TEMA 1.2: ARQUITECTURA DE RED VPS/SSH

1. SSH → SECURE SHELL

Es un protocolo o servicio que hace posible que, a través de la red podamos acceder a otro equipo de manera segura.
Funcional a nivel de aplicación y está vinculado al puerto 22 mediante tcp. La información se envía de forma cifrada. Dos tipos:

-CIFRADO SIMÉTRICO

Tanto el emisor como el receptor tienen **una única** clave común y secreta.
Se utiliza para cifrar y descifrar toda aquella información que enviemos y recibamos.
La información se transmite y en principio es ilegible para un tercero, puesto que previamente ha sido cifrada. Pero, al inicio de la conexión entre emisor y receptor hay un intercambio de la clave (y esto no se hace de forma cifrada)
Esta clave **se comparte al inicio de la conexión**, en este momento es cuando un tercero podría interceptar la clave y descifrar la información que se envíe..

-CIFRADO ASIMÉTRICO

En este caso, el emisor y el receptor **tienen 2 claves,** una **pública** y otra **privada,** una cada uno. (La clave pública puede ser compartida)
La clave privada debe permanecer siempre en local, contenida en un archivo (puede estar protegido por una contraseña). Esta clave en ningún momento de la conexión ssh va a ser compartida.

1. **Si ciframos con la clave pública, sólo** podrá ser descifrada **con la clave privada del receptor.** Esto hace que la información solo la pueda leer el destinatario.
2. **Si ciframos con la clave privada**, esta podrá ser descifrada **con la clave pública.** (Firma digital) Se trata de autenticar de que una persona ha escrito un mensaje.

Cuando un cliente envía un mensaje y lo cifra con su clave pública, el servidor lo descifra con su clave privada. Si lo manda con la clave privada, se descifra con la pública. Para clave digital.

Mensaje: sólo puede ser descifrado por C. Pública si ha sido cifrado por C. Privada y viceversa.

2. VPS → VIRTUAL PRIVATE SERVER

Es una forma de obtener un servidor para nuestras aplicaciones o páginas web de forma barata. Cuando contratamos un VPS, nos asignan una serie de recursos enteramente a nuestra disposición (memoria y procesador). Es el término medio entre un servidor compartido al uso y un servidor exclusivo. Es la opción para páginas con un nivel de tráfico mediano.
Debian es servidor y Ubuntu es cliente.

TEMA 2: ARQUITECTURA DE WEB. IMPLANTACIÓN Y ADMINISTRACIÓN DE SERVIDORES WEB.

1. INTRODUCCIÓN

Con la evolución y el acceso libre a internet, ha surgido la publicación de pgs. web donde se almacena y consulta contenido.
Es un punto donde anunciar productos, dar publicidad a aficiones o capacidades personales...

Las pgs web (su mayoría en formato HTML), requieren ser alojadas en máquinas que dispongan de espacio en disco para almacenar imágenes, archivos HTML, códigos o archivos de video en directorios específicos así como, deben ser capaces de entender todo tipo de extensión de archivos.

Las páginas deberán estar diseñadas (como medida de seguridad) considerando la incorporación de protocolos de comunicación seguros, como, el protocolo de transferencia de hipertexto(HTTPS) que utiliza claves y estrategias de cifrado propias de las herramientas del protocolo de capa de conexión segura (SSL).

-Servidores web → máquinas que alojan las pgs web. Los requerimientos más relevantes son **el espacio del disco** necesario para poder almacenar **y una buena conexión a internet.**

-Funcionamiento servidores web → podemos estar un tiempo sin peticiones o, de repente, tener una avalancha de peticiones. Esto hace que suelen tener un número bajo de procesos en espera. A medida que sean necesarios, se van arrancando nuevos. No todas las peticiones consumen lo mismo. **Aquellas páginas web que ejecuten programas de interacción con el usuario o requieran cifrado (HTTPS) consumen más recursos.**

2. SERVIDORES WEB

Sirven para almacenar contenidos de internet y facilitar su disponibilidad de forma constante y segura. Si visitamos una página web, es en realidad un servidor web el que envía los componentes individuales de dicha página a nuestro ordenador. (para que una página sea accesible en cualquier momento, el servidor web debe estar permanentemente online). **Toda página necesita un servidor especial para su contenido.**
Independientemente de si tienes servidor web propio o de si se alquila uno externo (grandes empresas cuentan con uno propio y administradores, la mayoría, uno externo), siempre se necesita un software para gestionar los datos de la página y mantenerla actualizada. Posibilidad de elegir entre varias soluciones de software para servidores para diferentes aplicaciones y SO.

2.1 Tecnología de servidores web

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

Abrir página web → el usuario pone la URL, el navegador envía una solicitud al servidor web, quien responde, por ejemplo, entregando una página HTML. Ésta puede estar alojada como documento estático o ser generada de forma dinámica, en lo que el servidor web tiene que ejecutar el código de programa(ej: java o PHP) antes de tramitar su respuesta. El navegador interpreta la respuesta, lo que genera automáticamente más solicitudes al servidor a propósito de archivos CSS, imágenes integradas...

-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.

-Un servidor web puede entregar los contenidos simultáneamente a varios pcs o navegadores web. La cantidad de solicitudes(request) y velocidad dependen entre otras cosas del hardware y la carga (nº de solicitudes) del host. La complejidad también es importante, los contenidos dinámicos necesitan más recursos que los estáticos.

-La selección del equipo adecuado para el servidor y la decisión de si debe ser dedicado, virtual o compartido, se debe hacer pensando siempre en evitar sobrecargas en el servidor. 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 servidor, por lo que la web puede que no esté disponible durante ese periodo de inactividad.

3. PROTOCOLO HTTP

3.1 Historia

Es el motor que da vida a Internet. Es la base para la web(www)

-Protocolo HTTP → basado en el envío de mensajes entre cliente y servidor.

-La web fue creada en 1989 en el CERN. A raíz de la necesidad de disponer de múltiples grupos de científicos repartidos por el mundo y colaborando entre ellos (envío de informes, esquemas...) nació la web.

-Mediados de 1990 → los inicios del protocolo HTTP. Encontramos la **versión 0.9 (consistía en una sola línea),** que su única finalidad era transferir datos por internet en forma de páginas web escritas en HTML. En la **versión 1.0** surgió lo de transferir mensajes con encabezados que describen el contenido de los mensajes.

3.2 Versiones

PRIMERA VERSIÓN: HTTP/1 → el servidor no hacía más que transferir el archivo solicitado, solo podía manejar archivos HTML.

PRIMERA ESTÁNDAR OFICIAL: HTTP/1.1 → añadió mejoras:

-Una **conexión** podía ser **reutilizada.**

-**Enrutamiento se añadió a la especificación,** permitiendo realizar una segunda petición de datos, antes de que fuera respondida la segunda, **disminuyendo la latencia de comunicación.**
-**Las respuesta a peticiones podían ser divididas en sub-partes.**
-**La negociación de contenido, se añadieron a la especificación,** permitiendo que servidor/cliente acordasen el contenido a intercambiarse.
-**Gracias a la cabecera Host** pudo ser posible alojar varios dominios en la misma dirección IP.

PROTOCOLO DE MAYOR RENDIMIENTO: HTTP/2 → las webs se volvían más amplias y complejas. Se tiene que solicitar muchos megabytes de datos y enviar hasta 100 solicitudes HTTP.
HTTP/1.1 está pensado para procesar solicitudes una tras otra en una misma conexión, de manera que, cuanto más grande la página, más tarda en cargarse.
Por ello, se desarrolló el SPDY o Speedy, que permitió que se desarrollara en 2015, HTTP/2, que **acelera la carga de las páginas web** (42% de páginas web usan esta versión).

HTTP/3 → un punto débil de todas las versiones HTTP anteriores es el TCP (protocolo de control de transmisión) en el que se basan. TCP requiere que el receptor de cada paquete confirme la recepción antes de enviar otro paquete. Basta con que se pierda un paquete, para que el resto tengan que esperar a que dicho paquete sea transmitido de nuevo.
Para evitarlo, **HTTP/3 no funciona con TCP sino con UDP** (que no aplica este tipo de medidas). Después de UDP, se creó el protocolo QUIC, que será la base de HTTP/3.

3.3 El funcionamiento de HTTP

Gráficamente podemos resumir el proceso de comunicación HTTP como sigue:

-Usuario accede a una URL, seleccionando un enlace de un HTML o introduciendo directamente en el campo del cliente web.

-El cliente web descodifica la URL, separando en partes: el protocolo de acceso, dirección DNS o IP del servidor, el posible puerto opcional y el objeto requerido del servidor. http://direccion[:puerto][path]

Ejemplo: http://www.unaweb.com/documento.html

-Se abre 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, la dirección del objeto requerido(contenido URL), versión del protocolo HTTP y un conjunto dr variable de información (datos opcionales para el servidor, capacidades del navegador(browser)...)

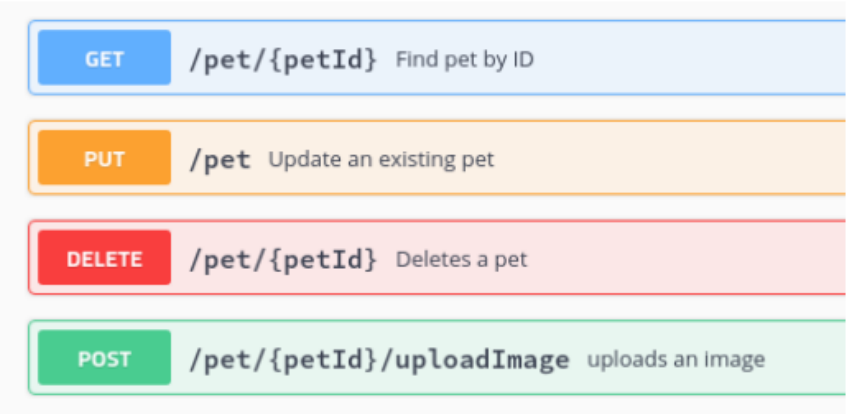
-El servidor devuelve respuesta al cliente, que consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.

-Se cierra la conexión TCP. Este proceso se repite en cada acceso al servidor HTTP.

EJEMPLO: 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).

3.3.1 Comandos o métodos HTTP

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



-HTTP/1.0 recoge 3 comandos para op. de envío y recepción de información y chequeo de estado:

- ➔ GET → para solicitar información/recurso al servidor. Cuando se pulsa sobre un enlace o se teclea una URL, el servidor HTTP envía el recurso correspondiente.
- ➔ HEAD → para solicitar información sobre el recurso(tamaño, tipo...). Usado por los gestores de cachés de páginas o servidores proxy, para saber cuando es necesario actualizar. Se puede comprobar la última fecha de modificación de un recurso antes de traer una nueva copia del mismo.
- ➔ POST → para enviar información al servidor. (por ej. los datos de un formulario). El servidor pasará esta información a un proceso encargado de su tratamiento.

-HTTP/1.1 incorpora:

- ➔ OPTIONS → devuelve los métodos HTTP que el servidor soporta para una URL específica. Para comprobar, por ejemplo, la funcionalidad de un servidor web mediante un recurso específico.
- ➔ DELETE → eliminar un recurso especificado en la URL,aunque pocas veces será permitido por un servidor web.
- ➔ TRACE → permite sondear para saber todos los dispositivos de red por los que pasa nuestra petición. Sabremos si pasan por dispositivos intermedios o proxys.
- ➔ PUT → inverso a GET. Nos permite poner un recurso en la URL que se especifique. Si no existe el recurso, lo crea y si no, lo reemplaza.

  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 → conjunto de líneas que el navegador envía al servidor. **Incluye:**

- Recurso solicitado, método que se aplicará y versión del protocolo.

- Campos del encabezado de solicitud → conjunto de líneas opcionales que permiten aportar información adicional sobre la solicitud y/o cliente (navegador, SO...). Estas líneas están formadas por el **nombre que describe el tipo encabezado, dos puntos (:) y el valor del encabezado**.
- Cuerpo de la solicitud → conjunto de líneas opcionales separadas de las líneas precedentes por una línea en blanco y que, por ej, permiten la transmisión de datos al servidor de un formulario a través de POST.

Una respuesta HTTP → conjunto de líneas que el servidor envía al navegador. Incluye:

- Línea de estado donde figura la versión del protocolo usada, código de estado/error y un texto con el significado de dicho código.
- Posibles códigos de estado se identifican con números de tres cifras y se clasifican en cinco grupos según informativos: (1xx), de éxito en la solicitud(2xx), para redireccionar la solicitud(3xx), por error generado en el cliente(4xx) o por errores generados por el servidor(5xx).
- Campos del encabezado de la respuesta → conjunto de líneas opcionales que aportan información adicional sobre la respuesta y/o el servidor.
- El cuerpo de la respuesta que contiene el recurso (objeto) solicitado.

3.3.2 Tipos MIME

-Se crearon los tipos MIME, para dar formato a mensajes no-ASCII, de forma que pudieran ser enviados por Internet e interpretados correctamente.

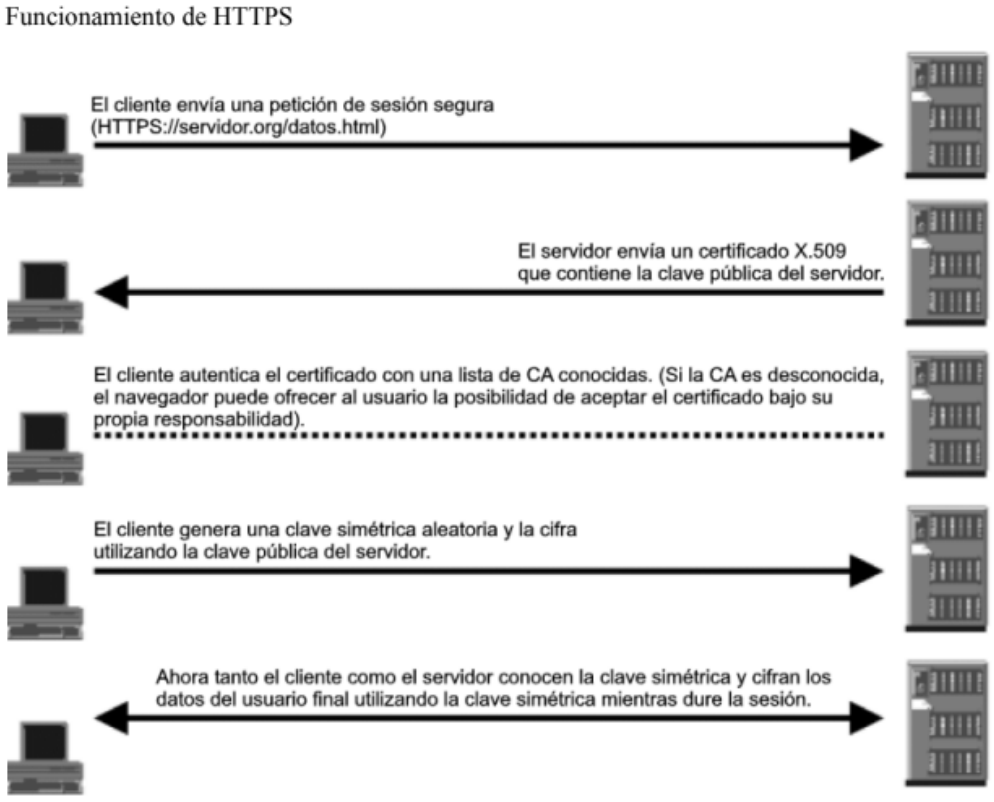
-Tipos de medios de internet, previamente conocidos como "tipos" o "tipos de contenido", es un estándar diseñado para indicar el tipo de información que presenta un archivo o conjunto de datos. Es un identificador útil para conocer el tipo de archivo antes de descargarlo y tener acceso a él.

Todo identificador tiene este formato:

Así pues, el "tipo" y el "subtipo" deben estar presentes en cualquier tipo de medio de Internet. Algunos ejemplos:

3.4 HTTPS

Es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto, es decir, la versión segura de HTTP.



4. APACHE & NGINX

Cuando vamos a poner en marcha un servidor web, lo primero que necesitamos es utilizar un SO (más del 95% es Linux), así como un software que se encargue de la gestión de BD(MYSQL habitualmente) y un software para gestionar el contenido dinámico de la web (PHP suele ser), así como un servidor web, los más conocido Apache y Nginx.
Nginx está orientado a mejorar el rendimiento, soportan mayores cargas de tráfico y usuarios que Apache, además de ofrecer funcionalidades como hacer de proxy. En sus orígenes era especialmente eficiente ofreciendo contenido estático.

La arquitectura de Apache está basada en la creación de un nuevo proceso por cada solicitud que se recibe, esto hace que consuma muchos recursos y reduce su escalabilidad. En cambio, Nginx tiene una arquitectura diferente, está orientada a eventos y posibilita que con cada proceso Nginx pueda atender muchas peticiones de usuarios. Con esto, Nginx pretende solventar el problemas de las 10mil conexiones es decir, la atención a muchas peticiones recurrentes. Nginx utiliza muchos menos recursos de CPU y de RAM.
La desventaja de este enfoque es que cuando un error tiene lugar en un proceso, afectará a todas las peticiones que ese proceso está atendiendo.

Nginx es muy buen servidor de páginas estáticas. En páginas dinámicas, Apache es claramente mejor.

TEORÍA PRÁCTICA 3:

¿Qué es un proxy? → Cuando utilizamos un proxy, vamos a pasar por un servidor proxy previamente a solicitar los recursos que necesitamos a un servidor web. La utilización de proxy nos aporta:

- Acceso a los sitios web que estén bloqueados en nuestro país. Si utilizamos el proxy, estas restricciones no surtirán efecto.
- Restricciones del acceso a determinados sitios web.
- Privacidad. Cuando accedemos a un sitio web por medio de un proxy, en el sitio web quedarán registradas las direcciones del proxy y no la nuestra como clientes. Por tanto, el rastreo es más complicado.

¿Qué es un proxy inverso? → el concepto es lo mismo que el anterior, solo que el proxy inverso se sitúa frente a otros servidores (llamados servidores origen). Estos servidores origen no son vistos por clientes. Nos permite los siguiente:

- Realizar un balanceo de carga: distribuye de forma equitativa las peticiones que recibe entre los servidores origen.
- Puede servir como un servidor de caché → en el proxy inverso se cargan partes estáticas de las webs, de forma que quitamos carga de trabajo a los servidores de origen.
- Privacidad y protección → como los clientes no pueden ver directamente los servidores origen, éstos van a permanecer más protegidos puesto que será más difícil que reciban ataques.