

Curso: “Desarrollo en Entorno Servidor con Java”

(IFC05CM15)



1. CONCEPTOS PREVIOS

1.1. ARQUITECTURA CLIENTE/SERVIDOR

- CONCEPTO, ELEMENTOS Y NIVELES
- TIPOS DE DESARROLLO WEB
- SERVIDORES DE APLICACIONES

1.2. ENTORNO DE TRABAJO

- MÁQUINA VIRTUAL DE WINDOWS 7 CON VIRTUAL BOX
- CREAR UNIDAD DE RED (MÁQUINA REAL-MÁQUINA VIRTUAL)

1.3. PROTOCOLO HTTP

- CARACTERÍSTICAS
- MENSAJES HTTP: PETICIÓN Y RESPUESTA
- CABECERAS REQUEST
- CABECERAS RESPONSE
- CÓDIGOS DE ESTADO
- HERRAMIENTAS PARA TESTEAR PÁGINAS WEB

1.4. INSTALACIÓN DE JDK 8 (Java Development Kit)

Autores:

José Ramón Rodríguez Sánchez

Juan Manuel Castro Ramos

1 – 10 de Julio, 2015

1.1. ARQUITECTURA CLIENTE/SERVIDOR

• CONCEPTO. ELEMENTOS. NIVELES

La **arquitectura cliente-servidor** es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos (servidores), y los demandantes (clientes). Se basa en la transferencia de archivos HTML y otros asociados (imágenes, sonido, css, JavaScript, etc.) entre ambos lados, mediante el protocolo HTTP.

Desde el punto de vista funcional, se puede definir como una arquitectura distribuida que permite a los usuarios finales, obtener acceso a la información de forma transparente, a pesar de las múltiples plataformas involucradas.

Características:

* Servicio de página web (HTML y asociados)

Es la unidad básica del modelo, el servidor lo proporciona y el cliente lo consume

* Recursos ofrecidos

Documentos HTML, CSS, JavaScript, imágenes, sonidos, etc.

* HTTP es un protocolo asimétrico

Los clientes realizan “peticiones” y los servidores esperan pasivamente y responden a cada petición.

* Transparencia

El cliente no necesita saber donde está físicamente situado el servidor ni el recurso que desea utilizar.

* Sistemas débilmente acoplados

La interacción entre cliente y servidor, está basada únicamente en el envío de mensajes mediante protocolo HTTP, lo que permite interactuar a diferentes clientes en diferentes plataformas con distintos servidores en distintas plataformas.

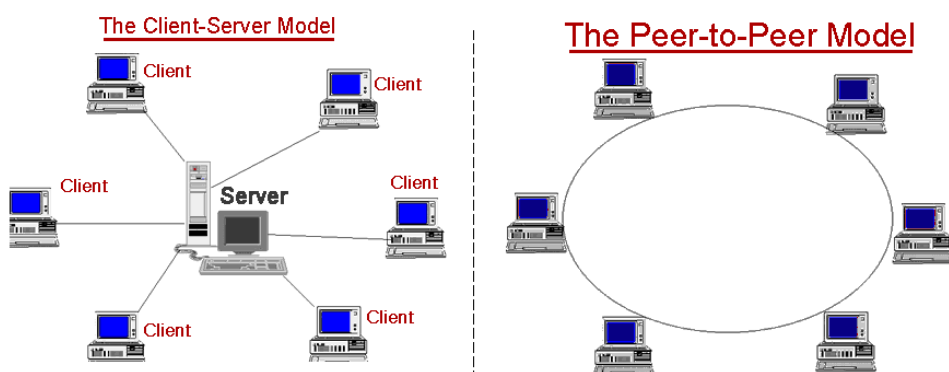


Figura 1.1: Modelos distribuidos. Cliente/Servidor vs. Peer-to-Peer

En el modelo cliente/servidor cada equipo tiene un rol diferente.

En el modelo Peer-to-Peer (p2p) cada equipo es a la vez cliente y servidor (datos)

Cliente.

Programa que solicita el establecimiento de las conexiones con el servidor. Una vez aceptada, envía una petición al servidor y espera la respuesta. Los clientes no comparten sus recursos. El tiempo de vida de la conexión es determinado y termina una vez que son atendidas sus peticiones.

Funciones:

- Interactuar con el usuario
- Realizar validaciones locales según la lógica de negocio
- Generar peticiones de información al servidor
- Recibir resultados del servidor
- Presentar los resultados recibidos con cierto formato (renderización)

Servidor.

Es un programa que comparte los recursos y servicios que se pueden obtener de una red. Acepta peticiones del cliente, realiza el procesamiento y devuelve el resultado al cliente. Debe soportar las peticiones de muchos clientes y sus fallos son críticos porque invalidan el servicio. El servidor comienza su ejecución antes de comenzar la interacción con el cliente y se mantiene después de terminarla (daemon).

Funciones:

- Aceptar peticiones de datos que hacen múltiples clientes
- Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos
- Realizar consultas a la bases de datos
- Generar una respuesta con formato
- Trasmitir los datos a los clientes

Ventajas del modelo cliente/servidor:

- Facilidad de mantenimiento y actualización de software del lado servidor que beneficia a múltiples clientes
- Aumenta la seguridad e integridad de los datos ya que se encuentran físicamente ubicados en los servidores, detrás de cortafuegos, IDS, etc. con personal dedicado
- Facilita la integración entre sistemas diferentes, ya que se pueden utilizar equipos con sistemas operativos distintos
- Las aplicaciones basadas en cliente/servidor son fácilmente escalables.
 - Escalado horizontal: El incremento del número de clientes es fácilmente absorbido mediante replicación de servidores (cluster de servidores)
 - Escalado vertical: Los servidores pueden descomponerse en grupos según la función que realizan aumentando el rendimiento global (función solicitud/respuesta web, función ejecución de aplicaciones y función servidor de datos)

Desventajas:

- Saturación del servidor por un elevado número de peticiones o ataques DDOS
- Paradas de funcionamiento del servidor por tareas de mantenimiento

Capas de la arquitectura Cliente/Servidor

1. Capa de presentación (interfaz de usuario)
Interacciona con el usuario, presenta los datos y recibe las entradas
2. Capa de aplicación/negocio (lógica de aplicación)
Implementa la lógica de la aplicación y aplica las reglas de negocio sobre los datos y las entradas de usuario. Se comunica con las otras capas.
3. Capa de datos (almacenamiento y acceso a datos)
Responsable de la gestión y almacenamiento permanente de los datos

Tipos de arquitecturas cliente/servidor por niveles

Dependiendo de si la capa de aplicación se implementa en el cliente, en un servidor o en varios servidores, tendremos una arquitectura cliente/servidor de 2, 3 o n niveles.

- Arquitectura de 2 niveles: cliente pesado y servidor ligero.

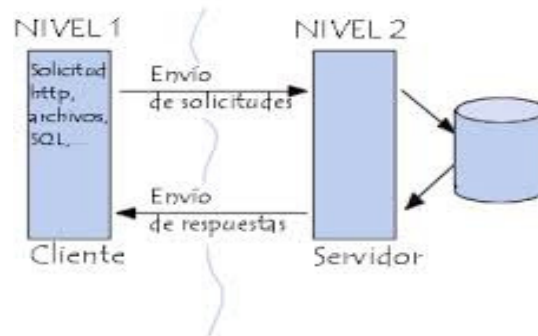


Figura 1.2: Arquitectura cliente/servidor de 2 niveles

El programa cliente se llama pesado cuando asume la mayor parte de la lógica de la aplicación. Ejemplos: cajero automático, terminal interactivo de negocio, juegos en red. En el caso de un cliente Terminal interactivo, tendría el siguiente reparto de funciones: Cliente.

Identificar cada producto mediante código de barras, calcular el importe total de la factura, calcular y desglosar los impuestos, aplicar ofertas y descuentos, emitir el ticket de compra, abrir y cerrar la caja, ejecutar transacciones bancarias con tarjeta de crédito Servidor.

Almacenar las ventas realizadas, descontar los productos comprados del stock de la tienda, proporcionar los precios de cada producto.

Ventajas:

Un cliente pesado aprovecha la capacidad de cálculo del equipo cliente, hoy en día los PC son potentes y el navegador infrutiliza su capacidad.

Un cliente pesado puede tener una interfaz compleja, y muy rica como en el caso de los juegos en red, que mejora mucho la experiencia del cliente frente a un navegador web.

Desventajas:

Un cliente pesado se tiene que instalar en todas las máquinas cliente, y actualizarse en todas ellas cada vez que haya una nueva versión del servidor (acoplamiento alto).

- Arquitectura de 3 niveles: cliente ligero, servidor de aplicaciones y servidor de datos

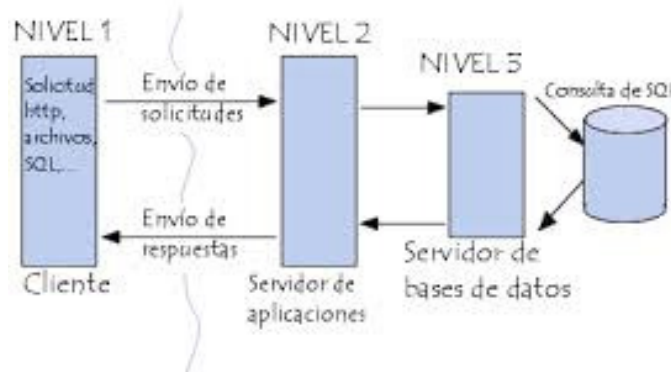


Figura 1.3: Arquitectura cliente/servidor de 3 niveles

Un cliente ligero prácticamente queda reducido a la función de presentación de los datos al usuario, se trata del típico navegador web. Puede implementar algunas reglas de negocio sobre los datos, por ejemplo, utilizando JavaScript para validación de formularios de entrada.

El servidor de aplicaciones, que implementa toda la lógica de la aplicación, es decir, la funcionalidad, todo lo que hace la aplicación. Recibe peticiones del cliente, traduce la petición en una consulta al servidor de datos, recibe el resultado, le da formato y lo envía al cliente.

El servidor de datos, se encarga básicamente del acceso a los datos y su almacenamiento permanente.

Esta es la arquitectura de las aplicaciones web que vamos a estudiar en este curso.

- Arquitectura multinivel: cliente ligero, servidores de aplicaciones por funcionalidad y servidor de datos

Las capas de negocio y de datos se pueden subdividir a su vez en subcapas como ocurre en un ERP.

- **TIPOS DE DESARROLLO WEB**

- **Desarrollo web de lado cliente**

El cliente web (navegador) es el encargado de “ejecutar” las aplicaciones en la máquina del usuario.

El servidor web (Apache) envía un documento HTML y el código de las aplicaciones al cliente y éste, mediante el navegador, las ejecuta.

Es necesario, por tanto, que el navegador tenga capacidad para interpretar ciertos lenguajes como JavaScript, Java Applets, ActionScript, etc.

Para que el navegador tenga dicha capacidad es preciso agregar los correspondientes plug-in.

Funcionamiento:

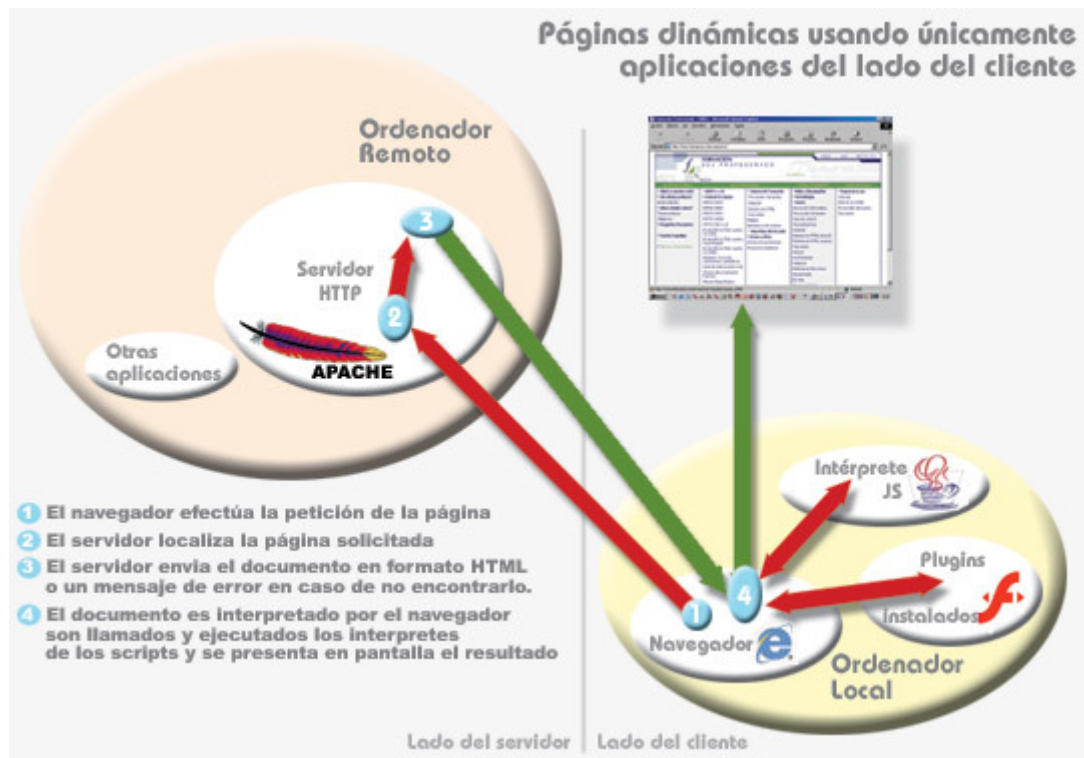


Figura 1.4: Desarrollo de lado cliente (Imagen de www.adelat.org)

- Desarrollo web de lado servidor

El servidor de aplicaciones ejecuta la aplicación en la máquina servidor.

Una vez ejecutada, genera un código HTML (puede que enriquecido con lenguajes de lado cliente).

Por último, envía dicho código al cliente mediante el protocolo HTTP.

El desarrollo de lado servidor suponen una gran ventaja, ya que al ejecutarse en la máquina servidor, el cliente no necesita ninguna capacidad adicional, además el desarrollador tiene más control sobre la ejecución.

Los lenguajes de lado servidor más habituales son PHP, .NET y JavaEE.

El servidor nunca envía las aplicaciones escritas en estos lenguajes al cliente, sólo el resultado de su ejecución, que es HTML (enriquecido).

Funcionamiento:

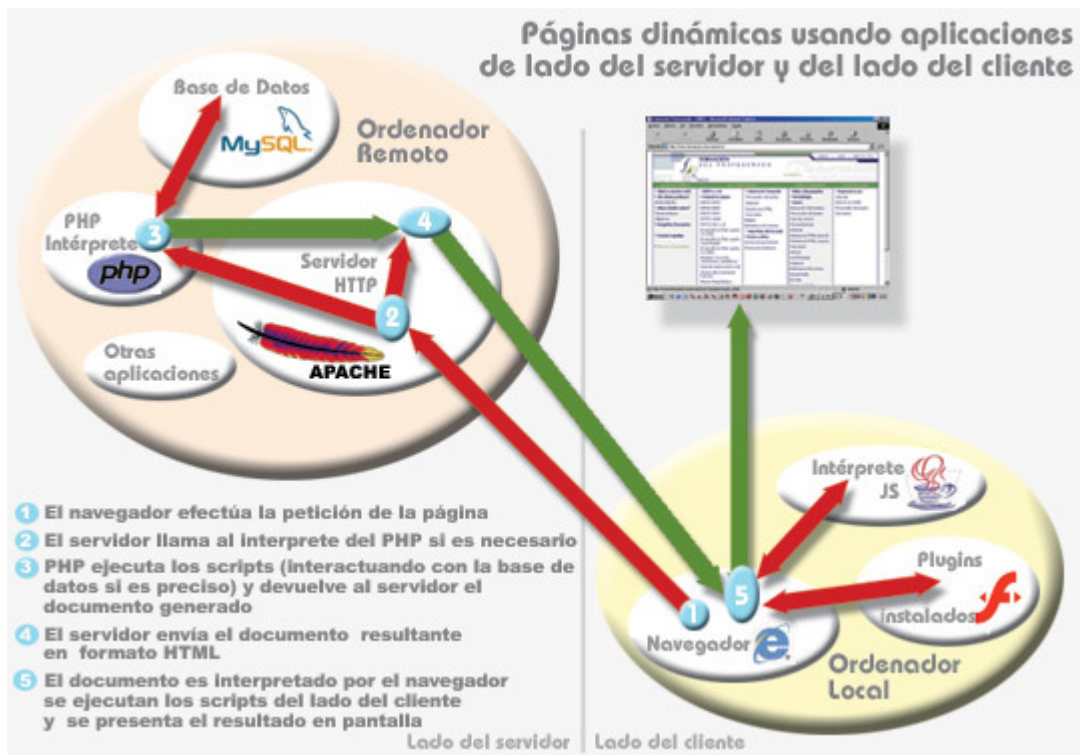


Figura 1.5: Desarrollo de lado servidor (Imagen de www.adelat.org)

• SERVIDOR DE APLICACIONES

Se trata de una aplicación que tiene la capacidad de interpretar lenguajes de lado servidor como JavaEE (no se suele aplicar este concepto a PHP o .NET aunque sería equivalente), soporta los EJB (Enterprise Java Beans) que son clases especiales para gestionar la lógica de negocio, permite la conectividad con otros servidores de aplicaciones para el funcionamiento de los servicios web, y además incorpora un servidor web para enviar los documentos HTML al cliente.

Los servidores de aplicaciones más conocidos de software propietario son:

- WebSphere de IBM
- WebLogic de Oracle

Los servidores de aplicaciones más conocidos de software libre son:

- TomEE de Apache Foundation (56 MB)
- JBoss de Red Hat (151 MB)
- GlassFish de Oracle (126 MB)

En muchas ocasiones no es necesario instalar un servidor de aplicaciones. Si nuestra aplicación no necesita una lógica de negocio muy compleja, basta con disponer de un contenedor de servlets.

Los contenedores de Servlets más conocidos de software libre son:

- Tomcat de Apache Foundation (9 MB)
- Jetty de Eclipse Foundation (12 MB)

Estadísticas de uso mundial de servidores de aplicaciones (tamaño de muestra= 623)

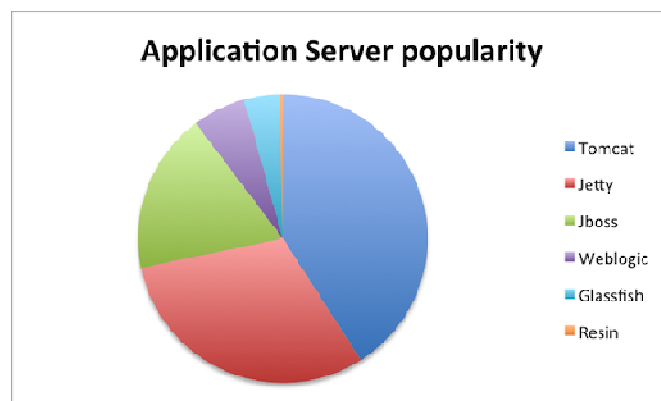


Figura 1.6: Fuente [Plumbr](#), 29 de Mayo de 2014

Tomcat: 41%

Jetty: 31%

JBoss/WildFly: 18%

WebLogic: 6%

GlassFish: 4%

1.2. ENTORNO DE TRABAJO

- MÁQUINA VIRTUAL W7 CON VIRTUALBOX

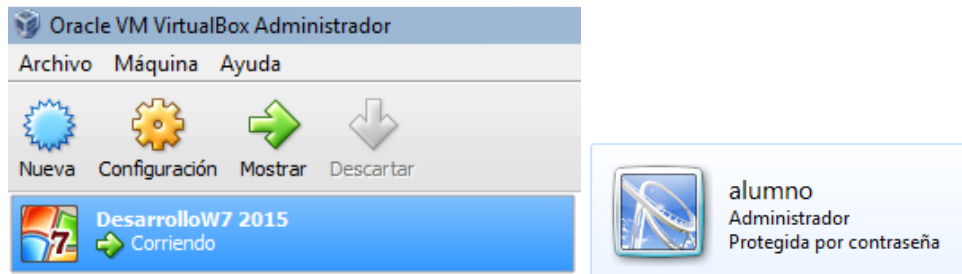


Figura 1.7: Oracle VM VirtualBox

CONFIGURACIÓN

| | |
|---|-----------------------|
| Sistema Operativo | Windows 7 de 32 bits |
| Usuario/Contraseña | alumno/alumno |
| Memoria RAM | 2 GB (mínimo) |
| Adaptador de red | Adaptador1 (Puente) |
| Ratón | Integrado sin captura |
| USB | desactivado |
| Extensiones de archivo | visibles |
| Detección de redes y compartir carpetas | desactivado |
| Fondo de escritorio | color gris |



Figura 1.8: Configuración de la máquina virtual

PROGRAMAS INSTALADOS

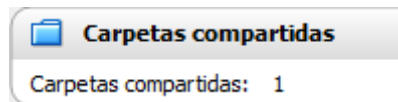
- Oracle VB Guest Additions (Integración del ratón)
- Navegadores Chrome y Firefox
- Editor Notepad++
- Wireshark con librería WinPcap (Capturas de red)
- Compresor 7-zip
- Lector de pdf FoxitReader

- **CREAR UNIDAD DE RED (Máquina real-máquina virtual)**

Arrancamos Oracle VM VirtualBox

1) Crear la carpeta compartida en la Máquina Virtual “DesarrolloW7” asociada a la unidad (DATOS)

- x Automontada
- (x Permanente)



2) En la MV, doble clic en Equipo:

- Clic en Conectar a unidad de red
- Dejar la unidad por defecto
- Clic en Examinar y si no hemos activado esta opción, aparece el error:
"Detección automática de redes no activada. Modificarla en Centro de Redes y Recursos Compartidos"

3) Abrir Centro de Redes y Recursos Compartidos (en barra de tareas)

Hacer clic en Cambiar configuración de uso compartido avanzado:

Si la red de la MV se configuró como Pública/ Casa o Trabajo:

- Pública/ Casa o Trabajo
- Detección de redes
- x Activar la detección de redes

4) Hacer un acceso directo en el escritorio a la recién creada unidad Z:

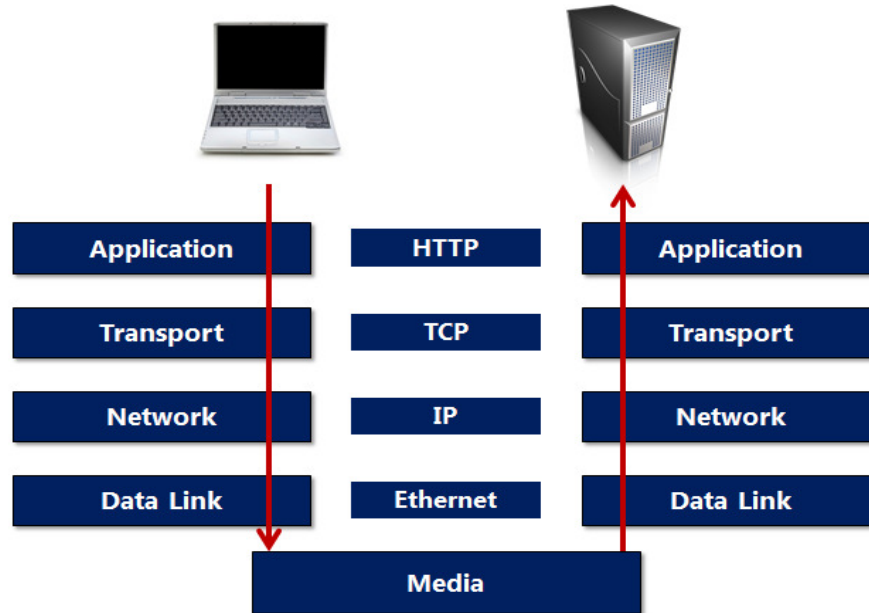
Opcionalmente

5) Crear acceso a la carpeta compartida en el servidor (Profesor) para tener acceso a todo el material compartido.

1.3. PROTOCOLO HTTP

- **Características**

- Funciona en la capa de aplicación de la arquitectura TCP/IP



- No se “preocupa” del transporte ni de la comprobación de errores de los paquetes, estas tareas las desempeñan capas inferiores como TCP o IP.
- “Navegar” en una página web consta de una serie de peticiones HTTP emitidas por el cliente.
- El servidor responde a cada una de ellas con:
 - La información solicitada
 - O bien con un código de error
- Es un protocolo sin estado, porque cada solicitud es completamente independiente de las demás, de manera que no se puede saber si existe alguna relación entre varias solicitudes.
(Más adelante veremos cómo solucionar este problema con las sesiones)
- Ha habido varias versiones desde 1991:
 - HTTP 0.9: [Versión original del protocolo](#)
 - HTTP/1.0: [RFC1945](#)
 - HTTP/1.1: [RFC 2616](#)

- **Mensajes HTTP: petición y respuesta**

Los mensajes http son de texto plano (ASCII) y contienen órdenes y parámetros.

Son de dos tipos: petición y respuesta.

Se componen de tres partes: Línea inicial de petición, cabecera y cuerpo

Mensajes de petición.

Tienen lugar cuando:

- Se escribe una URL en la barra de direcciones de un navegador
- Se hace click en un enlace
- Se envía un formulario HTML
- Se produce una invocación Ajax
- Se produce una invocación con lenguaje de lado servidor

La línea inicial de petición tiene:

Método {GET | POST | OPTIONS} + URI relativa + Versión {HTTP/1.1 | 1.0}

Cabeceras:

En cada línea va una sola cabecera.

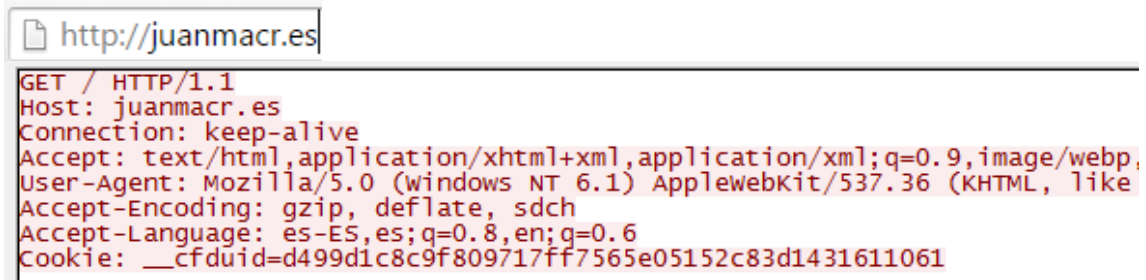
Cada cabecera está formada por un par 'nombre: valor'.

Detrás de la última cabecera hay una línea en blanco para separar el cuerpo (si tiene).

Cuerpo:

Contiene parámetros o ficheros para enviar al servidor. Puede ser vacío.

Ejemplo obtenido con Wireshark:



```
GET / HTTP/1.1
Host: juanmacr.es
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Accept-Encoding: gzip, deflate, sdch
Accept-Language: es-ES,es;q=0.8,en;q=0.6
Cookie: __cfduid=d499d1c8c9f809717ff7565e05152c83d1431611061
```

Figura 1.9: Mensaje de petición (color rojo)

Línea de petición:

GET / HTTP/1.1

Cabeceras:

Host: juanmacr.es

Connection: keep-alive

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp

User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.152 Safari/537.36

Accept-Encoding: gzip, deflate, sdch

Accept-Language: es-ES,es;q=0.8,en;q=0.6

Cookie: __cfduid=d499d1c8c9f809717ff7565e05152c83d1431611061

(Espacio)

Cuerpo:

Mensajes de respuesta.

· Línea inicial de respuesta:

Versión HTTP + Código de estado o código de error + Texto explicativo del código

· Cabeceras:

En cada línea va una sola cabecera.

Cada cabecera está formada por un par 'nombre: valor'.

Detrás de la última cabecera hay una línea en blanco para separar el cuerpo

· Cuerpo del mensaje:

Dependerá del recurso solicitado, en general será texto HTML.

Las imágenes y resto de contenido multimedia no se envían directamente en el código HTML. Cuando un navegador recibe una etiqueta comprueba si tiene copia válida en su caché, y si no la tiene, envía **otra** petición http al servidor.

Por lo tanto, una misma página web suele generar múltiples peticiones http.

Ejemplo:

```
HTTP/1.1 200 OK
Date: Tue, 19 May 2015 17:25:59 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.3.20
Server: cloudflare-nginx
CF-RAY: 1e918113733a0dc3-MAD
Content-Encoding: gzip

f43
.....ZKs.8.>~...Q%uQ$%Y.^T.ZvM..l.X.....q.`R.
...(.a...0...P?h...@R.....E<..D"...5yq|>.....).6&..
a...g...lzk...nC.....8...F.F.1Yc.@../...8...A.
[.....#q...o7...s...J/...m...7x..Vg....
{.3.....4r..DG...D8..{.GA.....8n.v.QDp..O.>.BD.....>
```

Figura 1.10: Mensaje de respuesta (color azul)

Línea inicial de respuesta:

HTTP/1.1 200 OK

Cabeceras:

| | |
|-------------------------------------|------------------------------|
| Date: Tue, 19 May 2015 17:25:59 GMT | → Fecha de la respuesta |
| Content-Type: text/html | → Tipo de contenido enviado |
| Transfer-Encoding: chunked | → Respuesta troceada |
| Connection: keep-alive | → Conexión persistente |
| Vary: Accept-Encoding | → Respuesta comprimida y sin |
| X-Powered-By: PHP/5.3.20 | → Servidor de aplicaciones |
| Server: cloudflare-nginx | → Servidor web/proxy inverso |
| CF-RAY: 1e918113733a0dc3-MAD | → Cookie de Cloudflare |
| Content-Encoding: gzip | → Compresión usada |

(Espacio)

Cuerpo:

```
f43
.....ZKs.8.>~...Q%uQ$%Y.^T.ZvM..l.X.....q.`R.
...
```

Hacemos doble clic sobre la línea que contiene la respuesta:

| | | | | |
|----|--------------|-------------|------|-----------------------------|
| 19 | 104.28.14.29 | 192.168.1.5 | HTTP | HTTP/1.1 200 OK (text/html) |
|----|--------------|-------------|------|-----------------------------|

Así obtenemos toda la trama de respuesta, y el contenido del cuerpo está al final:

```
19 4.407001000 104.28.14.29 192.168.1.5 HTTP 74 HTTP/1.1 200 OK (text/html)
+ Frame 19: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
+ Ethernet II, Src: HitronTe_ee:f4:e2 (78:8d:f7:ee:f4:e2), Dst: Giga-Byt_94:ae:cc (00
+ Internet Protocol Version 4, Src: 104.28.14.29 (104.28.14.29), Dst: 192.168.1.5 (19
+ Transmission Control Protocol, Src Port: http (80), Dst Port: 49285 (49285), Seq: 4
+ [5 Reassembled TCP Segments (4431 bytes): #13(1460), #14(1460), #16(1256), #17(235)
+ Hypertext Transfer Protocol
- Line-based text data: text/html
<!DOCTYPE html>\n
<html>\n
  \n
  <head>\n
    <title>Juanmacr Ciclos Formativos</title>\n
    <META HTTP-EQUIV="Cache-Control" CONTENT="max-age=2592000, must-revalidate">\n
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>\n
    <meta name="description" content="ciclos Formativos de Inform\303\241tica y Comun
    \t\t sitio web de ayuda al estudiante"/>\n
    <meta name="keywords" content="ciclo formativo fp informatica asir dam daw smr"/>
    <script type="text/javascript">\n
      //<![CDATA[\n
      [truncated] try{if (!window.CloudFlare) {var CloudFlare=[{verbose:0,p:0,byc:0,owl
      //}]}>\n
    </script>\n
    <link href="juanmacr.rss" rel="alternate" type="application/rss+xml" title="RSS 2.
    <link href="garceta.rss" rel="alternate" type="application/rss+xml" title="RSS 2.
    <link rel="shortcut icon" href="imagenes/jmcr.ico" type="image/x-icon"/>\n
    <link rel="stylesheet" href="2col_NavIzq.css" type="text/css"/>\n
    [truncated] <style type="text/css">#navBar{width:220px;margin-left:10px;}#navBar ;
    </head>\n
    \n
    <body>\n
      \n
      <div id="header">\n
        <h1>SITIO WEB PARA EL ESTUDIANTE DE CICLOS FORMATIVOS DE <br/>\n
        INFORM\303\201TICA Y COMUNICACIONES</h1>\n
        <p><a href="http://juanmacr.es">Inicio</a></p>\n
        <div id="main_menu">\n
          <ul>\n
            <li><a href="http://juanmacr.es">Inicio</a></li>\n
            <li><a href="http://juanmacr.es/ASIR.html">ASIR</a></li>\n
            <li><a href="http://juanmacr.es/DAM.html">DAM</a></li>\n
            <li><a href="http://juanmacr.es/DAW.html">DAW</a></li>\n
            <li><a href="http://juanmacr.es/SMR.html">SMR</a></li>\n
            </ul>\n
          </div>\n
        </div>\n
      </body>\n
    </html>
```

Figura 1.11: Contenido del cuerpo

Como ver cabeceras con el navegador.

Google Chrome dispone de la herramienta incorporada al navegador, Botón derecho → “Inspeccionar elemento” pestaña “Network”, que permite visualizar los mensajes de petición y respuesta de forma muy simple:

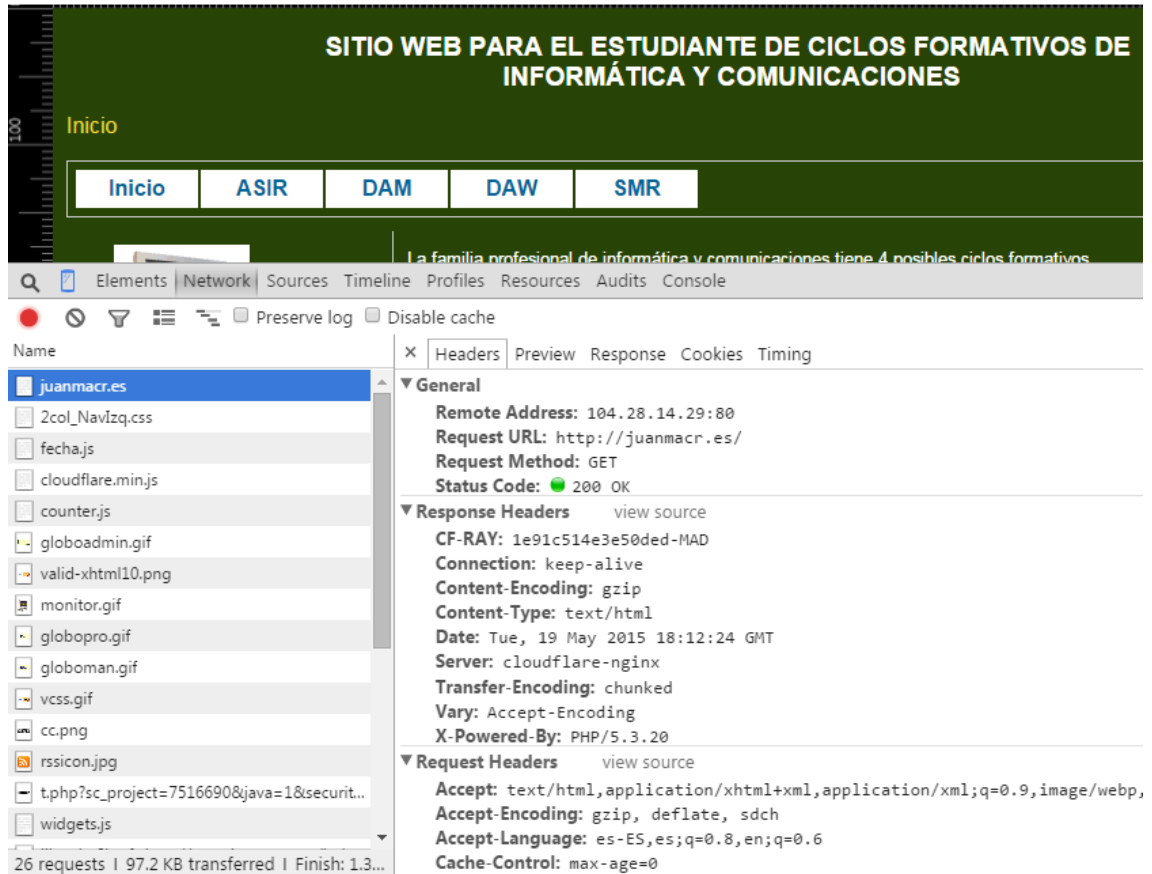


Figura 1.12: Inspeccionar elemento de Google Chrome

- **Cabeceras HTTP tipo Request (petición)**

| Cabecera Request | Significado | Ejemplo |
|--------------------------|---|--------------------------------|
| Host | IP/Nombre de dominio del servidor | <i>www.google.es</i> |
| User-Agent | Cliente web (navegador) | <i>Mozilla/5.0</i> |
| Accept | Determina el tipo de contenido MIME que se espera en la respuesta | <i>text/html</i> |
| Accept-Language | Determina el idioma aceptable para la respuesta | <i>es-Es</i> |
| Accept-Encoding | Determina el tipo de compresión aceptada por el navegador para la respuesta | <i>gzip, deflate</i> |
| Referer | Indica quién invoca el recurso (enlaces dentro de una página) | <i><página origen></i> |
| Cookie | Pareja de valores que almacena datos del cliente | <i>__cfduid=d499d1c8c9f809</i> |
| Connection | El cliente solicita que la conexión sea persistente, para enviar varias solicitudes utilizando la misma conexión TCP | <i>keep-alive</i> |
| Cache-Control | Especifica quien debe realizar el cacheo de la respuesta. – public: servidores proxy – private: el cliente web – no cache: nadie – no store: debe ser borrado – max-age= tiempo en seg | <i>max-age=0</i> |
| Content-Type | Determina el tipo MIME del cuerpo para peticiones POST | <i>application/x-file</i> |
| If-Modified-Since | El cliente indica al servidor, que solicita el recurso solamente si ha cambiado desde una fecha determinada. | <i><fecha></i> |
| Authorization | Para la autenticación de usuarios. Puede ser Basic, que es insegura porque no cifra el usuario ni la contraseña. O bien Digest que es algo más segura. | <i>Basic S2FycG92Og==</i> |

Cabeceras Request menos frecuentes: Pragma (para HTTP/1.0), Range, X-Requested-With, If-match, etc.

- **Cabeceras HTTP tipo Response (respuesta)**

| Cabecera Response | Significado | Ejemplo |
|----------------------------|--|---------------------------------------|
| Server | Nombre del software de servidor | <i>Apache/2.4.2 (Win32)</i> |
| Date | Fecha y hora en que fue enviado el mensaje de respuesta | <i>Tue, 19 May 2015 08:12:31</i> |
| Connection | Tipo de conexión establecida para la respuesta. - keep-alive: persistente - close: se cierra al enviar | <i>keep-alive</i> |
| Content-Encoding | Codificación empleada para comprimir la respuesta | <i>gzip, deflate</i> |
| Content-Type | Tipo MIME de la respuesta | <i>text/css</i> |
| Content-Disposition | Abre un cuadro de diálogo para guardar un archivo asociado a la respuesta | attachment; filename="fichero.txt" |
| Expires | Establece una fecha para la cual el contenido enviado se considera obsoleto | <i><fecha></i> |
| Last-Modified | Fecha de la última modificación | <i><fecha></i> |
| Set-Cookie | Solicita al navegador que guarde una cookie en el cliente | <i>UID=admin;</i> |
| Transfer-Encoding | Indica la codificación usada para transferir de forma segura los datos | <i>chunked</i> |
| Cache-Control | Indica el cacheo que debe realizar el navegador | <i>max-age=0</i> |
| Keep-Alive | Para cada conexión persistente, establece el máximo número de solicitudes y el máximo de tiempo permitido | <i>timeout=5, max=100</i> |
| X-Powered-By | Indica la tecnología usada por el servidor de aplicaciones | <i>PHP/5.3.20</i> |

Cabeceras Response menos frecuentes: X-Cache, X-UA, X-Content-Type-Options, WWW-Authenticate, etc.

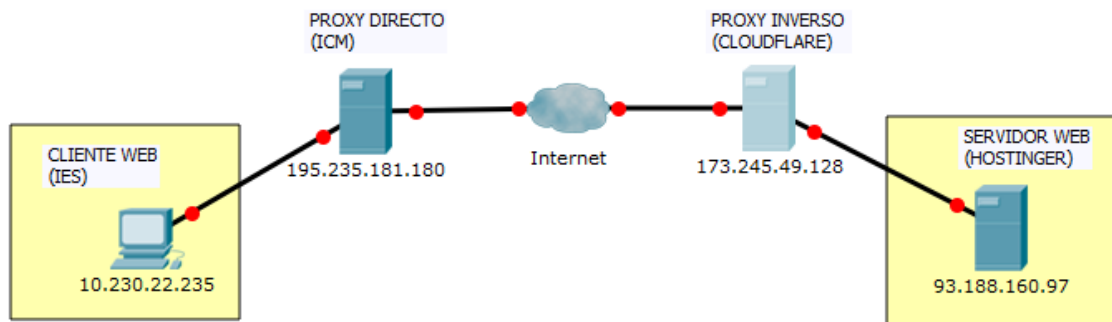
Actividad:

Utilizar la opción Follow TCP Stream del analizador de tráfico Wireshark, para ver la serie de solicitudes-respuesta de cada conexión TCP.

Analiza las cabeceras de solicitud y respuesta. Utiliza la captura realizada anteriormente.

Ejemplo 1:

Ver cabeceras HTTP mediante script de servidor y con Proxy intermedio.



CABECERAS HTTP

Mediante variables super globales

Tu dirección IP es: 195.235.181.180
Tu dirección IP antes de Proxy es: 10.230.22.235
Puerto: 48378

Servidor web (Host): juanmacr.es
Dirección IP del Servidor web: 93.188.160.97
Ruta al script: /home/u779585507/public_html/cabeceras.php

Mediante la función `apache_request_headers()`

```
Host: juanmacr.es
Cookie: __cfduid=d246b4d413210165191120ff4492809411425457554; _ga=GA1.2.864011297.1411972754
X-Real-IP: 173.245.49.128
X-Forwarded-Host: juanmacr.es
X-Forwarded-Server: juanmacr.es
X-Forwarded-For: 10.230.22.235
Connection: close
CF-IPCountry: ES
CF-RAY: 1ea77142b17114c7-CDG
X-Forwarded-Proto: http
CF-Visitor: {"scheme":"http"}
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0
Accept-Language: es-ES,es;q=0.8,en;q=0.6
If-Modified-Since: Fri, 22 May 2015 09:16:44 GMT
X-BlueCoat-Via: be4a77bdc34af56a
CF-Connecting-IP: 195.235.181.180
```

Mediante la función `apache_response_headers()`

```
Array ( [X-Powered-By] => PHP/5.4.10 )
```

Figura 1.13: Cabeceras HTTP cuando hay proxy

Ejemplo 2:

Ver cabeceras HTTP mediante la función phpinfo()

HTTP Headers Information

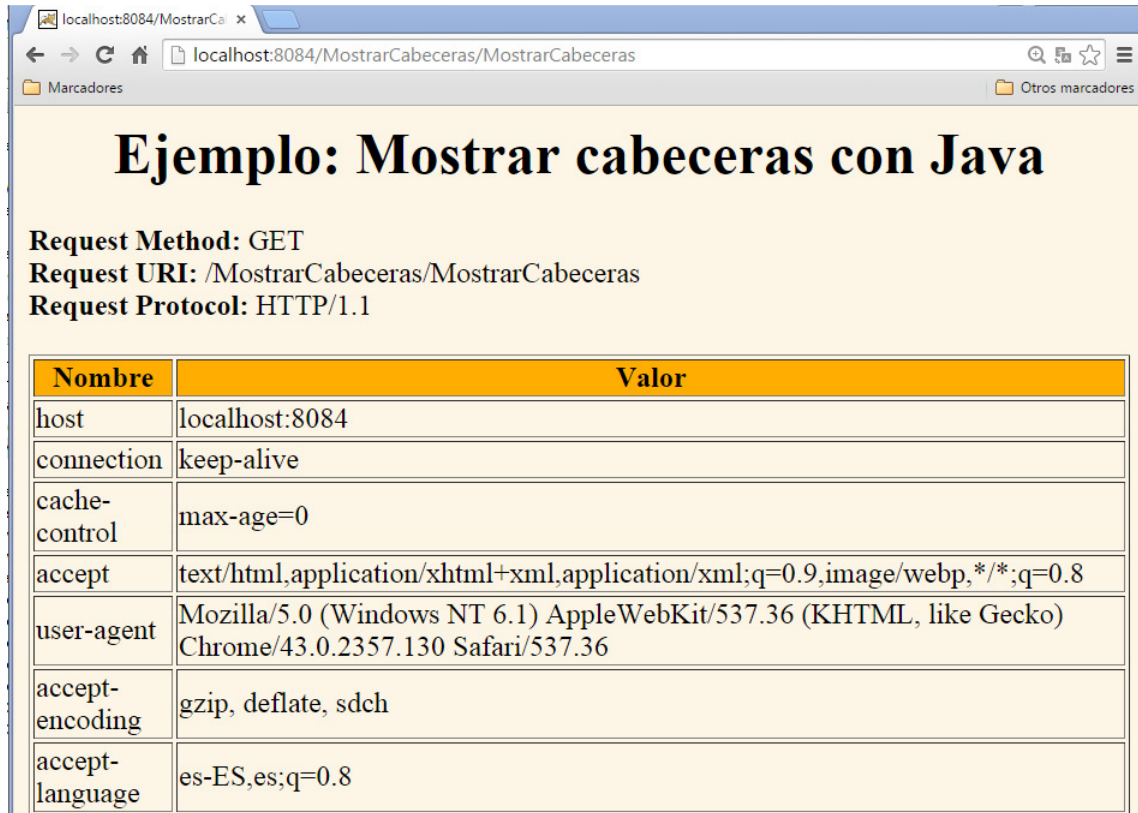
| HTTP Request Headers | |
|-----------------------|---|
| HTTP Request | GET /ver.php HTTP/1.0 |
| Host | www.juanmacr.es |
| Cookie | _cfduid=d499d1c8c9f809717ff7565e05152c83d1431611061; _ga=GA1.2.2021861000.1432056356 |
| X-Real-IP | 188.114.110.176 |
| X-Forwarded-Host | www.juanmacr.es |
| X-Forwarded-Server | www.juanmacr.es |
| Connection | close |
| CF-IPCountry | ES |
| CF-RAY | 1e9b4b59cd940de7-MAD |
| X-Forwarded-Proto | http |
| CF-Visitor | {"scheme":"http"} |
| Accept | text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 |
| User-Agent | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.152 Safari/537.36 |
| Accept-Language | es-ES,es;q=0.8,en;q=0.6 |
| CF-Connecting-IP | 79.108.109.173 |
| HTTP Response Headers | |
| X-Powered-By | PHP/5.3.20 |
| Connection | close |
| Content-Type | text/html |

Figura 1.14: phpinfo()

Podemos observar en las últimas figuras 1.13 y 1.14 como el Proxy inverso Cloudflare añade sus propias cabeceras, por esa razón no coinciden con las cabeceras de la petición inicial que analizamos con Wireshark y Chrome.

Ejemplo 3:

Ver cabeceras HTTP con un Servlet de Java



Ejemplo: Mostrar cabeceras con Java

Request Method: GET
Request URI: /MostrarCabeceras/MostrarCabeceras
Request Protocol: HTTP/1.1

| Nombre | Valor |
|-----------------|--|
| host | localhost:8084 |
| connection | keep-alive |
| cache-control | max-age=0 |
| accept | text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 |
| user-agent | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.130 Safari/537.36 |
| accept-encoding | gzip, deflate, sdch |
| accept-language | es-ES,es;q=0.8 |

- **Códigos de estado**

| Código | Estado | Descripción |
|------------------------|--------------------------------------|--|
| Peticiones correctas | | |
| 200 | OK | Petición correcta |
| 201 | Creado | La petición es correcta y se ha creado un nuevo recurso |
| 204 | Sin contenido | No hay un documento nuevo, se muestra el anterior |
| Redirecciones | | |
| 301 | Movido permanentemente | El recurso se encuentra en otra ubicación, que se indica en la respuesta |
| 302 | Movido temporalmente | Igual que el anterior pero de forma temporal |
| 302 | Ver otra | La respuesta se encuentra en otra URI |
| 304 | No modificado | Existe copia actualizada en caché y el cliente solicita una petición condicional del tipo If-modified-since |
| Errores en el cliente | | |
| 400 | Solicitud incorrecta | Error de sintaxis |
| 401 | No autorizado | Ha fallado la autenticación |
| 402 | Pago requerido | |
| 403 | Prohibido | El usuario no tiene permisos para acceder al recurso |
| 404 | No encontrado | No se pudo encontrar el recurso solicitado |
| Errores en el servidor | | |
| 500 | Error interno | Puede ser provocado por un servlet que no funciona correctamente |
| 503 | Servicio no disponible | Servidor saturado o en mantenimiento temporalmente |
| 504 | Tiempo de espera de pasarela agotado | Ocurre cuando un servidor no recibe respuesta de otro servidor remoto. (Fallo de la red, fallo del ISP, problemas en la resolución DNS, etc.) |

- Herramientas para testear una página web

www.webpagetest.org

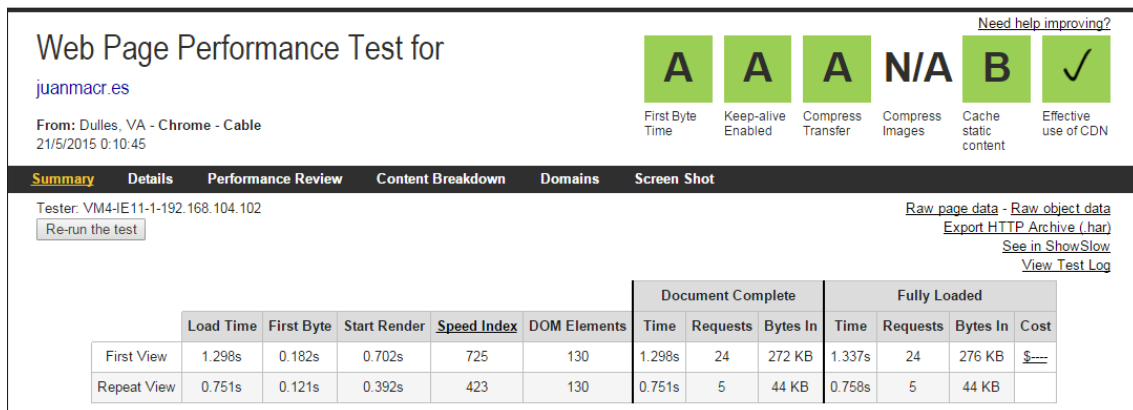


Figura 1.15: webpagetest.org

Parámetros analizados:

- First Byte Time = Tiempo hasta el primer Byte
- Keep-alive Enabled = Conexiones persistentes Activado
- Compress Transfer = Compresión de archivos en la respuesta
- Compress Images = Imágenes comprimidas
- Cache static content = Cacheo de contenido estático (navegadores y proxies directos)
- Effective use of CDN = Uso de real de un CDN (Content Distribution Network)

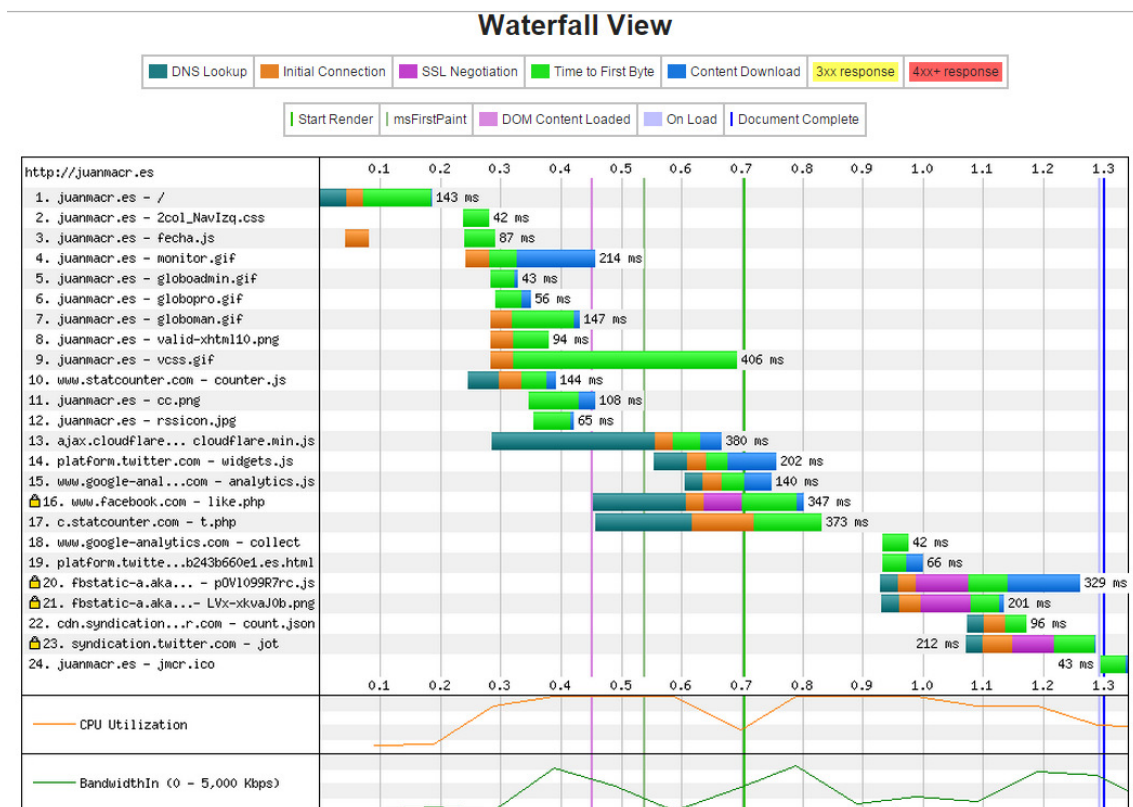


Figura 1.16: Vista en cascada de la respuesta

1.4. INSTALACIÓN DE JDK 8 (Java Development Kit)

1) Instalación de JDK8 en la MV:

- Acceder con el Explorador de Windows a la ubicación del instalable .exe del JDK
- Si hacemos doble clic en el archivo, suele aparecer un error de "La ruta especificada no existe"

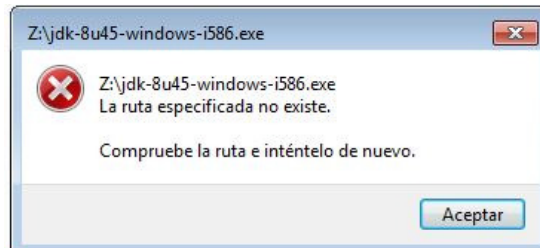


Figura 1.17: "La ruta especificada no existe"

- Se debe copiar el instalable en el sistema de ficheros de la MV, por ejemplo, en el Escritorio
- Hacemos doble clic sobre el instalable ya copiado en la MV

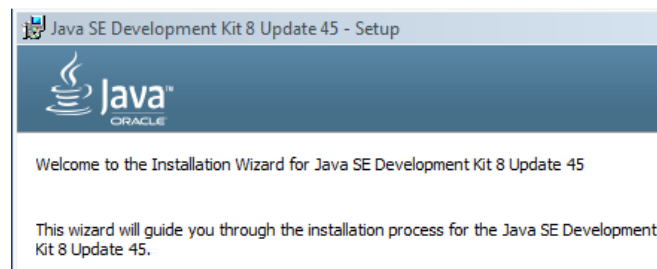


Figura 1.18: Installation Wizard for JDK 8

2) Comprobar la ruta de la instalación.

Por defecto → C:\Archivos de Programa\Java\jdk1.0.8_45

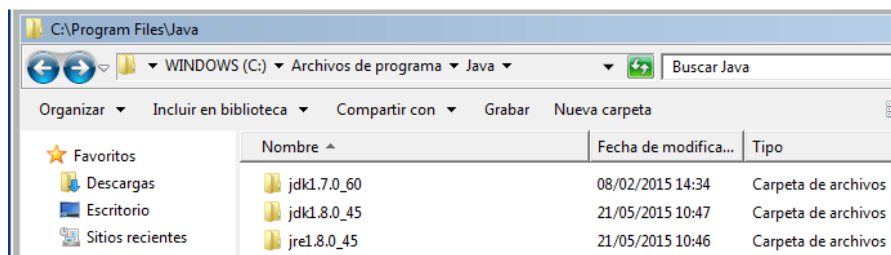


Figura 1.19: Ruta al JDK

3) Comprobar que el JDK incluye el JRE

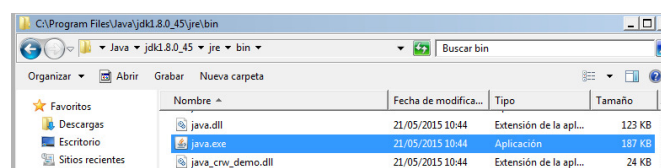


Figura 1.20: JRE incluido en JDK

4) Comprobar la variable de entorno JAVA_HOME:

