Protocolo HTTP

Servidores web, proxy y balanceadores

📧 José Domingo Muñoz

🏫 IES Gonzalo Nazareno · Dos Hermanas

📚 SRI · Servicios de Red e Internet

01

El protocolo HTTP

Concepto, mensajes y métodos

¿Qué es HTTP?

> " ***HyperText Transfer Protocol*** *es el protocolo de la* ***capa de aplicación*** *que permite la comunicación entre clientes, servidores y proxies en la web.*

CARACTERÍSTICAS FUNDAMENTALES

- Última versión ampliamente desplegada: **HTTP/1.1** (con HTTP/2 y HTTP/3 en uso creciente)
- Basado en el esquema **petición / respuesta**: el cliente pide, el servidor responde
- Mensajes en **texto plano**, legibles y depurables
- Protocolo **sin estado**: el servidor no recuerda quién hizo cada petición

> ℹ️ La falta de estado se compensa más tarde con **cookies** y **sesiones**.

HTTP sobre TCP

HTTP usa **TCP** como transporte (puerto **80** por defecto, **443** en HTTPS). Cada conversación se apoya en una conexión TCP previamente establecida.

CONEXIONES PERSISTENTES (KEEP-ALIVE)

- En **HTTP/1.0** se abría una conexión TCP **por cada recurso**
- En **HTTP/1.1** una misma conexión sirve **varias peticiones y respuestas**
- Reduce la latencia y el coste de los *handshakes* TCP
- Mejora notable en páginas con muchos recursos (CSS, JS, imágenes…)

HTTP en la pila TCP/IP

CAPA	PROTOCOLO
Aplicación	**HTTP**, HTTPS, FTP, SMTP…
Transporte	**TCP** (puerto 80/443)
Red	IP
Enlace	Ethernet, Wi-Fi…

ℹ️ HTTPS es **HTTP sobre TLS**: cifra el contenido pero conserva la misma semántica de peticiones y respuestas.

Estructura de los mensajes

PETICIÓN

- **Línea inicial**: método, URL del recurso y versión
- **Cabeceras** de petición (clave-valor)
- **Cuerpo** opcional (típico en `POST` y `PUT`)

```
GET /index.html HTTP/1.1
Host: www.ejemplo.com
User-Agent: curl/8.5
```

RESPUESTA

- **Línea de estado**: versión, código y descripción
- **Cabeceras** de respuesta
- **Cuerpo** con el recurso solicitado

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
```

Métodos de petición

- **GET** — Solicita un recurso. Los datos viajan en la URL (*query string*)
- **HEAD** — Igual que `GET` pero pide **sólo las cabeceras**
- **POST** — Envía datos al servidor en el **cuerpo** del mensaje
- **PUT** — Almacena el documento enviado en la URL indicada
- **DELETE** — Elimina el recurso referenciado por la URL
- **OPTIONS**, **PATCH**, **TRACE**, **CONNECT** — usos más específicos

> ℹ️ El protocolo no impone que un método sea seguro: lo es la **aplicación** que lo implementa. Una petición `GET` nunca debería modificar estado.

Códigos de estado

Cifra de tres dígitos que clasifica la respuesta. La primera cifra indica el grupo:

CÓDIGO	FAMILIA	SIGNIFICADO
1xx	Informativo	Información intermedia
2xx	Éxito	La petición se procesó correctamente (`200 OK`, `201 Created`)
3xx	Redirección	El recurso está en otra URL (`301`, `302`, `304`)
4xx	Error del cliente	Petición incorrecta (`400`, `401`, `403`, `404`)
5xx	Error del servidor	Fallo en el servidor (`500`, `502`, `503`)

Cabeceras HTTP

Información de la forma **clave: valor** asociada a peticiones y respuestas.

GENÉRICAS Y DE PETICIÓN

- `Host` — dominio solicitado
- `User-Agent` — cliente que hace la petición
- `Accept` — tipos MIME aceptados
- `Accept-Language` — idiomas
- `Cookie` — cookies enviadas
- `Authorization` — credenciales

DE RESPUESTA

- `Content-Type` — tipo MIME del cuerpo
- `Content-Length` — tamaño en bytes
- `Server` — software del servidor
- `Set-Cookie` — define una cookie
- `Location` — URL destino en redirecciones
- `Cache-Control` — directivas de caché

02

Funcionalidades del servidor web

Virtual Hosts, redirecciones, autenticación y cookies

Virtual Hosts

Un **Virtual Host** permite que un mismo servidor web atienda **varios sitios web** desde una **única dirección IP**.

CARACTERÍSTICAS

- Aloja **múltiples dominios** en el mismo servidor
- Cada sitio tiene su propio contenido, configuración y registros de acceso
- Es una técnica fundamental en **Apache** y **Nginx**
- Tipo más habitual: **basado en nombre** — el servidor distingue el sitio por la cabecera `Host` de la petición

> ℹ️ Sin la cabecera `Host` el servidor no sabría a qué sitio dirigir la petición: por eso es **obligatoria** en HTTP/1.1.

Alias

Un **alias** asocia una URL a una **ruta diferente** dentro del servidor, sin mover los archivos físicamente.

```
Alias /imagenes/ "/var/www/recursos/img/"

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

- Una petición a `http://servidor/imagenes/logo.png` accede realmente a `/var/www/recursos/img/logo.png`
- Útil para **reorganizar** rutas o **proteger** la ubicación real de los recursos

Redirecciones

Cuando un recurso ha **cambiado de ubicación**, el servidor indica al cliente la nueva dirección con un código **3xx**.

CÓDIGOS TÍPICOS

- **301 Moved Permanently** — cambio definitivo, cacheable
- **302 Found** — cambio temporal
- **307 / 308** — variantes que conservan el método

CÓMO FUNCIONA

- El servidor responde con el código y la cabecera `Location:`
- El cliente realiza una **nueva petición** a esa URL
- Las herramientas como `curl` necesitan `-L` para seguir la redirección

Autenticación

Mecanismo para que el cliente se identifique antes de obtener un recurso.

BÁSICA (*BASIC*)

- Credenciales codificadas en **Base64**
- Cabecera: `Authorization: Basic dXNlcjpwYXNz`
- **No cifra**: sólo segura sobre **HTTPS**

RESUMEN (*DIGEST*)

- Las credenciales viajan como **hash**
- Cabecera: `Authorization: Digest username="...", response="..."`
- Más segura que Basic en HTTP plano

Control de acceso

Restringe quién puede acceder a determinados recursos o directorios. Puede combinar varios criterios:

- **Dirección IP** o red de origen
- **Nombre de usuario y contraseña**
- **Nombres de dominio** del cliente
- **Cabeceras** HTTP o autenticación previa

> ℹ️ En Apache se configura con directivas como `Require ip`, `Require valid-user` o `Require host`; en Nginx con bloques `allow` / `deny` y `auth_basic`.

Cookies

> *Pequeños fragmentos de información que el **navegador** guarda a petición del **servidor** mediante la cabecera `Set-Cookie`.*

¿PARA QUÉ SIRVEN?

- Guardar información de **sesión**
- **Comercio electrónico**: carrito de la compra
- **Personalización**: idioma, tema, preferencias
- **Seguimiento** de visitas y publicidad
- Recordar **login** y contraseña

ℹ️ Las cookies permiten **mantener estado** entre peticiones que, por diseño, son independientes.

Sesiones

HTTP es **stateless**: las sesiones añaden la noción de estado a las aplicaciones web.

LO QUE GUARDA EL SERVIDOR

- **Identificador** de sesión
- **Usuario** asociado
- **Tiempo de expiración**
- Datos temporales de la aplicación

LO QUE GUARDA EL CLIENTE

- Sólo el **identificador** de sesión
- Habitualmente en una **cookie**
- En cada petición lo envía al servidor
- El servidor recupera el resto de su lado

03

Servidores web

Apache, Nginx y comparativa

¿Qué es un servidor web?

- Programa que **implementa el protocolo HTTP**
- Sirve **páginas estáticas** (HTML, CSS, JS, imágenes...)
- Sirve **páginas dinámicas** generadas por lenguajes como **PHP**, **Python**, **Java**...
- Suele apoyarse en un **servidor de aplicaciones** para ejecutar el código
- Implementa funcionalidades del protocolo: **redirecciones, autenticación, negociación de contenido...**

ℹ️ En la práctica, muchas de esas funcionalidades se acaban implementando en la propia **aplicación web**.

Apache HTTP Server

- Servidor web **HTTP de código abierto**, multiplataforma
- Implementa **HTTP/1.1** y soporta **HTTP/2** mediante módulos
- Desarrollado por la **Apache Software Foundation** (proyecto *httpd*)
- Surgió en 1995 a partir del servidor **NCSA HTTPd**
- **Servidor web más usado en Internet entre 1996 y 2020**
- Destaca por su **estabilidad**, **modularidad** y comunidad
- Integración sencilla con **PHP, Python, Perl, CGI**...
- Versión estable actual: **2.4** (la 2.5 en desarrollo)

Nginx

- Servidor **web y proxy inverso** ligero, de **alto rendimiento**
- Actúa también como **proxy de correo** (IMAP/POP3)
- **Software libre** (BSD simplificada); existe versión comercial **Nginx Plus**
- Creado por **Igor Sysoev** en 2004 para el portal ruso *Rambler*
- Diseñado para **superar a Apache** en archivos estáticos y muchas conexiones simultáneas
- Usa **menos memoria** y procesa más peticiones por segundo
- **Menos flexible** en configuraciones dinámicas (no hay `.htaccess`)
- Menos módulos integrados, pero **mejor escalabilidad**

Comparativa Apache vs Nginx

APACHE

- Modelo basado en **procesos / hilos**
- Configuración por **directorio** (`.htaccess`)
- Enorme catálogo de **módulos**
- Ideal para entornos con **PHP** y configuración heterogénea

NGINX

- Modelo **asíncrono** orientado a eventos
- Configuración **centralizada**
- Mejor rendimiento sirviendo **estáticos**
- Excelente como **proxy inverso** y balanceador

04

Proxy, proxy inverso y balanceador

Intermediarios HTTP

Proxy

Servidor que **intermedia las peticiones** entre los clientes de una red y los servidores de Internet.

¿PARA QUÉ SE USA?

- Cuando los equipos de la red **no acceden directamente a Internet**
- **Filtrar** y controlar el tráfico HTTP: por dominio, URL, contenido, horario...
- Hacer de **caché** de respuestas para acelerar accesos posteriores
- Mejorar la **seguridad**, anonimato y rendimiento

> ℹ️ Software clásico: **Squid**.

Proxy inverso

Recibe las peticiones de los clientes y las **redirige a uno o varios servidores internos**. El cliente no accede directamente a los servidores backend.

USOS HABITUALES

- **Proteger** los servidores internos (ocultando su ubicación)
- **Distribuir carga** entre varios backend
- **Caché** de contenidos estáticos o respuestas frecuentes
- Centralizar **TLS/SSL**, compresión y autenticación

> ℹ️ Ejemplos: **Apache**, **Nginx**, **Varnish**, **Traefik**.

Balanceador de carga

Distribuye las peticiones entrantes entre **varios servidores backend** para optimizar el rendimiento, la disponibilidad y la tolerancia a fallos.

CARACTERÍSTICAS

- Puede implementarse por **hardware** o **software**
- Reparte la carga según un **algoritmo de distribución**
- Evita la **sobrecarga** de un único servidor
- Permite ofrecer servicio aunque caiga uno de los nodos

ℹ️ Ejemplos: **Nginx**, **HAProxy**, **Apache mod_proxy_balancer**.

Algoritmos de balanceo — estáticos

ROUND ROBIN

Las peticiones se reparten **secuencialmente** entre los servidores. Requiere instancias equivalentes y sin estado.

STICKY ROUND ROBIN

Variante con **afinidad**: un mismo cliente se dirige siempre al mismo servidor.

ROUND ROBIN PONDERADO

Cada servidor tiene un **peso**: los más potentes reciben más peticiones.

HASH

Se aplica una función *hash* (sobre IP, URL...) que **determina** qué servidor atiende cada petición.

Algoritmos de balanceo — dinámicos

CONEXIONES MÍNIMAS

La nueva petición se envía al servidor con **menos conexiones activas** en ese momento.

MENOR TIEMPO DE RESPUESTA

Se elige el servidor con el **tiempo de respuesta más bajo** según mediciones recientes.

ℹ️ Los algoritmos dinámicos se adaptan al estado real del sistema, pero requieren más **monitorización**.

05

Ejercicio práctico

Peticiones HTTP con `curl`

La herramienta `curl`

`curl` es una herramienta de línea de comandos para hacer peticiones HTTP (y muchos otros protocolos). Permite **ver y depurar** el tráfico HTTP sin un navegador.

OPCIONES BÁSICAS

OPCIÓN	PARA QUÉ SIRVE
`curl <url>`	Petición **GET**
`curl -L <url>`	Sigue **redirecciones**
`curl -I <url>`	Petición **HEAD** (sólo cabeceras)
`curl -X POST -d "campo=valor" <url>`	Petición **POST** con datos en el cuerpo
`curl -v <url>`	Modo *verbose*: muestra cabeceras enviadas y recibidas

Ejercicios

1. Realiza una petición para ver las **cabeceras** de `https://dit.gonzalonazareno.org`. ¿Qué **código de estado** devuelve? ¿Qué significa? ¿En qué cabecera se encuentra la URL a la que hay que acceder para obtener el recurso?
2. Realiza una petición **GET** a `https://dit.gonzalonazareno.org`. ¿Qué tipo de **redirección** devuelve? Realiza una nueva petición que **siga** la redirección.
3. Con las **herramientas para desarrolladores** del navegador (en Firefox: *Herramientas para desarrolladores* → *Red*), inspecciona `https://dit.gonzalonazareno.org/gestiona/`. ¿Cuántas peticiones se han realizado para mostrar la página? Identifica las cabeceras más importantes.

Ejercicios (continuación)

4. Obtén el **cuerpo** de la respuesta de `https://dit.gonzalonazareno.org/gestiona/`.
5. Usando el método **GET**, manda tu nombre a la página `https://http.josedomingo.org/index2.php`.
6. Usando el método **POST** (que envía el contenido en el cuerpo), manda tu nombre a la misma página.

> ℹ️ Compara las dos últimas peticiones: el contenido viaja en lugares **distintos** y la página debería distinguirlas.

Pistas para resolverlos

```
# 1 — Cabeceras (HEAD)
curl -I https://dit.gonzalonazareno.org

# 2 — GET y seguir redirección
curl https://dit.gonzalonazareno.org
curl -L https://dit.gonzalonazareno.org

# 4 — Cuerpo de la respuesta
curl https://dit.gonzalonazareno.org/gestiona/

# 5 — GET con parámetros en la URL
curl "https://http.josedomingo.org/index2.php?nombre=Pepe"

# 6 — POST con datos en el cuerpo
curl -X POST -d "nombre=Pepe" https://http.josedomingo.org/index2.php
```

SRI · UNIDAD 3 — PROTOCOLO HTTP Y SERVIDORES WEB

¡Gracias!

HTTP → Apache y Nginx en la práctica

📧 José Domingo Muñoz 🏫 IES Gonzalo Nazareno · Dos Hermanas 📚 SRI