

2º DAW

Despliegue de aplicaciones web

Alberto Serrano Ibaibarriaga

Desarrollo de Aplicaciones Web

Índice

1. Implantación de arquitecturas web.....	2
1.1. Introducción.....	2
1.2. Servicios web.....	3
1.3. Arquitecturas web (redes).....	4
1.3.1. Arquitectura cliente-servidor.....	4
1.3.2. Arquitectura peer-to-peer (P2P).....	6
1.3.3. Arquitectura cliente-servidor de aplicaciones.....	8
1.4. Protocolos.....	9
1.4.1. Protocolo HTTP.....	10
1.4.2. Protocolo HTTPS.....	11
1.4.3. Protocolo FTP.....	12

1. Implantación de arquitecturas *web*

1.1. Introducción

En la **World Wide Web (WWW)** las aplicaciones *web* y demás contenidos siguen unos formatos de datos estandarizados. Gracias a esto, pueden localizarse a través de **navegadores web (web browsers)**, aplicaciones instaladas en el **cliente** que envían peticiones a un **servidor** y éste responde con los datos solicitados.

Los **estándares WWW** especifican muchos mecanismos necesarios para construir aplicaciones *web*, por ejemplo:

- **Modelo estandarizado de nombres:** todos los contenidos de la WWW, se denominan según un **URL (Uniform Resource Locator)**.
- **Especificación de los contenidos:** a todos los contenidos en la WWW se les especifica un determinado tipo, permitiendo de esta forma que los navegadores los identifiquen e interpreten correctamente.
- **Formatos de los contenidos:** todos los navegadores soportan un conjunto de formatos estandarizados, como por ejemplo **HTML (HyperText Markup Language)**, **ECMAScript (JavaScript)** o **CSS (Cascading Style Sheets)**.
- **Protocolos:** estos permiten que cualquier navegador pueda comunicarse con cualquier servidor. El más común en WWW es **HTTP (HyperText Transfer Protocol)**, que opera sobre el conjunto de protocolos **TCP/IP (Transfer Control Protocol/Internet Protocol)**.

Esta infraestructura permite crear y acceder a una inmensidad de aplicaciones y servicios.

1.2. Servicios web

Un **servicio web** es definido por el **World Wide Web Consortium (W3C)** como un sistema de *software* designado para soportar la interacción interoperativa de máquina a máquina a través de una red.

Un servicio *web* realiza una tarea específica o un conjunto de tareas, y se configura mediante una **descripción de servicio** en notación **XML (eXtensible Markup Language)**, denominada **Web Services Description Language (WSDL)**. La descripción de servicio proporciona todos los detalles necesarios para interactuar con el servicio, incluidos los formatos de mensaje (que detallan las operaciones: obtención, eliminación, modificación...), los protocolos de transporte y la ubicación.

Otros sistemas utilizan mensajes **SOAP (Simple Object Access Protocol)**, que se trata de un formato de mensaje XML utilizado normalmente sobre HTTP, para interactuar con el servicio.

La interfaz WSDL oculta los detalles de cómo se implementa el servicio, y el servicio se puede utilizar independientemente de la plataforma de *hardware* o *software* en la que se implementa e independientemente del lenguaje de programación en el que está escrito.

Las aplicaciones basadas en servicios *web* son implementaciones en todas las tecnologías, con acoplamientos flexibles y orientados a componentes. Los servicios se pueden utilizar individualmente o en conjunto con otros servicios para llevar a cabo una operación compleja.

De cara a los servicios *web*, existen tres partes fundamentales:

- **Proveedor del servicio web:** el que lo diseña, desarrolla e implementa, poniéndolo disponible para su uso, ya sea dentro de la misma organización o en público.
- **Consumidor del servicio:** el que accede para obtener lo que el servicio proporciona.

- **Agente del servicio:** el que sirve como enlace entre proveedor y consumidor a efectos de publicación, búsqueda y localización del servicio.

1.3. Arquitecturas web (redes)

Una **arquitectura web** es la **planificación y el diseño de los componentes técnicos, funcionales y visuales de un sitio web, antes de que sea desarrollado e implementado**. Los diseñadores y desarrolladores la utilizan como un medio para ejecutar su trabajo.

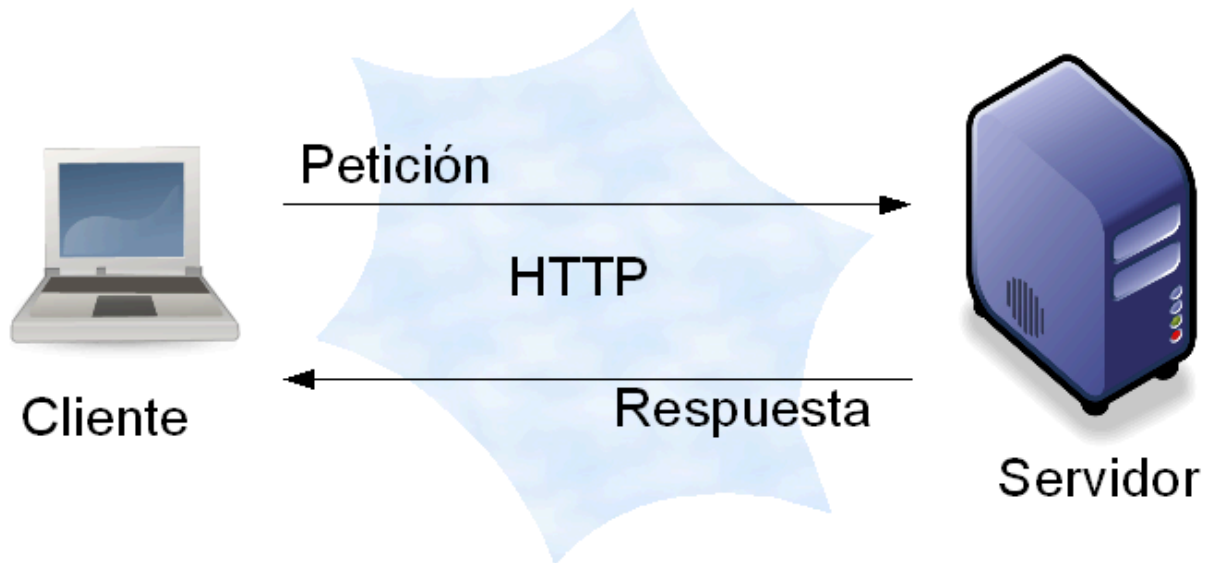
Por lo tanto, se trata de la **estructura conceptual** de la WWW. Esta infraestructura del Internet es posible gracias a los tres principales **protocolos de transmisión** de datos (TCP/IP, HTTP, HTTPS), los **formatos de representación** (HTML, CSS, XML) y los **estándares de direccionamiento** (*Uniform Resource Identifier* o URI, URL).

1.3.1. Arquitectura cliente-servidor

El modelo de desarrollo web se apoya, en una primera aproximación desde un punto de vista centrado en el *hardware*, en lo que se conoce como **arquitectura cliente-servidor**, que define una comunicación donde existen **dos actores, llamados nodos o hosts, el cliente y el servidor**, de forma que el primero es quién conecta con el segundo para solicitar algún servicio.

En el caso que nos ocupa, el desarrollo web, los clientes solicitan que se les sirva una web (archivos HTML, CSS y JS), así como otros datos (archivos de imagen, audio...). Entonces, el servidor se encuentra ejecutándose ininterrumpidamente a la espera de que los diferentes clientes realicen una solicitud. A la solicitud que hacen los clientes se le llama **petición (request)** y a lo que el servidor devuelve a dicho cliente le llamamos **respuesta (response)**.

También hay que tener en cuenta que esta arquitectura cliente-servidor plantea la posibilidad de numerosos clientes atendidos por un mismo servidor. Es decir, **el servidor será un software multitarea** (normalmente ubicado en una máquina específica, con recursos y capacidad destinados exclusivamente para servir) que será capaz de **atender peticiones simultáneas** de numerosos clientes.



Ventajas:

- **Centralización del control:** los accesos, recursos y la integridad de los datos son controlados por el servidor, de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema. Esta centralización también facilita la tarea de actualizar datos u otros recursos.
- **Escalabilidad:** se puede aumentar la capacidad del servidor (incluso fragmentar o clonar las tareas en varios de ellos, que pueden estar distribuidos para abarcar más zonas territoriales) y clientes fácilmente. Eso sí, a más clientes y menos servidores se podrá dar un **cuello de botella**.
- **Fácil mantenimiento:** se pueden dividir y clonar las tareas, tal como se mencionó en el punto anterior, en múltiples ordenadores servidores. Si uno de ellos falla, otros pueden sustituir temporalmente sus funciones. Por otra parte, que se caiga un cliente no supone un problema al resto. Esta independencia de los cambios también se conoce como **encapsulación**.

- **Existen tecnologías suficientemente desarrolladas** para la arquitectura C/S que brindan gran nivel de seguridad a las transacciones, amigabilidad de la interfaz y facilidad de uso.

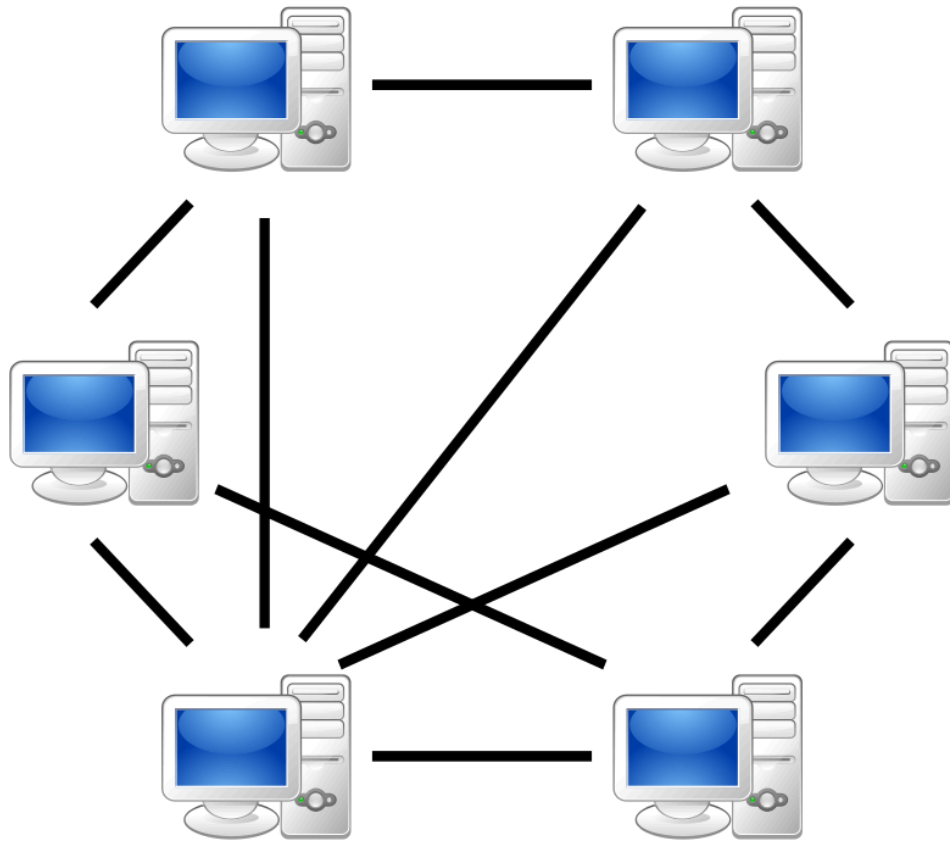
Desventajas:

- **La congestión del tráfico:** cuando una gran cantidad de clientes envían peticiones simultáneas al mismo servidor, puede ser que se sature y ocurra un **desbordamiento (overflow)**.
- **Falta de robustez:** cuando un servidor está caído, las peticiones de los clientes no pueden ser satisfechas. Gran dependencia del servidor.
- **El software y el hardware de un servidor son generalmente muy determinantes:** el *hardware* regular de un PC no es lo suficientemente potente como para servir a tantos clientes. Hay que invertir más para lograr buenos resultados.
- **El cliente no dispone de los recursos que puedan existir en el servidor:** por ejemplo, si la aplicación es una *web*, no podemos escribir en el disco duro del cliente o imprimir directamente sobre las impresoras sin sacar antes la ventana previa de impresión de los navegadores.

1.3.2. Arquitectura *peer-to-peer* (P2P)

La arquitectura **Peer-To-Peer (P2P)** es descentralizada y distribuida, en la que los **nodos** individuales de la red (**peers**) **actúan tanto como servidores como clientes**, en contraste con el modelo cliente-servidor, en el que los nodos clientes acceden a los recursos que proporcionan los nodos servidores.

En una red P2P, las tareas, como realizar un *streaming* de audio o vídeo, se comparten entre múltiples puntos interconectados que ponen una parte de sus recursos (CPU, almacenamiento, ancho de banda) disponibles directamente por otros puntos de la red, sin la necesidad de disponer de servidores que realicen la coordinación centralizada.



Ventajas:

- **Escalabilidad:** millones de usuarios potenciales. Cuantos más nodos estén conectados, mejor será su funcionamiento, al contrario que en el modelo C/S.
- **Robustez:** los clientes pueden localizar los recursos sin depender del correcto funcionamiento de un único servidor.
- **Descentralización:** ningún nodo es imprescindible para el funcionamiento de la red.
- **Distribución de costes:** no vuelca toda la carga sobre un único nodo de la red, responsable de suplir al resto.

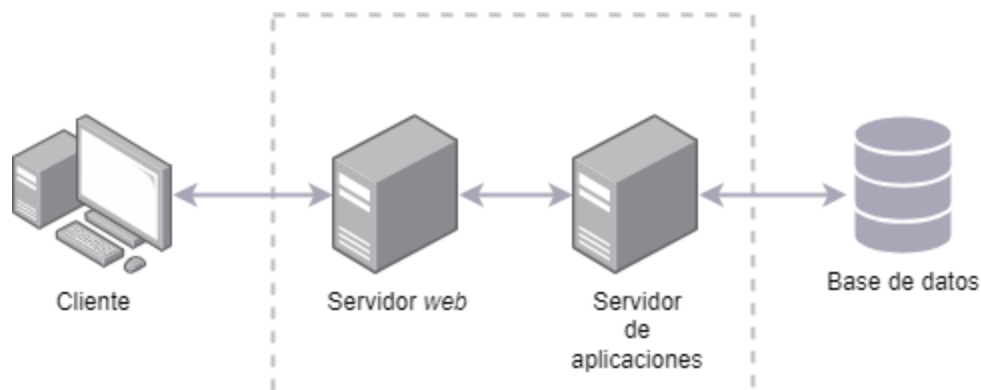
Desventajas:

- **Falta de fiabilidad de los recursos:** no hay garantías de que los recursos obtenidos desde un nodo sean realmente los deseados. En el modelo C/S, el recurso obtenido de un servidor es único, y no ha podido ser modificado por otro nodo.
- **Difícil mantenimiento:** la actualización de los recursos compartidos debe realizarse en todos los nodos.

1.3.3. Arquitectura cliente-servidor de aplicaciones

La función que realiza un servidor de aplicaciones es diferente, ya que **los recursos que va a manipular no son meros archivos estáticos, sino que contienen el código que tiene que ejecutar, pudiendo dar después una respuesta**. Es decir, un servidor *web* en solitario enviaría al cliente el recurso solicitado tal cual (archivo: HTML, XML...), mientras que el servidor de aplicaciones ejecuta una serie de operaciones y envía al cliente el resultado a través del servidor *web*.

El servidor *web* y el servidor de aplicaciones pueden residir en una misma máquina, como se refleja en el siguiente esquema. El servidor *web* recibe la petición de un recurso, y si este corresponde con un recurso dinámico, transfiere la parte correspondiente al servidor de aplicaciones, el cual devolverá de nuevo al servidor *web* el recurso de respuesta, que será devuelto al cliente.

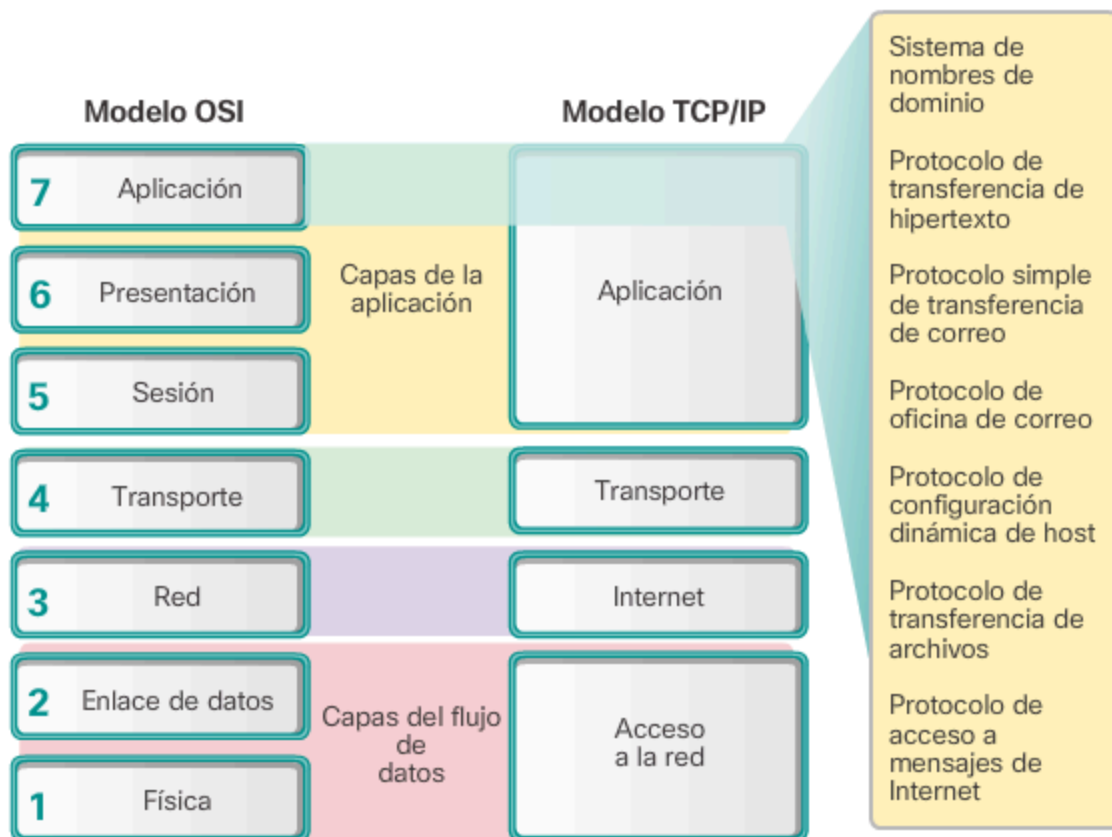


Es muy frecuente que el servidor de aplicaciones deba **conectarse con una base de datos** para obtener la información que requiera el cliente. Dicha base de datos puede residir en la misma máquina que el servidor o en otro *host*.

Por supuesto, ambos servidores también pueden estar separados, incluso pueden no pertenecer a la misma entidad. De igual forma, es posible que un servidor *web* pueda contactar con varios servidores de aplicaciones, y, por tanto, distintas bases de datos.

1.4. Protocolos

En el tema que nos ocupa, sólo nos fijaremos en la última capa del **modelo por capas de Internet (pila TCP/IP)**, la **capa de aplicación**. Es en esta es donde se encuentran los protocolos de la *web*, los que usan navegadores y servidores (*web* y aplicaciones) para comunicarse.



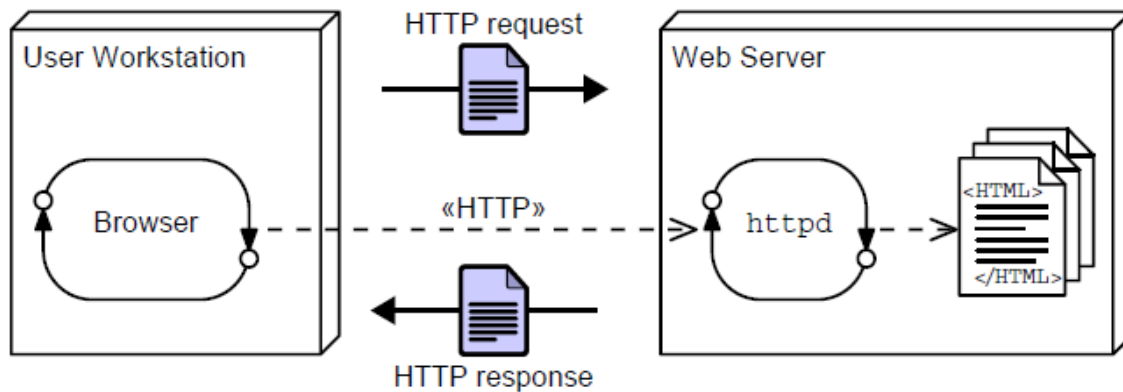
A destacar:

- **HTTP:** *HyperText Transfer Protocol*. Protocolo de comunicación para la web.
- **HTTPS:** *HTTP Secure*. Protocolo seguro de comunicación para la web. Surge de aplicar una capa de seguridad, utilizando **SSL/TLS** (protocolos criptográficos), al protocolo HTTP.
- **Telnet:** protocolo que establece una línea de comunicación basada en texto (consola) entre un cliente y un servidor. Desde su aparición, se utilizó ampliamente como vía de comunicación remota con el sistema operativo, ya que permitía la ejecución remota de órdenes. Con el tiempo ha ido cayendo en desuso a favor de un protocolo seguro que lo sustituye: SSH.
- **SSH:** *Secure Shell*. Protocolo seguro de comunicación ampliamente utilizado para la gestión remota de sistemas, ya que permite la ejecución remota de comandos. Surge como reemplazo para el protocolo no seguro Telnet.
- **SCP:** *Secure Copy*. Es un protocolo seguro basado en RCP (*Remote Copy*), que permite transferir ficheros entre un equipo local y otro remoto, o entre dos equipos remotos. Utiliza SSH, por lo que garantiza la seguridad de la transferencia, así como de la autenticación de los usuarios.
- **FTP:** *File Transfer Protocol*. Es un protocolo que se utiliza para la **transferencia de archivos entre un equipo local y otro remoto**. Su principal problema es que, tanto la autenticación como la transferencia, se realizan en texto plano, por lo que se considera un protocolo no seguro.
- **SFTP:** SSH FTP. Es una versión del protocolo FTP que utiliza SSH para cifrar, tanto la autenticación del usuario, como la transferencia de los archivos. Es la opción segura al uso de un protocolo como FTP.

1.4.1. Protocolo HTTP

El protocolo HTTP es un protocolo para la **transferencia de páginas web** (hipertexto) entre los clientes (navegadores web) y un servidor web. Cuando un usuario, a través del navegador, quiere un documento (página web), este lo solicita mediante una petición

HTTP al servidor. Este último le contestará con una respuesta HTTP y el documento, si dispone de él.




Hay que tener en cuenta que, al contrario que el resto de protocolos que estamos viendo en esta parte, **HTTP no tiene estado**. Eso significa que un servidor *web* no almacena ninguna información sobre los clientes que se conectan a él. Así, cada petición/respuesta supone una conexión única y aislada. En cualquier caso, utilizando ciertas tecnologías en el lado servidor, es posible hacer aplicaciones *web* que puedan establecer sesiones o *cookies* para almacenar ese estado y “recordar”, de alguna manera, a los clientes en sucesivas conexiones.

1.4.2. Protocolo HTTPS

El protocolo **HTTPS (HTTP Secure)** es un protocolo de comunicación segura a través de Internet. Se basa en la comunicación del protocolo HTTP, pero añade una capa de seguridad adicional, cifrando el contenido a nivel de transporte con TSL (*Transport Layer Security*) o SSL (*Secure Socket Layer*). TSL es un sucesor mejorado de SSL.

Su principal utilidad es el cifrado de los mecanismos de autenticación en la *web*, justamente cuando el usuario envía sus credenciales al servidor para validar su sesión. Es el momento más crítico en una comunicación, aunque actualmente se está utilizando abiertamente durante toda la comunicación entre cliente y servidor en la *web* por cuestiones de privacidad e integridad.



Con respecto a la integridad, la autenticación que brinda HTTPS evita así ataques como el de *Man-in-the-Middle* (MitM). De esta forma, podemos estar seguros de que el sitio con el que estamos comunicándonos es el que creemos que es.

1.4.3. Protocolo FTP

El protocolo **FTP (File Transfer Protocol)** es un protocolo para la transferencia de ficheros entre dos máquinas: una máquina cliente y otra máquina remota o servidor, donde se alojan dichos ficheros. La idea es que la máquina remota sirva como repositorio de información y sean los múltiples clientes los que se conecten a ella para descargar o subir ficheros.

También existe su versión segura, **FTPS (FTP SSL)**, que implementa cifrado en la comunicación mediante SSL. Por otro lado, también se creó **SFTP (SSH FTP)**, que es FTP implementado sobre la tecnología de SSH.