.
 - La **negociación del 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** pudo ser posible alojar **varios dominios en la misma dirección IP**.
 - HTTP/1.1 fue publicado originalmente como RFC en Enero de 1997. (**RFC2068**).

<https://tools.ietf.org/html/rfc2068>

HTTP/1.1 - Conexiones Persistentes

- Con HTTP/1.0, tras establecer una conexión TCP con el servidor en el puerto 80, se mandaba una petición y se recibía una respuesta.
- La **web** creció y se volvió **más compleja** y las **páginas** estaban compuestas por **más objetos**.
- **HTTP/1.1** soporta **conexiones persistentes**, que permiten enviar una petición y recibir una respuesta y después **mandar peticiones adicionales**. Esta estrategia también se conoce como **reutilización de conexiones**.
- Al amortizar los costos de configuración, inicio y liberación de TCP en múltiples solicitudes, **la sobrecarga relativa debido a TCP se reduce por solicitud**.
- También es posible canalizar peticiones (**pipelining**) que consiste en enviar la petición 2 antes de que la respuesta a la petición 1 haya llegado.



Conexión persistente y pipelining

- Cada conexión TCP requiere al menos **un tiempo de ida y vuelta** para establecerse.
- La transferencia de las mismas imágenes ocurre más rápido. ¿Por qué? Por el **control de congestión de TCP**.
 - TCP utiliza el **procedimiento de inicio lento** slow-start para aumentar el rendimiento hasta que **aprende el comportamiento de la ruta de la red**.
 - Por este motivo **varias conexiones TCP cortas tardan desproporcionadamente más** en transferir información que una conexión TCP más larga.
- La segunda y la tercera petición de la gráfica son en **pipeline**, y son lanzadas tan pronto tras leer la página se han identificado las imágenes que deben de ser solicitadas.

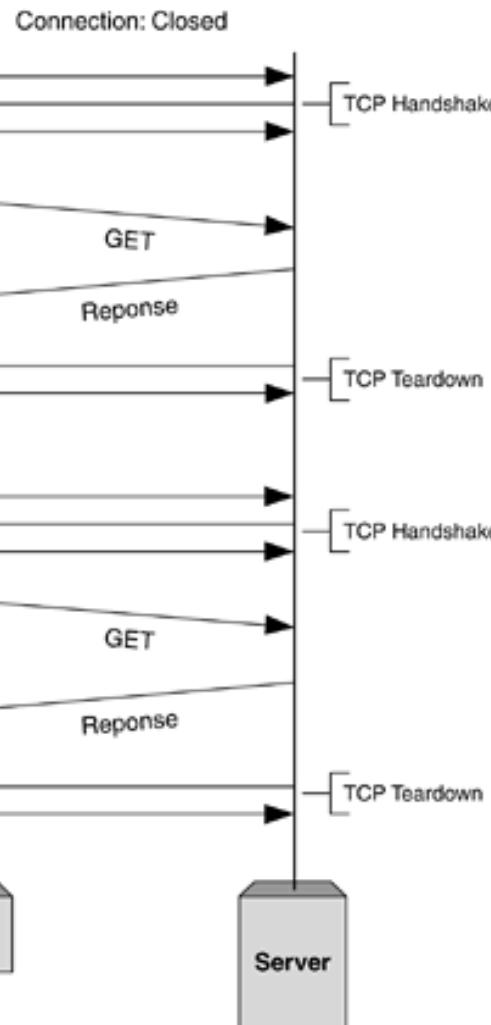
Conexión persistente ¿hasta cuándo?

- La conexión debe de seguir **abierta mientras el navegador está cargando la página**.
- Más tarde el usuario puede pinchar en un enlace que dé lugar a otra petición.
 - Si la conexión sigue abierta, se **reutilizaría**.
 - Si no, habría que establecerla de nuevo, con el **coste de establecimiento TCP** que supone.
- En general las conexiones son persistentes por un periodo corto de tiempo. Quizá de **60 segundos**, pero podría variar dependiendo de la congestión del servidor y de las conexiones que éste tenga abiertas.

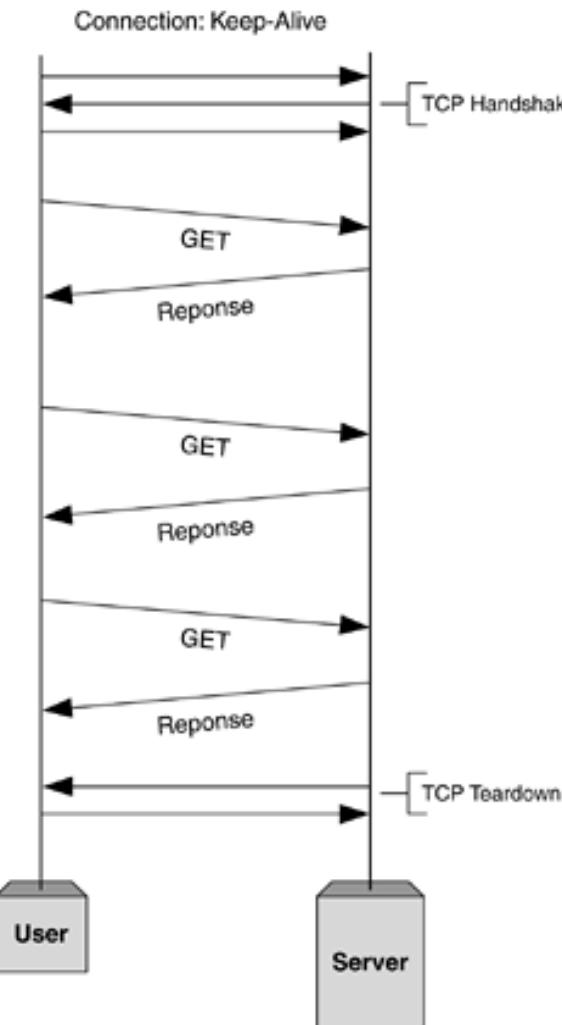
¿Por qué no mandar conexiones paralelas TCP?

- Es posible mandar **cada petición** en una **conexión TCP independiente**.
- Las conexiones se mandarían en **paralelo**.
- Este método de **conexiones paralelas TCP** era utilizado por los navegadores antes de las conexiones persistentes. Pero presentaba desventajas:
 - TCP realiza un **control de congestión independiente** por cada conexión.
 - **Las conexiones compiten entre sí**, lo que provoca **pérdidas adicionales de paquetes**.
 - Las conexiones independientes son más agresivas y provocan **sobrecargas adicionales**.
 - Las **conexiones persistentes son mejores** y se utilizan con preferencia a las conexiones paralelas porque **evitan la sobrecarga y no sufren problemas de congestión**.

HTTP/1.0



HTTP/1.1



HTTP/1.1 - El protocolo estándar

- **TCP Handshake:** Necesario para establecer una conexión TCP.
- **TCP Teardown:** Se cierra la conexión.
- La cabecera **Keep-Alive** se usaba antes de HTTP/1.1 pero **HTTP/1.1** hace que las **conexiones permanentes** sean el **comportamiento habitual**.

HTTP/1.1 - El protocolo estándar

HTTP/1.1 200 OK

Connection: Keep-Alive

Content-Encoding: gzip

Content-Type: text/html; charset=utf-8

Date: Thu, 11 Aug 2016 15:23:13 GMT

Keep-Alive: timeout=5, max=1000

Last-Modified: Mon, 25 Jul 2016 04:32:39 GMT

Server: Apache

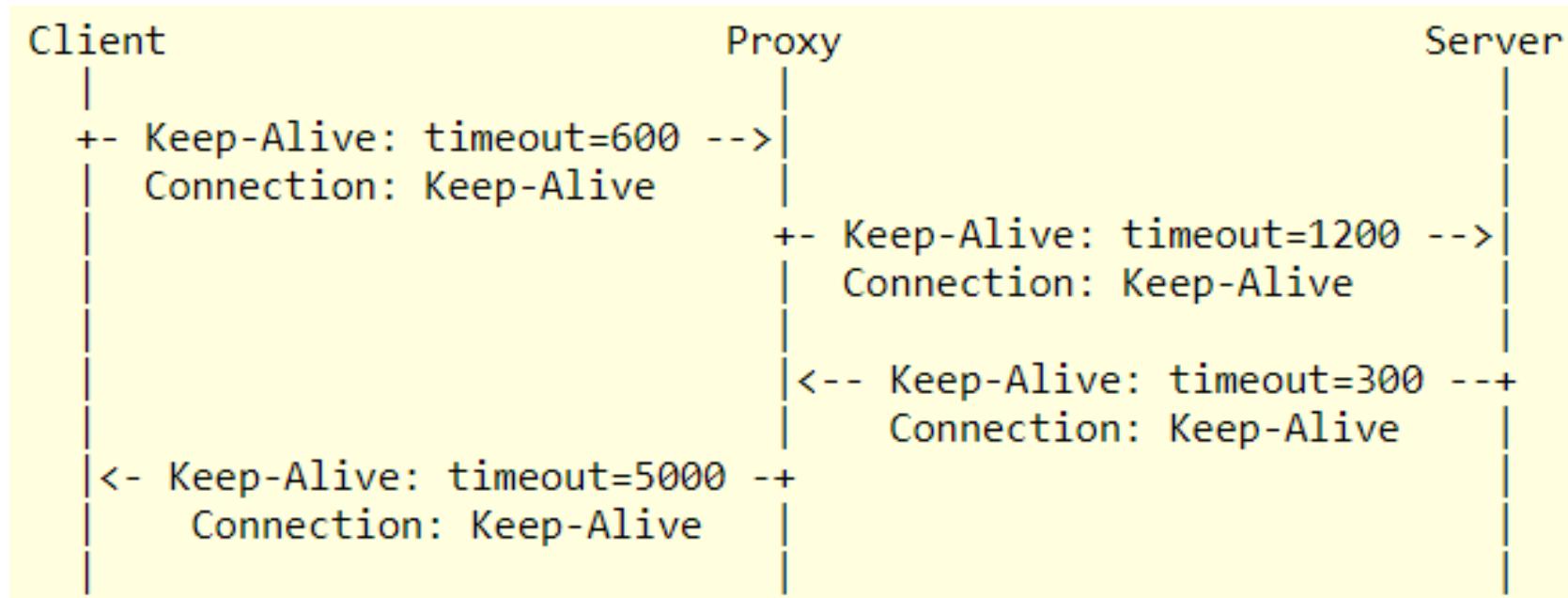
[body]

Hay que tener en cuenta que las webs con el paso del tiempo se volvían más complejas y estaban formadas por cada vez más archivos, por lo que reducir los tiempos de conexión es un crucial.

- El cliente, el servidor o cualquier intermediario pueden proveer información para la cabecera **Keep-Alive** de forma independiente.
- Un host puede añadir un parámetro **timeout** y **max** para fijar **el tiempo de desconexión** o limitar el **número de peticiones** por conexión.

<https://www.informit.com/articles/article.aspx?p=169578>

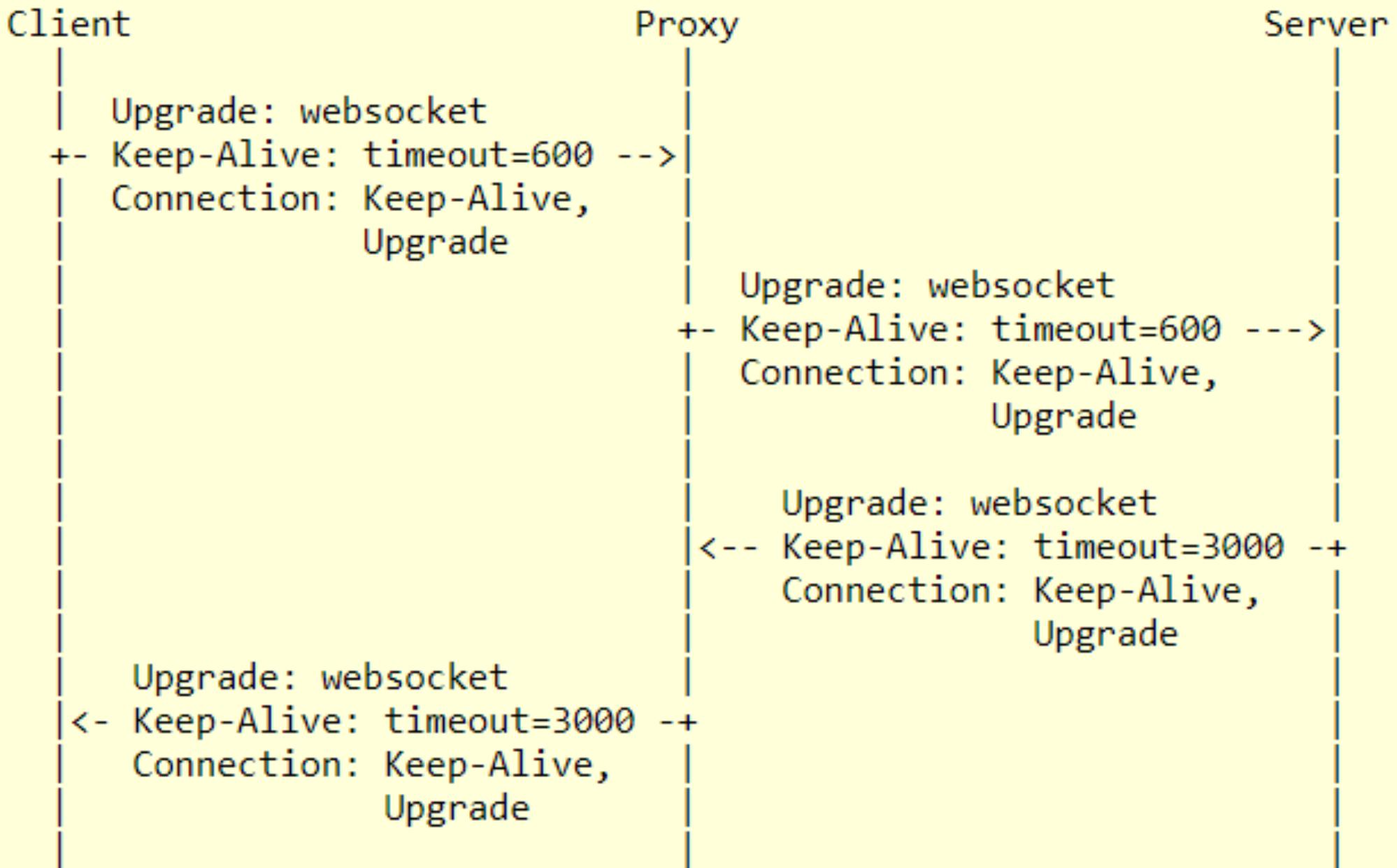
HTTP/1.1 - El protocolo estándar



- En este ejemplo de [IETF.org](#) se nos muestra cómo la cabecera **Keep-Alive** puede ser utilizada.
- Todas las conexiones TCP son **independientemente negociadas**.
- **Cada actor** negocia según **sus parámetros**.

HTTP/1.1 - El protocolo estándar

- Se introdujo la cabecera **Upgrade**, que permite empezar la conexión con un protocolo de uso común, como HTTP/1.1 y **cambiar a un protocolo mejorado**, como **HTTP/2.0 o Websocket** (RFC 6455).
- En los sucesivos protocolos, el parámetro **max** (maximum request count) no está presente. El protocolo superior puede proponer nuevas políticas para el parámetro **timeout**.
- En el ejemplo de **IETF** de la siguiente diapositiva se muestra un **upgrade de HTTP/1.0 a Websocket**.
- **El proxy ajusta el timeout** para reflejar el más bajo de los valores (su timeout y el del cliente) para que el servidor sea consciente de las características de la conexión y hace lo mismo al revés.
- La cabecera **Upgrade** es una **hop-by-hop header**: El proxy la consume, y es posible que se creen conflictos.
 - Más información: <https://nathandavison.com/blog/abusing-http-hop-by-hop-request-headers>



HTTP/1.1 - El protocolo estándar

- Gracias a su extensibilidad, el protocolo fue mejorado en dos revisiones:
 - RFC 2616 Junio 1999
 - RFC 7230 - RFC 7235 Junio 2014
- El protocolo HTTP/1.1 ha sido **muy estable durante ¡más de 15 años!**

<https://tools.ietf.org/html/rfc2616>

<https://tools.ietf.org/html/rfc7230>

<https://tools.ietf.org/html/rfc7235>

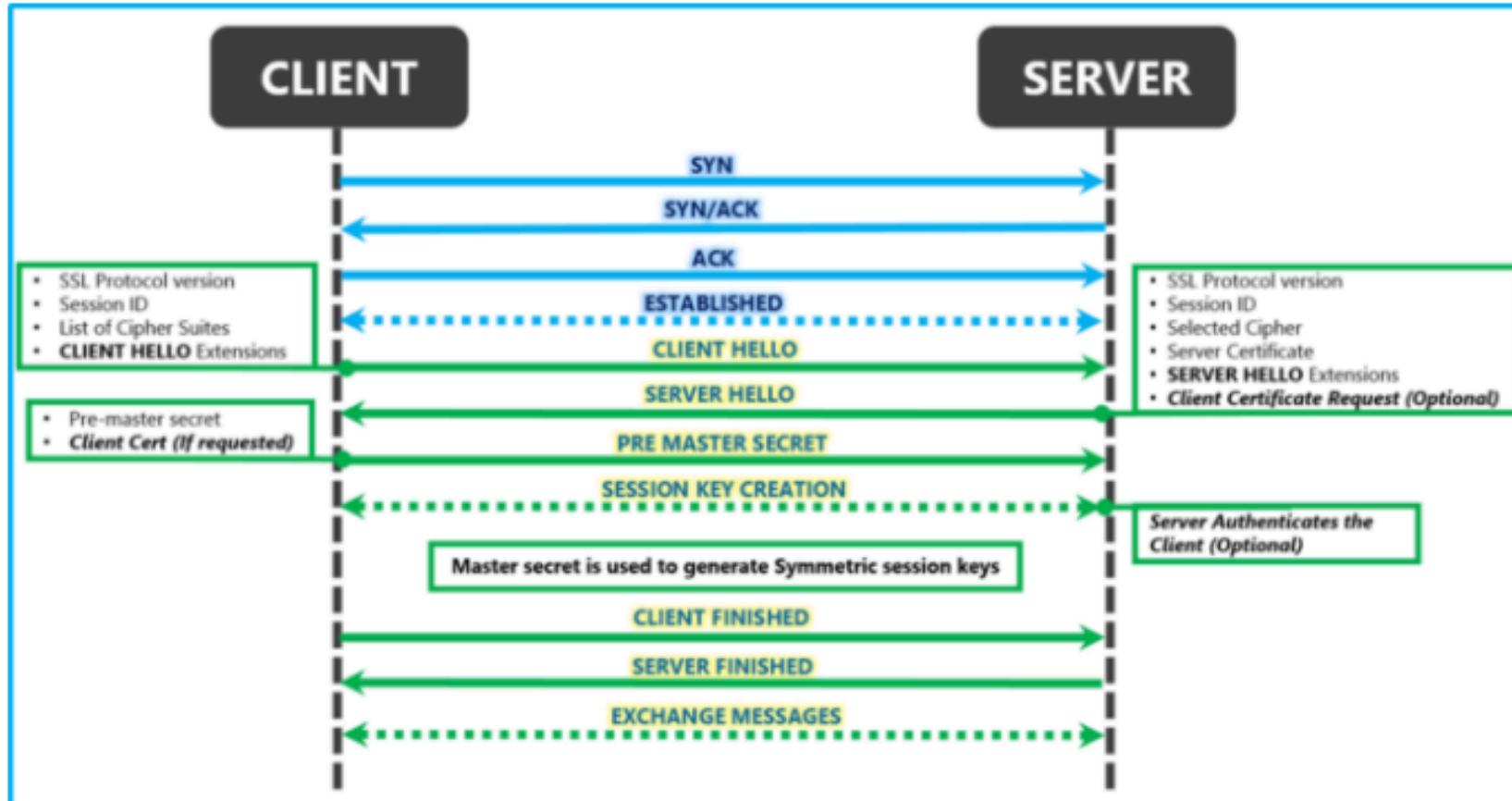
HTTP/1.1 - head of line blocking (HOL)

- Problema de bloqueo de cabeceras en la Capa de Aplicación
 - Es un problema que ocurre cuando **el número de conexiones en el navegador está completo** (típicamente 6 conexiones), y **las siguientes transferencias deben de esperar a que queden huecos libres**.
 - Además hay **peticiones que tardan más que otras**, por ejemplo una web dinámica construida con consultas a una base de datos. Y otros recursos que están listos tienen que esperar.
 - En teoría el **pipelining** ofrece una manera de **evitar el HOL**, pero es complicado de implementar en la práctica, es **propenso a errores y no es soportado por muchos servidores**.
 - Las páginas webs actuales constan de muchos elementos, quizá más de 100, por lo que **navegar por la web se vuelve muy lento**.
 - **HTTP/2** soluciona este problema con la **multiplexación**: varios archivos se envían por un mismo canal. Pero el problema del HOL, como veremos más adelante, **pasará de la capa de aplicación a la capa de transporte**, en el **TCP head of line blocking de HTTP/2**.
 - **QUIC** solucionará el problema del **HOL**.

HTTPS

- Hyper Text Transfer Protocol Secure (**HTTPS**) es la versión segura de HTTP.
- Utiliza el puerto **443** en lugar del puerto **80**.
- Utiliza SSL/TLS para encriptar las comunicaciones.
- **SSL** (Secure Socket Layer)
 - Fue desarrollada originalmente por Netscape a mediados de los 90s.
 - Define cómo el cliente y el servidor deben de comunicarse de forma segura entre ellos.
- **TLS** (Transport Layer Security) es el sucesor de SSL.
- Una conexión **HTTPS puede proteger** los datos de ataques man-in-the-middle y de otros problemas comunes de seguridad utilizando **encriptación bidireccional entre cliente y servidor**.

SSL/TLS Handshake: El mayor problema de HTTPS



Aunque HTTPS es seguro por su diseño, el handshake consume mucho tiempo para establecer una conexión, entre 1-2 segundos.

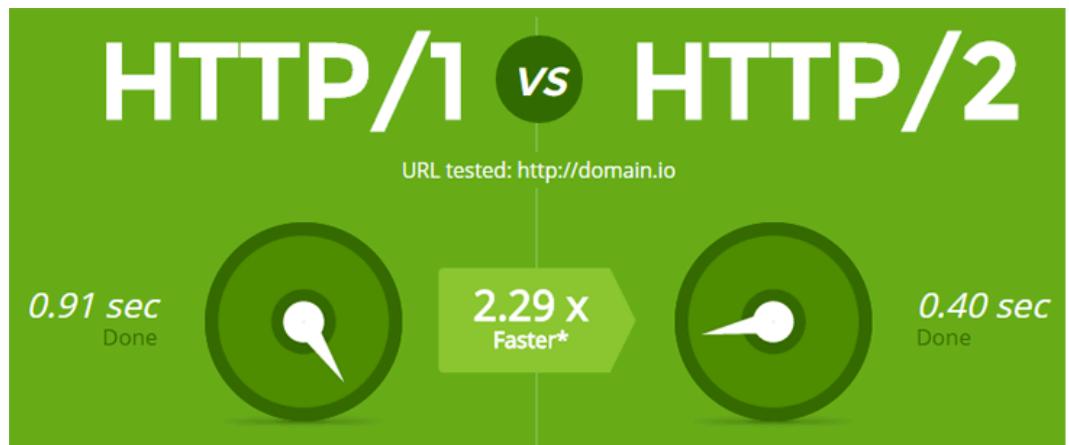
<https://blogs.msdn.microsoft.com/kaushal/2013/08/02/ssl-handshake-and-https-bindings-on-iis/>

HTTP/2 - Un protocolo para mayor rendimiento

- Las páginas web de hoy en día son mucho más complejas. Algunas, de hecho, son aplicaciones.
 - + contenido visual, + scripts, + interactividad, + peticiones...
- En 2010 **Google** demostró un **proceso alternativo para el intercambio de datos** entre clientes y servidores: **SPDY**, definiendo una **mejora en los tiempos de respuesta**.
- **SPDY** sirvió como base para el desarrollo del protocolo **HTTP/2**.
- **HTTP/2 no modifica la semántica de aplicación de HTTP** (todos los conceptos básicos continúan sin cambios). Sus mejoras se enfocan a **cómo se empaquetan los datos y en el transporte**.
- Se añade el **uso de una única conexión**, la **compresión de cabeceras** o el servicio '**server push**'.

HTTP/2 - Un protocolo para mayor rendimiento

- Server Push
- Compresión de Cabeceras
- Formato binario
- Multiplexed Streams



HTTP/2 - Server Push

- Es una técnica que permite al **servidor HTTP/2 enviar recursos al cliente antes de que éste los solicite.**
- Puede ayudar a **cargar las páginas más rápido**, por ejemplo:
- Supongamos que una página web está formada por index.html, style.css e script.js.
 - Antes de HTTP/2 el cliente pediría index.html, lo recibiría y vería que tiene que solicitar también style.css y script.js.
 - Con HTTP/2 **el servidor puede tomar la iniciativa** y enviar el script y la hoja de estilos antes de que sean solicitados, **reduciendo el tiempo de carga.**

HTTP/2 - Compresión de Cabeceras

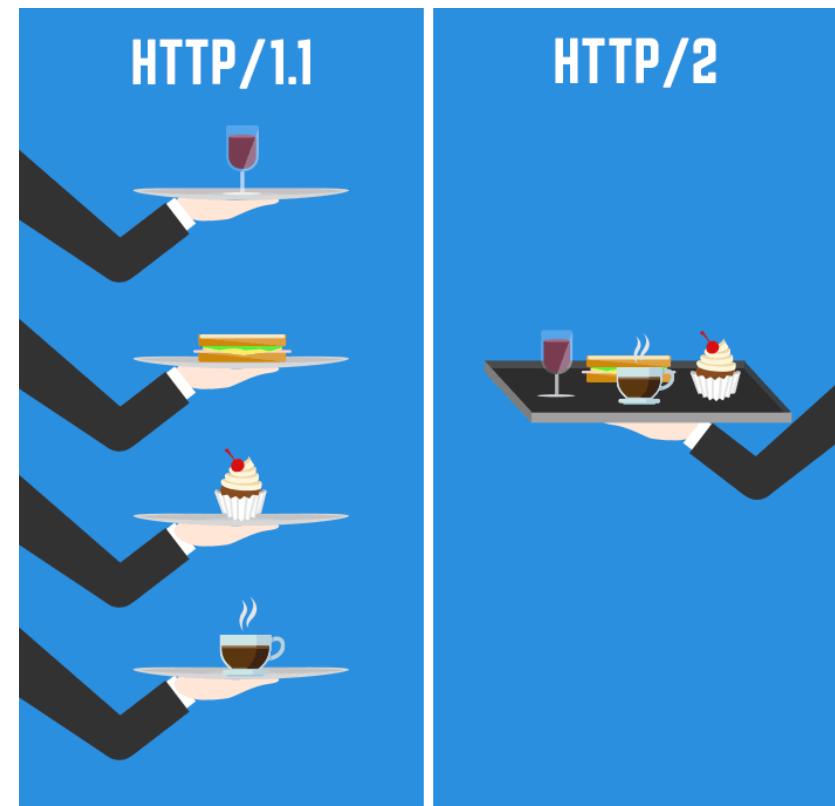
- En HTTP/1.X cada solicitud enviada tiene una pequeña pieza de información adicional que son los encabezados HTTP o HEADERS.
- En **HTTP/2**, cuando se establece una conexión, se **empaquetan todas las cabeceras en un bloque comprimido y son mandadas como una unidad: eliminan duplicación y retardo.**
- Una vez recibidas, se decodifica el bloque de cabeceras del otro lado.

HTTP/2 - Formato binario

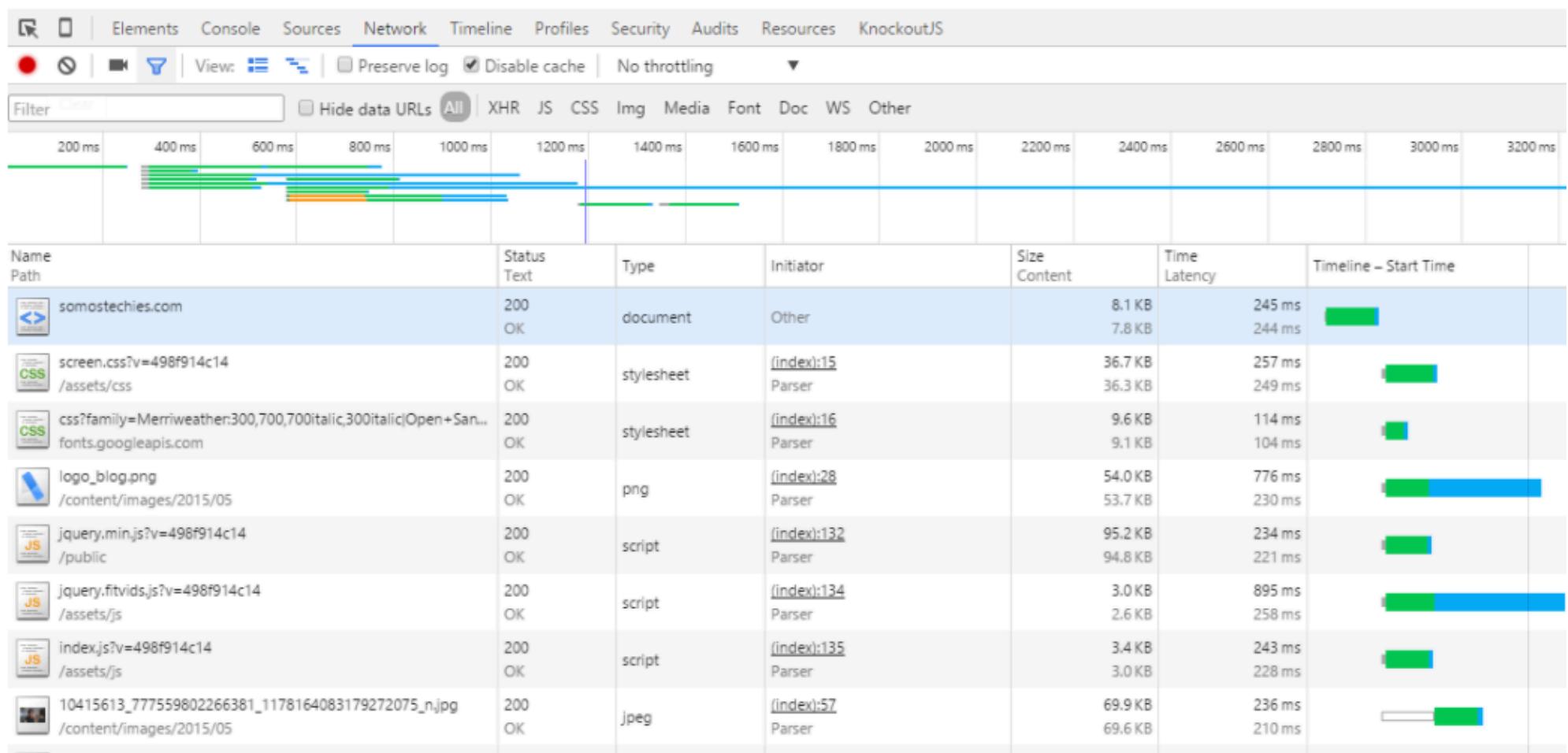
- Los **protocolos binarios** en comparación con los protocolos basados en texto como HTTP/1.X.
 - Son más **eficientes** para interpretar.
 - Más **compactos** al ser transportados.
 - Mucho **menos** propensos a **errores**.
- En los protocolos basados en texto hay que lidiar con problemas como los espacios en blanco, la capitalización, los finales de línea...
- La desventaja es que HTTP ya no es usable a través de **telnet** ni se puede leer la comunicación.

HTTP/2 - Multiplexed Streams

- La **conexión persistente HTTP**, también llamada **HTTP Keep-Alive** o reutilización de la conexión:
 - Es la idea de usar **una sola conexión TCP** para enviar y recibir **múltiples solicitudes y respuestas**. (HTTP/1.1)
- **HTTP/2** usa la misma idea y lo lleva más allá para permitir que múltiples **solicitudes/respuestas concurrentes** se **multiplexen en una sola conexión**.
 - Peticiones pueden realizarse sobre la **misma conexión**.
 - **No está sujeto a mantener el orden en los mensajes ni a otras restricciones** que tenían los protocolos anteriores HTTP/1.x



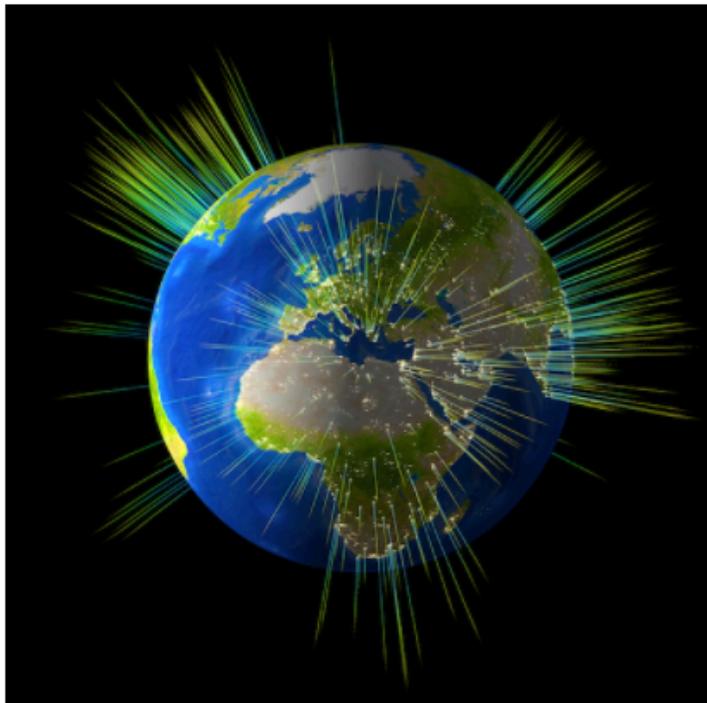
HTTP/2 - Multiplexed Streams



HTTP/2 vs HTTP/1.1

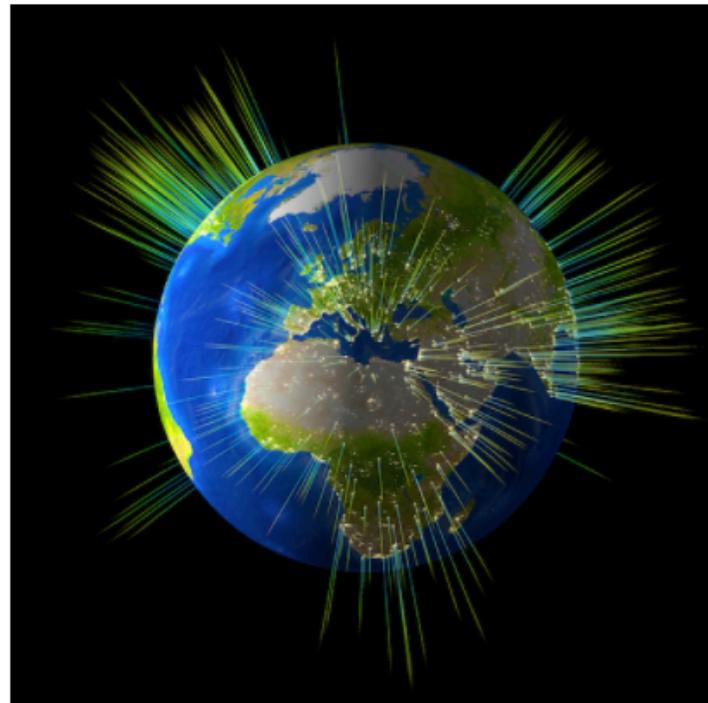
HTTP/1.1

Latency: ms
Load time: 4.27s



HTTP/2

Latency: ms
Load time: 0.93s



<https://http2.akamai.com/demo>

HTTP/2 - Estándar

- Estandarizado oficialmente en mayo del **2015**.
- En julio de 2016 un 8.7% de todos los sitios webs lo estaban usando ya, representando el 68% de su tráfico.
- Los sitios con más tráfico lo adoptaron rápidamente, **ahorrando en sobrecargas y en costes**.
- La adopción era esperada, el uso de HTTP/2 **no requiere 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.
- Según se actualizan navegadores y servidores, su uso aumenta sin que requiera un esfuerzo de los desarrolladores web.

HTTP/2

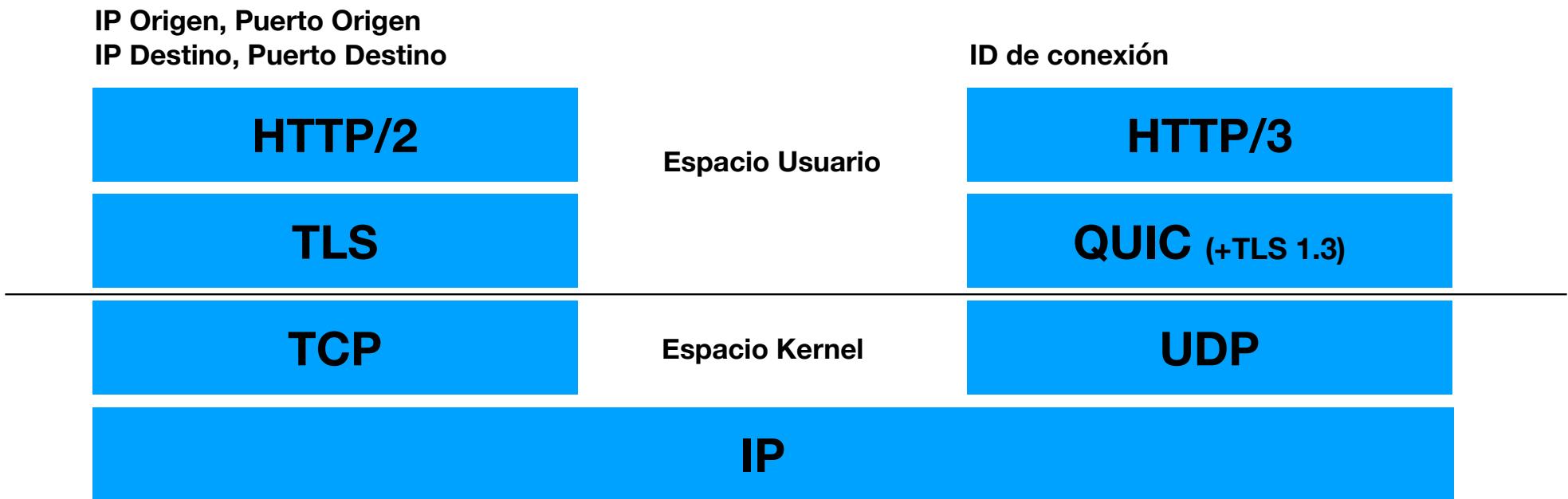
- Ha sido estandarizado de forma oficial en mayo de 2017 pero quizá pronto sea sustituido por HTTP/3.
- Algunas de sus características:
 - Soporte para la cabecera **Alt-Svc**, que permite **disociar la identificación de una ubicación**, con respecto a un recurso pedido, permitiendo un uso más inteligente de los mecanismos de cacheo de memoria de los **CDN** (Content Delivery Network).
 - La introducción de la cabecera **Client-Hints**, que permite al navegador o cliente comunicar proactivamente sus necesidades o restricciones de hardware.
 - La introducción de prefijos de **seguridad en la cabecera Cookie**, para garantizar que una Cookie no ha sido alterada.
- La evolución de HTTP demuestra su capacidad de **ampliación, simplificación, flexibilidad y extensibilidad** y de **corrección de errores**.

HTTP/2 - TCP Head of Line Blocking

- **Problema del Bloqueo de Cabecera de Línea TCP.**
- **HTTP/2** soluciona el problema de HTTP 1.1 y su uso ineficiente de conexiones TCP, reutilizando las conexiones (keep-alive).
- **Múltiples peticiones y respuestas pueden transmitirse por la misma conexión al mismo tiempo**, pero... pueden verse **afectadas por la pérdida de paquetes** (por congestión de la red por ejemplo).
 - TCP no sabe nada de esta abstracción, por lo que para él es un flujo de bytes sin un significado particular.
 - Cuando **un paquete TCP se pierde**, se crea un espacio en el flujo y **TCP necesita llenarlo reenviando el paquete afectado al detectar la pérdida**.
- Mientras dura este proceso, **ninguno de los bytes recibidos pueden entregarse a la aplicación**, incluso si no se perdieron y pertenecen a una solicitud HTTP completamente independiente.
- Por este motivo se producen **retardos innecesarios** ya que **TCP no sabe si la aplicación puede procesar los bytes que ha entregado sin los que faltan**.



- **QUIC** (Quick UDP Internet Connections)
- Creado por **Google**, pero ha evolucionado y actualmente no se parece mucho a la versión inicial.
- Es un **nuevo protocolo de transporte encriptado por defecto**, que provee de mejoras diseñadas para acelerar el tráfico HTTP, además de para hacerlo **más seguro**, con la idea de **reemplazar TCP y TLS en la web**.
- Provee de **autenticación y encriptación por defecto**, que eran típicamente gestionadas por un protocolo de una capa superior como TLS.
- Tiene soporte para **multiplexar diferentes streams HTTP en diferentes streams de transporte QUIC**, compartiendo la misma conexión y sin handshake adicionales.
- La **pérdida de paquetes** en un stream **no afecta a otras peticiones**. Ya no gestiona los paquetes TCP, **los gestiona QUIC**.
- Puede **reducir dramáticamente el tiempo necesario para renderizar una web** (con CSS, Javascript, imágenes y otros recursos), particularmente cuando las redes están congestionadas con altas tasas de pérdidas de paquetes.
- Está diseñado para funcionar **encima de los datagramas UDP**.



- Es complicado reemplazar **TCP** o **UDP** → Hay **muchos equipos viejos en la red**. Los navegadores se actualizan cada semana, los servidores cada poco tiempo pero los routers, switches, etc. ¡tardan años!
- **UDP** no otorga **garantías de entrega** de mensajes. **Se encarga QUIC** de hacer eso en el espacio de usuario.
- Se **evitan handshakes** de **TCP** y de **TLS**.
- QUIC no sólo se va a usar para web, **se utilizará para otros servicios** ofreciendo sus ventajas.



- Para poder funcionar, QUIC debe romper algunas de las suposiciones que muchas aplicaciones de red daban por sentadas y esto hace que **el despliegue de QUIC sea más difícil**.
- Está diseñado para **entregarse sobre datagramas UDP**, para facilitar la implementación y evitar problemas provenientes de **dispositivos de red que eliminan paquetes de protocolos desconocidos**.
- Esto permite también que **las implementaciones de QUIC vivan en el espacio del usuario**, de modo que **los navegadores podrán implementar nuevas funciones del protocolo** sin esperar actualizaciones de sistemas operativos.



- **Enrutadores NAT:**

- Los enrutadores NAT pueden hacer un **seguimiento de las conexiones TCP** que pasan a través de ellos a través de la **dirección de origen, puerto de origen, dirección de destino y puerto de destino**.
- Pueden observar los paquetes **TCP SYN, ACK y FIN** transmitidos a través de la red y ver **cuándo empieza y cuándo termina una conexión**.
 - Así pueden gestionar con precisión la vida útil de los enlaces NAT, la asociación entre las direcciones y puertos externos e internos.
- Con QUIC esto no es posible. **Los enrutadores NAT actuales no entienden QUIC**, por lo que recurren al manejo predeterminado y menos preciso de puertos UDP, que implica el uso de tiempos de espera arbitrarios y a veces muy cortos, lo que podría afectar a conexiones de larga duración.

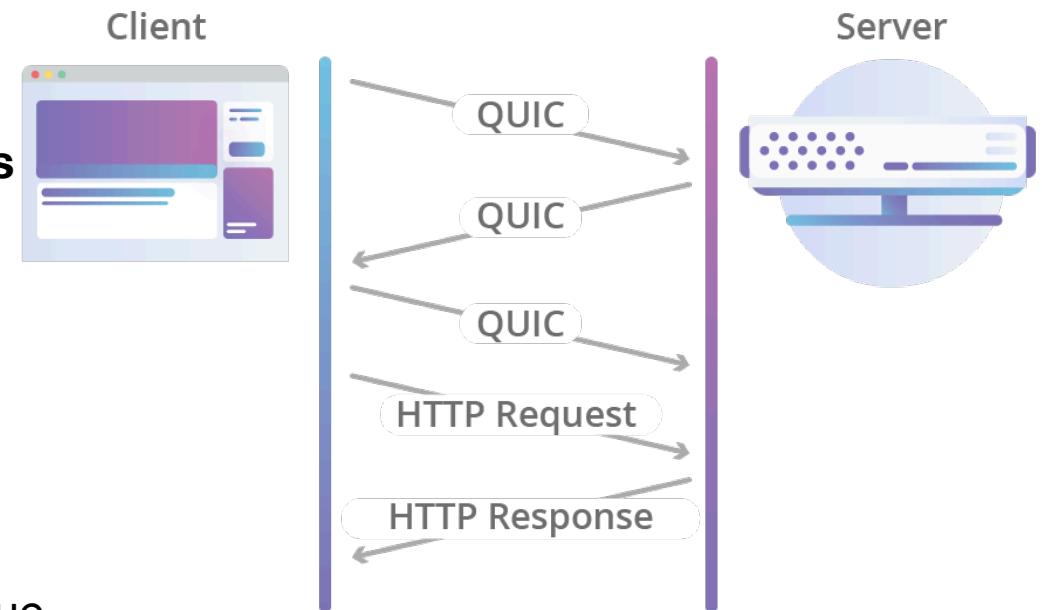


- **Migración de conexión**
 - QUIC pretende ofrecer **migración de conexión**:
 - Por ejemplo, un móvil podrá migrar conexiones QUIC entre redes de datos móviles y WIFI cuando una red WIFI conocida esté disponible. (Por ejemplo, al entrar en una cafetería).
 - También se podría migrar de Ethernet a Wifi en un portátil.
 - Para lograrlo intenta introducir el **concepto de ID de conexión**, una identificación arbitraria de longitud variable, en lugar del clásico IP de origen, puerto de origen, IP de destino, puerto de destino.
 - Esto también plantea un problema para los operadores de red, donde una IP de destino puede identificar a cientos e incluso miles de servidores.



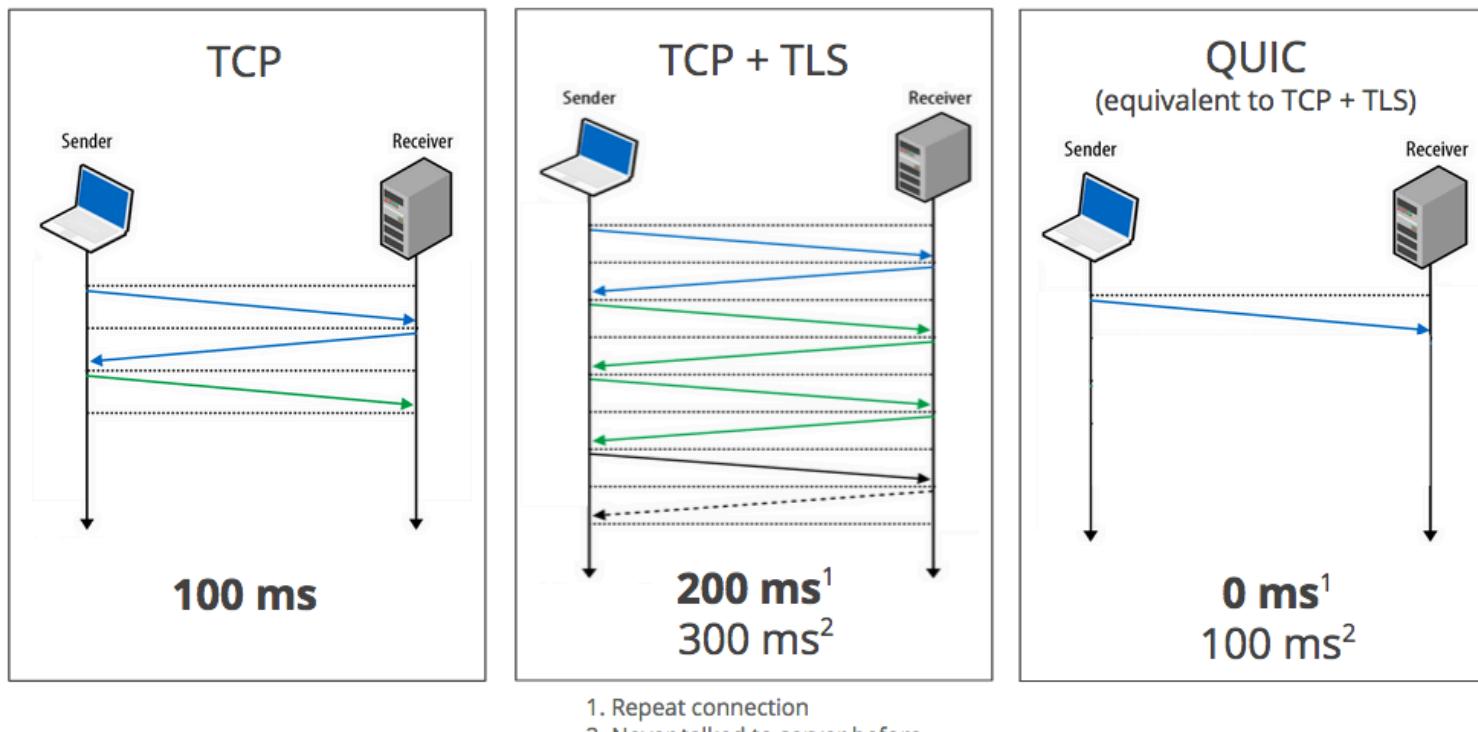
- Quick al utilizar UDP es mucho más flexible que TCP y permite los beneficios de HTTP/2 sin el problema del head of line blocking de TCP.
- Permite **establecer conexiones mucho más rápido**.
- Mediante la cabecera **Alt-Svc**, un servidor puede notificar a un navegador que ofrece el servicio de HTTP/3.
- Los navegadores más utilizados ya soportan HTTP/3 (Safari, Firefox, Chrome, Edge) aunque hay que activarlo manualmente.

HTTP Request Over QUIC





Zero RTT Connection Establishment



Comparación del tiempo de establecimiento de una conexión con TCP, TCP + TLS y QUIC



Retos:

- 3-7% de las conexiones por HTTP/3 fallan. **El cliente puede volver atrás** a HTTP/2 o HTTP/1. Le es transparente.
- Requiere **servidores más potentes** (2x o 3x la carga).
- **QUIC** está a **nivel de usuario**, no de kernel, por lo que habrá múltiples implementaciones y aparecerán incompatibilidades.
- Faltan **herramientas** de análisis y de tests.
- Crecerá despacio, pero **a largo plazo sustituirá a TCP**.

HTTP: SEGURIDAD

Introducción a la seguridad en web

- En ocasiones debemos introducir en la web **información confidencial**:
 - Autenticarnos en servidores de correo.
 - Entrar en la página de nuestro banco.
 - Etc.
- Como la **información puede viajar sin cifrar**, puede ser **interceptada** por un usuario malintencionado.
- HTTP no es un protocolo seguro (HTTP/3 sí)
- Es necesario implementar medidas de seguridad como:
 - **SSL**: Secure Socket Layer
 - **TSL**: Sucesor de SSL
- Para paliar este problema surge **HTTPS**.

HTTPS

- **HTTPS** (HyperText Transfer Protocol Secure): O protocolo seguro de Transferencia de Hipertexto.
- Se apoya en una conexión previamente establecida en la **capa de transporte** mediante la utilización de **SSL** o **TLS**, que encripta la información susceptible de comprometer la seguridad: usuario y contraseña, por ejemplo.
- Este tipo de conexiones requiere de mecanismos de cifrado basados en el uso de **clave pública** y la **utilización de certificados**.
- El puerto que se suele utilizar para HTTPS es el **443**, en lugar del 80.
- El cliente utilizará **https** en la URI en lugar de http.

Firma Digital

- La **firma digital** ofrece el soporte para la **autenticación** e **integridad** de los datos, así como para el **no repudio** en origen. Se basa en los mecanismos que ofrece la **criptografía asimétrica**, donde un mensaje cifrado con una **clave pública** sólo puede ser descifrado con la **clave privada** y viceversa.
 - El receptor puede verificar la identidad del transmisor.
 - El transmisor no puede repudiar el contenido del mensaje.
 - El receptor no puede confeccionar el mensaje él mismo.
- **Creación:** Se calcula algún **hash** o resumen del documento, como **SHA** y se cifra utilizando la **clave privada**. El resultado de este valor se conoce como la **firma digital del documento**.
- **Comprobación:** La firma se verifica utilizando la **clave pública del firmante**. Se obtiene el valor hash del documento y se comparan los dos resúmenes. Si coinciden, **la firma es válida**.

Certificado Digital

- Un **certificado digital** es un documento mediante el cual, una autoridad de certificación (un tercero confiable) **garantiza la vinculación de identidad de un sujeto o entidad y su clave pública**.
 - (Sin el tercero de confianza, se pueden llevar a cabo ataques MiTM.)
 - Los certificados se usan para **comprobar que una clave pública pertenece a un individuo** o entidad.
 - Gracias al certificado digital se pueden garantizar que las partes son quienes dicen ser que son.
 - El formato estándar de certificados digitales es el **X.509**.

Certificado Digital

- El estándar **X.509** surge por la necesidad de que distintos sistemas puedan **interoperar** entre ellos.
- El certificado debe contener al menos lo siguiente:
 - **Contenido fundamental:** identidad y clave pública asociada:
 - La identidad del propietario del certificado (identidad a certificar).
 - La clave pública asociada a esa identidad.
 - Datos imprescindibles para la **validación:**
 - La identidad de quién expide y firma el certificado.
 - El algoritmo criptográfico utilizado para firmar el certificado.

Certificado Digital

- Un certificado emitido por una **entidad de certificación autorizada** debe estar firmado digitalmente por ella y además, contener lo siguiente:
 - Nombre, dirección y domicilio del suscriptor.
 - Identificación del suscriptor nombrado en el certificado.
 - El nombre, la dirección y el lugar donde realiza actividades la entidad de certificación.
 - La clave pública del usuario.
 - La metodología para verificar la firma digital del suscriptor impuesta en el mensaje de datos.
 - El número de serie del certificado.
 - La fecha de emisión y expiración del certificado.

Certificado Digital

Certificados

Propósito planteado: <Todos>

Entidades de certificación intermedias Entidades de certificación raíz de confianza Editor

Emitido para	Emitido por	Fecha de...	Nombre descriptivo
Copyright (c) 1997 ...	Copyright (c) 1997 Mi...	31/12/1999	Microsoft Timest...
DigiCert High Assur...	DigiCert High Assuran...	10/11/2031	DigiCert
Entrust.net Secure ...	Entrust.net Secure Se...	25/05/2019	Entrust
Equifax Secure Cer...	Equifax Secure Certifi...	22/08/2018	GeoTrust
Equifax Secure Glo...	Equifax Secure Global...	21/06/2020	Equifax Secure ...
FNMT Clase 2 CA	FNMT Clase 2 CA	18/03/2019	Fabrica Nacional ...
GeoTrust Primary C...	GeoTrust Primary Cer...	17/07/2036	GeoTrust
GlobalSign Root CA	GlobalSign Root CA	28/01/2028	GlobalSign
Go Daddy Class 2 C...	Go Daddy Class 2 Cer...	29/06/2034	Go Daddy Class ...

Importar... Exportar... Quitar Avanzadas

Propósitos planteados del certificado
Correo seguro, Autenticación del servidor

Ver Cerrar

Obtener más información acerca de [certificados](#)

(Leer más... | 369 bytes más | <Comentarios?>)

Certificado

General Detalles Ruta de certificación

Mostrar: <Todos>

Campo	Valor
Versión	V3
Número de serie	36 f1 1b 19
Algoritmo de firma	sha1RSA
Emitor	FNMT Clase 2 CA, FNMT, ES
Válido desde	jueves, 18 de marzo de 1999 ...
Válido hasta	jueves, 18 de marzo de 2019 1...
Aapunto	FNMT Clase 2 CA, FNMT, ES
Clave pública	RSA (1024 bits)

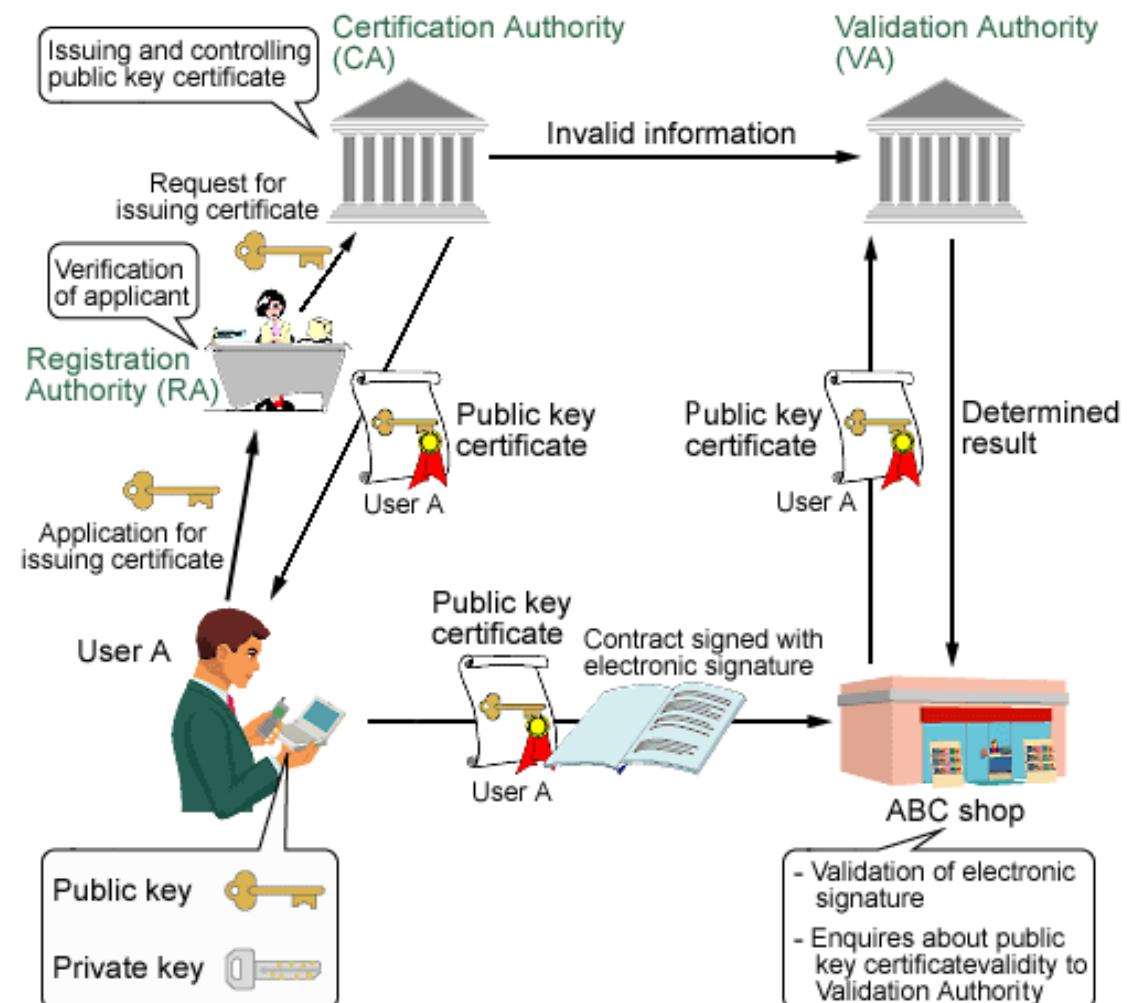
OU = FNMT Clase 2 CA
O = FNMT
C = ES

Modificar propiedades... Copiar en archivo... Aceptar

Más información acerca de los [detalles del certificado](#).

PKI

- **PKI (Public Key Infrastructure):** Infraestructura de clave pública.
- Es todo lo necesario: **HW y SW**, para las **comunicaciones seguras** mediante el uso de **certificados digitales y firmas digitales**.
- Gracias a ella se alcanzan los objetivos de **autenticidad, confidencialidad, integridad y no repudio**.



PKI

- Las PKI están compuestas de:
 - La **autoridad de certificación, CA** (Certificate Authority), la entidad de confianza encargada de emitir y revocar certificados digitales.
 - La **autoridad de registro, RA** (Registration Authority), encargada de **controlar la generación de certificados** procesando las **peticiones** de los usuarios y **comprobando la identidad** de los mismos. Por último, solicita a la CA la expedición del certificado digital.
 - Las **autoridades de los repositorios** donde se almacenan los certificados emitidos y revocados.
 - Todo el **software** necesario para poder utilizar los certificados digitales.
 - **Política de seguridad** definida para las comunicaciones.

OpenSSL

Cryptography and SSL/TLS Toolkit

- Es una implementación robusta, de nivel comercial y Open Source con todas las características de los protocolos **SSL** y **TLS**, además de bibliotecas con propósitos criptográficos.
 - **Creación** de certificados digitales.
 - **Instalación** de Certificados digitales.
 - **Manejo** de Certificados digitales
 - Generar y firmar certificados
 - Revocar Certificados
 - Renovar un certificado
 - Visualizar un certificado.

```
sudo apt-get update  
sudo apt-get install openssl
```

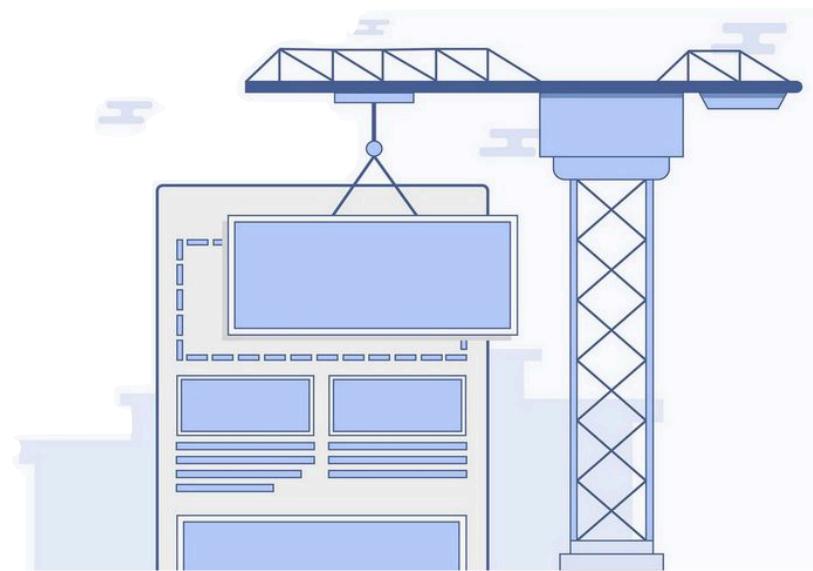
SERVIDORES WEB

Servidores web: características generales

- Deben tener la capacidad de servir al menos los **ficheros estáticos** que se encuentran en el disco.
- Con el tiempo se les han ido añadiendo **nuevas funcionalidades**:
 - Capacidad de servir páginas estáticas y dinámicas
 - Servidores virtuales
 - Seguridad y autenticación
 - Protocolos adicionales
 - Prestaciones extra
 - Módulos

Servidor web: Páginas estáticas

- Son aquellas cuyo contenido es **estático**, no suele variar.
- Se suelen crear o modificar **manualmente**.
- La **interacción** que ofrece al usuario es extremadamente **limitada**.
- No utilizan ningún intérprete o lenguaje de script para ser generadas o procesadas.
- Normalmente son documentos HTML: **.html** o **.htm**
- Son características de la **Web 1.0**.
- También se usan en páginas que no se vayan a modificar, que no ofrezcan interacción o que su contenido no vaya a variar de forma frecuente.



Servidor web: Páginas dinámicas

- Su contenido varía.
- Antes de servirse al cliente **son procesadas** por un programa.
- Es necesario utilizar algún tipo de lenguaje o intérprete.
 - **Páginas dinámicas del lado del cliente:**
 - El proceso corre por parte del navegador.
 - Lenguajes del lado del cliente: Javascript
 - **Páginas dinámicas del lado del servidor:**
 - Son procesadas por el servidor antes de ser servidas.
 - Lenguajes del lado del servidor: PHP, JSP, ASP, Swift, etc.
 - Son propias de la **Web 2.0** en adelante.



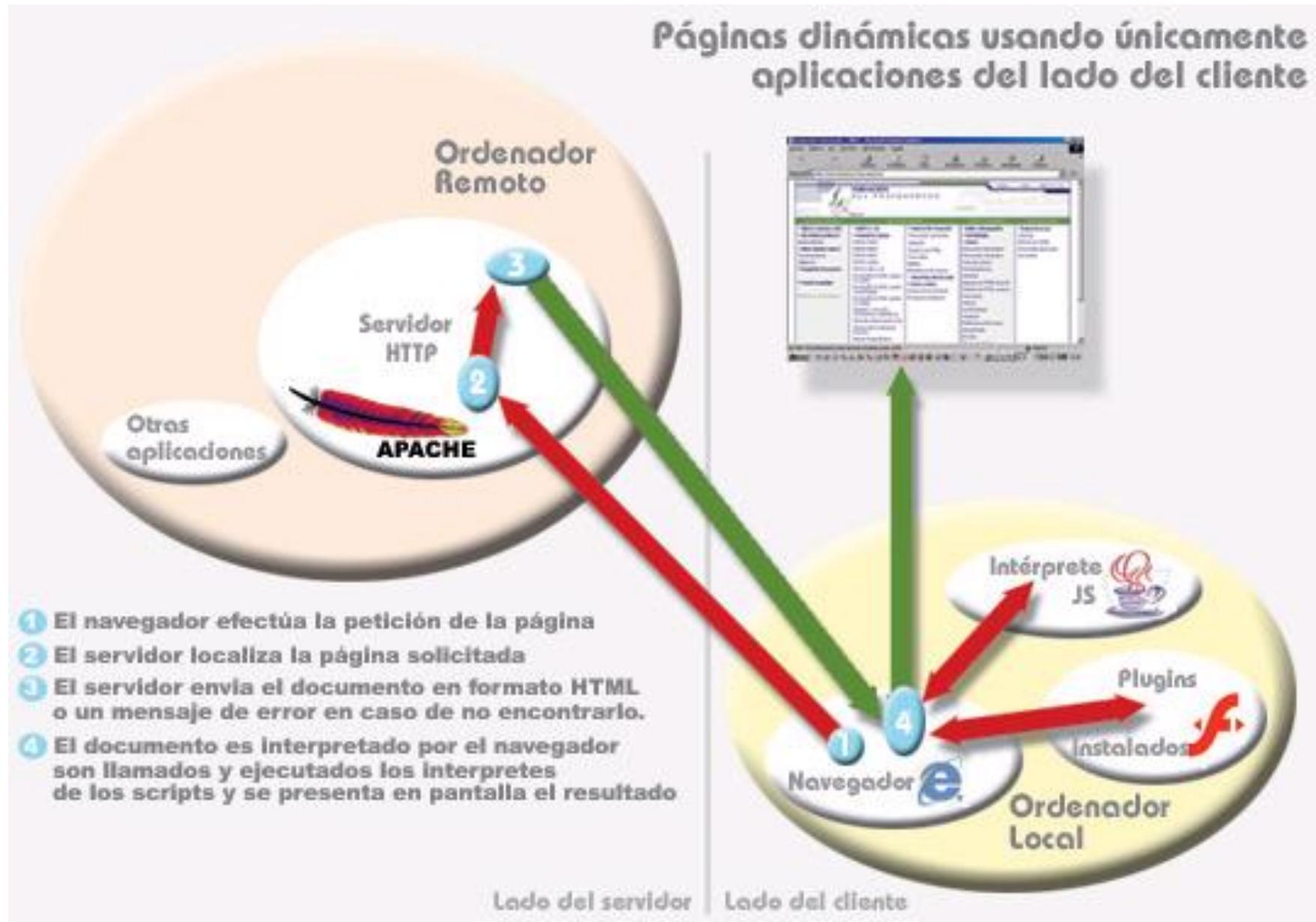
Servidor web: Tecnologías

- **PHP**: Lenguaje libre y muy utilizado cuyo código se combina con HTML para formar **páginas dinámicas**.
- **ASP** (Active Server Pages): Al igual que PHP, se interpreta en el servidor y el resultado se envía al cliente.
- **Java**: Lenguaje de programación multiplataforma. Se usa para applets (apps), servlets (páginas generadas por código) o JSP (Java Server Pages) que son parecidos a PHP.
- **Javascript**: Es el lenguaje más popular para hacer **páginas dinámicas del lado del cliente**. También es posible usarlo como lenguaje de backend e incluso para hacer apps.
- **Node.js** es un entorno de ejecución basado en el motor **V8** de **Chrome**. Es altamente escalable y útil para crear programas de red como servidores web.
- **CGI** (Common Gateway Interface): Es una interfaz que permite a los documentos HTML intercambiar datos con programas escritos en **C** o en **Perl**, por ejemplo.
- **AJAX** (Asynchronous JavaScript and XML): Permite hacer webs dinámicas e **intercambiar datos** con el servidor **sin tener que refrescar la página**.

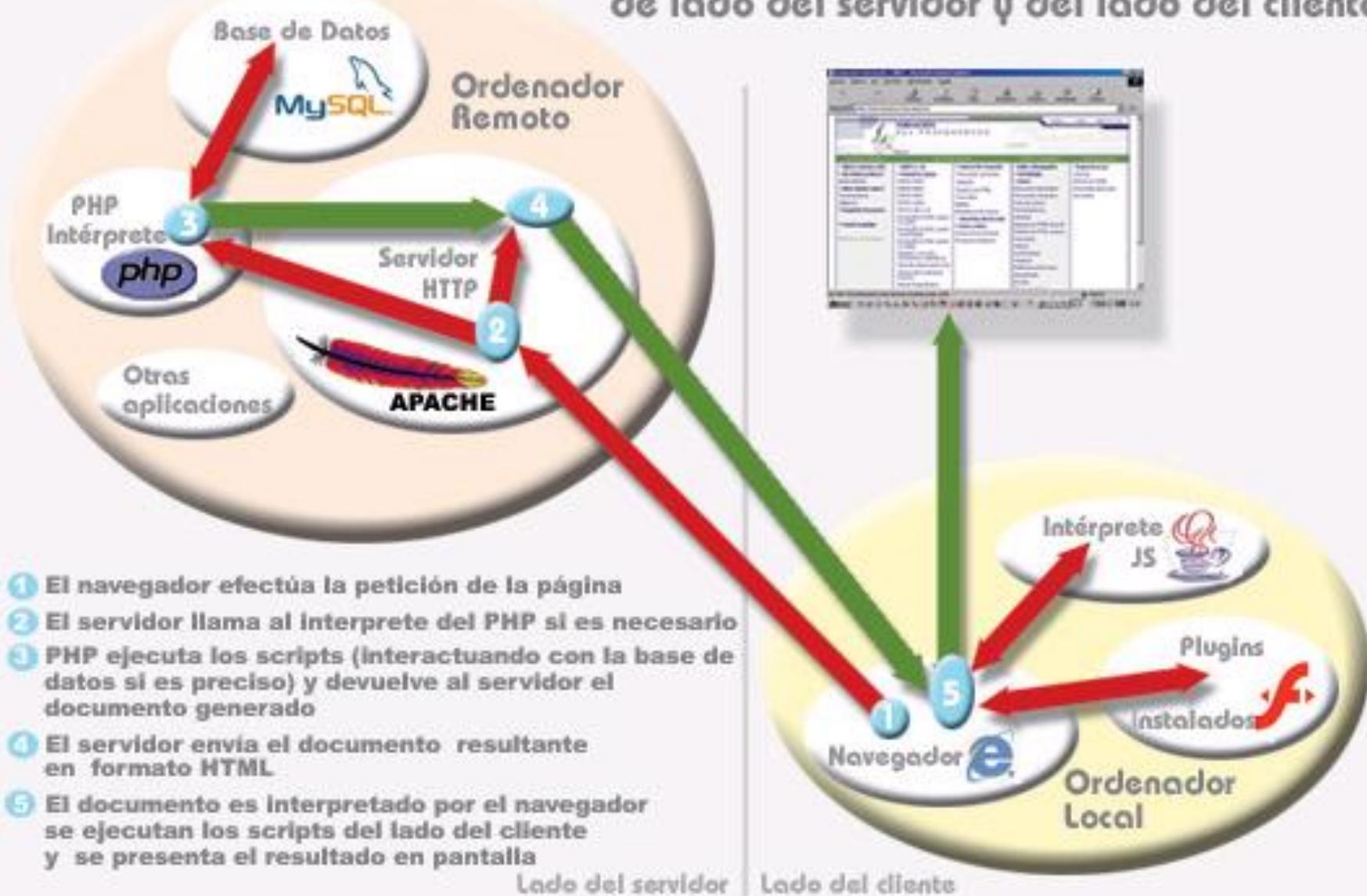
```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
%@page import="java.lang.* ,java.util.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h1>Cálculo de la tabla de multiplicar del 5.-</h1>
<%
    int indice;
    for(indice=0;indice<=10;indice++)
    {
%
        <h<%=indice%>>5 * <%= indice %> = <%= indice*5 %> </h<%=indice%>>
<%
    }
%
</body>
</html>
```

Ejemplo de página dinámica programada en JSP

Páginas dinámicas usando únicamente aplicaciones del lado del cliente



Páginas dinámicas usando aplicaciones de lado del servidor y del lado del cliente



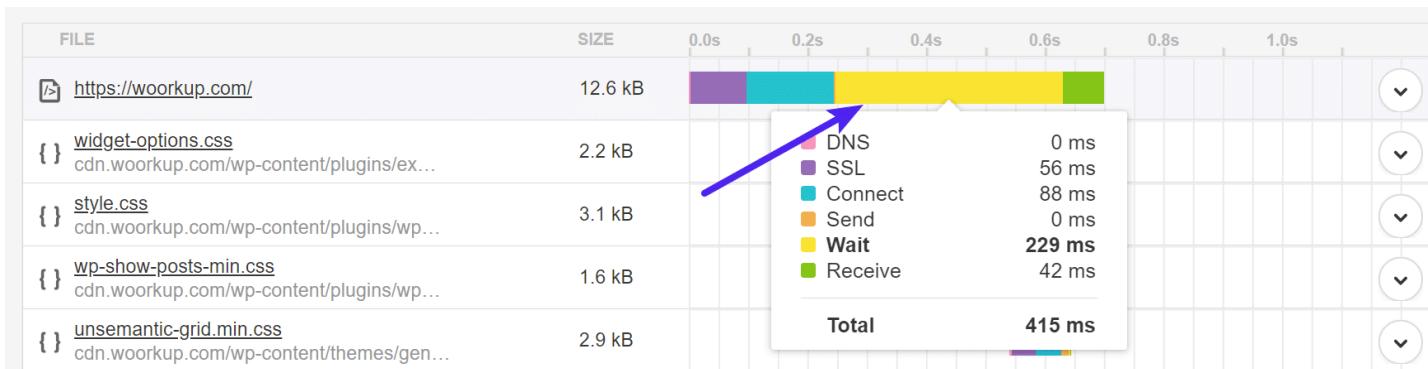
- Generar páginas dinámicamente consume CPU y memoria en el servidor, que es quizás el recurso más valioso.t
- Es interesante configurar algún sistema de caché y servir versiones estáticas de las páginas hasta que éstas cambien.
- De esta manera, se mejoran también los tiempos de respuesta: Mejor experiencia de usuario y mejor posicionamiento SEO.

Caché de páginas dinámicas

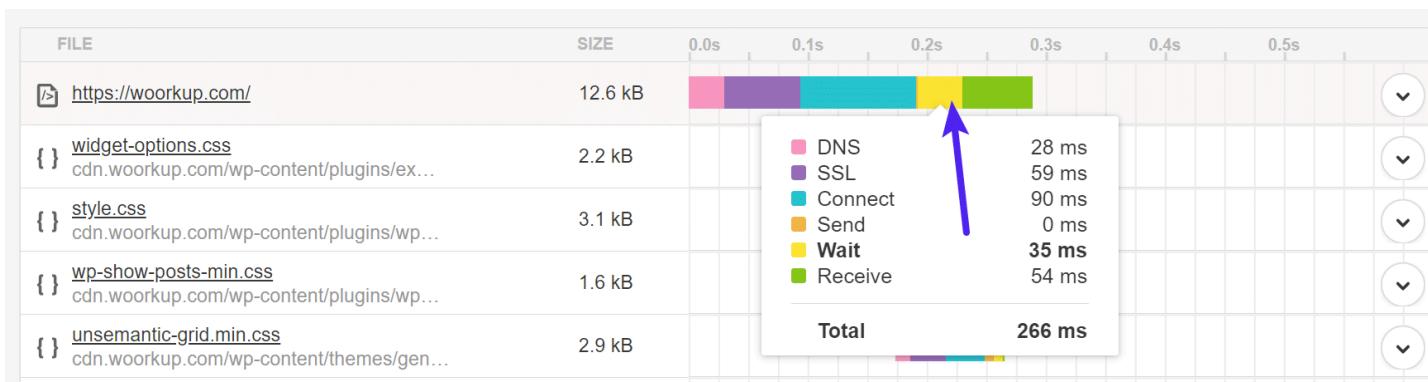
- Servir páginas dinámicas es **mucho más costoso** que servir páginas estáticas:
 - Hay que ejecutar un programa que genere la página.
 - Crear una página puede dar lugar a **varias consultas** o realizar consultas complejas hacia la base de datos.
 - **Se tarda tiempo** en realizar todas las acciones necesarias para construir la página antes de servirla al usuario.
- La idea consiste en **generar la página una vez**, y **sólo actualizarla en el caso de que varíe** su contenido, **guardando una versión estática** de la misma para enviar a los clientes que la soliciten.
- **Ventajas:**
 - Se **reduce la carga del servidor**: Se pueden atender a **más clientes con el mismo hardware**, por lo que se ahorra dinero.
 - Se responde antes a los clientes: **Mejora de experiencia de usuario**.
 - Mejor posicionamiento web: **¡Los buscadores posicionan mejor las páginas rápidas!**

Caché de páginas dinámicas

Sin caché



Con caché



Servidor web: Hosts Virtuales

- **Host Virtuales:** Mediante el uso de hosts virtuales, los servidores web pueden **alojar varios dominios en una máquina.**
- **1. Basados en direcciones IP:**
 - Por cada **IP** se puede tener un servidor web virtual.
 - El equipo necesita tener conectados **varios adaptadores de red**, cada uno con una IP.
- **2. Basados en puertos:**
 - Un servidor web usará un **puerto** de escucha por cada servidor virtual.
 - Por ejemplo, escuchando en el puerto 80 (http) y en el puerto 443 (https).
- **3. Basados en nombres:**
 - El servidor web puede alojar varios sitios web pertenecientes a **distintos dominios** usando la **misma dirección** y el **mismo puerto** de escucha.

Algunos servidores web

Servidor	Descripción
Apache.	Es el más usado en Internet. Desarrollado por Apache Software Foundation. Es software libre, gratuito y multiplataforma (Unix, Linux, Windows, Netware).
Microsoft IIS.	Es software propietario y sólo se puede instalar en sistemas Windows.
Sun Java System Web	Está desarrollado por Sun y se usa principalmente en sistemas Sun. Actualmente es software libre, gratuito y multiplataforma.
Tomcat.	Al igual que Apache está desarrollado por Apache Software Foundation. Es software libre, gratuito y multiplataforma. Usa una tecnología distinta a la de Apache. Permite trabajar con algunas páginas dinámicas con las que no puede trabajar Apache. Puede trabajar como un servidor de aplicaciones.
Nginx.	Es un servidor que consume muy pocos recursos. Se puede instalar en sistemas Unix/Linux y Windows. Es software libre y gratuito.
Lighttpd.	Es también un servidor que consume muy pocos recursos y muy fácil de configurar con su herramienta gráfica de configuración. Es software libre, gratuito y multiplataforma.
Cherokee.	Su principal objetivo es ser rápido y muy funcional. Es software libre, gratuito y multiplataforma.

SERVIDOR APACHE

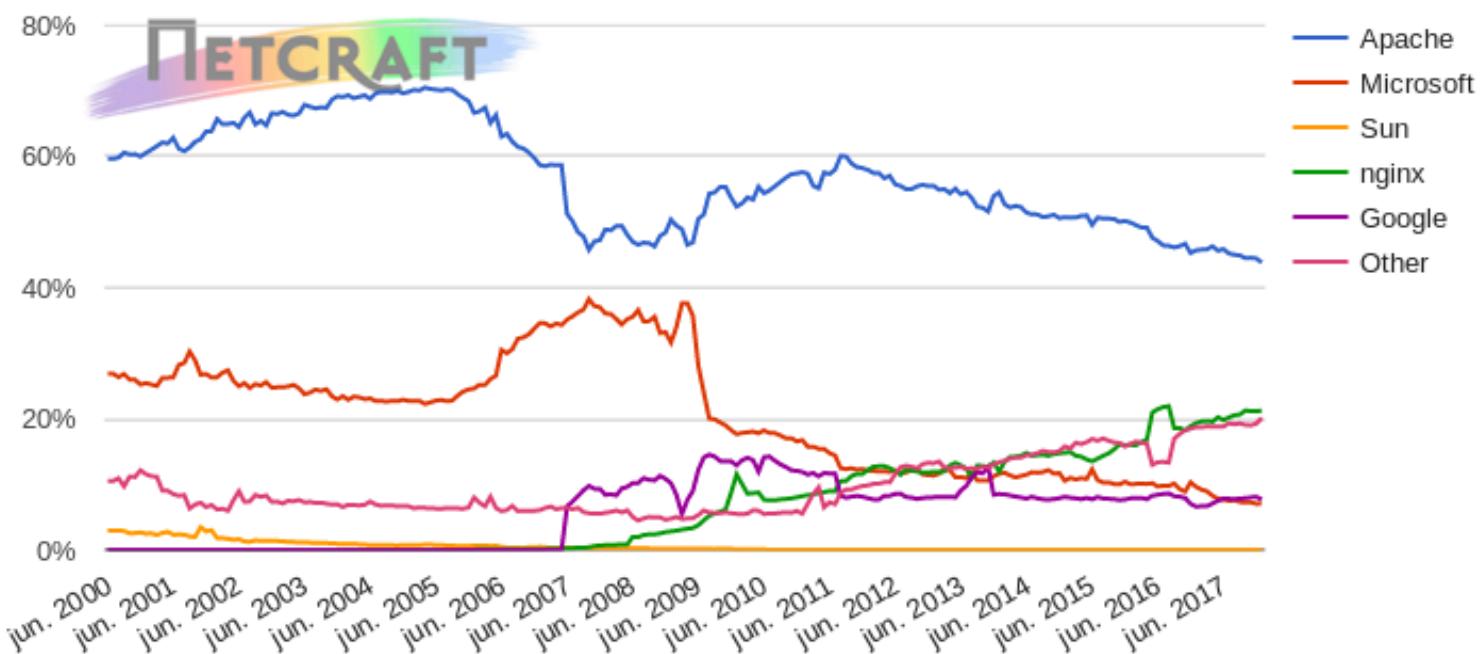
Apache Web Server

- Es un servidor web HTTP de **código abierto**.
- **Licencia Apache**.
- **Multiplataforma**: Windows, Macintosh, Linux, BSD...
- Desarrollado y mantenido por usuarios bajo la supervisión de la **Apache Software Foundation**.
- Es el servidor web más utilizado.
- Modular, extensible, altamente configurable, muy popular, etc.



<https://www.apache.org/>

Web server developers: Market share of active sites



Developer	December 2017	Percent	January 2018	Percent	Change
Apache	76,427,234	44.44%	75,212,932	43.82%	-0.63
nginx	36,333,280	21.13%	36,504,577	21.27%	0.14
Google	13,946,934	8.11%	13,270,339	7.73%	-0.38
Microsoft	12,133,888	7.06%	12,147,278	7.08%	0.02

Documentación oficial

 **HTTP SERVER PROJECT**

Módulos | Directivas | Preguntas Frecuentes | Glosario | Mapa del sitio web

Versión 2.4 del Servidor HTTP Apache

Apache > Servidor HTTP > Documentación

Apache HTTP Server Versión 2.4 Documentación

Idiomas disponibles: [da](#) | [de](#) | [en](#) | [es](#) | [fr](#) | [ja](#) | [ko](#) | [pt-br](#) | [ru](#) | [tr](#) | [zh-cn](#)

Buscar en Google

Notas de la versión Nuevas funcionalidades en Apache 2.3/2.4 Nuevas funcionalidades en Apache 2.1/2.2 Nuevas funcionalidades en Apache 2.0 Actualizar a la versión 2.4 desde la 2.2 Licencia Apache	Guía de Usuario Empezando Enlazando Direcciones y Puertos Ficheros de Configuración Secciones de Configuración Almacenamiento de Contenido en Caché Negociación de Contenido Objetos Compartidos Dinámicamente (DSO) Variables de Entorno Ficheros de Log Cumplimiento del Protocolo HTTP Mapeo de URLs al Sistema de Ficheros Optimización del Rendimiento Consejos de seguridad Configuración básica del Servidor Cifrado SSL/TLS Ejecución de Susex para CGI Reescritura de URL con mod_rewrite Servidores Virtuales	How-To / Tutoriales Índice de Tutoriales Autenticación y Autorización Control de Acceso CGI: Contenido Dinámico Ficheros .htaccess Inclusiones del Lado del Servidor (SSI) Directorios Web por Usuario (public_html) Guía de configuración de Proxy Inverso Guía HTTP/2	Notas Sobre Plataformas Específicas Microsoft Windows Sistemas Basados en RPM (Redhat / CentOS / Fedora) Novell NetWare	Otros Temas Preguntas Frecuentes Mapa del Sitio Documentación para Desarrolladores Contribuir en la Documentación Otras Notas Wiki
---	--	---	---	---

Idiomas disponibles: [da](#) | [de](#) | [en](#) | [es](#) | [fr](#) | [ja](#) | [ko](#) | [pt-br](#) | [ru](#) | [tr](#) | [zh-cn](#)

Copyright 2020 The Apache Software Foundation.
Licencia bajo los términos de la [Apache License, Version 2.0](#).

Módulos | Directivas | Preguntas Frecuentes | Glosario | Mapa del sitio web

<http://httpd.apache.org/docs/2.4/>

Instalación

- Para instalar Apache en sistemas GNU/Linux:
 - **apt-get install apache2**
- Para controlar el servicio apache2
 - **apache2ctl [-k start|restart|gracefull|graceful-stop|stop]**
- La opción **graceful** es un reinicio suave, **se terminan de servir las peticiones establecidas** y cuando se finaliza, se hace un reinicio del servidor.

Instalación: apache2ctl

- También podemos utilizar la herramienta **apache2ctl** para obtener más información del servidor:
 - **apache2ctl -t** → Comprueba la sintaxis del fichero de configuración.
 - **apache2ctl -M** → Lista los módulos cargados.
 - **apache2ctl -S** → Lista los sitios virtuales y las opciones de configuración.
- El servidor está gestionado por **Systemd**, por lo que para controlar el arranque, reinicio y parada utilizaremos:
 - **systemctl [start|stop|restart|reload|status] apache2.service**

Ficheros de configuración

- El **fichero de configuración principal** es:
 - **/etc/apache2/apache2.conf**
 - En este fichero se incluyen los ficheros que forman parte de la configuración de Apache2

```
...
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
...
Include ports.conf
...
IncludeOptional conf-enabled/*.conf
IncludeOptional sites-enabled/*.conf
```

- Ficheros **mods-enabled** → Módulos activos.
- Ficheros **sites-enabled** → Sitios virtuales activos.
- Directorio **conf-enabled** → Añadimos ficheros de configuración adicionales.
- Fichero **ports.conf** → Se especifica los puertos de escucha del servidor.

Opciones para servidores virtuales

- Por defecto, se indican las opciones de configuración del directorio **/var/www** y de todos sus subdirectorios.
- Por lo tanto los **DocumentRoot** de los **virtual host** que se crean deben ser subdirectorios de este directorio.
- Encontramos en el fichero **/etc/apache2/apache2.conf** lo siguiente:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

- Podemos indicar como directorio raíz de nuestros virtual host otro directorio... (tendríamos que descomentar)

```
#<Directory /srv/>
#    Options Indexes FollowSymLinks
#    AllowOverride None
#    Require all granted
#</Directory>
```

Añadir una nueva configuración

- Si queremos añadir una configuración adicional para nuestro servidor, podemos guardarla en un fichero, dentro del directorio:
 - **/etc/apache2/conf-available**
- Para añadir dicho fichero de configuración, a la configuración general del servidor usamos la instrucción:
 - **a2enconf prueba** → Esta instrucción crea un enlace simbólico en el directorio `/etc/apache2/conf-enabled`.
- Para desactivar una configuración utilizamos:
 - **a2disconf prueba**

Variables de entorno

- Apache ofrece un mecanismo para almacenar la información en variables especiales que se llaman **variables de entorno**.
- Esta información puede ser utilizada para **controlar diversas operaciones**, como almacenar los datos en ficheros de registro (log files) o controlar el acceso al servidor.
- Podemos encontrar estas variables definidas en el fichero
 - **/etc/apache2/envvars**

Algunas directivas

- Algunas directivas que podemos encontrar en el fichero:
 - **/etc/apache2/apache2.conf**
- **Timeout** → Define en segundos el tiempo que el servidor esperará por recibir y transmitir durante la comunicación (por defecto, 300 segundos).
- **KeepAlive** → Define si las conexiones persistentes están activadas (por defecto, lo están).
- **MaxKeepAliveRequests** → Establece el número máximo de peticiones permitidas por cada conexión persistente (por defecto, 100).
- **KeepAliveTimeout** → Establece el número de segundos que el servidor esperará tras haber dado servicio a una petición antes de cerrar la conexión (por defecto, 5 segundo).

Otras directivas

- **User** → Define el usuario que ejecuta los procesos de apache2.
- **Group** → Define el grupo al que corresponde el usuario.
- **LogLevel** → Controla el nivel de información que se guarda en los ficheros de logs.
- **LogFormat** → Controla el formato de la información que se guarda en los ficheros de logs.
- **Directory** o **DirectoryMatch** → Declara un contexto para un directorio y todos sus subdirectorios.
- **Files** o **FilesMatch** → Declara un contexto para un conjunto de ficheros.

Probando el servidor (1)

- El servidor apache se instala por defecto con la configuración de un servidor virtual.
 - **/etc/apache2/sites-available/000-default.conf**
- Por defecto **este sitio virtual está habilitado**, por lo que podemos comprobar que existe un enlace simbólico a este fichero en el directorio:
 - **/etc/apache2/sites-enabled**
- Una de las directivas más importantes que nos encontramos en el fichero de configuración es **DocumentRoot**, donde se indica el directorio donde van a estar guardados los ficheros de nuestro sitio web.
 - **/var/www/html**

Probando el servidor (2)

- En el fichero de la configuración general **/etc/apache2/apache2.conf** nos encontramos las opciones de configuración del directorio padre indicado en la directiva **DocumentRoot**. (suponemos que todos los hosts virtuales van a estar guardados en subdirectorios de este directorio):
- Por lo tanto podemos acceder desde un navegador a la IP de nuestro servidor (también podemos usar el nombre del servidor) y accederemos a la **página de bienvenida** que se encuentra en:
 - **/var/www/html/index.html**
 - Por defecto, los **errores** de nuestro sitio virtual se guardan en **/var/log/apache2/error.log**
 - Por defecto, los **accesos** a nuestro servidor se guardan en **/var/log/apache2/access.log**

```
...
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
...
```

Introducción a Virtual Hosting

- **Virtual Hosting** es un mecanismo para hacer funcionar varias páginas web en un mismo servidor.
 - Basados en **direcciones IP** → Cada sitio tiene una IP diferente.
 - Basados en **nombres diferentes** → Cada sitio tiene un nombre diferente de dominio.
 - Basados en **puertos diferentes** → Cada sitio tiene un puerto diferente.
- El servidor apache instala por defecto un host virtual en
 - **/etc/apache2/sites-available/000-default.conf**
- Podemos habilitar o deshabilitar nuestros hosts virtuales con los comandos **a2ensite** y **a2dissite**.

```
<VirtualHost *:80>
    #ServerName www.example.com
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

- **ServerName** → Nombre por el que se va a acceder al vhost.
- **ServerAdmin** → Email del responsable.
- **ServerAlias** → Otros nombres por los que se puede acceder al sitio.
- **DocumentRoot** → Directorio donde se guardan los ficheros servidos en el vhost.
- **ErrorLog** → Fichero de logs de errores.
- **CustomLog** → Fichero de accesos al sitio

Configuración de Virtual Hosting (1)

- Se van a configurar dos sitios web distintos en el mismo servidor Apache.
 - **VHOST 1**
 - Nombre: www.pagina1.org
 - Directorio base → /var/ww/pagina1
 - **VHOST 2**
 - Nombre: www.pagina2.org
 - Directorio base → /var/ww/pagina2
- Los ficheros de configuración se encuentran en: **/etc/apache2/sites-available**
 - **cd /etc/apache2/sites-available**
 - **cp 000-default.conf pagina1.conf**
 - **cp 000-default.conf pagina2.conf**
- Modificamos los ficheros **pagina1.conf** y **pagina2.conf** para indicar el nombre que vamos a usar para acceder al host virtual (**ServerName**) y el directorio de trabajo (**DocumentRoot**).
- También podríamos cambiar el nombre del fichero donde se guardan los logs.

Configuración de Virtual Hosting (2)

- No es suficiente con crear los ficheros de configuración, es necesario crear un enlace simbólico a estos ficheros dentro del directorio **/etc/apache2/sites-enabled**
 - **a2ensite pagina1**
 - **a2ensite pagina2**
- La creación de los enlaces simbólicos se puede hacer con la instrucción:
 - **a2ensite nombre_fichero_configuración**
- Para desactivar un sitio utilizaríamos:
 - **a2dissite nombre_fichero_configuración**
- Hay que crear los directorios y los ficheros index.html necesarios en **/var/www** y **reiniciar el servicio**. Hay que tener en cuenta que los ficheros deben de ser del usuario y grupo que usa Apache: usuario: **www-data** y grupo: **www-data**.
- Para finalizar hay que editar el **fichero hosts en los clientes** y poner dos líneas donde se haga la conversión entre nombre y dirección IP.

Configuración de los puertos de escucha

- Para determinar los puertos de escucha del servidor web utilizamos la directiva Listen, que podemos modificar en el archivo:
 - **/etc/apache2/ports.conf**
- **Listen** le dice al servidor principal en qué direcciones y puertos tiene que escuchar.
 - Si no se usan directivas **VirtualHost**, el servidor se comporta de la misma manera con todas las peticiones que se acepten.
 - **VirtualHost** puede usarse para especificar un **comportamiento diferente** en una o varias direcciones y puertos.
 - Para implementar un **vhost** hay que indicarle primero al **servidor que escuche en aquellas direcciones y puertos a usar**.
 - Después se debe de crear una sección **VirtualHost** en una **dirección y puerto** específicos para determinar el **comportamiento de ese host virtual**.
 - Por defecto, los vhost definidos responden desde cualquier IP en el puerto 80. en el fichero /etc/apache2/sites-available/000-default.conf encontramos:

```
<VirtualHost *:80>
```

```
...
```

Ejemplo: Virtual Host basado en IP

- En este caso nuestro servidor debe tener configurado varias IPs (varias interfaces de red).
- Serviremos diferentes páginas por cada IP.

```
<VirtualHost 192.168.56.3:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/externa
    ErrorLog ${APACHE_LOG_DIR}/error_externa.log
    CustomLog ${APACHE_LOG_DIR}/access_externa.log combined
</VirtualHost>

<VirtualHost 172.22.0.1:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/interna
    ErrorLog ${APACHE_LOG_DIR}/error_interna.log
    CustomLog ${APACHE_LOG_DIR}/access_interna.log combined
</VirtualHost>
```

Ejemplo: Virtual Host basado en IP

- En este caso nuestro servidor debe tener configurado varias IPs (varias interfaces de red).
- Serviremos el mismo contenido en varias IP.

```
<VirtualHost 192.168.56.3 172.22.0.1>
    DocumentRoot /var/www/externa
    ServerName servidor.example.com
    ServerAlias servidor
    ...
</VirtualHost>
```

Ejemplo: Distintos sitios en distintos puertos

- En esta ocasión se definen dos puertos de escucha en el fichero `/etc/apache2/ports.conf`

```
Listen 80  
Listen 8080
```

- Y la configuración de los vhosts sería la siguiente:

```
<VirtualHost *:80>  
    ServerName servidor.example.com  
    DocumentRoot /var/www/externa  
</VirtualHost>  
  
<VirtualHost *:8080>  
    ServerName servidor.example.com  
    DocumentRoot /var/www/interna  
</VirtualHost>
```

Mapeo de URL: Opciones de directorios

- Cuando indicamos la configuración de un servidor por apache, por ejemplo con la directiva Directory, podemos indicar algunas opciones con la directiva Options como:
 - **All** → **Todas las opciones excepto MultiViews.**
 - **FollowSymLinks** → Se pueden seguir los enlaces simbólicos.
 - **Indexes** → Cuando **accedemos al directorio y no** se encuentra un **fichero por defecto** (indicado con la directiva DirectoryIndex del módulo mod_dir), por ejemplo el index.html, **se muestra la lista de ficheros** (esto lo realiza el módulo mod_autoindex).
 - **MultiViews** → Permite la **negociación del contenido**, mediante el módulo (mod_negotiation), como mostrar una página distinta en función del **idioma** del navegador.
 - **SymLinksIfOwnerMatch** → Se pueden **seguir enlaces simbólicos** solo cuando el **fichero destino es del mismo propietario** que el enlace simbólico.
 - **ExecCGI**: Permite ejecutar **script CGI** usando el módulo (mod_cgi).
 - Podemos **activar o desactivar una opción** en referencia con la **configuración de un directorio padre** mediante el **signo + o -**

Mapeo de URL: Opciones de directorios: Ejemplos

- En el fichero **/etc/apache2/apache2.conf** nos encontramos el siguiente código:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    ...

```

- A continuación se podrían cambiar las opciones del vhost pagina1 incluyendo en su fichero de configuración:

```
<Directory /var/www/pagina1>
    Options -Indexes +Multiviews
    ...

```

Mapeo de URL: Trabajando con Alias

- La directiva **Alias** nos permite que el servidor **sirva ficheros desde cualquier ubicación del sistema de archivo** aunque esté **fuerza del directorio indicado en DocumentRoot**.
- Además es necesario dar permiso de acceso al directorio.

```
Alias "/image" "/ftp/pub/image"
<Directory "/ftp/pub/image">
    Require all granted
</Directory>
```

- También es posible usar **expresiones regulares** y la directiva **AliasMatch**

```
AliasMatch "^/image/(.*)$" "/ftp/pub/image/$1"
```

Mapeo de URL: Negociación de contenidos

- Apache puede escoger la mejor representación de un recurso basado en las **preferencias proporcionadas por el navegador** para los distintos tipos de **medios, idiomas, conjuntos de caracteres y codificación**.
- A esta funcionalidad se le denomina **Negociación de contenido**.
- Los navegadores pueden indicar sus preferencias mediante cabeceras como **Accept-Language: es**.

Ejemplo: Negociación de contenidos con Multiviews

- Queremos acceder a la URL www.pagina1.org/internacional donde se muestre un index.html con el idioma adecuado según la cabecera Accept-Language enviada por el cliente.
- Lo primero es crear varios ficheros index.html con los distintos idiomas ofrecidos por el servidor.

```
# ls /var/www/html/internacional  
index.html.en  index.html.es
```

- A continuación se debe activar la opción **Multiviews** para el directorio con el que estamos trabajando.
- Por lo tanto en el fichero de configuración del vhost **/etc/apache2/sites-available/pagina1.conf** creamos una sección **Directory**:

```
...  
<Directory /var/www/html/internacional>  
    Options +Multiviews  
</Directory>  
...
```

- Ya sólo hay que configurar el idioma en el navegador y acceder a la URL y podremos ver cómo sirve las páginas según el idioma seleccionado.

Ejemplo: Negociación de contenidos con type-map (1)

- Un **handler** es una representación interna de Apache de una **acción que se va a ejecutar** cuando hay una **llamada a un fichero**.
- Generalmente los ficheros tienen **handlers implícitos**, basados en el **tipo** de fichero del que se trata.
- Normalmente, todos los ficheros son simplemente servidos por el servidor, pero algunos se tratan de forma diferente.
- En este caso se usará un fichero especial denominado **type-map** con **extensión var** al que se le creará un **handler** para manejarlo de una forma especial en la **negociación de contenidos**.

Ejemplo: Negociación de contenidos con type-map (2)

- Los ficheros de tipo mapa tienen una entrada para cada variante disponible.
- Estas entradas consisten en líneas de cabecera contiguas en formato HTTP.
- Las entradas para diferentes variables se separan con líneas en blanco. Las líneas en blanco no están permitidas dentro de una entrada.
- La configuración del directorio sería:

```
<Directory /var/www/html/internacional>
    DirectoryIndex index.var
    AddHandler type-map .var
</Directory>
...
```

- Y en el directorio /
var/www/html/internacional
además de
index.html.en,
index.html.es
tendremos un fichero
index.var con el
siguiente contenido:

URI: index

URI: index.html.en
Content-type: text/html
Content-language: en

URI: index.html.es
Content-type: text/html
Content-language: es

Mapeo de URL: Redirecciones

- La directiva **Redirect** es utilizada para pedirle al cliente que haga otra petición a una URL diferente.
 - **Permanentes:** El recurso ha sido movido de forma permanente (**301**).
 - **Temporales:** El recurso ha sido “encontrado” pero reside temporalmente en una dirección distinta, es decir, en otra URL (**302**).
- Redirecciones temporales:
 - Redirecciones permanentes:

```
Redirect "/service" "http://www.pagina.com/service"  
Redirect "/one" "/two"
```

```
Redirect permanent "/one" "http://www.pagina2.com/two"  
Redirect 301 "/otro" "/otra"
```

- De forma análoga a AliasMatch, existe la directiva **RedirectMatch** que utiliza también expresiones regulares:

```
RedirectMatch "(.*)\.gif$" "http://www.pagina2.org/$1.jpg"
```

Mapeo de URL: Páginas de errores personalizadas

- Apache ofrece la posibilidad de que los administradores puedan **configurar las respuestas** que muestra el servidor cuando se producen **errores** o problemas.
 - Desplegar un **texto diferente**, en lugar de los que aparecen por defecto.
 - **Redireccionar** a una **URL local**.
 - **Redireccionar** a una **URL externa**.
- La directiva **ErrorDocument** permite configurar la **página de error personalizada** para cada tipo de error:

```
ErrorDocument 403 "Sorry can't allow you access today"
```

```
ErrorDocument 404 /error/404.html
```

```
ErrorDocument 500 http://www.pagina2.org/error.html
```

Mapeo de URL: Páginas de errores personalizadas

- La directiva **ErrorDocument** la podemos utilizar en **varios ámbitos** de nuestra configuración. Por ejemplo, si la ponemos dentro de un vhost, las páginas de errores sólo se verán en ese vhost.
- Si queremos configurar las páginas de error personalizadas podemos hacerlo en un fichero de configuración:
 - **/etc/apache2/conf-available/localized-error-pages.conf**

Cambiando el idioma de las páginas de error personalizadas

- En el directorio **/usr/share/apache2/error** nos encontramos un **fichero tipo mapa** donde se encuentran definidas las páginas de error personalizadas para distintos idiomas.
- Por negociación de contenidos podemos activar la funcionalidad de que **sirva la versión adecuada**, para ello debemos descomentar en el fichero **/etc/apache2/conf-available/localized-error-pages.conf** el siguiente bloque:
 - La directiva **IfModule** exige tener activo los módulos **negotiation**, **alias** e **include**, que hay que activarlo:
 - **a2enmod include**

```
<IfModule mod_negotiation.c>
  <IfModule mod_include.c>
    <IfModule mod_alias.c>

      Alias /error/ "/usr/share/apache2/error/"

      <Directory "/usr/share/apache2/error">
        Options IncludesNoExec
        AddOutputFilter Includes html
        AddHandler type-map var
        Order allow,deny
        Allow from all
        LanguagePriority en cs de es fr it nl sv pt-br ro
        ForceLanguagePriority Prefer Fallback
      </Directory>

      ErrorDocument 400 /error/HTTP_BAD_REQUEST.html.var
      ErrorDocument 401 /error/HTTP_UNAUTHORIZED.html.var
      ErrorDocument 403 /error/HTTP_FORBIDDEN.html.var
      ErrorDocument 404 /error/HTTP_NOT_FOUND.html.var
      ErrorDocument 405 /error/HTTP_METHOD_NOT_ALLOWED.html.var
      ErrorDocument 408 /error/HTTP_REQUEST_TIME_OUT.html.var
      ErrorDocument 410 /error/HTTP_GONE.html.var
      ErrorDocument 411 /error/HTTP_LENGTH_REQUIRED.html.var
      ErrorDocument 412 /error/HTTP_PRECONDITION_FAILED.html.var
      ErrorDocument 413 /error/HTTP_REQUEST_ENTITY_TOO_LARGE.html.var
      ErrorDocument 414 /error/HTTP_REQUEST_URI_TOO_LARGE.html.var
      ErrorDocument 415 /error/HTTP_UNSUPPORTED_MEDIA_TYPE.html.var
      ErrorDocument 500 /error/HTTP_INTERNAL_SERVER_ERROR.html.var
      ErrorDocument 501 /error/HTTP_NOT_IMPLEMENTED.html.var
      ErrorDocument 502 /error/HTTP_BAD_GATEWAY.html.var
      ErrorDocument 503 /error/HTTP_SERVICE_UNAVAILABLE.html.var
      ErrorDocument 506 /error/HTTP_VARIANT_ALSO_VARIES.html.var
    </IfModule>
  </IfModule>
</IfModule>
```

```
<IfModule mod_negotiation.c>
    <IfModule mod_include.c>
        <IfModule mod_alias.c>

            Alias /error/ "/usr/share/apache2/error/"

            <Directory "/usr/share/apache2/error">
                Options IncludesNoExec
                AddOutputFilter Includes html
                AddHandler type-map var
                Order allow,deny
                Allow from all
                LanguagePriority en cs de es fr it nl sv pt-br ro
                ForceLanguagePriority Prefer Fallback
            </Directory>

            ErrorDocument 400 /error/HTTP_BAD_REQUEST.html.var
```

Control de acceso

- El control de acceso hace referencia a todos los medios que proporcionan una forma de **controlar el acceso a cualquier recurso**.
- La directiva **Require** proporciona diferentes maneras de **permitir o denegar el acceso** a los recursos.
- Además puede ser usada junto con las directivas: **RequireAll**, **RequireAny** y **RequireNone**.
- Estos requerimientos pueden ser **combinados de forma compleja y arbitraria** para cumplir cualesquiera que sean las **políticas de acceso**.

Directiva Require

- **Require all granted** → El acceso es permitido incondicionalmente
- **Requiere all denied** → El acceso es denegado incondicionalmente
- **Require user userid [userid] ...** → El acceso es permitido solo si los usuarios indicados se han autenticado.
- **Require valid-user** → El acceso es permitido a los usuarios válidos.
- **Require ip 10 172.20 192168.2** → El acceso es permitido si se hace desde el conjunto de direcciones especificadas.
- **Require host dominio** → El acceso es permitido si se hace desde el dominio especificado.
- **Require local** → El acceso es permitido desde localhost.
- Se puede usar el operador **not** para indicar la denegación: **Require not ip 10.0**

Apache 2.2 ← Versiones anteriores

- En las versiones 2.2 y anteriores de Apache se utilizaban otras directivas para controlar el acceso: **Allow**, **Deny** y **Order**, que están obsoletas y serán quitadas en futuras versiones.

```
<Directory "/var/www">
    Order allow,deny
    Allow from all
</Directory>
```

Más info https://httpd.apache.org/docs/trunk/es/mod/mod_access_compat.html

All requests are denied:

```
Order deny,allow  
Deny from all
```

```
Require all denied
```

All requests are allowed:

```
Order allow,deny  
Allow from all
```

```
Require all granted
```

Only hosts in the example.org domain are allowed access:

```
Order Deny,Allow  
Deny from all  
Allow from example.org
```

```
Require host example.org
```

<https://www.the-art-of-web.com/system/apache-authorization/>

Autenticación básica

- El servidor web apache puede acompañarse de distintos módulos para proporcionar la autenticación.
- La autenticación básica viene de serie mod_auth_basic.
- La configuración que podríamos añadir en el fichero de definición del vhost sería algo así:

```
<Directory "/var/www/pagina1/privado">
    AuthUserFile "/etc/apache2/claves/passwd.txt"
    AuthName "Palabra de paso"
    AuthType Basic
    Require valid-user
</Directory>
```

```
<Directory "/var/www/pagina1/privado">
    AuthUserFile "/etc/apache2/claves/passwd.txt"
    AuthName "Palabra de paso"
    AuthType Basic
    Require valid-user
</Directory>
```

- El método de autentificación básica se indica en la directiva **AuthType**:
 - <http://httpd.apache.org/docs/2.4/es/mod/core.html#authtype>
- **Directory** → El directorio a proteger.
- **AuthUserFile** → Fichero que guardará la información de usuarios y contraseñas, que tendrá que estar en un directorio que no sea visible desde el Apache.
- **AuthName** → Personalizamos el mensaje que aparecerá en la ventana del navegador que nos pedirá la contraseña.
- Para el control de acceso, es decir, qué usuarios tienen permiso para obtener el recurso, utilizamos las directivas **AuthGroupFile**, **Require user**, **Require group**.

Autenticación básica: Fichero de contraseñas

```
$ htpasswd /etc/apache2/claves/passwd.txt carolina
```

New password:

Re-type new password:

Adding password for user carolina

- El fichero de contraseñas se genera mediante la utilidad **htpasswd**.
- Para crear el fichero de contraseñas con la introducción del primer usuario tenemos que añadir la **opción -c**.
- Las contraseñas no se guardan en texto plano, se guarda su **hash**.

```
$ htpasswd /etc/apache2/claves/passwd.txt carolina
```

New password:

Re-type new password:

Adding password for user carolina

Autenticación básica: Fichero de contraseñas

- Si lo que se desea es permitir a un **grupo de usuarios**, se puede crear un **archivo de grupo** que asocie los nombres de los grupos con los usuarios, para permitirles el acceso.
- El contenido de este fichero será:

```
NombreGrupo: usuario1 usuario2 usuario3
```

- La directiva que se tendría que usar es **AuthGroupFile** y para permitir el acceso a grupos se utilizaría:

```
Require group NombreGrupo
```

Autenticación digest

- Es un tipo de **autenticación** que soluciona el **problema de la transferencia de contraseñas en claro sin necesidad de usar SSL**.
- El procedimiento es similar a la autenticación básica pero cambiando algunas de las directivas y utilizando la utilidad **htdigest** en lugar de **htpassword** para crear el fichero de contraseñas.
- Más información: <https://plataforma.josedomingo.org/pledin/cursos/apache24/curso/u17/>

Políticas de acceso en Apache 2.2

- El control de acceso se hacía con las directivas **Order**, **Allow** y **Deny**.
- Además existe otra directiva **Satisfy**
 - **Satisfy All** → Se tienen que cumplir todas las condiciones para obtener el acceso.
 - **Satisfy Any** → Basta con que se cumpla una de las condiciones.
- El uso de este tipo de directivas hacía muy complicado crear políticas de acceso complejas.

```
<Directory /dashboard>
    Order deny,allow
    Deny from all
    Allow from 10.1
    Require group admins
    Satisfy any
</Directory>
```

Fichero .htaccess

- **.htaccess (hypertext access)**
- Es un archivo de **configuración distribuida** popularizado por Apache.
- Nos permite **definir distintas directivas de configuración para cada directorio** sin necesidad de editar el archivo de configuración principal de Apache.
 - A veces **no tendremos acceso al archivo de configuración principal** de Apache.
 - Permite **deshabilitar listar ficheros en un directorio**, proteger el directorio con **autenticación básica**, hacer que **algunos archivos no estén accesibles**, crear una **lista de IPs prohibidas...**

Fichero .htaccess

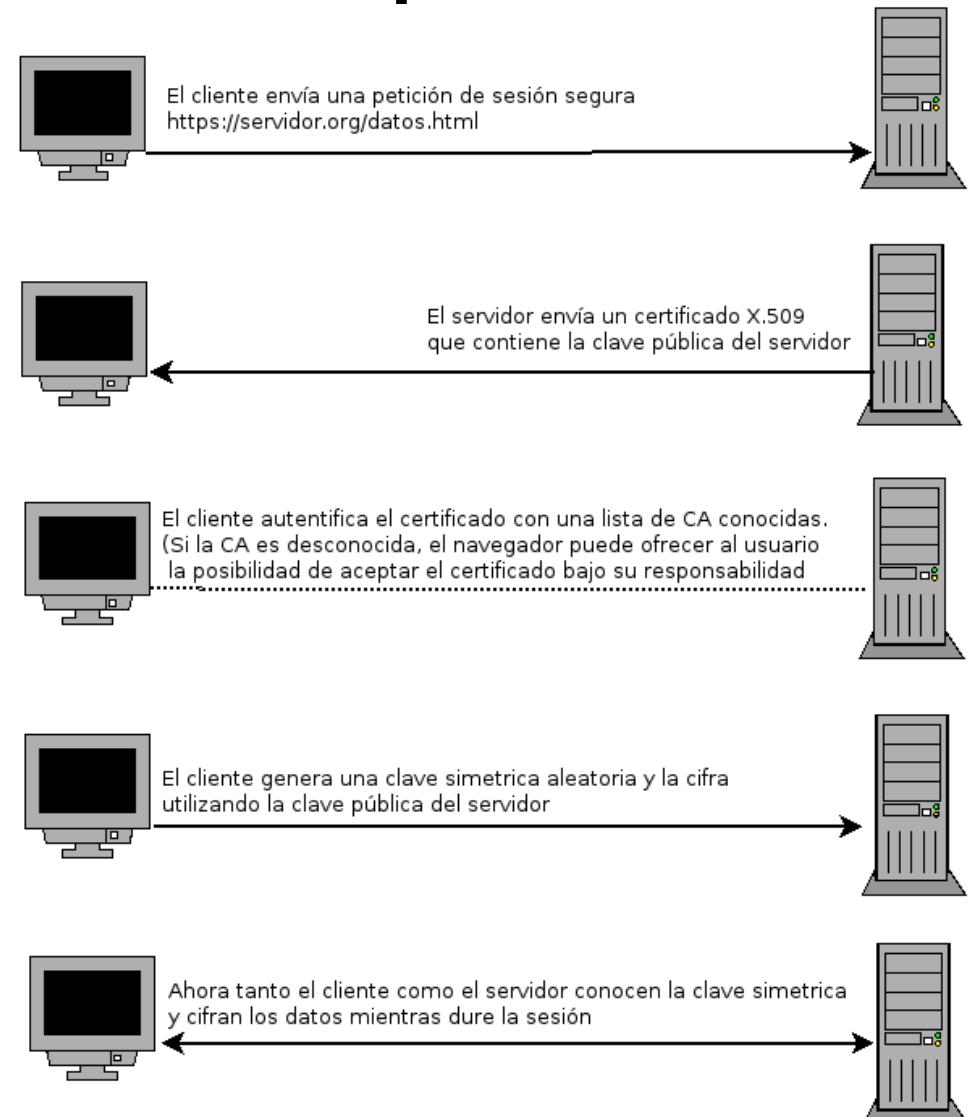
- Para permitir el uso de ficheros .htaccess o restringir las directivas que se pueden aplicar en él, utilizamos la directiva **AllowOverride** que irá acompañada de una o varias opciones:
 - **All** → Se pueden usar todas las directivas permitidas.
 - **None** → Se ignora el fichero .htaccess (valor por defecto).
 - **AuthConfig** → Directivas de **autenticación y autorización**: AuthName, AuthType, AuthUserFile, Require...
 - **FileInfo** → Directivas relacionadas con el **mapeo de URL**: Redirecciones, módulo rewrite, ...
 - **Indexes** → Directiva que controlan la **visualización de listado de ficheros**.
 - **Limit** → Directivas para controlar el **control de acceso**: Allow, Deny y Order

Fichero .htaccess

- Para permitir el uso de ficheros .htaccess o restringir las directivas que se pueden aplicar en él, utilizamos la directiva **AllowOverride** que irá acompañada de una o varias opciones:
 - **All** → Se pueden usar todas las directivas permitidas.
 - **None** → Se ignora el fichero .htaccess (valor por defecto).
 - **AuthConfig** → Directivas de **autenticación y autorización**: AuthName, AuthType, AuthUserFile, Require...
 - **FileInfo** → Directivas relacionadas con el **mapeo de URL**: Redirecciones, módulo rewrite, ...
 - **Indexes** → Directiva que controlan la **visualización de listado de ficheros**.
 - **Limit** → Directivas para controlar el **control de acceso**: Allow, Deny y Order

Introducción a HTTPS en Apache

- Utiliza el protocolo **SSL** (actualmente **TLS**) para el cifrado de datos.
- El servidor utiliza por defecto el puerto **443/tcp**.
- Utiliza mecanismos de cifrado de clave pública y las claves públicas se denominan certificados.
- El formato de los certificados está especificado por el estándar **X.509** y normalmente son emitidos por una entidad denominada **Autoridad Certificadora** (CA por sus siglas en inglés).
- En el caso de HTTPS, **la función principal de la CA es demostrar la autenticidad del servidor** y que pertenece legítimamente a la persona u organización que lo utiliza. Dependiendo de los criterios utilizados para comprobar la autenticidad del servidor se emiten diferentes tipos de certificados X.509 (actualmente se usa el llamado Extended Validation Certificate).
- **El navegador contiene una lista de certificados de CA en las que confía** y acepta inicialmente sólo los certificados de los servidores emitidos por alguna de estas CA.
- Una vez **aceptado el certificado** de un servidor web, el **navegador utiliza éste para cifrar los datos** que quiere enviar al servidor mediante el protocolo **HTTPS** y cuando llegan al servidor sólo **éste podrá descifrarlos** ya que es el único que **posee la clave privada** que los descifra.



Certificado X.509 para nuestro servidor

- Tenemos que conseguir un **certificado X.509** para nuestro servidor.
- Hay **CA** que proporcionan **certificados más robustos**, con un nivel más elevado de **confianza**, pero normalmente **no son gratuitos**.
- Para obtener un certificado gratuito podemos utilizar **CAcert** o **Let's Encrypt**.
- Obtener certificado con **CAcert**:
 - <https://plataforma.josedomingo.org/pledin/cursos/apache24/curso/u28/index.html>
- Obtener certificado con **Let's Encrypt**:
 - <https://letsencrypt.org/es/docs/certificates-for-localhost/>
- Pero vamos a tener problemas, ya que no podemos crear un certificado para “**localhost**” y no disponemos de un dominio como por ejemplo **miweb.com**.
- La mejor opción para **desarrollo local** consiste en **generar nuestro propio certificado autofirmado**.

Certificado X.509 para nuestro servidor

- Cualquiera puede crear su **propio certificado** sin ayuda de una **AC**.
- La única diferencia es que **sólo tú vas a confiar en tus certificados**, y nadie más, pero para desarrollo o pruebas es suficiente.
- La manera más sencilla para generar una **llave privada** y un **certificado autofirmado para localhost**.
- A continuación se puede **configurar el servidor con localhost.crt** y **localhost.key** e instalar **localhost.crt** en tu lista de **raíces confiadas localmente**.
- **Tutorial:** <https://www.youtube.com/watch?v=1daMCJeh5yM>
- Probablemente también sea interesante **crear redirecciones de http a https**:
 - En la configuración del **vhost** podemos incluir un **redirect**:
 - **redirect permanent / https://tudominio.com**

Políticas de acceso en Apache 2.4

- El **control de acceso** se determina con la directiva **Require**.
- Las **políticas de acceso** las podemos indicar usando las directivas:
 - **RequireAll** → Todas las condiciones dentro del bloque se deben cumplir para obtener acceso.
 - **RequireAny** → Al menos una de las condiciones en el bloque se debe cumplir.
 - **RequireNone** → Ninguna de las condiciones se deben cumplir para permitirse el acceso.

```
<Directory /dashboard>
  <RequireAny>
    Require ip 10.1
    Require group admins
  </RequireAny>
</Directory>
```

Políticas de acceso en Apache 2.2 vs Apache 2.4

Apache 2.2

```
Order Allow,Deny
```

```
Allow from all
```

```
Deny from 212.100.100.100
```

Apache 2.4

```
<RequireNone>
    Require ip 212.100.100.100
</RequireNone>
```

o bien:

```
<RequireAll>
    Require all granted
    Require not ip 212.100.100.100
</RequireAll>
```

Políticas de acceso: Ejemplo complejo:

```
<RequireAny>
  <RequireAll>
    Require user root
    Require ip 123.123.123.123
  </RequireAll>
  <RequireAll>
    <RequireAny>
      Require group sysadmins
      Require group useraccounts
      Require user anthony
    </RequireAny>
    <RequireNone>
      Require group restrictedadmin
      Require host bad.host.com
    </RequireNone>
  </RequireAll>
</RequireAny>
```

MÁS INFORMACIÓN

Más información:

<https://gorkamudev.wordpress.com/tag/rtt/>

<https://blog.cloudflare.com/http3-the-past-present-and-future/>

<https://blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2/>

<https://developer.mozilla.org/es/docs/Web/>

<https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/cabecera-http/>

<https://uniwebsidad.com/tutoriales/los-codigos-de-estado-de-http?from=librosweb>

<https://medium.com/platform-engineer/evolution-of-HTTP-69cfe6531ba0#:~:text=HTTP%20has%20four%20versions%20%E2%80%94%20HTTP,future%20will%20be%20HTTP%2F2.0.>

<https://plataforma.josedomingo.org/pledin/cursos/apache24/curso/u14/index.html>