



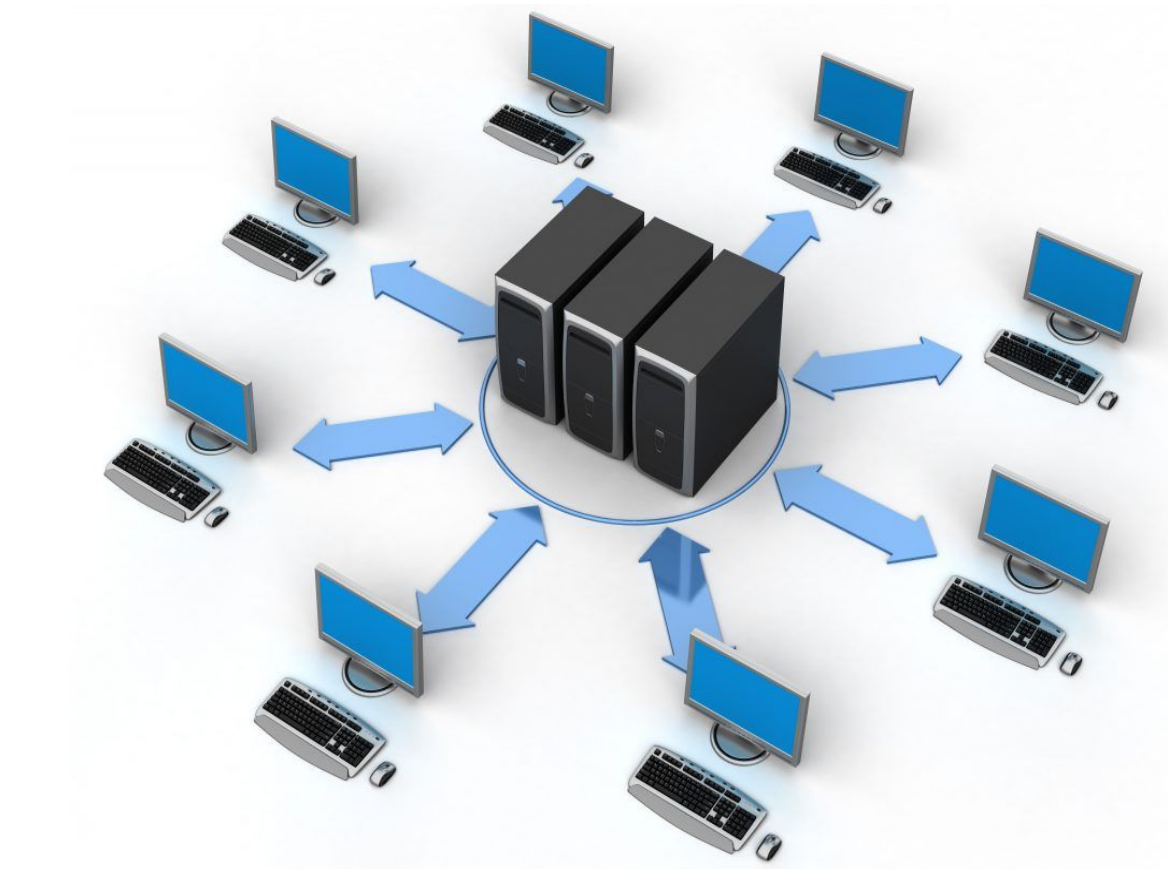
Arquitectura Web. Implantación y administración de servidores web

Introducción

Con la evolución y el acceso libre a Internet, uno de los principales alicientes que han surgido es la publicación de páginas web donde se pueden almacenar unos contenidos bastante atractivos para nosotros y que, al mismo tiempo, pueden ser consultados desde cualquier del mundo para todos.

Cabe decir que, con la popularización de Internet, tanto empresas como usuarios han visto la necesidad de establecer un punto desde donde anunciar sus productos, o bien, a título particular, dar publicidad a las aficiones o capacidades personales mediante la publicación de páginas web.

Las páginas web, en su mayoría en formato HTML, requieren ser alojadas en máquinas que dispongan de espacio en disco para almacenar archivos HTML, imágenes, bloques de código o archivos de vídeo en directorios específicos y, al mismo tiempo, deben ser capaces de entender todo tipo de extensión de los archivos que son enviados en ambos sentidos de la comunicación.



Paralelamente, no podemos dejar de lado la importancia de las medidas de seguridad ante los peligros existentes en Internet. Para ello, las páginas deberán estar diseñadas considerando la incorporación de protocolos de comunicación seguros como, por ejemplo, los desarrollados con el protocolo seguro de transferencia de hipertexto (HTTPS, Hyper Text Transfer Protocol secure) que utilizan claves y estrategias de cifrado propias de las herramientas del protocolo de capa de conexión segura (SSL, secure sockets layer).

Las máquinas que alojan las páginas web reciben la categoría de servidores web. Desde el punto de vista de los servidores, los requerimientos más relevantes son el espacio de disco necesario para poder almacenar la estructura de la página web y una buena conexión de red para que el consumo de la unidad de procesamiento central (CPU, central processing unit) sea bastante bajo.

El funcionamiento de los servidores web es especial ya que, como si se tratara de un diente de sierra, tienen consumos de recursos puntuales porque podemos estar un tiempo sin peticiones y, de repente, tener una avalancha de peticiones. Esto hace que los servidores web suelen tener un número bajo de procesos en espera. A medida que resultan necesarios, se van arrancando nuevos.

Cabe decir que no todas las peticiones consumen el mismo, y, por ejemplo, aquellas páginas web que ejecuten programas de interacción con el usuario o requieran cifrado (HTTPS) consumen más recursos que otras páginas web con menos interacción.

¿Qué es un servidor web?

Los servidores web sirven para almacenar contenidos de Internet y facilitar su disponibilidad de forma constante y segura. Cuando visitas una página web desde tu navegador, es en realidad un servidor web el que envía los componentes individuales de dicha página directamente a tu ordenador. Esto quiere decir que para que una página web sea accesible en cualquier momento, el servidor web debe estar permanentemente online.



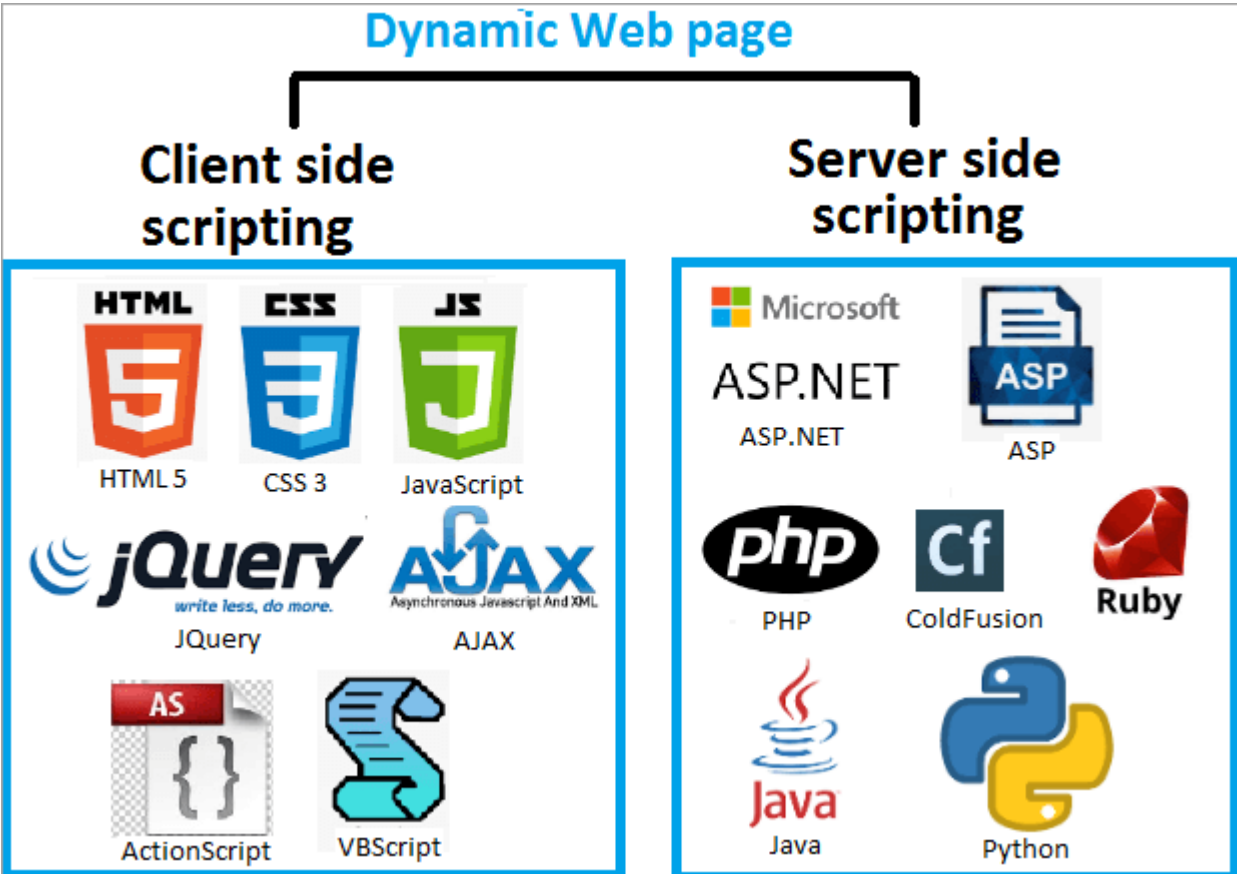
Toda página accesible en Internet necesita un servidor especial para sus contenidos web. A menudo, las grandes empresas y organizaciones cuentan con un servidor web propio para disponer sus contenidos en Intranet e Internet. Sin embargo, la mayoría de administradores recurren a los centros de datos de proveedores de alojamiento web para sus proyectos. Independientemente de si tienes un servidor web propio o de si alquilas uno externo, siempre necesitarás un software para gestionar los datos de tu página y mantenerla actualizada. En este sentido, tienes la posibilidad de elegir entre varias soluciones de software para servidores web diseñadas para diferentes aplicaciones y sistemas operativos.

Tecnología de servidores web

Principalmente, el software de un servidor HTTP es el encargado de proporcionar los datos para la visualización del contenido web.

Para abrir una página web, el usuario solo tiene que escribir el URL correspondiente en la barra de direcciones de su navegador web. El navegador envía una solicitud al servidor web, quien responde, por ejemplo, entregando una página HTML. Esta puede estar alojada como un documento estático en el host o ser generada de forma dinámica, lo que significa que el servidor web tiene que ejecutar un código de programa (p. ej., Java o PHP) antes de tramitar su respuesta.

El navegador interpreta la respuesta, lo que suele generar automáticamente más solicitudes al servidor a propósito de, por ejemplo, imágenes integradas o archivos CSS (hojas de estilos).



El protocolo utilizado para la transmisión es HTTP (o su variante cifrada HTTPS), que se basa, a su vez, en los protocolos de red IP y TCP (y muy rara vez en UDP). Un servidor web puede entregar los contenidos simultáneamente a varios ordenadores o navegadores web. La cantidad de solicitudes (requests) y la velocidad con la que pueden ser procesadas depende, entre otras cosas, del hardware y la carga (número de solicitudes) del host. Sin embargo, la complejidad del contenido también juega un papel importante: los contenidos web dinámicos necesitan más recursos que los contenidos estáticos.



La selección del equipo adecuado para el servidor y la decisión de si este debe ser dedicado, virtual o en la nube, se debe hacer pensando siempre en evitar sobrecargas en el servidor. Aunque se haya encontrado un servidor web que se adapta perfectamente a las necesidades del proyecto, siempre se corre el riesgo de que se presenten fallos en él como consecuencia de imprecisiones técnicas o cortes de energía en el centro de datos del host. Aunque no es muy frecuente, durante un período de inactividad de este tipo (downtime), la web no estará disponible.

Otras funciones de los servidores web

Aunque su principal función es la transferencia de contenido web, muchos programas de servidor web ofrecen características adicionales:

Seguridad	Cifrado de la comunicación entre el servidor web y el cliente vía HTTPS
Autenticación del usuario	Autenticación HTTP para áreas específicas de una aplicación web
Redirección	Redirección de una solicitud de documento por medio de Rewrite Engine
Redirección	Almacenamiento en caché de documentos dinámicos para la respuesta eficiente de solicitudes y para evitar una sobrecarga del servidor web
Asignación de cookies	Envío y procesamiento de cookies HTTP

Además del software del servidor, un host puede contener otro tipo de programas, como por ejemplo un servidor FTP para la carga de archivos o un servidor de base de datos para contenidos dinámicos. En general, existen diferentes tipos de servidores web que pueden ser utilizados para numerosos propósitos, por ejemplo, los servidores de correo, los servidores de juegos o los servidores proxy.

El protocolo HTTP

Historia

El protocolo de transferencia de hipertexto (HTTP, Hypertext Transfer Protocol) es el motor que da vida a Internet, ya que es la base para la web (www, world wide web).

Desde un punto de vista histórico, la web fue creada en 1989 en el Consejo Europeo para la Investigación Nuclear (CERN, Centro Europeene pour la Recherche Nucléaire), con sede en Ginebra, justo en la frontera entre Suiza y Francia. Cabe decir que este organismo disponía (y dispone) de una amplia plantilla de científicos de diferentes países de Europa que trabajan en sus aceleradores de partículas. En consecuencia, muchos equipos de trabajadores están integrados por miembros de nacionalidades diferentes. Además, muchos de los experimentos que se realizan destacan por su complejidad y requieren años y años de planificación y de construcción de equipamientos.

Fue a raíz de la necesidad de disponer de múltiples grupos de científicos repartidos por el mundo y colaborando entre ellos (enviándose informes, dibujos, esquemas, fotos y todo tipo de documentos) que nació la web.



Es en los inicios del protocolo HTTP, a mediados del año 1990, cuando encontramos la versión 0.9. Esta versión tenía como única finalidad transferir datos por Internet en forma de páginas web escritas en lenguaje de marcado de hipertexto (HTML, HyperText Markup Language). A partir de la versión 1.0 del protocolo surgió la posibilidad de transferir mensajes con encabezados que describían el contenido de los mensajes.

Versiones

La primera versión: HTTP/1

La historia de HTTP empezó en 1989, cuando Tim Berners-Lee y su equipo del CERN (Suiza) empezaron a desarrollar la World Wide Web. La versión inicial de HTTP fue bautizada con el número de versión 0.9, consistía en una sola línea y solo permitía solicitar un archivo HTML del servidor cada vez.

El servidor entonces no hacía más que transferir el archivo solicitado, de manera que esta versión del protocolo solo podía manejar archivos HTML.

El primer estándar oficial: HTTP/1.1

HTTP/1.1 aclaró ambigüedades y añadió numerosas mejoras:

- Una conexión podía ser reutilizada, ahorrando así el tiempo de re-abrirla repetidas veces.
- Enrutamiento('Pipelining' en inglés) se añadió a la especificación, permitiendo realizar una segunda petición de datos, antes de que fuera respondida la primera, disminuyendo de este modo la latencia de la comunicación.
- Se permitió que las respuestas a peticiones, podían ser divididas en sub-partes.
- La negociación de contenido, incluyendo el lenguaje, el tipo de codificación, o tipos, se añadieron a la especificación, permitiendo que servidor y cliente, acordasen el contenido más adecuado a intercambiarse.
- Gracias a la cabecera, Host, pudo ser posible alojar varios dominios en la misma dirección IP.

Un protocolo de mayor rendimiento HTTP/2

Según pasaban los años, las páginas web se volvían cada vez más amplias y complejas. Para cargar una web moderna en el navegador, este tiene que solicitar muchos megabytes de datos y enviar hasta cien solicitudes HTTP. HTTP/1.1 está pensado para procesar solicitudes una tras otra en una misma conexión, de manera que cuanto más compleja sea una página web, más tardará en cargarse y mostrarse.

Por esta razón, Google desarrolló un nuevo y experimental protocolo, el SPDY o Speedy, que despertó un gran interés entre los desarrolladores y permitió que en 2015 se publicara la versión HTTP/2 del protocolo. Este estándar incluye múltiples mejoras que tienen como objetivo acelerar la carga de las páginas web.

La versión HTTP/2 se extendió rápidamente y las páginas web con mucho tráfico fueron de las primeras en adoptarla. Actualmente (con fecha de enero de 2020), según W3Techs, un 42 % de las páginas web utilizan la versión HTTP/2.



El futuro: HTTP/3

Un punto débil de todas las versiones de HTTP usadas hasta ahora es el protocolo de control de transmisión (TCP) en el que se basan. Este protocolo requiere que el receptor de cada paquete de datos confirme la recepción antes de que pueda enviarse el siguiente paquete. De este modo, basta con que se pierda un paquete para que todos los demás tengan que esperar a que dicho paquete sea transmitido de nuevo.

Para evitarlos, la nueva versión HTTP/3 no funcionará con TCP, sino con UDP, que no aplica este tipo de medidas correctivas. A partir de UDP, se ha creado el protocolo QUIC (Quick UDP Internet Connections), que será la base de HTTP/3.

Funcionamiento del protocolo HTTP

Ya hemos comentado que el protocolo HTTP tiene un funcionamiento bastante sencillo basado en el envío de mensajes entre cliente y servidor.

Gráficamente podemos resumir el proceso de comunicación HTTP como sigue:



1. **Un usuario accede a una URL**, seleccionando un enlace de un documento HTML o introduciéndola directamente en el campo correspondiente del cliente Web.
2. **El cliente Web descodifica la URL**, separando sus diferentes partes: el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (el valor por defecto es 80) y el objeto requerido del servidor. `http://direccion[:puerto][path]`
Ejemplo: `http://www.miweb.com/documento.html`
3. **Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.** En ese momento, se realiza la petición HTTP. Para ello, se envía el comando necesario (GET, POST, HEAD,...), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada y un conjunto variable de información, que incluye datos sobre las capacidades del navegador (browser), datos opcionales para el servidor, etc.
4. **El servidor devuelve la respuesta al cliente.** Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.
5. **Se cierra la conexión TCP. Este proceso se repite en cada acceso al servidor HTTP.** Por ejemplo, si se recoge un documento HTML en cuyo interior están insertadas 2 imágenes y 1 vídeo, el proceso anterior se repite cuatro veces, una para el documento HTML y tres más para los recursos (la dos imágenes y el vídeo).

Comandos o métodos HTTP

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado.

/books

GET	/books	Lists all the books in the database
DELETE	/books/{bookId}	Deletes a book based on their id
POST	/books	Creates a Book
PUT	/books/{bookId}	Method to update a book
GET	/books/{bookId}	Retrieves a book based on their id

El estándar HTTP/1.0 recoge únicamente tres comandos, que representan las operaciones de recepción y envío de información y chequeo de estado:

- **GET:** se utiliza para solicitar cualquier tipo de información o recurso al servidor. Cada vez que se pulsa sobre un enlace o se teclea directamente a una URL se usa este comando. Como resultado, el servidor HTTP enviará el recurso correspondiente.
- **HEAD:** se utiliza para solicitar información sobre el recurso: su tamaño, su tipo, su fecha de modificación... Es usado por los gestores de cachés de páginas o los servidores proxy, para conocer cuándo es necesario actualizar la copia que se mantiene del recurso. Con HEAD se podrá comprobar la última fecha de modificación de un recurso antes de traer una nueva copia del mismo.
- **POST:** sirve para enviar información al servidor, por ejemplo, los datos contenidos en un formulario. El servidor pasará esta información a un proceso encargado de su tratamiento.

La versión 1.1 del protocolo incorpora unos pocos comandos más como son: OPTIONS, PUT, DELETE, TRACE y CONNECT. Veamos algunos de ellos:

- **OPTIONS:** Devuelve los métodos HTTP que el servidor soporta para una URL específica. Esto puede ser utilizado para comprobar la funcionalidad de un servidor web mediante petición en lugar de un recurso específico.
- **DELETE:** sirve para eliminar un recurso especificado en la URL, aunque pocas veces sera permitido por un servidor web.
- **TRACE:** comando que permite hacer un sondeo para saber todos los dispositivos de la red por los que pasa nuestra petición. Así podremos descubrir si la petición pasa a través dispositivos intermedios o proxys antes de llegar al servidor Web.
- **PUT:** puede verse como el comando inverso a GET. Nos permite escribir datos en el servidor o, lo que es lo mismo, poner un recurso en la URL que se especifique. Si el recurso no existe lo crea sino lo reemplaza. La diferencia con POST puede ser algo confusa; mientras que POST está orientado a la creación de nuevos contenidos, PUT está más orientado a la actualización de los mismos (aunque también podría crearlos).

HTTP/2 no incluye métodos nuevos.

Ejemplo de petición y respuesta

Una solicitud HTTP es un conjunto de líneas que el navegador envía al servidor. Incluye:

- El recurso solicitado, el método que se aplicará y la versión del protocolo utilizada.

- Los campos del encabezado de solicitud: es un conjunto de líneas opcionales que permiten aportar información adicional sobre la solicitud y/o el cliente (navegador, sistema operativo, etc.). Cada una de estas líneas está formada por un nombre que describe el tipo de encabezado, seguido de dos puntos (:) y el valor del encabezado.
- El cuerpo de la solicitud: es un conjunto de líneas opcionales que deben estar separadas de las líneas precedentes por una línea en blanco y que, por ejemplo, permiten la transmisión de datos al servidor de un formulario a través del método POST.

```
GET https://aules.edu.gva.es/fp/login/index.php HTTP/1.1
Host: aules.edu.gva.es
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:82.0)
Gecko/20100101 Firefox/82.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie: MoodleSession=jilabl0pq6h3a3h16q154vc0uk;
BIGipServerP_PHP2_aules=1099308972.20480.0000;
BIGipServerPOOL_EDUCA_FP=251784970.22811.0000;
Cookie_AULES=553774858.22811.0000;
NAUSICA=98333959-8110-43f4-818d-c4ec1d7bc80b
Upgrade-Insecure-Requests: 1
```

La sintaxis de una respuesta HTTP es un conjunto de líneas que el servidor envía al navegador. Incluye:

```
HTTP/1.1 200 OK
Date: Wed, 18 Nov 2020 08:58:37 GMT
Server: Apache
Content-Language: es
Content-Script-Type: text/javascript
Content-Style-Type: text/css
X-UA-Compatible: IE=edge
Cache-Control: private, pre-check=0, post-check=0, max-age=0, no-transform
Pragma: no-cache
Expires:
Accept-Ranges: none
X-Frame-Options: sameorigin
Che: 30
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 40633

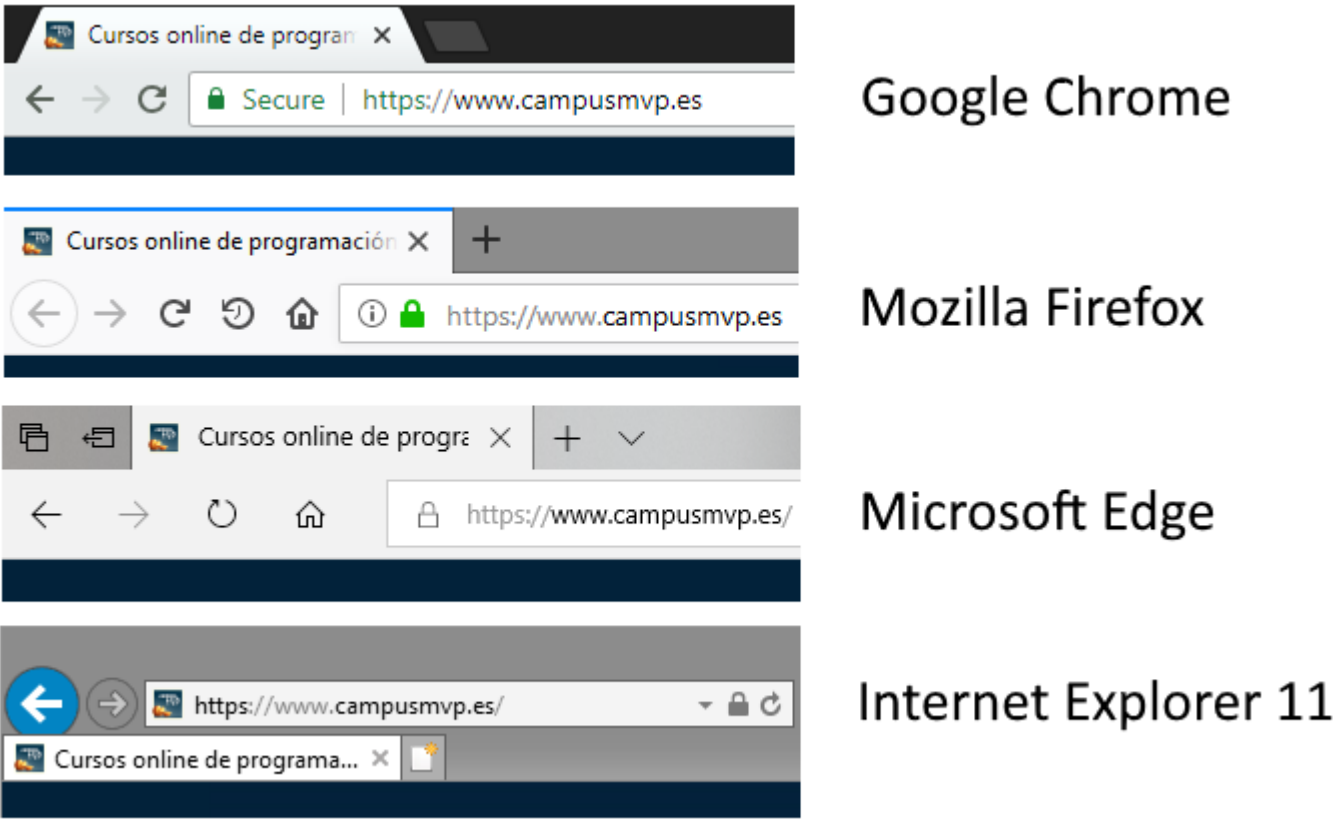
<!DOCTYPE html>
<html dir="ltr" lang="es" xml:lang="es">
<head>
  <title>AULES: Entrar al sitio</title>
  <link rel="icon" href="https://aules.edu.gva.es/fp/theme/image.php/adaptable/theme/1605268028/favicon" />

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="keywords" content="moodle, AULES: Entrar al sitio" />
<link rel="stylesheet" type="text/css" href="https://aules.edu.gva.es/fp/theme/yui_combo.php?rollup/3.17.2/yui-moodlesimple-min.css" /><script
id="firstthemesheet" type="text/css">/** Required in order to fix style inclusion problems in IE with YUI **</script><link rel="stylesheet"
type="text/css" href="https://aules.edu.gva.es/fp/theme/styles.php/adaptable/1605268028_1/all" />
<script>
//
var M = {}; M.yui = {};
M.pageloadstarttime = new Date();
M.cfg =
{"wwwroot":"https://aules.edu.gva.es/fp","sesskey":"xACirjvwhI","sessiontimeout":"3600","themerev":"1605268028","slasharguments":1,"theme":"adaptable",
,"iconsystemmodule":"core/icon_system_fontawesome","jsrev":"1605268028","admin":"admin","svgicons":true,"usertimezone":"Europa/Madrid","contextid":1,"langrev":1605268028,"templaterev":"1605268028"};var yuilConfigFn = function(me)
{if(/-skin|reset|fonts|grid|base/.test(me.name)){me.type='css';me.path=me.path.replace(/\.js/, '.css');me.path=me.path.replace(/\/yui2-skin/, '/assets/skins/sam/yui2-skin')}};
var yui2ConfioFn = function(me) {var</pre></div><div data-bbox="96 461 904 514" data-label="List-Group"><ul><li>• Una línea de estado donde figura el versión del protocolo usada, un código de estado/error y un texto con el significado de dicho código.</li><li>• Los posibles códigos de estado se identifican con números de tres cifras y se clasifican en cinco grupos según sean informativos (1xx), de éxito en la solicitud (2xx), para redireccionar la solicitud (3xx), por error generado en el cliente (4xx) o bien por errores generados en el servidor (5xx) → <b>Códigos de estado/error</b></li><li>• Los campos del encabezado de la respuesta. Conjunto de líneas opcionales que aportan información adicional sobre la respuesta y/o el servidor.</li><li>• El cuerpo de la respuesta que contiene el recurso (objeto) solicitado</li></ul></div><div data-bbox="88 529 190 538" data-label="Section-Header"><p>Cabeceras HTTP</p></div><div data-bbox="88 545 904 571" data-label="Text"><p>Las <b>cabeceras HTTP</b> son los parámetros que se envían en una petición o respuesta HTTP al cliente o al servidor para proporcionar información esencial sobre la transacción en curso. Estas cabeceras proporcionan información mediante la sintaxis <b>‘Cabecera: Valor’</b> y son enviadas automáticamente por el navegador o el servidor Web. → <a href="#">Cabeceras HTTP</a></p></div><div data-bbox="88 583 159 593" data-label="Section-Header"><p>Tipos MIME</p></div><div data-bbox="88 600 904 626" data-label="Text"><p>El protocolo HTTP fue diseñado para transportar por red ficheros en formato ASCII, formados por texto plano. Con el paso del tiempo, surgió la necesidad de incluir diferentes tipos de ficheros no ASCII en las aplicaciones por Internet (imágenes, vídeos, sonidos, etc.) y, como consecuencia de ello, fue necesario buscar una solución: había que transformar estos formatos a tipo ASCII (u otros juegos de caracteres compatibles) para su correcta recepción en el navegador web.</p></div><div data-bbox="88 632 904 650" data-label="Text"><p>Este problema ya había surgido en las aplicaciones de correo electrónico, cuando se necesitó enviar por MAIL ficheros no formados por texto plano, y por tanto, no compatibles con los juegos de caracteres permitidos.</p></div><div data-bbox="88 656 904 673" data-label="Text"><p>Para solucionar este problema se crearon los tipos MIME (Multipurpose Internet Mail Extensions), especificaciones para dar formato a mensajes no-ASCII, de forma que pudieran ser enviados por Internet e interpretados correctamente por los programas de correo locales.</p></div><div data-bbox="88 679 904 715" data-label="Text"><p>Tipos de medios de Internet, previamente conocido como "tipos " o "tipos de contenido", es un estándar diseñado para indicar el tipo de información que presenta un archivo o un conjunto de datos. En , este identificador puede ser útil para conocer el tipo de un archivo antes de descargarlo y tener acceso a él. Es una buena práctica proveer información de tipos de medios siempre que sea posible, como en el caso de los elementos que cuentan con atributos como type, enctype, formenctype y accept.</p></div><div data-bbox="258 720 742 859" data-label="Image"><img alt="Screenshot of a web browser window showing a 'MIME TYPES' window. The window displays a grid of icons representing various file formats: HTML, DOC, PNG, PPT, JPG, MP3, AVI, AI, PSD, PDF, BMP, EPS, TXT, MP4, C, CSS, JS, PHP, XML, ZIP, and GIF. The icons are color-coded and arranged in a 4x6 grid. The window has a title bar with standard minimize, maximize, and close buttons."/></div><div data-bbox="88 865 493 874" data-label="Text"><p>Todo identificador de tipo de medio de Internet debe ajustarse al siguiente formato:</p></div><div data-bbox="88 880 904 898" data-label="Text"><p>Así pues, el "tipo" y el "subtipo" deben estar presentes en cualquier tipo de medio de Internet. En la lista siguiente hay algunos ejemplos que contienen cada una da las partes delineadas anteriormente.</p></div><div data-bbox="88 912 141 923" data-label="Section-Header"><p>HTTPS</p></div><div data-bbox="88 930 904 948" data-label="Text"><p>El Protocolo seguro de transferencia de hipertexto (en inglés, Hypertext Transfer Protocol Secure o HTTPS) es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto, es decir, es la versión segura de HTTP.</p></div><div data-bbox="88 954 904 970" data-label="Text"><p>La web es insegura por naturaleza. Cuando se diseñaron los protocolos en los que está basada (TCP/IP) no se tuvieron en cuenta muchos de los problemas que tiene la Internet moderna. Y el protocolo HTTP para transferir páginas web, no añadió nada al respecto tampoco hasta mucho después, con la introducción del protocolo</p></div>
```

ra internet moderna. Y el protocolo HTTP, para transmitir paginas web, no añadio nada al respecto tampoco hasta mucho despues, con la introduccion del protocolo HTTPS (la "ese" es de "Seguro") allá por 1994 por la empresa Netscape. El protocolo HTTPS original utilizaba SSL (Secure Sockets Layer) como protocolo seguro de intercambio de claves y cifrado, pero en la actualidad está obsoleto y se emplea TLS (Transport Layer Security, que va por su versión 1.3). El estándar de HTTP sobre TLS, en realidad, no se configuró hasta mayo del año 2000.

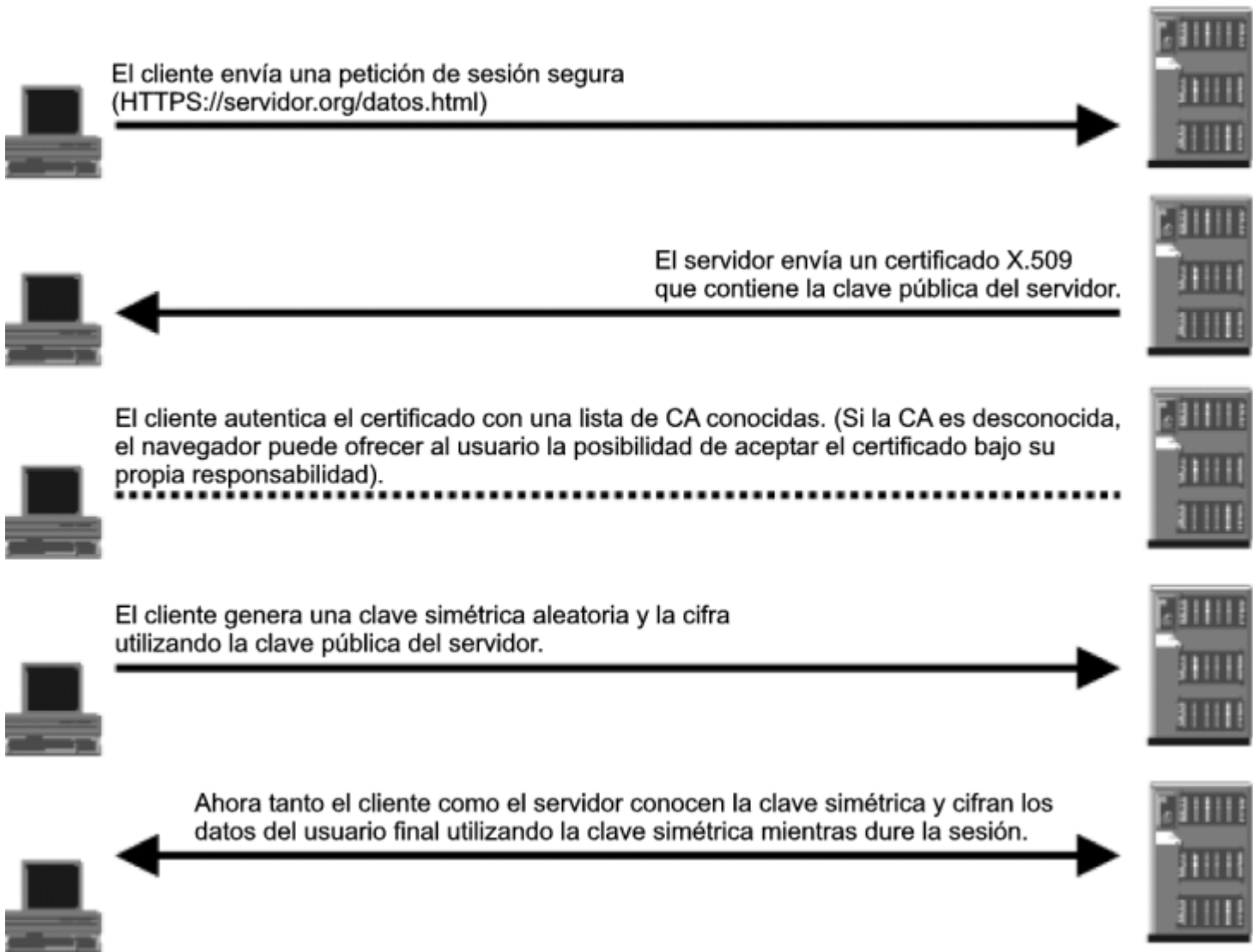
Tradicionalmente, los navegadores le han indicado a sus usuarios que se estaban conectando a un sitio seguro utilizando un iconito, generalmente uno con un candado.

Según el navegador el aspecto cambia un poco, pero todos muestran el proverbial "candadito" al lado de la dirección:



Es decir, lo importante aquí es que hasta ahora los navegadores consideran HTTP como la norma, y HTTPS como la excepción, y por eso lo marcan de esta manera.

Funcionamiento de HTTPS



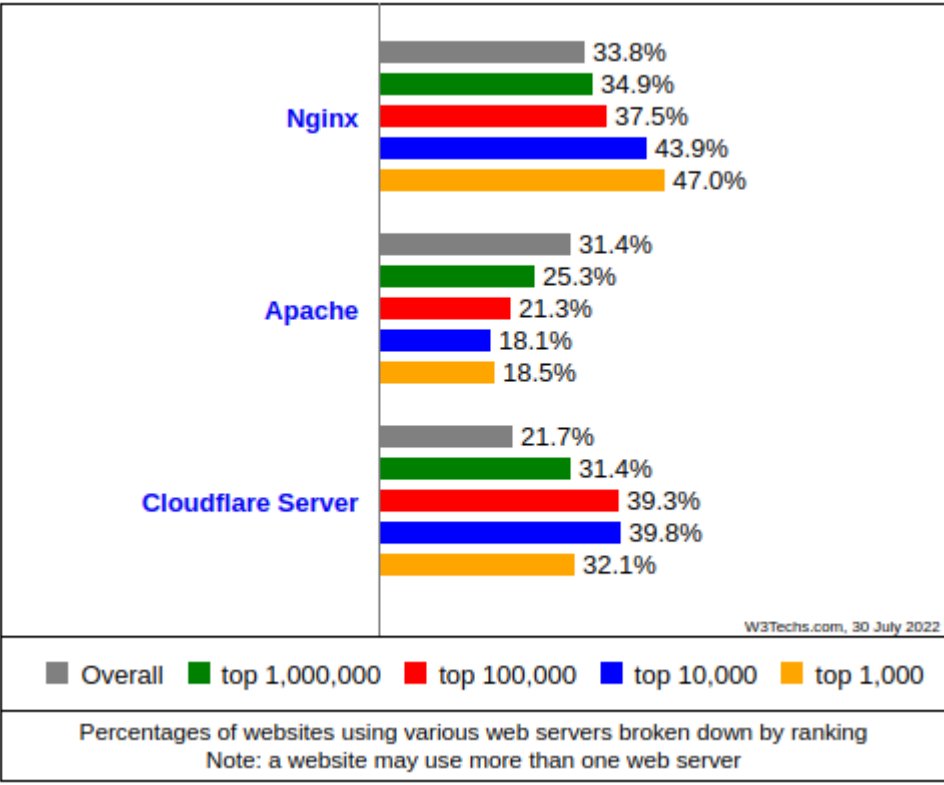
Servidores web: Apache vs Nginx

Cuando vamos a poner en marcha un servidor web, lo primero que necesitamos es utilizar un sistema operativo sobre el cual vamos a ejecutar los diferentes servicios, sistema operativo que en más del 95% de las ocasiones suele ser un sistema Linux, así como un software que se encargue de la gestión de las bases de datos, MySQL habitualmente, y un software para gestionar el contenido dinámicos de las webs, que suele ser PHP. Además de este software esencial, otra de las partes más importantes del servidor suele ser la elección del servidor web, y aquí es donde entran las dudas.

Cuando buscamos montar una web podemos elegir una gran cantidad de servidores web diferentes, desde Apache y Nginx, los más conocidos y utilizados con más de un 85% de uso entre ambos, hasta otros servidores menos conocidos como Microsoft IIS (si usamos un servidor Windows), LiteSpeed, Node.js, etc.

Los dos servidores más utilizados para montar páginas web hoy en día son Apache y Nginx, sin embargo, es imposible decir que uno es mejor que otro ya que cada uno de ellos tiene sus propias fortalezas y debilidades y puede mejorar mejor bajo ciertas circunstancias o simplemente ser más sencillo de utilizar.

Nginx está orientado a mejorar el rendimiento, soportando mayores cargas de tráfico y usuarios que Apache (Problema C10K), además de ofrecer otras funcionalidades como hacer de proxy. En sus orígenes era especialmente eficiente ofreciendo contenido estático.

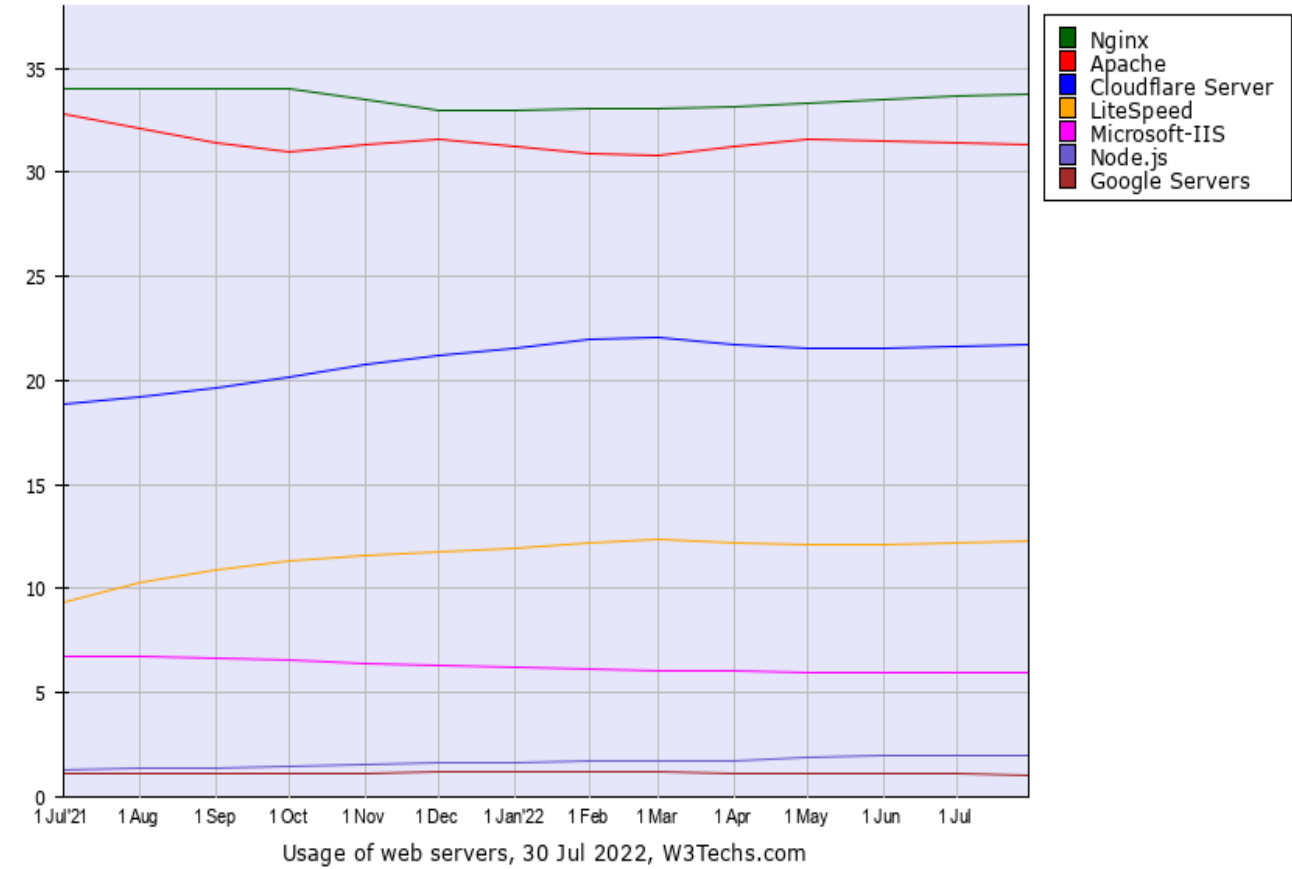


Después de ser lanzado, Nginx fue usado principalmente para servir archivos estáticos y como un balanceador de carga o proxy inverso en frente de instalaciones Apache.

Ejemplos de servicios de despliegue de páginas estáticas:

- Netlify
- Surge
- GitHub Pages
- GitLab Pages
- Firebase
- Vercel
- Neocities

Mientras evolucionaba la red, y la necesidad de exprimir hasta la última gota de la velocidad y eficiencia de uso de hardware con este, más sitios empezaron a reemplazar Apache con Nginx por completo, gracias a un software mucho más maduro.



Razones para usar Nginx

- 1. Es ligero
Nginx reduce el consumo de RAM.
- 2. Es multiplataforma y fácil de instalar
La mayoría de las grandes distribuciones de GNU/Linux, tienen Nginx en sus repositorios.
- 3. ¡Se puede usar junto a Apache!
Sí, como lo lees, algunas empresas solo usan Nginx para servir contenido estático y Apache para el contenido dinámico.
- 4. Caché
Puedes usar Nginx como caché, con algo de configuración, permitiendo mejorar la eficiencia de tu aplicación sin tocar la programación de la misma.
- 5. Balanceador de carga
Este servidor web puede funcionar como balanceador de carga, distribuyendo el tráfico entre varios servidores, permitiendo mayor escalabilidad.
- 6. Soporte comunitario y profesional
Nginx, Inc está detrás del desarrollo de Nginx, además de la comunidad en general, permitiendo tener un soporte tanto profesional como comunitario.
- 7. Compatibilidad con las aplicaciones web más populares
Nginx es compatible con una gran cantidad de CMS existentes en el mercado, y hay un muchos tutoriales y documentación para instalar estos bajo Nginx, como por ejemplo: Wordpress, Joomla, Drupal, phpBB ¡y más!

Práctica 2.1 – Instalación y configuración de servidor web Nginx

Instalación servidor web Nginx

Para instalar el servidor nginx en nuestra Debian, primero actualizamos los repositorios y después instalamos el paquete correspondiente:

```
sudo apt update

sudo apt install nginx
```

Comprobamos que nginx se ha instalado y que está funcionando correctamente:

```
systemctl status nginx
```

Info

Esta práctica se ha hecho con Nginx 1.18.0

Creación de las carpeta del sitio web

Igual que ocurre en Apache, todos los archivos que formarán parte de un sitio web que servirá nginx se organizarán en carpetas. Estas carpetas, típicamente están dentro de `/var/www`.

Así pues, vamos a crear la carpeta de nuestro sitio web o dominio:

```
sudo mkdir -p /var/www/nombre_web/html
```

Donde el nombre de dominio puede ser la palabra que queráis, sin espacios.

Ahí, dentro de esa carpeta html, debéis clonar el siguiente repositorio:

```
https://github.com/cloudacademy/static-website-example
```

Además, haremos que el propietario de esta carpeta y todo lo que haya dentro sea el usuario `www-data`, típicamente el usuario del servicio web.

```
sudo chown -R www-data:www-data /var/www/nombre_web/html
```

Y le daremos los permisos adecuados para que no nos de un error de acceso no autorizado al entrar en el sitio web:

```
sudo chmod -R 755 /var/www/nombre_web
```

Para comprobar que el servidor está funcionando y sirviendo páginas correctamente, podéis acceder desde vuestro cliente a:

```
http://IP-maq-virtual
```

Y os deberá aparecer algo así:



Lo que demuestra que todo es correcto hasta ahora.

Configuración de servidor web NGINX

En Nginx hay dos rutas importantes. La primera de ellas es `sites-available`, que contiene los archivos de configuración de los hosts virtuales o bloques disponibles en el servidor. Es decir, cada uno de los sitios webs que alberga el servido. La otra es `sites-enabled`, que contiene los archivos de configuración de los sitios habilitados, es decir, los que funcionan en ese momento.

Dentro de `sites-available` hay un archivo de configuración por defecto (default), que es la página que se muestra si accedemos al servidor sin indicar ningún sitio web o cuando el sitio web no es encontrado en el servidor (debido a una mala configuración por ejemplo). Esta es la página que nos ha aparecido en el apartado anterior.

Para que Nginx presente el contenido de nuestra web, es necesario crear un bloque de servidor con las directivas correctas. En vez de modificar el archivo de configuración predeterminado directamente, crearemos uno nuevo en `/etc/nginx/sites-available/nombre_web`:

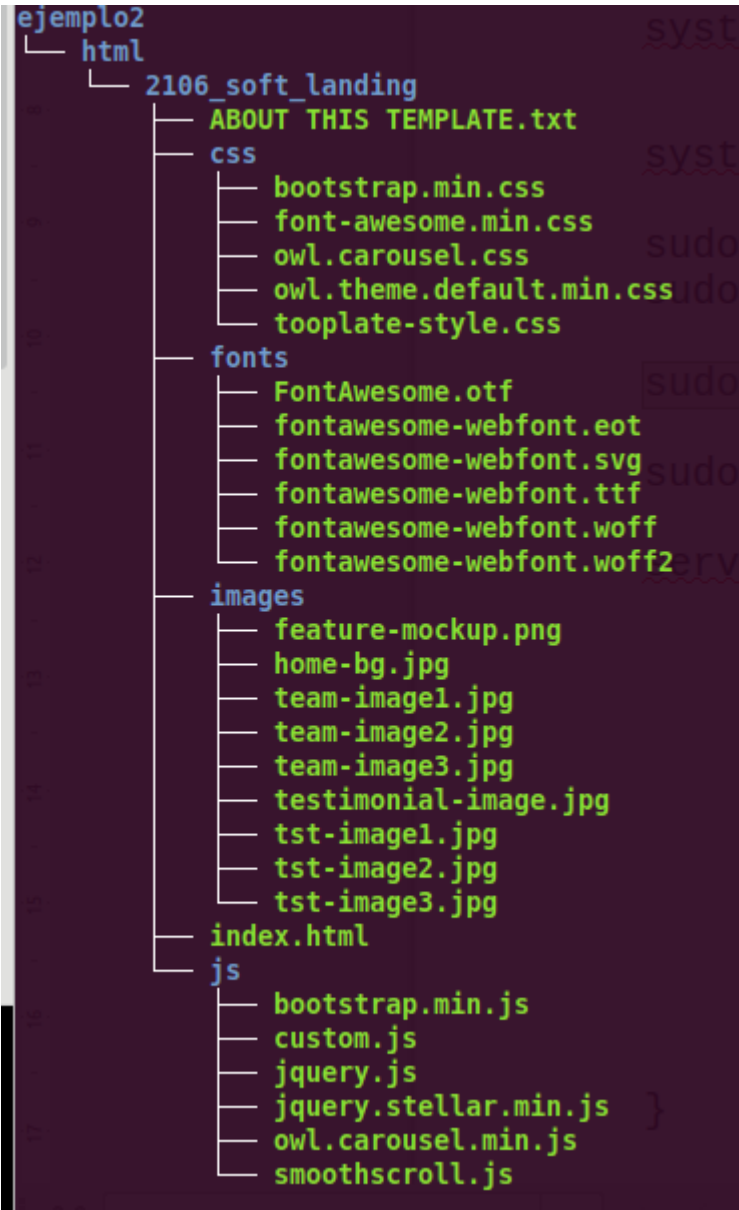
```
sudo nano /etc/nginx/sites-available/vuestro_dominio
```

Y el contenido de ese archivo de configuración:

```
server {
    listen 80;
    listen [::]:80;
    root /ruta/absoluta/archivo/index;
    index index.html index.htm index.nginx-debian.html;
    server_name nombre_web;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

Aquí la directiva `root` debe ir seguida de la ruta absoluta absoluta dónde se encuentre el archivo `index.html` de nuestra página web, que se encuentra entre todos los que habéis descomprimido.

Aquí tenéis un ejemplo de un sitio webs con su ruta (directorios que hay) antes del archivo `index.html`:



i Info

Ruta → /var/www/ejemplo2/html/2016_soft_landing

Y crearemos un archivo simbólico entre este archivo y el de sitios que están habilitados, para que se dé de alta automáticamente.

```
sudo ln -s /etc/nginx/sites-available/nombre_web /etc/nginx/sites-enabled/
```

Y reiniciamos el servidor para aplicar la configuración:

```
sudo systemctl restart nginx
```

Comprobaciones

Comprobación del correcto funcionamiento

Como aún no poseemos un servidor DNS que traduzca los nombres a IPs, debemos hacerlo de forma manual. Vamos a editar el archivo `/etc/hosts` **de nuestra máquina anfitriona** para que asocie la IP de la máquina virtual, a nuestro `server_name`.

Este archivo, en Linux, está en: `/etc/hosts`

Y en Windows: `C:\Windows\System32\drivers\etc\hosts`

Y deberemos añadirle la línea:

```
192.168.X.X nombre_web
```

donde debéis sustituir la IP por la que tenga vuestra máquina virtual.

Comprobar registros del servidor

Comprobad que las peticiones se están registrando correctamente en los archivos de logs, tanto las correctas como las erróneas:

- `/var/log/nginx/access.log` : cada solicitud a su servidor web se registra en este archivo de registro, a menos que Nginx esté configurado para hacer algo diferente.
- `/var/log/nginx/error.log` : cualquier error de Nginx se asentará en este registro.

i Info

Si no os aparece nada en los logs, podría pasar que el navegador ha cacheado la página web y que, por tanto, ya no está obteniendo la página del navegador sino de la propia memoria. Para solucionar esto, podéis acceder con el *modo privado* del navegador y ya os debería registrar esa actividad en los logs.

FTP

Si queremos tener varios dominios o sitios web en el mismo servidor nginx (es decir, que tendrán la misma IP) debemos repetir todo el proceso anterior con el nuevo nombre de dominio que queramos configurar.

¿Cómo transferir archivos desde nuestra máquina local/anfitrión a nuestra máquina virtual Debian/servidor remoto?

A día de hoy el proceso más sencillo y seguro es a través de Github como hemos visto antes. No obstante, el currículum de la Conselleria d'Educació me obliga a enseñaros un método un tanto obsoleto a día de hoy, así que vamos a ello, os presento al FTP.

El **FTP** es un protocolo de transferencia de archivos entre sistemas conectados a una red TCP. Como su nombre indica, se trata de un protocolo que permite transferir archivos directamente de un dispositivo a otro. Actualmente, es un protocolo que poco a poco va abandonándose, pero ha estado vigente más de 50 años.

El protocolo FTP tal cual es un protocolo inseguro, ya que su información no viaja cifrada. Sin embargo, en 2001 esto se solucionó con el protocolo **SFTP**, que le añade una capa SSH para hacerlo más seguro y privado.

SFTP no es más que el mismo protocolo FTP pero implementado por un canal seguro. Son las siglas de SSH File Transfer Protocol y consiste en una extensión de Secure Shell Protocol (SSH) creada para poder hacer transmisiones de archivos.

La seguridad que nos aporta **SFTP** es importante para la transferencia de archivos porque, si no disponemos de ella, los archivos viajarán tal cual por la red, sin ningún tipo de encriptación. Así pues, usando FTP tradicional, si algún agente consigue escuchar las transferencias, podría ocurrir que la información quedase al descubierto. Esto sería especialmente importante si los archivos que subimos contienen información confidencial o datos personales.

Dado que usar **SFTP** aporta mayor seguridad a las transmisiones, es recomendable utilizarlo, más aún sabiendo que realmente no hay mucha dificultad en establecer las conexiones por el protocolo seguro.

Configurar servidor SFTP en Debian

En primer lugar, lo instalaremos desde los repositorios:

```
sudo apt-get update
sudo apt-get install vsftpd
```

Ahora vamos a crear una carpeta en nuestro *home* en Debian:

```
mkdir /home/nombre_usuario/ftp
```

En la configuración de *vsftpd* indicaremos que este será el directorio al cual vsftpd se cambia después de conectarse el usuario.

Ahora vamos a crear los certificados de seguridad necesarios para aportar la capa de cifrado a nuestra conexión (algo parecido a HTTPS)

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/vsftpd.pem -out /etc/ssl/private/vsftpd.pem
```

Y una vez realizados estos pasos, procedemos a realizar la configuración de *vsftpd* propiamente dicha. Se trata, con el editor de texto que más os guste, de editar el archivo de configuración de este servicio, por ejemplo con *nano*:

```
sudo nano /etc/vsftpd.conf
```

En primer lugar, buscaremos las siguientes líneas del archivo y las eliminaremos por completo:

```
rsa_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
rsa_private_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
ssl_enable=NO
```


Tras ello, añadiremos estas líneas en su lugar

```
rsa_cert_file=/etc/ssl/private/vsftpd.pem
rsa_private_key_file=/etc/ssl/private/vsftpd.pem
ssl_enable=YES
allow_anon_ssl=NO
force_local_data_ssl=YES
force_local_logins_ssl=YES
ssl_tlsv1=YES
ssl_sslv2=NO
ssl_sslv3=NO
require_ssl_reuse=NO
ssl_ciphers=HIGH

local_root=/home/nombre_usuario/ftp
```

Y, tras guardar los cambios, reiniciamos el servicio para que coja la nueva configuración:

```
sudo systemctl restart --now vsftpd
```

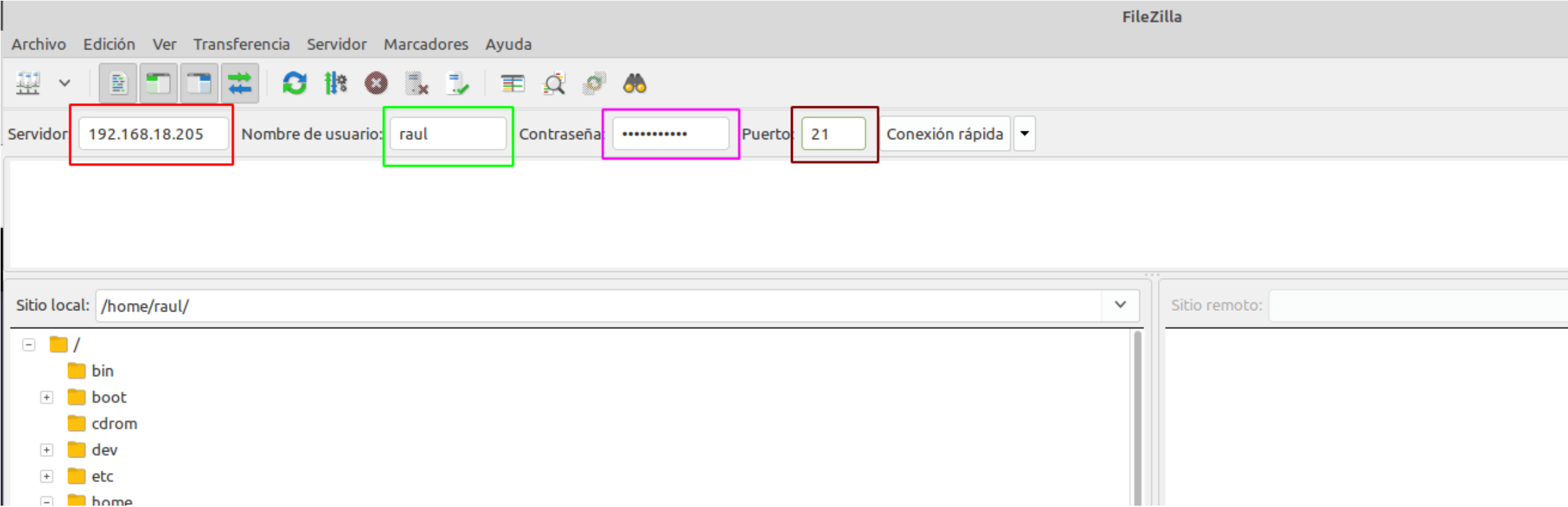
 **Tarea**

Configura un nuevo dominio (nombre web) para el .zip con el nuevo sitio web que os proporcionado. **En este caso debéis transferir los archivos a vuestra Debian mediante SFTP**

Tras acabar esta configuración, ya podremos acceder a nuestro servidor mediante un cliente FTP adecuado, como por ejemplo *Filezilla* de dos formas, a saber:

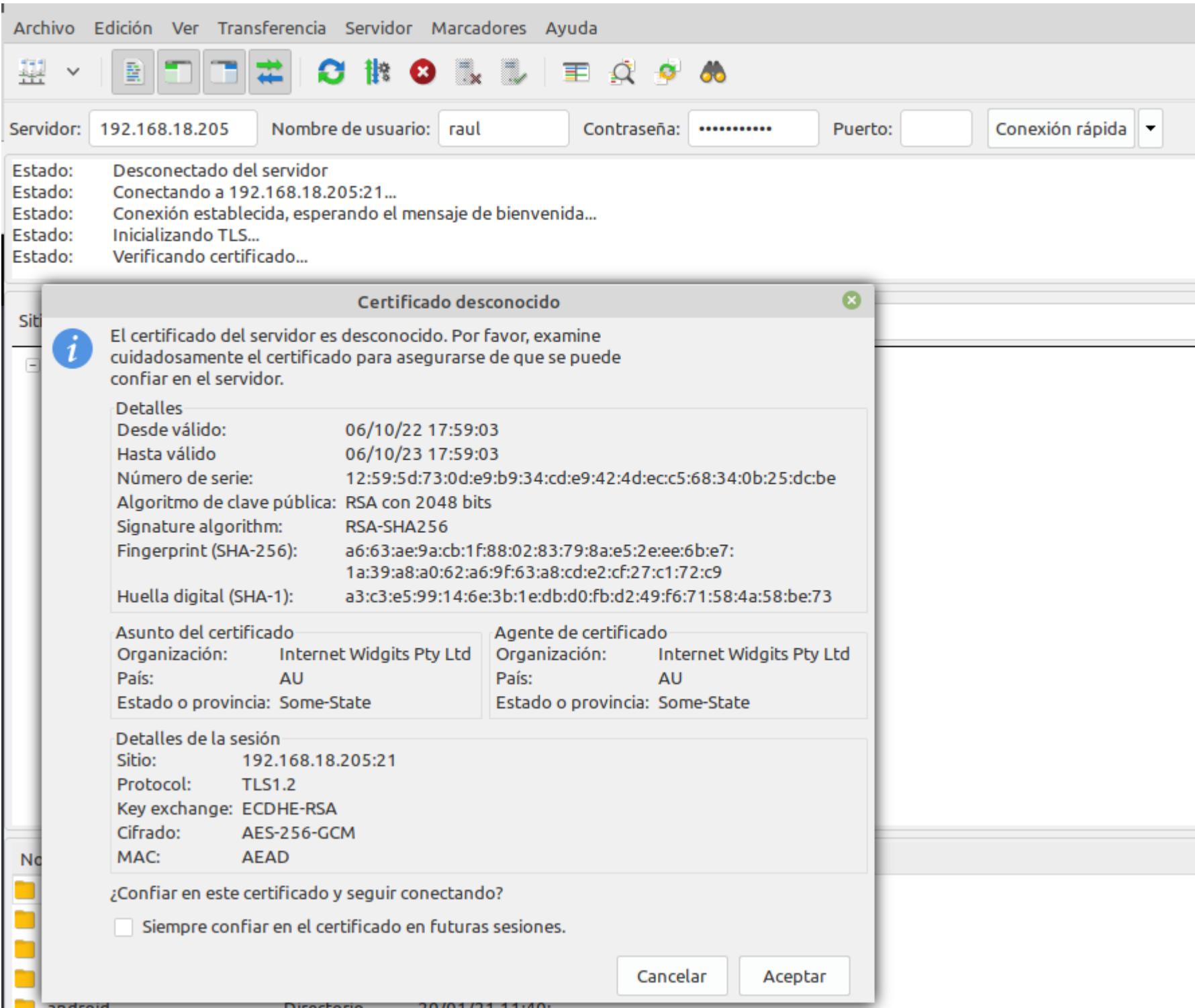
- Mediante el puerto por defecto del protocolo inseguro FTP, el 21, pero utilizando certificados que cifran el intercambio de datos convirtiéndolo así en seguro
- Haciendo uso del protocolo *SFTP*, dedicado al intercambio de datos mediante una conexión similar a SSH, utilizando de hecho el puerto 22.

Tras descargar **el cliente FTP** en nuestro ordenador, introducimos los datos necesarios para conectarnos a nuestro servidor FTP en Debian:



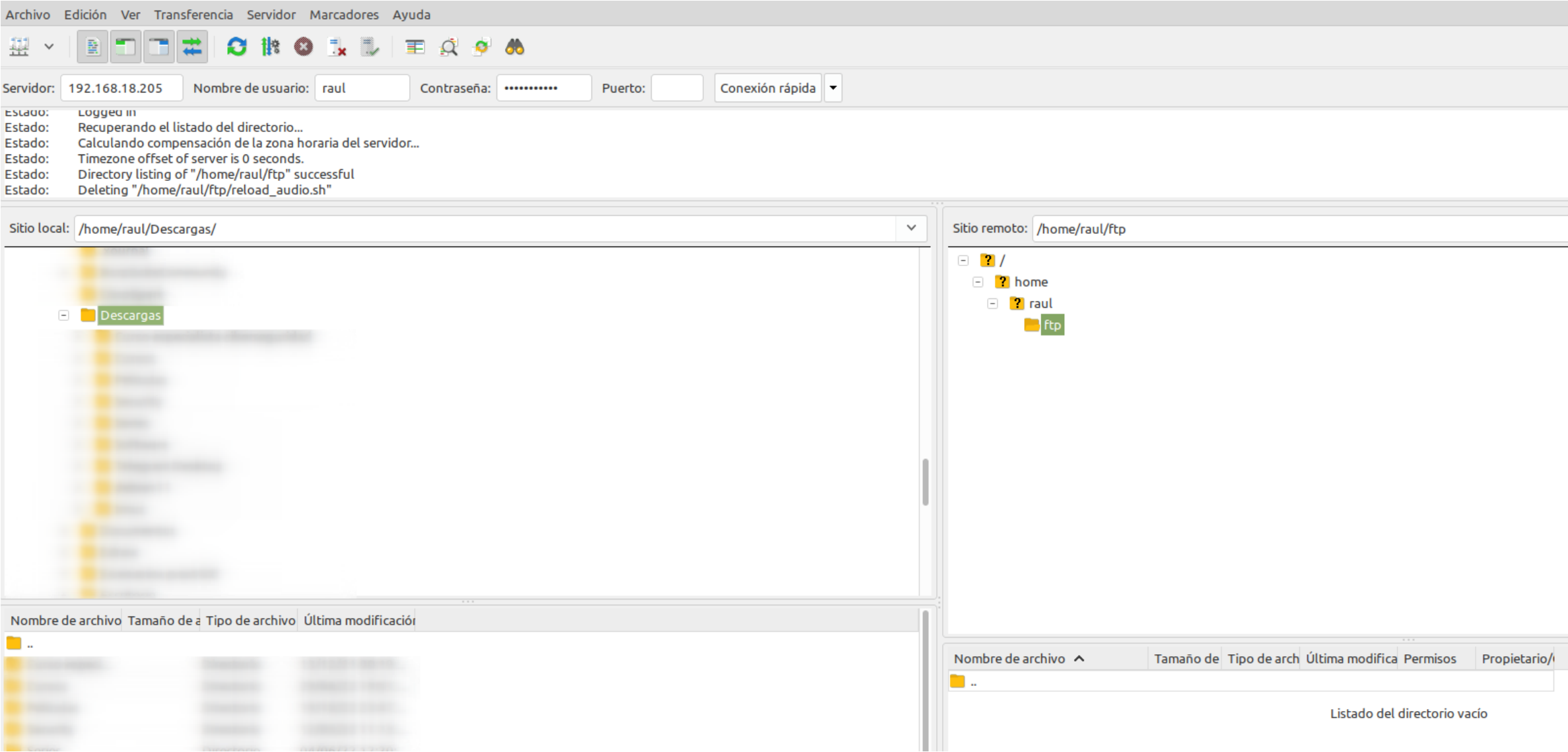
- La IP de Debian (recuadro rojo)
- El nombre de usuario de Debian (recuadro verde)
- La contraseña de ese usuario (recuadro fucsia)
- El puerto de conexión, que será el 21 para conectarnos utilizando los certificados generados previamente (recuadro marrón)

Tras darle al botón de *Conexión rápida*, nos saltará un aviso a propósito del certificado, le damos a aceptar puesto que no entraña peligro ya que lo hemos genrado nosotros mismos:

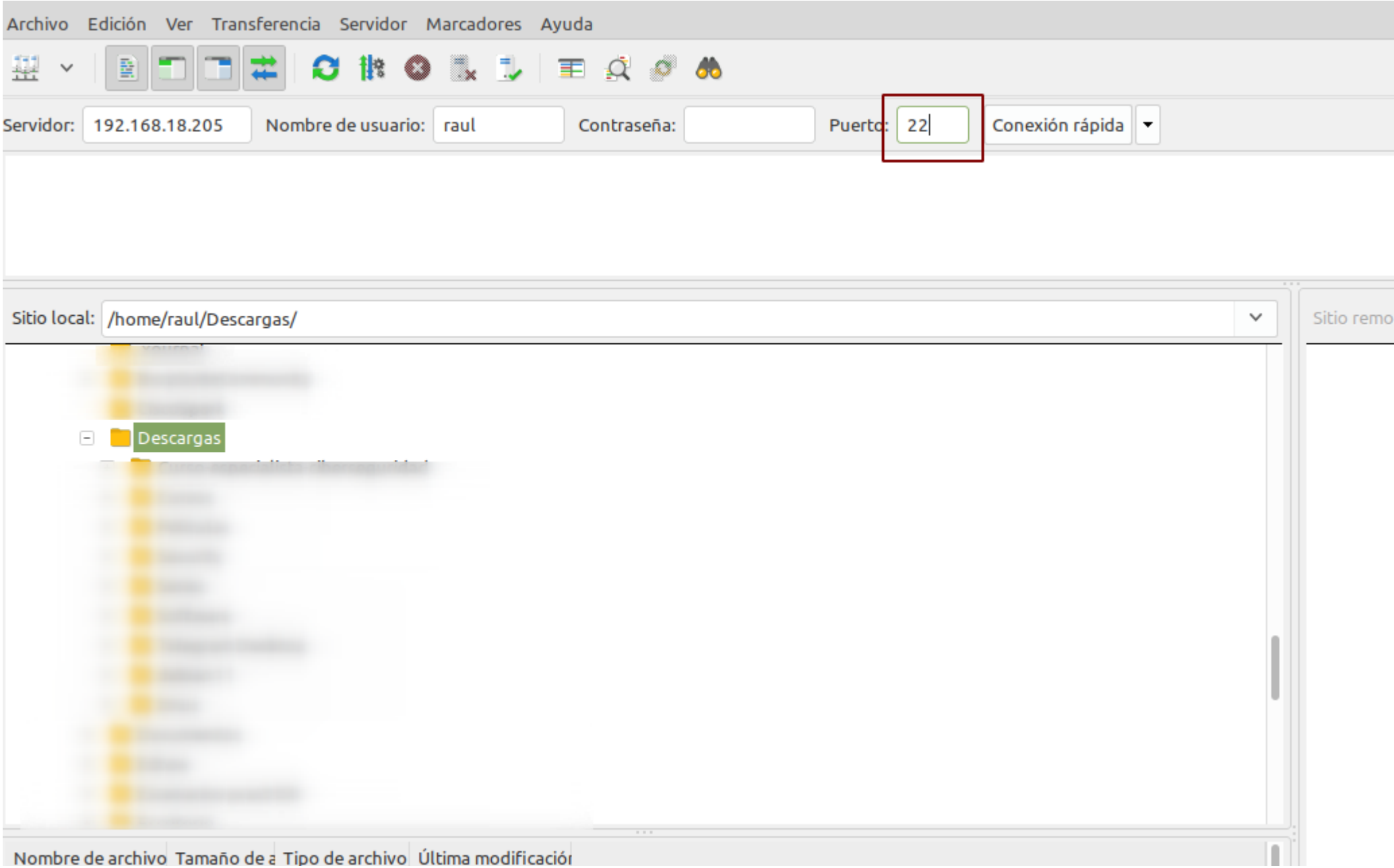


Nos conectaremos directamente a la carpeta que le habíamos indicado en el archivo de configuración `/home/raul/ftp`

Una vez conectados, buscamos la carpeta de nuestro ordenador donde hemos descargado el .zip (en la parte izquierda de la pantalla) y en la parte derecha de la pantalla, buscamos la carpeta donde queremos subirla. Con un doble click o utilizando *botón derecho > subir*, la subimos al servidor.

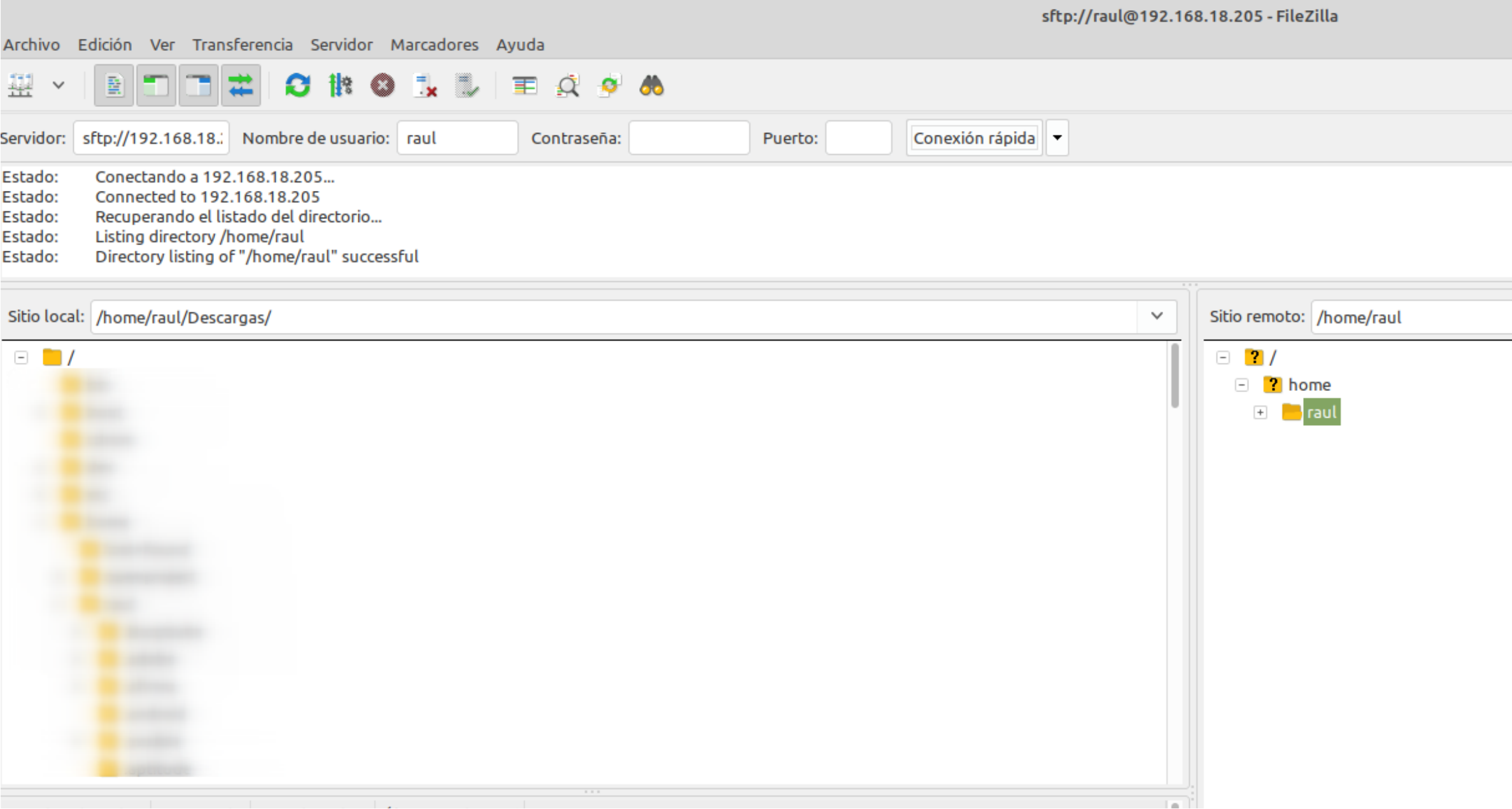
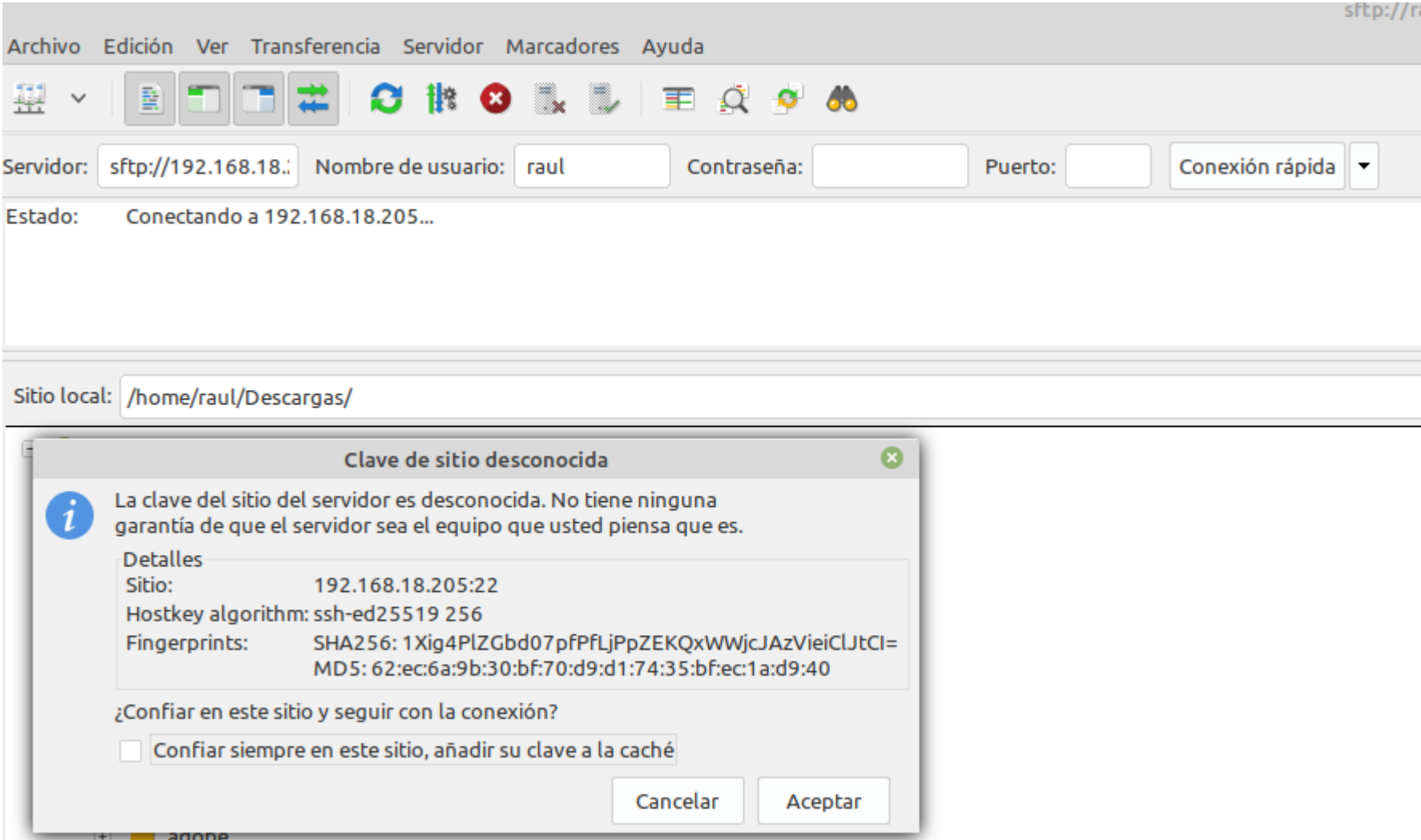


Si lo que quisiéramos es conectarnos por **SFTP**, exactamente igual de válido, haríamos:



Fijáos que al utilizar las claves de SSH que ya estamos utilizando desde la Práctica 1, no se debe introducir la contraseña, únicamente el nombre de usuario.

Puesto que nos estamos conectando usando las claves FTP, nos sale el mismo aviso que nos salía al conectarnos por primera vez por SSH a nuestra Debian, que aceptamos porque sabemos que no entraña ningún peligro en este caso:



Nombre de archivo	Tamaño de a	Tipo de archivo	Última modificaci

Y vemos que al ser una especie de conexión SSH, nos conecta al `home` del usuario, en lugar de a la carpeta `ftp`. A partir de aquí ya procederíamos igual que en el otro caso.

Recordemos que debemos tener nuestro sitio web en la carpeta `/var/www` y darle los permisos adecuados, de forma similiar a cómo hemos hecho con el otro sitio web.

El comando que nos permite descomprimir un `.zip` en un directorio concreto es:

```
unzip archivo.zip -d /nombre/directorio
```

Si no tuvieráis unzip instalado, lo instaláis:

```
sudo apt-get update && sudo apt-get install unzip
```

HTTPS

En este apartado le añadiremos a nuestro servidor una capa de seguridad necesaria. Haremos que todos nuestros sitios web alojados hagan uso de certificados SSL y se acceda a ellos por medio de HTTPS.

Para ello, a modo de prueba de concepto, nos generaremos unos certificados autofirmados y, en el fichero de configuración de nuestros hosts virtuales (los sitios web que hemos configurado), deberemos cambiar los parámetros necesarios.

Apoyaos en una búsqueda en Internet para conseguir vuestro objetivo.

Redirección HTTP a HTTPS

Cuando hayáis cumplido con la tarea de dotar de HTTPS a vuestros sitios web, podréis pasar a esta.

Fijáos que con el estado de la configuración actual, a vuestro sitio web se puede acceder aún de dos formas simultáneas, por el puerto 80 (HTTP e inseguro) y por el puerto 443 (HTTPS, seguro). Puesto que queremos dejar la configuración bien hecha y sin posibles fisuras, vuestro objetivo es que si el usuario accede a vuestro sitio web mediante el puerto 80 (HTTP) automáticamente, por motivos de seguridad, se le redirija a HTTPS, en el puerto 443.

Realizad la búsqueda de información adecuada para conseguir esta redirección automática mediante los cambios necesarios en vuestros archivos de hosts virtuales.

Cuestiones finales

Cuestión 1

¿Qué pasa si no hago el link simbólico entre `sites-available` y `sites-enabled` de mi sitio web?

Cuestión 2

¿Qué pasa si no le doy los permisos adecuados a `/var/www/nombre_web` ?

Evaluación

Criterio	Puntuación
Configuración correcta del servidor web	1 puntos
Comprobación del correcto funcionamiento del primer sitio web	2 puntos
Configuración correcta y comprobación del funcionamiento de una segunda web	1 puntos
Configuración de acceso HTTPS	2 puntos
Configuración de redirección HTTP --> HTTPS	1 punto
Cuestiones finales	1 puntos
Se ha prestado especial atención al formato del documento, utilizando la plantilla actualizada y haciendo un correcto uso del lenguaje técnico	2 puntos