

Implantación, configuración y administración de servidores web.

Caso práctico



, con su perfil de Técnico Superior en Desarrollo de Aplicaciones Web, va a

Por ese motivo, ha decidido

interna para los miembros de la empresa, los conocimientos que va a ir adquiriendo durante su vida laboral respecto a los trabajos que le toque realizar. De este modo, a los compañeros y compañeras que le sucedan o que tengan que realizar labores similares les será útil esta información.

Juan ha pensado en incluir un tema titulado **Implantación de arquitecturas web** que estructurará en varios apartados.

Para realizar este trabajo va a contar con la colaboración de sus amigos **Carlos** y **Ana**, estudiantes ambos de Ciclos de Informática.

le ha surgido un nuevo proyecto: una empresa con varias sucursales quiere montar una aplicación web por sucursal.

Ada, la directora, considera que para afrontar este proyecto y atender así la **configurar un nuevo equipo servidor, de cuya administración también se encargarán desde BK Programación**.

Para tal fin se reúne con **María** y con **Juan**.





**Materiales desarrollados y actualizados por el profesorado de la
Junta de Andalucía**

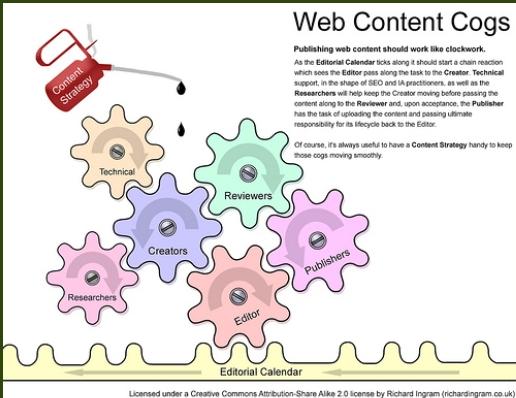


[Aviso legal](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

1.- Fundamentos de la web.

Caso práctico



[Richard Ingram \(CC BY-SA\)](#)

Juan, después de hablar con María, ha comenzado la



documentación de la wiki explicando los aspectos básicos de la arquitectura web, ya que considera que es un medio que es necesario conocer con precisión si se pretende realizar el despliegue de una aplicación web. Para ello ha pensado en

definir conceptos básicos de las interfaces web.

Cuando se habla de la web todo parece muy sencillo, pero detrás hay muchos estándares y protocolos que se ponen en marcha para poder comunicar aplicaciones entre sí. Aplicaciones que se ejecutarán generalmente en equipos diferentes. Una aplicación que se ejecutará en un sistema remoto que llamaremos servidor web y otra aplicación que se ejecutará en el sistema del cliente llamado navegador web.

El conjunto de tecnologías subyacentes en la web, basada en su mayoría en estándares abiertos, se ha convertido en el pilar fundamental de Internet. Hoy en día, el uso de estas tecnologías no se limita a la comunicación entre un navegador web y un servidor web, sino que se utiliza incluso para facilitar el intercambio de datos entre aplicaciones a través de lo que conocemos como servicio web.

El mejor ejemplo de intercambio de datos entre aplicaciones a través de tecnologías web es **REST**, donde usando protocolos y estándares ya conocidos como HTTP y XML, se permite la comunicación entre aplicaciones. Con tecnologías como esta podríamos crear una aplicación web (e incluso una aplicación móvil) que se comunicara con otra aplicación web, como por ejemplo Twitter o Facebook.

Reflexiona

Internet se ha convertido en algo imprescindible, pero, ¿qué tecnologías han hecho que Internet sea imprescindible? Responder a esa pregunta nos lleva a plantearnos otra, ¿qué significa estar en o formar parte de Internet?

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

1.1.- ¿Qué aspectos generales de arquitecturas web debes conocer?

La arquitectura World Wide Web (WWW) de Internet provee un modelo de programación sumamente poderoso y flexible. Las aplicaciones y los contenidos son presentados en formatos de datos estándar y son localizados por aplicaciones conocidas como "web browsers", que envían requerimientos de objetos a un servidor y éste responde con el dato codificado según un formato estándar.

Los estándares WWW especifican muchos de los mecanismos necesarios para construir un ambiente de aplicación de propósito general, por ejemplo:

- ✓ **Modelo estándar de nombres:** todos los servidores, así como el contenido de la WWW se denominan según un Localizador Uniforme de Recursos (Uniform Resource Locator: URL).
- ✓ **Contenido:** a todos los contenidos en la WWW se les especifica un determinado tipo, permitiendo de esta forma que los browsers (navegadores) los interpreten correctamente.
- ✓ **Formatos de contenidos estándar:** todos los navegadores soportan un conjunto de formatos y tecnologías estándar, por ejemplo HTML (Lenguaje de marcas de hipertexto), ECMAScript (JavaScript), CSS, etc.
- ✓ **Protocolos estándar:** éstos permiten que cualquier navegador pueda comunicarse con cualquier servidor web. El más comúnmente usado en WWW es HTTP (Protocolo de Transferencia de HiperTexto), que opera sobre el conjunto de protocolos TCP/IP (Protocolo de Control de Transmisión / Protocolo de Internet).



Esta infraestructura permite a los usuarios acceder a una gran cantidad de aplicaciones y servicios de terceros. También permite a los desarrolladores crear aplicaciones y servicios para una gran comunidad de clientes.

Los aspectos generales a destacar en una arquitectura web son los siguientes:

- ✓ Escalabilidad.
- ✓ Separación de responsabilidades.
- ✓ Portabilidad.
- ✓ Utilización de componentes en los servicios de infraestructura.
- ✓ Gestión de las sesiones del usuario.
- ✓ Aplicación de patrones de diseño.

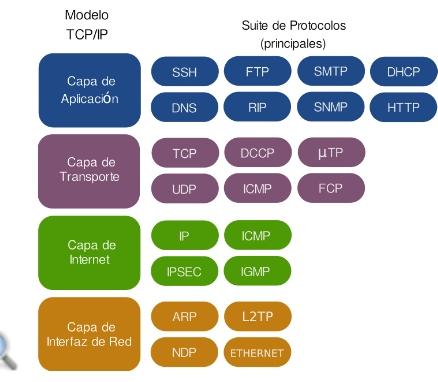
El esquema de funcionamiento de los servicios web requiere de tres elementos fundamentales:

1. **Proveedor del servicio web**, que es quien lo diseña, desarrolla e implementa y lo pone disponible para su uso, ya sea dentro de la misma organización o en público.
2. **Consumidor del servicio**, que es quien accede al componente para utilizar los servicios que éste presta.
3. **Agente del servicio**, que sirve como enlace entre proveedor y consumidor para efectos de publicación, búsqueda y localización del servicio.

1.2.- El servicio web e Internet.

Internet, tal y como la conocemos hoy día, utiliza para su funcionamiento la arquitectura de protocolos TCP/IP. Lo importante de TCP/IP es que permite la comunicación de equipos que están en diferentes redes a través de una arquitectura flexible.

Como supondrás, cuando accedemos a un sitio web a través de un navegador web, el navegador web inicia una o más comunicaciones con un sistema remoto y se produce un intercambio de paquetes que permite mostrar la página web. Veamos el proceso más detalladamente:



[Wikipedia / GISEPROJ CC BY-SA](#)

- ✓ El navegador realiza una petición de un recurso al servidor web (imagen, HTML, etc.).
- ✓ El servidor web recibe la petición y la procesa. Procesar la petición en el servidor web puede significar dos cosas diferentes:
 - ◆ Servir un recurso estático, es decir, el contenido de un archivo almacenado en el disco del servidor.
 - ◆ Ejecutar una aplicación en el servidor que genere contenido dinámico.
- ✓ El navegador web recibe el contenido del servidor y es el encargado de mostrarlo al usuario final, a través de un proceso conocido como renderizado. En el proceso de renderizado el navegador web convierte los datos recibidos en información que el usuario puede ver en la pantalla de su ordenador.
- ✓ Si para renderizar y representar la página web el navegador necesita más recursos (imágenes, archivos CSS, etc.), los volvería a pedir al navegador, siguiendo el proceso anterior.

El protocolo que permite todo ese proceso anterior es HTTP. HTTP es el protocolo que permite al navegador realizar una petición y al servidor generar una respuesta, y es importante entenderlo antes de configurar un servidor web.

Pero si ha triunfado la web y ha llegado hasta donde está hoy día, no es solo por el protocolo HTTP, sino también por otras cosas:

- ✓ El sistema DNS, que permite traducir un nombre de máquina en una dirección IP. El sistema DNS nos permite acceder a una máquina remota a través de un nombre (como www.gnu.org o www.debian.org).
- ✓ El uso de URLs (Uniform Resource Locator). Una URL es una nomenclatura pensada para poder nombrar recursos en la red de forma sencilla, por ejemplo: <http://www.gnu.org>.
- ✓ Por el ingenio de muchos desarrolladores y desarrolladoras web que con HTML, CSS y JavaScript, y tecnologías del lado del servidor, consiguen hacer páginas webs increíbles.

De todos estos conceptos, el que vamos a revisar ahora es el de URL. Para entender una URL lo mejor es analizar un ejemplo de URL y desglosarla en partes:

http://es.wikipedia.org/wiki/Localizador_de_recursos_uniforme

En la URL anterior aparecen las siguientes partes:

- ✓ **Esquema.** El esquema es donde pone http: y representa el protocolo usado para acceder al recurso. Los esquemas más usados son http:, https: (protocolo HTTPS) y ftp: (protocolo FTP), aunque hay otros esquemas como tel: (para hacer referencia a un número de teléfono), mailto: (para hacer referencia a un correo electrónico), y bastantes más.
- ✓ **Nombre de máquina o FQDN.** En este caso es es.wikipedia.org y hace referencia a la máquina a la que estamos accediendo.
- ✓ **Ruta hasta el recurso.** En este caso sería /wiki/Localizador_de_recursos_uniforme, y hace referencia al camino a seguir para llegar al recurso.

Si alguna URL necesita **usuario y contraseña** (por ejemplo, si usamos el esquema ftp), entonces podemos escribirlo de la siguiente forma:

- ✓ ftp://anonymous@ftp.rediris.es/mirror/Apache/: donde anonymous sería el nombre de usuario.
- ✓ ftp://anonymous:contraseña@ftp.rediris.es/mirror/Apache/: donde anonymous:contraseña son la combinación del usuario y su contraseña separados por dos puntos.

Y por último, como se verá más adelante, si un protocolo se usa en un puerto diferente al puerto por defecto o estándar, podemos usar el siguiente formato de URL:

- ✓ http://www.ejemplodedominio.com:8080/mirecurso; donde 8080 es el puerto que se usará en este caso para acceder al servicio remoto.

Autoevaluación

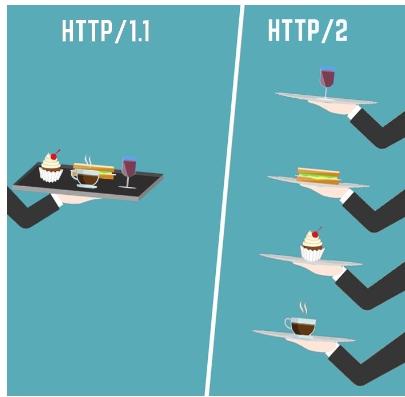
¿Cuál de las siguientes opciones corresponde con el esquema en una URL?

- html:
- ftp:
- www.wikipedia.org
- :8080

1.3.- El protocolo HTTP.

El protocolo HTTP permite realizar la transferencia de información entre un cliente web y un servidor web en Internet, dando lugar a lo que hoy día se conoce como World Wide Web. Este protocolo ha pasado por varias versiones, desde la inicial 0.9 hasta la actual 2.0 (HTTP/2). La versión más extendida actualmente es la versión HTTP/1.1, definido en el RFC 2616, pero la versión 2.0 se irá imponiendo poco a poco de forma natural. Veamos cuáles son las características principales de la versión HTTP/1.1:

- ✓ Es un protocolo del nivel de aplicación, por lo que está destinado a que aplicaciones en sistemas diferentes se comuniquen entre sí.
- ✓ Es un protocolo sin estado, no hay conciencia de las peticiones anteriores, simplemente se hace una petición de un recurso y se obtiene el recurso. Por lo que es un protocolo donde la comunicación consiste en realizar una petición y obtener una respuesta del servidor a dicha petición.
- ✓ Las peticiones de recursos desde un cliente web al servidor web se hacen en formato texto "legible" por un ser humano, también lo son las respuestas por parte del servidor.
- ✓ No existe el concepto de sesión y no es necesario autenticarse para realizar la petición de un recurso.



[Wikipedia / Diseño Web Valencia CC BY-SA](#)

Como puedes ver el protocolo HTTP es muy simple, por lo que cuando desarrollamos una aplicación web es necesario que cosas como la autenticación sea implementada por la aplicación web, dado que el protocolo no provee de mecanismos para ello.

Para el seguimiento del usuario, y en consecuencia, para la gestión de la información de sesión, se utilizan cookies. Las cookies son pequeños archivos de textos almacenados en el sistema del cliente, donde se guarda temporalmente información que proviene del servidor web. La aplicación web generará una "cookie" que el cliente web almacenará durante un periodo de tiempo, y el cliente web volverá a enviar al servidor web dicha cookie cada vez que se haga una petición a la misma aplicación web. De esta forma, la aplicación web que se ejecuta en un servidor web podrá almacenar en el cliente información, como por ejemplo los productos que el usuario visitó anteriormente o el listado de productos que tiene almacenado en el carrito de la compra.

Cuando un cliente web (un navegador por ejemplo) accede a un recurso, se sigue el siguiente procedimiento:

- ✓ El cliente web se conecta al servidor web.
- ✓ El cliente web envía una petición al servidor. Existen diferentes métodos para realizar la petición, los más comunes son los métodos **GET** y **POST**.
- ✓ El servidor envía una respuesta al cliente, donde, si todo ha ido bien, se incluye el tipo de contenido, su longitud y el contenido en sí.

Autoevaluación

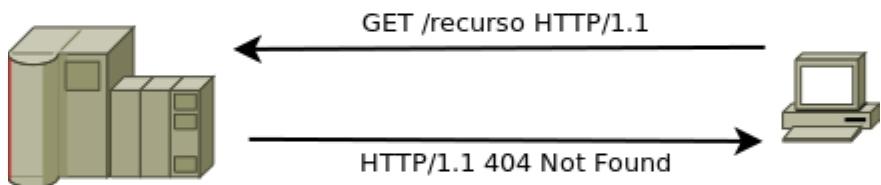
GET corresponde con...

- Una respuesta del servidor.
- Una cabecera de petición.
- Una petición de un cliente.
- Es el código de estado de la petición.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

1.4.- Uso del servicio HTTP.

Ahora que ya conocemos los fundamentos del protocolo HTTP, vamos a profundizar un poco más en él. Para empezar, veamos los principales métodos (también llamado "verbos") usados para realizar una petición al servidor web.



Salvador Romero. Elaboración propia. [CC BY-NC-SA](#)

- ✓ Método **GET**: se usa para obtener un recurso, y en general, no admite que el cliente web envíe datos al servidor. Si queremos usar este método para enviar datos al servidor los datos deben formar parte del recurso solicitado, usando por ejemplo una cadena de consulta (query string). Por ejemplo:
 - ◆ Si ponemos la url `http://www.estoesunejemplo.com/mipagina.php`, se generaría una petición HTTP del estilo a `GET /mipagina.php HTTP/1.1`.
 - ◆ Si ponemos la url `http://www.estoesunejemplo.com/mipagina.php?dato1=a&dato2=2`, se generaría una petición HTTP del estilo a `GET /mipagina.php?dato1=a&dato2=2 HTTP/1.1`. La diferencia con el anterior es que se ha añadido una cadena de consulta que se procesará internamente por la aplicación web (en este caso escrita en `php`) para generar el contenido de forma dinámica.
- ✓ Método **POST**: se usa para enviar datos al servidor, los datos no irán en el recurso solicitado (como es el caso del query string), sino que se enviarán después de las cabeceras de petición (cabeceras, línea en blanco y a continuación los datos). De esta forma, los datos enviados al servidor no se ven en la URL y quedan ocultos a los ojos del usuario. La respuesta del servidor será también contenido que se podrá visualizar en el navegador (generalmente una página web).
- ✓ Método **HEAD**: permite obtener datos de un recurso sin que el servidor lleve a enviarlo al cliente. Se obtendría por ejemplo la fecha de la última modificación de una imagen y su tamaño, lo cual es útil para mantener una caché en el navegador con los contenidos descargados más habitualmente o más recientemente.

Hoy día HTTP va más allá de la simple navegación web (usando un navegador web como Firefox o Chrome). Este protocolo también se utiliza en lo que se conoce como servicio web o Web Service (aplicaciones web que usan o proveen de un Web Service). Un Web Service es un API que permite a otra aplicación comunicarse remotamente para acceder a un servicio. Por ejemplo, con el Web Service de Twitter podemos publicar twits desde una aplicación de escritorio que hayamos realizado, para ello nuestra aplicación web se conectaría al API de Twitter vía HTTP. Las ventajas de los Web Services son innumerables, y no vamos a entrar aquí en detalle, y su base es el protocolo HTTP. En el contexto de un Web Service los métodos HTTP suelen tomar otra perspectiva (sobre todo cuando hablamos de servicios web que implementan un API REST):

- ✓ Método **GET**: se usa para obtener un recurso o entidad, por ejemplo, un mensaje de twitter que alguien haya publicado.
- ✓ Método **POST**: se usa para crear o enviar una entidad a la aplicación web, por ejemplo, un nuevo mensaje de twitter que nosotros vayamos a publicar.
- ✓ Método **PUT**: se utiliza para actualizar o reemplazar una entidad en un servidor web.
- ✓ Método **DELETE**: se utiliza para borrar un recurso o entidad del servidor web.

Existen más métodos interesantes, pero estos son los más usados. Después de realizar una petición como las anteriores, sea el método que sea, el servidor generará una respuesta HTTP que contendrá un código de estado de respuesta. El código de estado 200 OK ya se vio en el apartado anterior, pero existen otros códigos importantes que debes conocer:

- ✓ **1XX:** los códigos de estado en el rango de 100 son **respuestas informativas**.
 - ◆ **100 Continue:** significa que todo va bien por ahora y que el cliente debería seguir con la petición.
- ✓ **2XX:** los códigos de estado en el rango de 200 son respuestas de **éxito en la transferencia**, por ejemplo:
 - ◆ **200 OK:** significa que la respuesta ha tenido éxito y generalmente implica que los datos van en la respuesta (depende del método de la petición).
- ✓ **3XX:** los códigos de estado en el rango de 300 son respuestas de **redirección, el recurso ha cambiado de ubicación**.
 - ◆ **301 Moved Permanently:** significa que un recurso se ha movido a otro sitio de forma permanente. El servidor web incluye una cabecera llamada Location: con la nueva ubicación. El navegador realizará una petición a la nueva ubicación de forma transparente al usuario.
 - ◆ **302 Found:** significa que un recurso se ha movido a otro sitio de forma temporal. El funcionamiento es similar al 301.
 - ◆ Otras respuestas relacionadas con la redirección y el cambio de ubicaciones son la **303 See Other**, **307 Temporary Redirect** y **308 Permanent Redirect**. Tienen connotaciones ligeramente diferentes a los códigos de estado 301 y 302, pero la idea es similar.
- ✓ **4XX:** los códigos de estado en el rango de 400 son respuestas en las que se indica que hay un **error en la petición del cliente** o que algo ha ido mal al procesarla. Veamos algunos ejemplos:
 - ◆ **400 Bad Request:** el servidor no entiende la petición quizás porque está mal formada.
 - ◆ **401 Unauthorized:** es necesario que el usuario se autentique para acceder al recurso y dicha autenticación a fallado.
 - ◆ **403 Forbidden:** el acceso al contenido solicitado no está permitido.
 - ◆ **404 Not Found:** posiblemente el código de estado más conocido. El servidor responde con este código cuando el recurso solicitado no existe o no puede ser encontrado.
- ✓ **5XX:** los códigos de estado que están en el rango de 500 simbolizan que se ha recibido una petición correcta pero que **el servidor no ha podido completar la respuesta**. Veamos algunos ejemplos:
 - ◆ **500 Internal Server Error:** generalmente significa que hay un error en la configuración del servidor y que no se puede completar la respuesta.
 - ◆ **502 Bad Gateway:** suele indicar que el servidor al que se ha solicitado la petición actúa como una pasarela a otro servidor, y el segundo servidor no ha respondido adecuadamente.
 - ◆ **503 Service Unavailable:** el servidor no está disponible en este momento.

Para saber más

Un simple navegador, como Firefox, o Chrome, es una herramienta muy potente para analizar las cabeceras que se envían y reciben cuando se realiza una petición HTTP. Puedes ver como utilizar esta utilidad en Firefox (no hay que instalar nada, dado que ya va incorporado), en el siguiente enlace:

 [Como usar el monitor de red de Firefox.](#)

Para empezar, puedes activarlo con la combinación de teclas Ctrl+Shift+I.

Autoevaluación

Indica si la siguiente afirmación es verdadera o falsa:

Si después de una petición POST el servidor web responde con 404 es porque se debería haber utilizado una petición GET.

- Verdadero Falso

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

1.5.- Protocolo HTTP vs HTTPS.



[OpenClipartVectors](#) CCO

-¿Quieres conservar la información de forma confidencial? ¿Quieres transferir información de forma segura? Si estás pensando en este tipo de preguntas necesariamente estás pensando en el protocolo HTTPS y no en el protocolo HTTP.

El protocolo HTTPS permite que la información viaje de forma segura entre el cliente y el servidor, por la contra el protocolo HTTP envía la información en texto claro, esto es, cualquiera que accediese a la información transferida entre el cliente y el servidor puede ver el contenido exacto y textual de la información.

Para asegurar la información, el protocolo HTTPS requiere de certificados y siempre y cuando sean validados la información será transferida cifrada. Pero cifrar la información requiere un tiempo de computación, por lo

que será perjudicado el rendimiento del servidor web. Así, ¿es necesario que toda, absolutamente toda, la información sea transferida entre el cliente y servidor de forma cifrada? A lo mejor solamente es necesario que sea cifrada la autenticación a dicha información, por eso en algunas páginas web puede que el servidor esté configurado para que en todo el dominio esté cifrada su información o simplemente el intento de acceso a la misma.

Un servidor web, como Apache, puede emitir certificados, pero puede que en algún navegador sea interpretado como peligroso, esto suele ser debido a que los navegadores poseen en su configuración una lista de Entidades Certificadoras que verifican, autentican y dan validez a los certificados. ¿Tú, confiarías en un DNI que no fuese certificado por una entidad de confianza como el Ministerio del Interior? Pues, lo mismo le pasa a los navegadores, solamente confían en quien confían. Eso no quiere decir que no puedes crear tus certificados en un servidor web, de hecho muchas empresas lo hacen, sobre todo para sitios internos o externos en los que solamente puede acceder personal autorizado por la propia empresa. Ahora si, si utilizas certificados mediante Apache en un sitio visible a través de Internet y accesible por cualquier usuario, o bien eres una empresa o entidad en la que de por si confía el usuario o la imagen de la empresa o entidad quedará muy mal parada, ya que lo más probable es que el usuario no aceptará la comunicación, por visionar en el navegador un aviso de problema de seguridad.

El protocolo HTTPS utiliza cifrado sobre SSL/TLS que proporcionan autenticación y privacidad. Entonces, si necesitas que la información viaje cifrada debes emplear el protocolo HTTPS, en caso contrario el protocolo HTTP. Hay que dejar claro que la utilización del protocolo HTTPS no excluye ni impide el protocolo HTTP, los dos pueden convivir en un mismo dominio.



[geralt](#) CCO

Bien, pero, ¿cómo funcionan? Más o menos hemos visto ya el funcionamiento pero vamos a revisarlo para entender la diferencia con HTTPS. Cuando escribes una dirección URL en el navegador, por ejemplo http://www.debian.org/index.es.html, antes de ver la página en el navegador existe todo un juego de protocolos que entran en acción, sin profundizar en todos ellos, vamos ver el proceso que ocurre:

- ✓ Se traduce el dominio DNS por una dirección IP (en este caso www.debian.org se traduce a una dirección IP).
- ✓ Una vez obtenida la dirección IP se accede al sistema remoto que tiene dicha IP, concretamente al puerto 80, que es el puerto TCP asignado por defecto al protocolo HTTP.
- ✓ El cliente web entonces realiza la petición HTTP, solicitando un recurso (en este caso podría ser GET /index.es.html HTTP/1.1, por ejemplo).
- ✓ Si el servidor web aloja la página, ésta será transferida a tu navegador en la respuesta HTTP.

Sin embargo cuando escribes en el navegador una dirección URL con llamada al protocolo HTTPS, el procedimiento es similar al anterior pero un poco más complejo:

- ✓ Se traduce el dominio DNS por una IP.
- ✓ Una vez que se obtiene la IP, se busca en el servidor web que aloja la página solicitada el puerto 443, puerto TCP asignado por defecto al protocolo HTTPS.
- ✓ Pero ahora, antes de transferir la página a tu navegador se inicia una negociación SSL, en la que entre otras cosas el servidor envía su certificado de seguridad al navegador (el navegador, aunque es poco habitual, también puede enviar el suyo), y si el certificado está firmado por un Entidad Certificadora de confianza, el cliente web aceptará el certificado y se usará un canal seguro de comunicación, donde toda la información irá cifrada.
- ✓ Una vez verificado el certificado servidor, se realiza el proceso de petición HTTP y respuesta HTTP pero de forma cifrada.

Puedes hacer que un servidor web para una determinada página acepte los protocolos HTTP y HTTPS en puertos TCP distintos del 80 y 443 respectivamente. Eso sí, cuando visites la página web, en la dirección URL debes especificar el puerto TCP. Por ejemplo, si nuestro servidor atiende peticiones HTTP en el puerto 8080 deberíamos poner http://www.tupagina.local:8080, de esta forma el cliente web enviará la petición de la página www.tupagina.local al puerto 8080. Del mismo modo si nuestro servidor atiende peticiones HTTPS en el puerto 4333, deberíamos escribir la url en el navegador de la siguiente forma: https://www.tupagina.local:4333. Como ves, puedes cambiar los puertos por defecto, pero ten en cuenta que cualquiera que quisiera acceder a esas páginas debería saber el puerto TCP de la solicitud. Entonces, ¿quiere decir que aunque no escribas el puerto TCP en las direcciones URL estas se interpretan en el puerto 80 y 443 para el protocolo HTTP y HTTPS respectivamente? Pues si, así es. Es lo mismo escribir http://www.tupagina.local:80 que http://www.tupagina.local y es lo mismo escribir https://www.tupagina.local:443 que https://www.tupagina.local.

Debes conocer

En la página oficial de  [warriorsofttheNet](#) puedes encontrar un vídeo muy ameno sobre el funcionamiento de Internet. Aunque tiene algunos fallos conceptuales cómo puedes ver  [aquí](#), junto con subtítulos y un resumen del vídeo.

[Resumen textual alternativo](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

2.- Implantación de arquitecturas web.

Caso práctico



Juan, en un principio, considera importante realizar un resumen claro y conciso del concepto de arquitecturas web para incluir en su wiki, en donde pretende abarcar como mínimo los siguientes puntos:

- ✓ Evolución de los servicios web.
- ✓ Tecnologías asociadas a las aplicaciones web.
- ✓ Tipos de aplicaciones web.
- ✓ Arquitecturas web.
- ✓ etc.

También pretende incluir en otro de los puntos el servidor web Apache, abarcando **la instalación, la configuración básica e inicio de Apache**, y está pensando documentar los mismos puntos para el servidor Tomcat.

Finalmente, piensa crear un último punto en donde explicar **el proceso de despliegue de una aplicación web**; explicando el despliegue de **contenido estático** y el de **una aplicación web Java**, la **estructura de carpetas y recursos** de una aplicación web y **el descriptor del despliegue**.

Sus amigos **Carlos** y **Ana**, están muy interesados en comenzar a ayudarle a **Juan**, porque saben que van a aprender de forma práctica y van a poner en uso lo que están aprendiendo en sus estudios.

De forma genérica podríamos decir que la arquitectura web es un modelo compuesto de tres capas:

- 1.- **Capa de Base de Datos**, donde estarían todos los datos de la aplicación que se pretende administrar mediante el servicio web y emplearía una plataforma del tipo MySQL, PostgreSQL, etc.
- 2.- En una segunda capa estarían los **servidores de aplicaciones web**, ejecutando aplicaciones de tipo Apache, Tomcat, Resin, etc.
- 3.- En una tercera capa estarían los **clientes del servicio web** al que accederían mediante un navegador web como Firefox, Internet Explorer, Opera, Chrome, Safari, etc.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

2.1.- ¿Cómo evolucionan los servicios web?

Caso práctico



Juan ha enviado un correo electrónico a **Ana** solicitándole ayuda para poder documentar la evolución de los servicios web en la wiki de su empresa.

Ana ha accedido encantada a su petición, ya que está muy interesada en colaborar con la empresa en la que **Juan** trabaja, llegando incluso a pedirle a éste que le cree un usuario para poder acceder a la wiki interna de **BK programación** y ella misma documentar parte de los puntos que **Juan** tiene pensado redactar.

La evolución del uso de Servicios web en las organizaciones está fuertemente ligada al desarrollo de Internet como red prestadora de servicios. Entre los factores que han impulsado el uso de los servicios web se encuentran:

- ✓ **El contenido se está volviendo más dinámico:** los sitios web actuales proporcionan contenidos "instantáneos". Un Servicio web debe ser capaz de combinar contenido proveniente de fuentes muy diferentes.
- ✓ **El ancho de banda es menos costoso:** actualmente un Servicio web puede entregar tipos variables de contenidos como vídeo o audio. A medida que crezca el ancho de banda, los servicios web deben adaptarse a nuevos tipos de contenidos.
- ✓ **El almacenamiento es más barato:** un Servicio web debe ser capaz de manejar cantidades masivas de datos, y debe poder hacerlo de forma inteligente.
- ✓ **El éxito de la computación extendida** se está volviendo más importante: con cientos de millones de dispositivos como teléfonos móviles, agendas electrónicas, etc. existentes actualmente, estamos llegando a un momento en el cual las computadoras están dejando de ser el dispositivo más común en Internet. A medida que las plataformas se hacen más diversas, tecnologías como XML se volverán más importantes. Un servicio web no puede exigir que los usuarios ejecuten, por ejemplo, un navegador web tradicional en alguna versión de Microsoft Windows; por el contrario, los servicios web deben servir a todo tipo de dispositivos, plataformas y navegadores, entregando contenido sobre una amplia variedad de tipos de conexión.



dreia CC BY-SA

Estos factores, unidos a los beneficios proporcionados por los servicios web en la organización y los buenos productos disponibles para su desarrollo, han hecho que su utilización se extienda sin mayores obstáculos.

En términos generales, cuando se empiezan a utilizar servicios web en una organización, éstos se desarrollan e implementan como servicios simples, que poco a poco se van integrando hasta llegar a servicios web mucho más complejos.

En los orígenes del mundo web nos situábamos ante un entorno estático, con páginas en formato HTML que raramente sufrían modificaciones o actualizaciones y en las que apenas había interacción con el usuario. A esta generación de contenido web se la denominaba generalmente como **Web 1.0**.

Poco a poco los contenidos webs fueron haciéndose más versátiles y empezaron a surgir las primeras aplicaciones web, aplicaciones que se ejecutaban en el servidor y cuyo objeto era generar contenido dinámico, para lo cual generalmente se necesitaba interaccionar con una base de datos. Muchos desarrolladores denominan a esta generación web la **Web 1.5**, aunque otros lo ubican todo dentro de la web 1.0.

La **Web 2.0** es la transición que se ha dado desde las aplicaciones tradicionales hacia aplicaciones que funcionan a través de la web y que están fuertemente enfocadas al usuario final. En este nuevo entorno existen una serie de nuevas tecnologías que, en general, tienen como objetivo:

- ✓ Transformar software de escritorio para orientarlo hacia la web.
- ✓ Separar hojas de estilo.
- ✓ Potenciar el trabajo colaborativo y la utilización de redes sociales.
- ✓ Dar control total a los usuarios en el manejo de su información.

Hoy día a la web 2.0 también se la suele denominar **web social**. Dado que en ella los usuarios han dejado de ser usuarios pasivos, meros receptores de contenidos, y han pasado a formar parte activa de la web, creando contenido y conocimiento (blogs, wikis, redes sociales y otros servicios en la nube, como almacenamiento web o servicios de streaming). La web social también se caracteriza por fomentar las relaciones sociales.

Pero podemos ir un poco más lejos, también hay autores que hablan del concepto de **web 2.5** o **web simbiótica**, que sería donde encajarían hoy día mejor las redes sociales actuales y otros servicios de la web. En esta generación web los individuos depositan sus datos en servicios web gratuitos (buscadores, correo electrónico, redes sociales, etc.) y a cambio se acepta que dichos servicios usen sus datos para hacerles llegar contenidos específicos, hechos a medida del usuario, la mayor parte de las veces como anuncios o recomendaciones que están orientadas a influir en la opinión del usuario.

Pero eso no es todo, si hablamos de hacia donde caminan los servicios web deberíamos hablar de la **web 3.0**, donde se habla de una web en la que el contenido no es solo accesible por navegadores, sino también por otro tipo de aplicaciones, convirtiendo la web en una gran base de datos, y donde las aplicaciones web incluyen metadatos en los contenidos de manera que otras aplicaciones, no solo personas, puedan hacer uso de esos datos. Estos y otros conceptos, como por ejemplo la inclusión de inteligencia artificial en la web se engloban en la web 3.0.

Reflexiona

Hemos hablado de los factores que han impulsado la web y de la evolución de la misma. Hay quien ya habla de la web del futuro como la **Web 4.0**. Es difícil ubicarse entre tanta terminología y tecnologías, ¿cómo crees que será la web del futuro?

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

2.2.- ¿Cuáles son las Tecnologías asociadas a las aplicaciones web?

Caso práctico



Ana Ramirez de Arellano (CC BY-NC-ND)

Carlos, debido a su afición por el diseño web, ha ofrecido a su amigo **Juan** realizar un estudio sobre las tecnologías que se emplean actualmente en esta materia. A **Juan** le ha parecido una idea magnífica, puesto que le va a servir para el desarrollo de su wiki.



Javier Benek (CC BY-NC-SA)

Las aplicaciones web emplean páginas dinámicas, éstas se ejecutan en un servidor web y se muestran en el navegador de un equipo cliente que es el que ha realizado previamente la solicitud. Cuando una página web llega al navegador, es posible que también incluya algún programa o fragmento de código que se deba ejecutar. Ese código, normalmente en lenguaje JavaScript, **lo ejecutará el propio navegador**. Es por ello que en este apartado nos centraremos en las tecnologías asociadas a las aplicaciones web que se ejecutarán tanto del lado del servidor como del cliente, especificando lo que corresponda en cada uno de los casos.

✓ **Tecnologías en el lado del servidor:**

- ◆ **CGI (Common Gateway Interface):** la "Interface Común de Entrada" es uno de los estándares más antiguos relacionado con las aplicaciones web. Está pensado para permitir que un cliente web pueda acceder a un programa que se ejecuta en un servidor web, donde este generará contenido dinámico. **Este estándar** es utilizado para acceder a información almacenada en bases de datos, para implementar motores de búsqueda, gestionar datos de formularios, generar emails de forma automática, foros, comercio electrónico, juegos en línea, etc. Las rutinas de CGI son habitualmente escritas en lenguajes interpretados como Perl o por lenguajes compilados como C.
- ◆ **ASP (Active Server Pages):** las "Páginas Activas" se ejecutan del lado del servidor, de este modo se forman los resultados que luego se mostrarán en el navegador de cada equipo cliente en virtud de la petición que se ha realizado. Un buen ejemplo de ello son los buscadores, donde un usuario realiza una petición de información y el servidor nos entrega un resultado a medida de nuestra petición. Existen versiones de ASP para Unix y Linux, a pesar de que fue una tecnología desarrollada por Microsoft para la creación dinámica de páginas web ofrecida junto a su servidor IIS. Hoy en día ASP ha evolucionado hasta  **ASP.NET**, y se ha convertido en un framework de desarrollo web libre.
- ◆ **PHP (Hypertext Preprocessor):** este lenguaje es, al igual que ASP, ejecutado en el lado del servidor. PHP es similar a ASP y puede ser usado en circunstancias similares. Es muy eficiente, permitiendo el acceso a bases de datos empleando servidores como  **MySQL** y, por lo tanto, suele utilizarse para crear páginas dinámicas complejas.
- ◆ **Java:** en el ecosistema de Java existe un conjunto de tecnologías pensadas para desarrollo de aplicaciones web que se ejecutan en el lado del servidor, generalmente enmarcadas en lo que se conoce como Java EE (Java Enterprise Edition). Las tecnologías más conocidas en el entorno web de Java son **JSP** (JavaServer Pages), **JSF** (JavaServer Faces) y los **servlets**. JSP es en conceptualmente similar a PHP y ASP, pero con el trasfondo de java.
- ◆ **JavaScript:** dentro de la idea de "full stack development", en la que se persigue que un mismo desarrollador o desarrolladora pueda implementar tanto la parte que se ejecuta en el servidor (back-end) como la página que se visualiza en el cliente (front-end), han surgido cada vez más tecnologías que permiten desarrollar en JavaScript aplicaciones web que se ejecutan en el servidor (la misma usada en el navegador web), como es el caso de Express.js y Hapi.js (ambos funcionan sobre Node.js), y Meteor.js. Pero esta idea no es nueva, ya en 1995 el servidor web Netscape Enterprise Server ya contemplaba el uso de javascript en el servidor (server-side javascript).

✓ **Tecnologías en el lado del cliente:**

- ◆ **HTML (HiperText Markup Language):** como es de suponer, ya sabrás que HTML es el lenguaje de marcas que se utiliza maquetar el contenido de una página web. Hoy día va por la versión 5 (HTML5).
- ◆ **CSS (Cascading Style Sheets):** las "Hojas de Estilo en Cascada" se usan para formatear las páginas web; se trata de separar el contenido de un documento de su presentación. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS.
- ◆ **Java:** es un lenguaje que también podemos encontrarlo en el lado del cliente, ejecutándose de forma incrustada en un navegador web. A las aplicaciones Java que se pueden embeber en una página web se las conoce como "applets".
- ◆ **JavaScript:** posiblemente el uso más extendido de este lenguaje es hacer las páginas web más interactivas. Es un lenguaje que como ya se introdujo antes, puede interpretarse y ejecutarse en un navegador web. Es útil para realizar tareas tales como mover imágenes por la pantalla, crear menús de navegación interactivos, juegos, etc. En las páginas web suele preferirse JavaScript porque es aceptado por muchos más navegadores que VBScript (creado por Microsoft).
- ◆ **VBScript (Visual Basic Scripting):** fue la respuesta de Microsoft a JavaScript. VBScript es un lenguaje que ha entrado en desuso, dado que solo se podía usar casi exclusivamente en el navegador Microsoft Internet Explorer. El código en VBScript puede, además, estar diseñado para su ejecución en el lado del cliente o en el del servidor, la diferencia es que un código que se ejecuta en el lado del servidor no es visible en el lado del cliente. Éste recibe los resultados, pero no el código.



[the::beastieux CC BY-SA](#)

Autoevaluación

¿Podemos ver una página web sin que intervenga un servidor web?

- Sí.
- No.

2.3.- Tipos de aplicaciones web.

Caso práctico



Tras una gran labor de documentación acerca de las tecnologías de aplicaciones web, **Carlos** es consciente de que existe una gran diversidad de aplicaciones disponibles en la web. Por ello, ha decidido documentar un apartado denominado "**Tipos de aplicaciones web**" que puede ser de utilidad para la wiki que está creando su amigo **Juan**.



[Link576 \(CC BY-NC-SA\)](#)

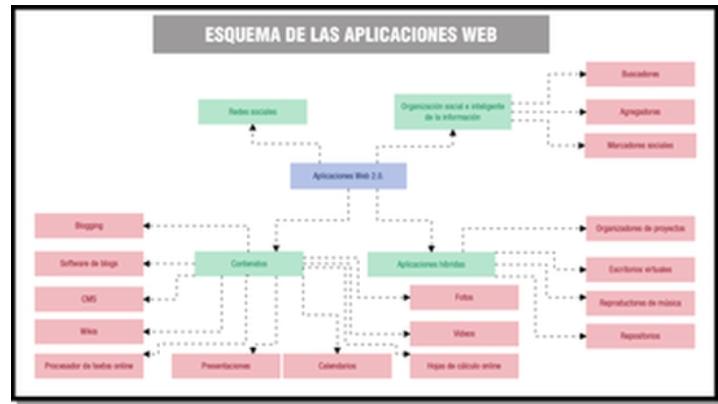
Cualquier proyecto que se quiera desarrollar en Internet, bien sea comercio electrónico, reservas de billetes de vuelo on-line, información meteorológica, registro de usuarios, simuladores de hipotecas, etc., conlleva el desarrollo de una aplicación web. En definitiva, una aplicación web es una plataforma orientada a automatizar los procesos de servicios que se quieran ofrecer a usuarios.



Establecer una clasificación de los tipos de aplicaciones web es una tarea compleja debido a la dificultad existente para poder establecer algún parámetro en función del cual establecer dicha clasificación, junto con la innumerable cantidad de aplicaciones existentes en el actual entorno **web 2.0**.

En función de cómo se presenta la aplicación web junto con el contenido que pretende mostrar, se ha establecido la siguiente clasificación:

- ✓ **Página web Estática.** Está implementada en HTML y puede mostrar en alguna parte de la página objetos en movimiento tales como banners, GIF animados (Formato gráfico de intercambio), vídeos, etc.
- ✓ **Página web Animada.** Se realiza con la tecnología FLASH; ésta permite que una página web presente el contenido con ciertos efectos animados continuados. El uso de esta tecnología permite diseños más vanguardistas, modernos y creativos.
- ✓ **Página web Dinámica.** Existen muchos lenguajes de programación que son la base para la mayoría de páginas web dinámicas. Los que destacamos aquí son los lenguajes PHP y ASP. Estos lenguajes permiten una perfecta estructuración del contenido. Por una parte crearíamos la estructura de las páginas web y por otra, almacenaríamos el contenido en determinados archivos. A partir de ahí, crearíamos el código de llamada, que insertaría el contenido en la propia página web estructurada. Este es el principio básico que siguen los lenguajes de programación. A partir de aquí se desarrollan aplicaciones para poder gestionar el contenido a través de un panel de control.
- ✓ **Portal.** Es un sitio web que en su página principal permite el acceso a múltiples secciones que, por lo general, son foros, chats, cuentas de correo, buscador, acceso registrado para obtener ciertas ventajas, las últimas noticias de actualidad, etc.
- ✓ **Tienda virtual o comercio electrónico.** Sitio web que publica los productos de una tienda en Internet. Permite la compra on-line a través de tarjeta de crédito, domiciliación bancaria o transferencia bancaria en general. Ofrece al administrador un panel de gestión para poder subir los productos, actualizarlos, eliminarlos, etc.
- ✓ **Página web con "Gestor de Contenidos".** Se trata de un sitio web cuyo contenido se actualiza a través de un panel de gestión por parte del administrador del sitio. Este panel de gestión suele ser muy intuitivo y fácil de usar. En aquellas páginas web que requieran una actualización constante, se suele incorporar este panel de gestión para que la web pueda controlarse día a día por parte del cliente.



Elaboración Propia / Ministerio de Educación.
(Uso educativo no comercial)

Reflexiona

Cuando adquirimos un equipo informático nuevo, existen una serie de aplicaciones imprescindibles que es necesario instalar junto con los drivers de nuestro equipo para poder empezar a utilizarlo. Entre estas aplicaciones encontramos aplicaciones ofimáticas, antivirus, aplicaciones de mensajería, compresores, visualizadores, reproductores multimedia, etc.

¿En algún momento te has parado a pensar qué cantidad de aplicaciones web hay disponibles en Internet para sustituir a las que tienes pensado instalar en el equipo?

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

2.4.- Arquitecturas web. Modelos.

Caso práctico



[mrjoro \(CC BY-NC\)](#)

sus ventajas e inconvenientes.

En la wiki que **Juan** está desarrollando junto con sus amigos, ha decidido integrar un punto en el cual explicar los diversos tipos de modelos de arquitecturas web que se han utilizado a lo largo del tiempo. La finalidad es tener en cuenta las características de cada uno de ellos y, en función de las mismas, distinguir



Se puede establecer que la arquitectura de un sitio web comprende los sistemas de organización y estructuración de los contenidos junto con los sistemas de recuperación de información y navegación que provea el sitio web, con el objetivo de servir de ayuda a los usuarios a encontrar y manejar la información.

Centraremos el estudio de los modelos de arquitectura web relacionados, en función de cómo implementan cada una de las capas establecidas en una aplicación web:



On Alien Cinema CC BY-NC-SA

- 1.- **Capa de presentación**, es la encargada de la navegabilidad, validación de los datos de entrada, formateo de los datos de salida, presentación de la web, etc.; se trata de la capa que se presenta al usuario.
- 2.- **Capa de negocio**, es la que recibe las peticiones del usuario y desde donde se le envían las respuestas; en esta capa se verifican que las reglas establecidas se cumplen.
- 3.- **Capa de acceso a datos**, es la formada por determinados gestores de datos que se encargan de almacenar, estructurar y recuperar los datos solicitados por la capa de negocio.

La evolución experimentada por los medios informáticos en los últimos años ha convivido con otra evolución paralela, la evolución de la arquitectura de las aplicaciones web, que permite aprovechar las nuevas características que éstas ofrecen. De esta forma, el modelo arquitectónico de las aplicaciones de Internet ha sufrido dos grandes saltos, con algún paso intermedio, desde la aparición de los primeros portales web. Los distintos modelos de aplicación sobre los que se ha ido desarrollando, según diversos autores, se podrían clasificar del siguiente modo:

- ✓ **Modelo 1.** En este caso las aplicaciones se diseñan en un modelo web CGI, basadas en la ejecución de procesos externos al servidor web, cuya salida por pantalla era el HTML que el navegador recibía en respuesta a su petición. Presentación, negocio y acceso a datos se confundían en un mismo script perl.
- ✓ **Modelo 1.5.** Aplicado a la tecnología java se da con la aparición de las JSP y los servlets. En este modelo, las responsabilidades de presentación recaen en las páginas JSP, mientras que los beans incrustados en las mismas son los responsables del modelo de negocio y acceso a datos.
- ✓ **Modelo 2.** Como evolución del modelo anterior, con la incorporación del patrón MVC (modelo vista controlador) en este tipo de aplicaciones, se aprecia la incorporación de un elemento controlador de la navegación de la aplicación. El modelo de negocio queda encapsulado en los javabeans que se incrustan en las páginas JSP.
- ✓ **Modelo 2X.** Aparece con el objetivo de dar respuesta a la necesidad, cada vez más habitual, de desarrollar aplicaciones multicanal, es decir, aplicaciones web que pueden ser atacadas desde distintos tipos de clientes remotos. Así, una aplicación web multicanal podrá ejecutarse desde una PDA, desde un terminal de telefonía móvil, o desde cualquier navegador HTML estándar. El medio para lograr publicar la misma aplicación para distintos dispositivos es emplear plantillas XSL para transformar los datos XML.

Para saber más

Esta web está pensada como un curso en español de Java básico. Pretende tener una interacción con los lectores, de forma que se puedan resolver las dudas que surjan.

 [Java básico en lenguaje español.](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

2.5.- Plataformas web libres y propietarias.

Caso práctico



El diseño de aplicaciones web requiere de un entorno funcional, que permita el desarrollo de cada uno de los

componentes que forman la aplicación web, junto con un entorno complejo de herramientas que ofrecen al cliente los servicios de las aplicaciones web; para todo ello, el mercado pone a disposición de los programadores un abanico de herramientas software, que **Juan** pretende analizar en la wiki de la empresa **BK programación**, estableciendo el criterio de clasificación que considera primordial: **software libre o software propietario**.



[logan_x \(CC BY\)](#)

Una plataforma web es el entorno de desarrollo de software empleado para diseñar y ejecutar un sitio web. En términos generales, una plataforma web consta de cuatro componentes básicos:

- 1.- El **sistema operativo**, bajo el cual opera el equipo donde se hospedan las páginas web y que representa la base misma del funcionamiento del computador. En ocasiones limita la elección de otros componentes.
- 2.- El **servidor web** es el software que maneja las peticiones desde equipos remotos a través de la Internet. En el caso de páginas estáticas, el servidor web simplemente provee el archivo solicitado, el cual se muestra en el navegador. En el caso de sitios dinámicos, el servidor web se encarga de pasar las solicitudes a otros programas que puedan gestionarlas adecuadamente.
- 3.- El **gestor de bases de datos** se encarga de almacenar sistemáticamente un conjunto de registros de datos relacionados para ser usados posteriormente.
- 4.- Un **lenguaje de programación interpretado** que controla las aplicaciones de software que corren en el sitio web.

Diferentes combinaciones de los cuatro componentes señalados, basadas en las distintas opciones de software disponibles en el mercado, dan lugar a numerosas plataformas web, aunque, sin duda, hay dos que sobresalen del resto por su popularidad y difusión: LAMP y WISA.

La plataforma LAMP trabaja enteramente con componentes de **software libre** y no está sujeta a restricciones propietarias. El nombre LAMP surge de las iniciales de los componentes de software que la integran:

- ✓ Linux: sistema operativo.
- ✓ Apache: servidor web.
- ✓ MySQL: gestor de bases de datos.
- ✓ PHP: lenguaje interpretado PHP, aunque a veces se sustituye por Perl o Python.

La plataforma **WISA** está basada en tecnologías desarrolladas por la compañía Microsoft; se trata, por lo tanto, de **software propietario**. La componen los siguientes elementos:

- ✓ Windows: sistema operativo.
- ✓ Internet Information Services (IIS): servidor web.
- ✓ SQL Server: gestor de bases de datos.
- ✓ ASP o ASP.NET: como lenguaje para scripting del lado del servidor.

Existen otras plataformas, como por ejemplo la configuración Windows-Apache-MySQL-PHP que se conoce como **WAMP**. Es bastante común pero sólo como plataforma de desarrollo local.

De forma similar, un servidor Windows puede correr con IIS, y con MySQL y PHP. A esta configuración se la conoce como plataforma **WIMP**.

Entre los servidores web más utilizados hoy en día, aparte de Apache e IIS tenemos también a **Nginx**, un servidor web software libre que está demostrando un alto rendimiento, y que es capaz de atender una gran cantidad de peticiones simultáneas, y que según las estadísticas tiene un mejor rendimiento que sus competidores al servir contenido estático. En su contra, es un servidor cuya configuración es menos flexible que otros.

Nginx también puede integrarse con PHP, por lo que podemos encontrar plataformas tipo LNMP, resultado de integrar Linux con Nginx, MySQL o  MariaDB y PHP. Esta opción es también posible en Windows, dando lugar a plataformas WNMP (Windows + Nginx + MySQL o MariaDB + PHP).

Existen muchas otras plataformas que trabajan con distintos sistemas operativos (Unix, MacOS, Solaris), servidores web (incluyendo algunos que se han cobrado relativa popularidad como Lighttpd y LiteSpeed), bases de datos (MariaDB,  PostgreSQL,  MongoDB) y otros lenguajes de programación.

Para saber más

XAMPP es una forma fácil de instalar y usar el servidor web Apache con un sistema gestor de bases de datos (MariaDB), PHP y Perl (en versiones anteriores de XAMPP el motor de bases de datos incluido era MySQL, pero en versiones recientes se ha sustituido por MariaDB, el cual, es un proyecto derivado de MySQL).

XAMPP, basta con descargarlo, extraerlo y comenzar. En este momento hay cuatro versiones de XAMPP, para: Linux, Windows, Mac OS X y Solaris.



Te proponemos los siguientes enlaces con interesantísimos vídeos sobre la instalación y uso de XAMPP en Windows.



Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

2.6.- Escalabilidad.

Las aplicaciones web se ejecutan en un entorno donde el número de clientes que solicitan el servicio puede variar en gran medida en función del momento. Es por ello que hay una característica de esencial importancia como es la escalabilidad, al que **Juan** ha dedicado un apartado de su wiki para documentar esta característica.

En el entorno en que se ubican las aplicaciones web, uno de los principales factores que puede afectar al rendimiento de las mismas es el número de usuarios, ya que éste puede verse incrementado de forma vertiginosa en un periodo de tiempo relativamente corto. El éxito o el fracaso de un sitio web orientado al usuario común vendrá determinado, entre otros aspectos, por el dimensionamiento del sistema sobre el que se instala y soporta el software que sustenta dicho sitio. En consecuencia, uno de los requisitos fundamentales de una aplicación web es que sea completamente escalable sin que un aumento de los recursos dedicados a la misma suponga modificación alguna en su comportamiento o capacidades.

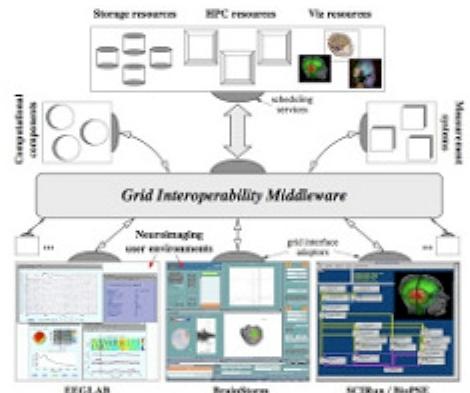
La escalabilidad de un sistema web puede ser:

- ✓ **Verticalmente:** de manera ascendente "upgrades" a cada nodo.
- ✓ **Horizontalmente:** consiste en aumentar el número de nodos.
- ✓ **Cluster:** consiste en crear agrupaciones de servidores.

Escalabilidad vertical.

Habitualmente, la separación lógica en capas se implementa de tal forma que se permita una separación física de las mismas. Interponiendo elementos conectores que actúen de middlewares es posible distribuir la aplicación de forma vertical (una máquina por cada capa del sistema), e incluso si esto no fuera suficiente, distribuyendo los elementos de una misma capa entre distintas máquinas servidoras.

Escalabilidad horizontal.



[Sam Churchill CC BY](#)

Se trata de clonar el sistema en otra máquina de características similares y balancear la carga de trabajo mediante un dispositivo externo. El balanceador de carga puede ser:

- ✓ **Balanceador Software:** por ejemplo, habitualmente encontramos un servidor web Apache junto con el módulo mod_jk, que permite la redirección de las peticiones http que a tal efecto sean configuradas entre las distintas máquinas que forman la granja de servidores. Este tipo de balanceadores examinan el paquete http e identifican la sesión del usuario, guardando registro de cuál de las máquinas de la granja se está encargando de servir a dicha sesión. Este aspecto es importante, dado que nos permite trabajar (de cara al diseño de la aplicación) apoyándonos en el objeto sesión propio del usuario y almacenando información relativa a la sesión del mismo, puesto que tenemos la garantía de que todas las peticiones de una misma sesión http van a ser redireccionadas hacia la misma máquina.
- ✓ **Balanceador hardware:** se trata de dispositivos que, respondiendo únicamente a algoritmos de reparto de carga (Round Robin, LRU (Menos Usado Recientemente), etc.), redireccionan una petición http del usuario a la máquina que, según dicho algoritmo, convenga que se haga cargo de la petición. Son mucho más rápidos que los anteriores, dado que se basan en commutación de circuitos y no examinan ni interpretan el paquete http. Sin embargo, el no garantizar el mantenimiento de la misma sesión de usuario en la misma máquina, condiciona seriamente el diseño, dado que fuerza a que la información relativa a la sesión del usuario sea almacenada por el implementador del mismo, bien en cookies o bien en base de datos.
- ✓ **Balanceador hardware http:** se trata de dispositivos hardware pero que examinan el paquete http y mantienen la relación usuario-máquina servidora. Mucho más rápidos que los balanceadores software, pero algo menos que los hardware, suponen hoy en día una de las soluciones más aceptadas en el mercado.



phrenologis CC BY-NC

Cluster.

Con la aparición de los servidores de aplicaciones en cluster se abre una nueva capacidad de escalabilidad que, dependiendo de cómo se aplique, podría clasificarse como vertical u horizontal. Un cluster de servidores de aplicaciones permite el despliegue de una aplicación web corriente, de forma que su carga de trabajo vaya a ser distribuida entre la granja de servidores que forman el cluster, de modo transparente al usuario y al administrador. El cluster, mediante el mecanismo de replicación de sesión, garantiza que sea cual sea la máquina que sirva la petición http, tendrá acceso a la sesión del usuario (objeto HttpSession en Java). Este tipo de sistemas, debido precisamente a la replicación de sesión, suele presentar problemas de rendimiento.

Debes conocer

En el siguiente vídeo puedes comprobar cómo realizar la instalación del paquete XAMPP en un equipo con sistema operativo Ubuntu 10.04

[Resumen textual alternativo](#)

Autoevaluación

¿En qué tipo de escalabilidad se emplean los balanceadores de carga?

- Horizontal.
- Vertical.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

3.- Servidor web Apache.

Caso práctico

En la empresa **BK programación**, **Ada** ha solicitado a **María** la instalación de un servidor web para albergar, entre otras cosas, la wiki que **Juan**, junto con sus amigos, están construyendo.

María, por su parte, ha decidido instalar el **servidor web Apache** por ser uno de los más empleados.

Le va a aportar también a **Juan** la información que pueda interesarle acerca de este servidor web para su wiki.



Hoy en día utilizamos Internet como una herramienta común: para el trabajo, para el ocio...

Pero sin duda el elemento fundamental que usamos no es otro que el **navegador**, gracias al cual podemos sacar partido a todo lo que se encuentra en Internet: comprar entradas para el cine, acceder a nuestra cuenta bancaria, averiguar el tiempo que hará el fin de semana... Pero nada de esto tendría sentido si detrás de cada página web a la que accedemos no existiera **un servidor web**, que permite que la página esté accesible 24x7 (24 horas al día y 7 días a la semana, es decir, siempre).

Detrás de cada página web debe existir un servidor web que ofrezca esa página, bien a los internautas, a los trabajadores y trabajadoras de una empresa, por tratarse de una página web interna de la empresa, no accesible a Internet, o a toda persona que disponga de una conexión de red con la cual pueda acceder a la página.

La configuración del servidor web dependerá de las páginas web que ofrezca, así la configuración no será la misma si la página posee contenido estático o no, o si se necesita que modifique el contenido según interacción del usuario, o si se necesita de comunicación segura en la transición de información, o si se debe tener en cuenta el control de acceso a determinados sitios de la página. Por lo tanto según las páginas web que se ofrezcan el servidor web deberá estar configurado para tal fin: con soporte  PHP, con soporte de cifrado, con soporte de control de acceso, etc.

Pero ¿un servidor web pueda alojar varias páginas web o solamente una? Es más, ¿puede alojar   sitios,  dominios de Internet o solamente uno, esto es, permite hosts virtuales? Pues, un servidor web puede alojar varias páginas, sitios, dominios de Internet, pero hay que tener en cuenta que la elección del servidor web será muy importante para la configuración y administración de uno o múltiples sitios. No obstante, debemos preguntarnos:

- ✓ ¿Puede el servidor web ser modular —es decir, fácilmente se le pueden añadir o quitar características—?
- ✓ O por contra, si queremos añadirle una funcionalidad que no posea en la instalación base, ¿debemos desinstalarlo e instalarlo de nuevo?
- ✓ Por ejemplo: hasta ahora el servidor web solamente ofrecía páginas estáticas, pero queremos ofrecer también páginas web dinámicas. ¿Qué hacemos: modular o nueva instalación?
- ✓ También tenemos que pensar que todo puede crecer y lo que ahora era un servidor web que ofrecía x número de páginas, ahora necesitamos que ofrezca x*y, con lo cuál tenemos que prever la **escalabilidad** del servidor web, y también la **estabilidad**: ¿cómo se comporta ante múltiples conexiones simultáneas?

De nada servirá tener instalado un servidor web sin saber cómo se va a comportar ofreciendo el servicio, con lo cual será muy importante previamente y durante el funcionamiento del servidor establecer unas pruebas de funcionamiento del mismo y registrar lo acontecido.

Por todo lo anterior, veremos cómo configurar y administrar el servidor **Apache (apache2)**, ya que soporta:

- ✓ páginas web estáticas;
- ✓ páginas web dinámicas;
- ✓ hosts virtuales;
- ✓ seguridad mediante cifrado;
- ✓ autenticación y control de acceso;
- ✓ modularización;
- ✓ monitorización de archivos de registro.



[mayhem CC BY-NC-SA](#)

3.1.- Características de Apache Http Server.

Reflexiona



[msarturlr \(CC BY-NC-SA\)](#)

¿Alguna vez te has parado a pensar qué existe detrás de una página web?

¿Por qué al escribir un nombre en un navegador puedes visionar una página web?

¿Por qué no tienes acceso a determinadas páginas?

¿De qué modo puedes impedir el acceso a determinados sitios de una página: por directorio, por usuario?

¿Cómo se puede establecer una comunicación segura en una transición bancaria?

✓ ...

Un servidor web es un programa que se ejecuta de forma continua en un ordenador (también se utiliza el término para referirse al ordenador que lo ejecuta), se mantiene a la espera de peticiones por parte de un cliente (un navegador de Internet) y contesta a estas peticiones de forma adecuada, sirviendo una página web que será mostrada en el navegador o mostrando el mensaje correspondiente si se detectó algún error.

Uno de los servidores web más populares del mercado y el más utilizado actualmente es **Apache**, de código abierto y gratuito, disponible para Windows y GNU/Linux, entre otros.

En cuanto a su arquitectura podemos destacar lo siguiente:

- ✓ Estructurado en módulos. Cada módulo contiene un conjunto de funciones relativas a un aspecto concreto del servidor. El administrador puede elegir qué módulos incluir dependiendo de la funcionalidad que necesite.
- ✓ El ejecutable binario principal es httpd (que significa http daemon). Los módulos pueden ir incluidos directamente en el httpd, si son compilados de forma estática, o bien pueden ser cargados más tarde si han sido compilados por separado. Los módulos compilados por separado se conocen como DSOs (objetos dinámicos) y pueden ser cargados en función de las necesidades.
- ✓ La funcionalidad de estos módulos puede ser activada o desactivada al arrancar el servidor. Esto se logra gracias a los módulos compilados como DSO, los cuales se podrán cargar o no en función de nuestras necesidades. Los módulos compilados de forma estática siempre están disponibles.
- ✓ Los módulos de Apache se pueden clasificar en tres categorías:
 - ◆ Módulo **base**: son un conjunto de módulos que **están siempre presente** dado que han sido incluidos en tiempo de compilación. El módulo **core** (núcleo) es un módulo base que está siempre presente en nuestro servidor y que se encarga de las **funciones básicas** del servidor web.
 - ◆ Módulos **multiproceso**: encargados de la unión de los puertos de la máquina, aceptando las peticiones y atendiéndolas.
 - ◆ Módulos **adicionales**: se encargan de añadir funcionalidad al servidor.



[jaaron CC BY](#)

El servidor Apache se desarrolla dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation. La licencia de software, bajo la cual el software de la fundación Apache es distribuido, es una parte distintiva de la historia de Apache HTTP Server (y de la comunidad de código abierto). La **Licencia Apache** permite la distribución de derivados de código abierto y cerrado a partir de su código fuente original.

Para saber más

Esta web sirve como manual de referencia, guía de usuario, tutoriales prácticos, etc., **sobre el servidor web Apache**. Se trata de la web oficial de Apache Software Foundation.



[Documentación Servidor HTTP Apache en español \(versión 2.0\)](#)



[Documentación Servidor HTTP Apache \(versión actual\)](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

3.2.- Instalación en sistema basados en Linux.

Caso práctico



En la empresa **BK programación**, **Ada** ha solicitado a **María** la instalación de un servidor web para albergar, entre otras cosas, la wiki que **Juan**, junto con sus amigos, está construyendo.



The Apache Software Foundation (Licencia Apache 2.0.)

María debe instalar el **servidor Apache** debido a que es uno de los más empleados. Se trata de una herramienta de código libre y que funciona en multitud de plataformas; en este caso, será instalado en una máquina con un sistema operativo Linux basado en Debian, quedando el servidor totalmente operativo.

¿Qué necesitas para realizar la **instalación de un servidor Apache** en una máquina con una distribución Linux basada en Debian (por ejemplo Ubuntu)?

Empezamos por identificarnos en la máquina con el usuario root (a través del comando sudo su por ejemplo) y, a continuación, ejecutamos:

```
root@localhost: ~#apt-get install apache2
```

Con el comando apt-get anterior instalamos Apache 2.X desde el repositorio de la distribución Linux que estemos usando, por lo que deberemos tener nuestro equipo conectado a Internet. Si queremos montar una plataforma LAMP completa, necesitaremos instalar un motor de bases de datos como MariaDB o MySQL, e instalar el interprete de PHP como módulo del servidor Apache. Podemos consultar los módulos relacionados con PHP y proceder a su instalación con los siguientes comandos:

```
root@localhost: ~#apt-cache search php
```


- ✓ Strongly Encouraged (for example those using public computing resources) - Usually referring to the use of multiple personal machines or to use multiple devices for a single user account.
- ✓ Weakly Encouraged (for example those using public computing resources) - Usually referring to the use of multiple accounts for a single user.

From https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-1.3

✓ New PFS methods (A-D) are required to be supported by all implementations.

✓ The new PFS methods (A-D) are required to be supported by all implementations.

✓ The new PFS methods (A-D) are required to be supported by all implementations.

Cloud providers are encouraged to do the following:

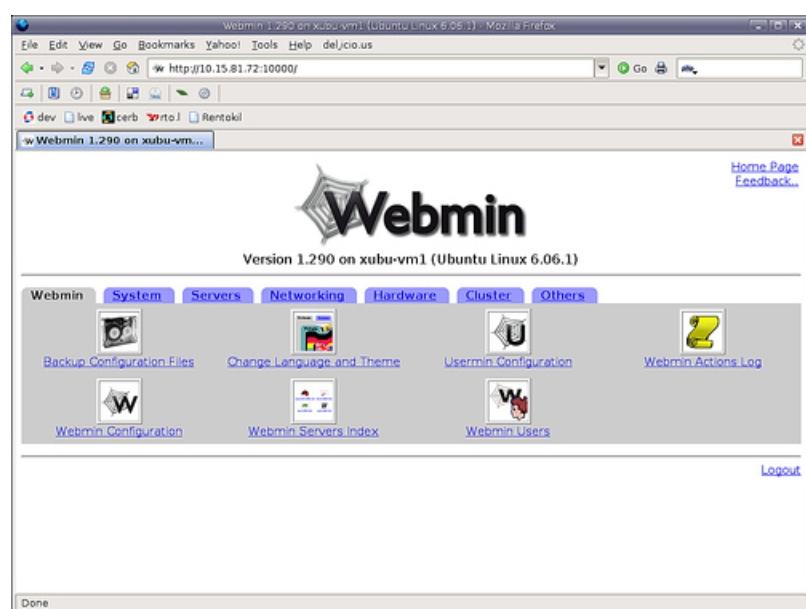
✓ Implement the new PFS methods (A-D).

✓ Encourage their users to include the new PFS methods in their certificates and the certificate revocation lists.

Compliance with this section (or other cases of resistance to the upgrade) [is optional](#).

✓ nope

✓ auto make install



[fse8info CC BY-SA](#)

Para saber más

Apache también se puede instalar en otros sistemas operativos como Windows, NetWare o BSD. En el siguiente enlace encontrarás una lista con soluciones Apache ya compiladas para Windows:

 [Descargar Apache HTTP Server para Windows.](#)

Debes conocer

En vídeo práctico se explica como instalar y configurar un servidor web Apache en una máquina virtual con el sistema operativo Ubuntu:

[Resumen textual alternativo](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

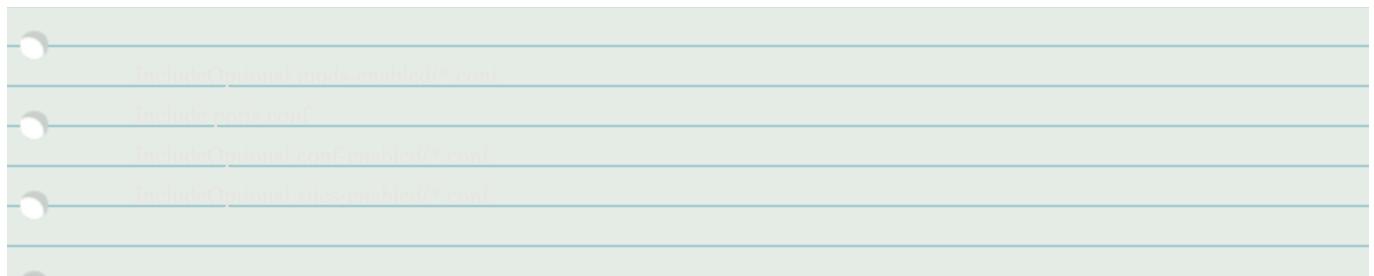
3.3.- Introducción a la configuración de Apache.

Apache se configura colocando directivas en archivos de configuración de texto plano. El archivo principal de configuración se llama generalmente apache2.conf o httpd.conf (aunque esto puede cambiar dependiendo de la instalación realizada y del sistema). Y, ¿dónde encontramos dicho archivo? Dependiendo de la instalación variará su ubicación:

- En distribuciones de Linux tales como Ubuntu o Debian se le llama apache2.conf y se puede encontrar en la carpeta /etc/apache2/.
- En distribuciones de Linux tales como CentOS, RedHat o Fedora se le llama httpd.conf y suele estar en la carpeta /etc/httpd/conf/.
- En distribuciones tales como SUSE Linux se le llama httpd.conf y suele estar en la carpeta /etc/apache2/.

Como puedes comprobar, dependiendo del caso, estará en un lugar u otro, y la cosa cambia todavía más si realizas una instalación en Windows u otros sistemas operativos.

Desde el archivo de configuración principal se cargan otros archivos de configuración mediante la directiva `Include` e `IncludeOptional`. Esto permite separar la configuración de Apache en varios archivos, facilitando así su administración. En ambas directivas se pueden usar comodines para incluir muchos archivos de configuración simultáneamente. Si echamos un vistazo al archivo apache2.conf de una distribución como Ubuntu podremos ver lo siguiente:



La diferencia fundamental entre la directiva `Include` e `IncludeOptional` es que `Include` producirá un error en la configuración si no existe ningún archivo que coincidan con el patrón indicado, mientras que `IncludeOptional` no.

Pero, ¿qué es una directiva de configuración? **Una directiva es básicamente una regla de funcionamiento que imponemos a Apache.** Las directivas tienen dos partes, el **nombre** de la directiva (Include, por ejemplo) y unos **parámetros** (ports.conf, por ejemplo).

A la hora de configurar Apache debemos tener en cuenta lo siguiente:

- Todas las directivas deben colocarse en alguno de esos archivos de configuración.
- **Apache2 sólo reconocerá los cambios realizados en los archivos de configuración cuando se inicie o se reinicie.**
- Solo podrá haber una directiva por línea, si una directiva ocupa más de una línea podemos acabar la línea en "\" y continuar en la siguiente línea.
- Las directivas se pueden escribir tanto en mayúsculas como en minúsculas, pero no así sus parámetros, en los que hay situaciones que si se distinguen mayúsculas de minúsculas.

Cada directiva permite configurar un aspecto concreto de Apache y está asociada a un módulo concreto. Por ejemplo, si consultamos la documentación de Apache veremos que la directiva `Include` está asociada al módulo **core**, como podemos ver en la siguiente imagen, que corresponde a una captura de pantalla de la documentación oficial de Apache 2.4:

The screenshot shows a section titled "Include Directive". It contains the following information:

- Description:** Includes other configuration files from within the server configuration files
- Syntax:** `Include file-path|directory-path|wildcard`
- Context:** server config, virtual host, directory
- Status:** Core
- Module:** core
- Compatibility:** Directory wildcard matching available in 2.3.6 and later

Below the main content, there is a small search icon and two links at the bottom:

Obra derivada de la documentación oficial de Apache bajo licencia Apache License 2.0
Captura realizada por Salvador Romero / Apache License 2.0

En la imagen anterior se puede ver la sintaxis de la directiva, el módulo al que pertenece e incluso el contexto, que explicaremos a continuación. **Lo importante de aquí es entender que si el módulo asociado a la directiva no está activo, dicha directiva no se podrá usar.**

Además de las directivas, en la configuración de Apache podremos tener secciones. Las secciones son contenedores que pueden incluir en su interior directivas que se aplican en un contexto concreto. Tienen un aspecto como el siguiente:

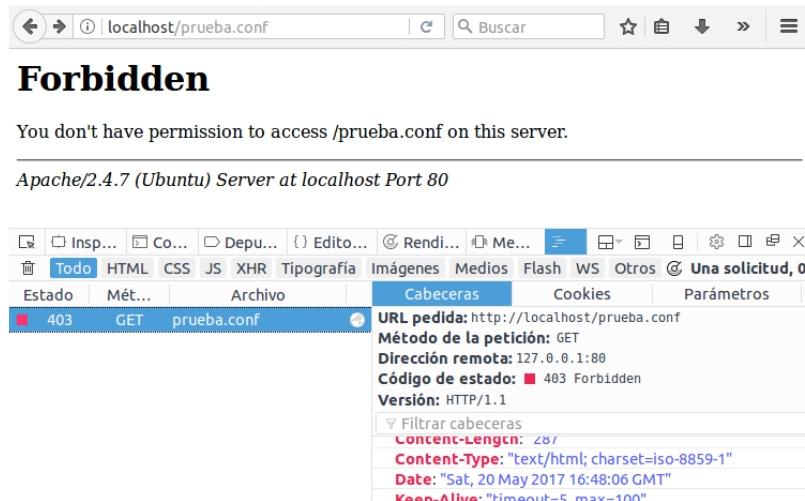


En este caso, la sección `Directory` permite aplicar directivas de forma específica sobre un directorio del servidor. Aquí el contexto es el directorio `/var/www/html/configuracion/`, y cuando se realice una petición web que requiera que el servidor web Apache acceda a dicho directorio, se denegará el acceso, tal y como se establece a través de la directiva `required` y los parámetros `all denied`. El resultado es que el servidor web Apache no servirá contenido de dicha carpeta, y devolverá un código de estado 403 al cliente web.

Una sección puede contener en su interior más secciones, como el siguiente ejemplo:



En este caso, la sección `Files` está dentro de la sección `Directory`, y esto hace que la directiva `Require all denied` se aplique a aquellos archivos con extensión `.conf` que hay dentro de la carpeta `/var/www` (o en cualquier subcarpeta). El resultado es que se limitará el acceso solo a dichos archivos, tal y como se puede ver en la siguiente imagen:



Obra derivada de una captura de pantalla del navegador Firefox, bajo licencia Mozilla Public License 2.0 (MPL).
Captura realizada por Salvador Romero ([Mozilla Public License 2.0](#))

Por lo tanto, cuando vayamos a poner una directiva en la configuración (o una sección) tendremos que consultar la documentación para saber si la podemos poner dentro de una sección concreta o no. Para esto tendremos que mirar en que **contexto** se puede utilizar cada directiva o sección. Los contextos son los siguientes:

- **server config**: es lo que llamamos configuración global y nos indica que podemos poner la directiva fuera de cualquier sección, dado que se aplicará a todo el servidor web en su conjunto. Si en la directiva el único contexto posible es **server config**, la directiva solo se podrá poner a nivel global, y nunca dentro de una sección, ni en un archivo .htaccess.
- **virtual host**: hace referencia a las secciones <VirtualHost> que se verán más adelante.
- **directory**: hace referencia a las secciones <Directory>, <Location> y <Files> entre otras. Es decir, secciones relativas a archivos y directorios en el servidor, o relativas al recurso pedido en la petición HTTP (como es el caso de <Location>).
- **.htaccess**: las directivas válidas en este contexto podrán ser usadas dentro de los archivos .htaccess. Los archivos .htaccess son archivos de texto plano que, si están habilitados, se pueden crear en un directorio para establecer directivas específicas a ese directorio (y subdirectorios).

Autoevaluación

¿Cuál de las siguientes opciones sobre directivas es falsa?

- Se pueden poner dentro de una o más secciones dependiendo del contexto.
- Tienen dos partes, nombre de la directiva y parámetros.
- Se aplican aunque no se reinicie Apache.
- Las directivas se pueden escribir tanto en minúsculas como en mayúsculas.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

3.4.- Iniciar y detener Apache.

Caso práctico



En la
empresa **BK**



The Apache Software Foundation (Licencia Apache 2.0)

programación, Ada ha solicitado a **María** la instalación de un servidor web para albergar, entre otras cosas, la wiki que **Juan**, junto con sus amigos, está construyendo.

María debe instalar el servidor Apache debido a que es uno de los más empleados. También debe indicar todos y cada uno de los pasos para arrancar el servidor, así como las indicaciones oportunas para la accesibilidad a las páginas de la wiki desde el servidor Apache.

El proceso de iniciar y detener el servidor web de Apache es muy diferente dependiendo del sistema. Si hemos decidido, por ejemplo, compilarlo manualmente en Ubuntu e instalado en la ruta /opt/apache, tal y como se explicaba en apartados anteriores, podemos iniciar, detener y reiniciar el servicio de la siguiente forma:

- Accedemos al directorio donde se ha compilado Apache (en este caso /opt/apache) con el comando cd /opt/apache
- Ejecutamos el comando ./bin/apachectl start para iniciar apache (estrago apachectl start)
- Ejecutamos el comando ./bin/apachectl stop para detener apache (estrago apachectl stop)

En un principio, si no hemos cambiado la configuración base (archivo httpd.conf en este caso), el servidor debería arrancarse al ejecutar el primero de los tres comandos anteriores, y a través de un navegador web podríamos ver la página que se sirve al poner la URL `http://localhost/` (si accedemos desde otro equipo debemos usar la IP del servidor web), donde veríamos la página de bienvenida de Apache.

Pero si estamos en un sistema Linux, lo normal, no es compilar Apache, sino instalarlo a través del repositorio de paquetes de la distribución que estemos usando. Si estamos en una máquina con Debian o Ubuntu podemos iniciar, pararlo y detenerlo con los siguientes comandos:

- Accedemos al directorio Apache con el siguiente de los tres comandos siguientes (el primero es el servicio apache2, el segundo apache2 y el tercero es el servicio apache2 stop)
- Con los tres comandos podemos reiniciar apache2 (estrago apache2 &)
- El servicio apache2 responderá con apache2: [re]start: [ok] (el ultimo apache2 restart)

Ten en cuenta que los comandos anteriores normalmente hay que ejecutarlos con un usuario administrador, por lo que en muchos casos será necesario convertirse en administrador del sistema:

- Usando la opción -s (silenciosa) podemos ejecutar el comando sin tener que ser administrador
- Accedemos al directorio apache2 (estrago apache2 &)

O bien ejecutar el comando directamente con privilegios:

- Usando el comando sudo podemos ejecutar el comando sin tener que ser administrador (estrago sudo apache2 &)

Para saber más

Es imposible abordar todos los sistemas operativos, pero aquí te incluimos algunos enlaces que te pueden ser útiles a la hora de gestionar el servicio de Apache en diferentes sistemas operativos:

 [Como instalar Apache, PHP y MariaDB en CentOS](#)

 [Como iniciar, detener y reiniciar servicios usando systemctl](#) (usado por fedora, CentOS y RedHat)

 [Iniciar, detener y reiniciar servicios en Windows 10](#)

La directiva Listen permite especificar en que puertos del servidor atenderá peticiones [HTTP](#) nuestro servidor web:

Listen 80

Listen 8080

Listen 192.168.0.1:80

Listen 192.168.0.2:8080

En Debian y Ubuntu esta directiva normalmente se ubica en el archivo /etc/apache2/ports.conf, y se puede especificar más de un puerto donde nuestro servidor web escuchará:

Si nuestro equipo tiene además más de una dirección [IP](#) asociada (ya sea porque tenga más de una interfaz de red o porque tenga direcciones [IP](#) asignadas a diferentes interfaces virtuales de red), podemos hacer que se atienda un puerto en una de las direcciones [IP](#) y otro puerto diferente en otra de las direcciones [IP](#), por ejemplo:

En Linux y Unix, el proceso de arranque de Apache requiere (salvo excepciones) que el proceso se inicie con el usuario root (Administrador). Una vez que el servidor Apache se ha iniciado y ha completado algunas tareas preliminares, tales como abrir sus ficheros [log](#), lanzará varios procesos, procesos hijo, que hacen el trabajo de escuchar y atender las peticiones de los clientes. El proceso principal, httpd, continúa ejecutándose como root, pero los procesos hijo se ejecutan con menores privilegios de usuario. El usuario y el grupo usado para los procesos hijo en Apache se establece con la directiva User y [Group](#) respectivamente. En Ubuntu suele ser el usuario www-data y el grupo www-data.

El demonio httpd se debería invocar empleando el script de control apachectl, que es el que se encarga de fijar variables de entorno y pasa al demonio (httpd) cualquier opción que se le pase como argumento por línea de comandos.

El script apachectl es capaz de interpretar los argumentos start, restart, y stop y traducirlos en las señales apropiadas para httpd, tal y como se puede ver en los ejemplos anteriores.

Recomendación

Después de realizar un cambio de configuración en Apache es muy importante verificar que los cambios realizados en los archivos de configuración no contienen errores. Esto, en Linux y Unix podemos hacerlo con el comando siguiente:



Puedes consultar más información sobre el comando apachectl en el siguiente enlace:

 [Uso del comando apachectl](#)

Reflexiona

Si hemos configurado del servidor para que escuche el puerto 8080 en vez del puerto 80, ¿qué cambios en la configuración se han hecho y como podremos acceder a las páginas después del cambio desde un navegador web?

[Mostrar retroalimentación](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

4.- Configuración básica de Apache.

Caso práctico

Ada, la Directora de **BK Programación**, se reúne con **María** y con **Juan** para tratar la mejor manera de implantar, configurar y mantener el nuevo servidor para atender la aplicación web que le han encargado, para una empresa con varias sucursales.



—Hola **María** —dijo **Ada**—, nos han ofrecido un nuevo proyecto relacionado con servicios web, pienso que podemos afrontarlo, pero quería saber tu opinión. ¿Con la infraestructura que tenemos ahora ves necesario el montaje de otro equipo servidor dedicado a este proyecto o con lo que tenemos nos arreglamos?

—Pienso que tal como estamos ahora, sí o sí —dijo María—, independientemente de los recursos que consuma este nuevo proyecto necesitamos la configuración de otro equipo servidor. Además debemos configurar dos entornos: el de pruebas y el de producción. ¿Para cuándo sería el proyecto?

—El proyecto debemos entregarlo con fecha final dentro de tres meses.

—Entonces, creo que si todo sigue su cauce normal no tendremos ningún tipo de problema para la ejecución del proyecto. ¿Qué recursos humanos habías pensado destinar al proyecto?

—Ahora disponemos de todo el personal de la empresa y cuento contigo y con **Juan** para que os coordinéis las funciones de este proyecto.

—Pues por mí, no veo objeción al mismo.

—Bien —asintió **Ada**—, entonces no se hable más, tendremos que configurar otro equipo servidor y aceptamos el proyecto.

Así, la empresa **BK Programación** envió un presupuesto a la empresa del proyecto, ésta lo aprobó y comenzó el trabajo.



Para afrontar el nuevo proyecto al que se enfrentan en la reunión se determinó que **María** sería la encargada del montaje, configuración y administración del nuevo equipo servidor y **Juan** el encargado de coordinar con el resto del personal la creación y funcionamiento de las aplicaciones web del proyecto.

María, entonces, se puso manos a la obra y determinó el siguiente escenario de trabajo para el equipo servidor de este proyecto:

- ✓ Sistema Operativo: **Debian GNU/Linux 6.0**
- ✓ Servidor Web: **Apache (apache2)**
 - ➔ Configuración de Red:
 - ➔ Servidor Web: **192.168.200.250**
 - ➔ Cliente de pruebas (desde donde se lanza el navegador): **192.168.200.100**

Citas para pensar

“

Se debe hacer todo tan sencillo como sea posible, pero no más sencillo.

Albert Einstein

Hay que tener en cuenta que en el escenario las IP empleadas son IP privadas, sin existencia en Internet, por lo cual siempre que se haga referencia a las mismas a través de nombre de dominios, deberá existir un servidor DNS que las resuelva en local o bien en su defecto deberán existir las entradas correspondientes en el fichero del sistema local /etc/hosts.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

4.1.- Servicio de ficheros estáticos.

Reflexiona

¿Es necesario que todas las páginas web se modifiquen constantemente?

¿Un blog sería útil si el contenido no sufre cambios?

¿Y un manual? ¿Si actualizamos un manual la página deja de ser estática?



The Apache Software Foundation - [Licencia Apache 2.0](#)

Todas aquellas páginas web que durante el tiempo no cambian su contenido no necesariamente son estáticas. Una página estática puede modificarse, actualizando su contenido y seguir siendo estática.

¿Y entonces, si puede cambiar, qué hace que sea estática o dinámica?

Debemos diferenciar cuando accedemos a una página web entre **código ejecutable en el lado del servidor y en el lado del cliente** —equipo que solicita la página mediante el cliente web (navegador). Si al acceder a una página web no es necesaria la intervención de código en el lado del servidor —por ejemplo código PHP— o en el lado del cliente —por ejemplo JavaScript— entonces entenderemos que la **página es estática**. Si por el contrario es necesaria la intervención de dicho código en el lado del servidor y/o en el lado del cliente, entenderemos que la **página es dinámica**.

Ofrecer páginas estáticas es simple, puesto que solamente se necesita que el servidor web disponga de soporte html/xhtml/css o incluso solamente html/xhtml. En cuanto a configuración y administración del servidor es el caso más simple: solamente se necesita un soporte mínimo base para la instalación del servidor Apache, esto es, no se necesita por ejemplo soporte PHP. En cuanto a rendimiento del servidor, sigue siendo el caso más beneficioso: no necesita de ejecución de código en el lado del servidor para visionar la página y tampoco necesita ejecución de código en el lado del cliente, lo que significa menos coste de CPU y memoria en el servidor y en el cliente, y por lo tanto una mayor rapidez en el acceso a la información de la página.

Para poder **ofrecer páginas estáticas** mediante el servidor Apache simplemente se copia la página en la ruta configurada en la directiva DocumentRoot. La directiva DocumentRoot permite indicar la raíz de los archivos que el servidor web Apache servirá:

DocumentRoot /var/www/html

En el ejemplo anterior, Apache podrá servir los archivos que hay en dicho directorio al recibir una petición. Dada la configuración anterior, podríamos hacer lo siguiente:

- ✓ Creamos la carpeta /var/www/html/prueba.
- ✓ Crearíamos el documento /var/www/html/prueba/miprueba.html (el documento debe estar dentro de la carpeta raíz indicada por la directiva DocumentRoot).
- ✓ Ahora podríamos ir al navegador web y, sin reiniciar el servidor web, podríamos acceder a nuestra página web poniendo la url <http://localhost/prueba/miprueba.html>. Fíjate que en la URL no aparece la carpeta /var/www/html sino solamente las carpetas que se han agregado a partir de la carpeta raíz.

El ejemplo anterior muestra lo fácil que es agregar contenidos estáticos. Podemos agregar más contenidos o cambiar los existentes. Si cambias un documento [HTML](#) o una imagen que estás visualizando en el navegador web en ese momento, recuerda que tienes que actualizar o recargar la página (se puede hacer simplemente pulsando la tecla F5), para que el navegador vuelva a realizar una petición [HTTP](#) y vuelva a cargar el contenido.

La directiva DocumentRoot podemos encontrarla en diferentes partes de la configuración de Apache, a nivel global o dentro de una o más secciones VirtualHost. De hecho, no es obligatorio que aparezca a nivel global, puede ser que aparezca dentro de un VirtualHost directamente. Se puede decir que es una práctica habitual que la configuración por defecto incluya un VirtualHost (también llamado VirtualHost por defecto), que determine cual es la ruta de los documentos estáticos a servir. Más adelante se hablará de que es un VirtualHost.

Otro detalle a tener en cuenta es que en muchas ocasiones no escribimos el recurso completo al que estamos accediendo. Por ejemplo, si escribes la [URL](#) <http://httpd.apache.org/docs/current/> no estamos indicando que archivo queremos obtener del servidor, da la impresión de que estamos accediendo a una carpeta remota llamada docs/current/. El hecho es que Apache, cuando no se especifica el archivo a obtener, buscará lo que se llama el DirectoryIndex (índice de directorio), que no es más que un archivo con un nombre específico que se servirá cuando no se ha especificado un recurso concreto.

Normalmente el DirectoryIndex es index.html (aunque este puede cambiar), por lo que si ponemos por ejemplo la [URL](#) <http://httpd.apache.org/docs/current/>, en realidad estamos accediendo a <http://httpd.apache.org/docs/current/index.html> (de hecho ambas URLs funcionan). DirectoryIndex es a su vez una directiva que podemos configurar en nuestro servidor Apache a nivel global y dentro de varias secciones (por ejemplo dentro de un VirtualHost o dentro de un Directory):



El código anterior muestra un ejemplo de nombres de archivo usados como DirectoryIndex. En Ubuntu y Debian es normal encontrar esta directiva en el archivo de configuración /etc/apache2/mods-available/dir.conf.

Cuando se instala Apache en un GNU/Linux Debian o Ubuntu (en una versión igual o superior a 14.04), se crea una serie de rutas en el equipo servidor similar a la estructura siguiente:

Rutas de interés en la instalación de Apache (apache2) en una distro Debian y Ubuntu

Ruta 1	Rutas 2, 3, 4 y 5.
/etc/apache2/sites-available /etc/apache2/sites-enabled /var/www/html /var/www /var/www/available /var/www/enabled /var/www/sites-available /var/www/sites-enabled	/etc/apache2/sites-available/000-default.conf /etc/apache2/sites-enabled/000-default.conf /var/www/html /var/www/sites-available /var/www/sites-enabled /etc/apache2/apache2.conf

En la instalación de Apache anterior, si no hemos modificado nada con respecto a la configuración inicial, la directiva DocumentRoot está configurada en el archivo /etc/apache2/sites-available/000-default.conf, en una sección VirtualHost, y apunta a la carpeta /var/www/html.

Para saber más

Te proponemos que hagas un viaje por la página web de documentación de Apache.

-  [Página web oficial de documentación de Apache \(apache 2.2\)](#)
-  [Página web oficial de documentación de Apache \(apache 2.4\)](#)
-  [Página web oficial de documentación de Apache \(todas las versiones\)](#)

La actual versión y la que se instala por defecto en Ubuntu 14.04 y posteriores es Apache 2.4. **Hay que tener cuidado con algunos cambios de nombres de ficheros (en 2.4 hay ficheros con extensión .conf que en la versión 2.2 no suelen tener extensión .conf). Además, también cambian algunas directivas y algunos comandos como los de rotar logs.**

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

4.2.- Directivas y secciones de configuración.

Ya hemos hablado de los contextos en los cuales podemos encontrar directivas y de algunas directivas, y hemos tratado superficialmente las secciones que hay asociadas a cada contexto. Ahora es el momento de ahondar en las secciones más importantes.

Algunas de estas secciones ya las hemos tratado superficialmente, como la sección Directory, que permite aplicar un conjunto de directivas de forma específica a un directorio y sus subdirectorios de nuestro servidor. O la sección Files que permite aplicar un conjunto de directivas de forma específica a un conjunto de archivos.

Es importante saber que la directiva Directory, y otras directivas, pueden contener caracteres comodín y expresiones regulares. Las expresiones regulares son expresiones que permiten expresar un patrón de búsqueda, si el directorio al que se está accediendo tras la petición web encaja con el patrón de búsqueda entonces se aplicarán las restricciones. Veamos un ejemplo:

•	Directory /var/www/html/*
•	Options +Indexes
•	AllowOverride All
•	Directory

El carácter comodín en el ejemplo anterior es el asterisco (*) el cual significa literalmente "una secuencia de cero o más símbolos alfanuméricos". En dicho ejemplo, la sección Directory anterior se aplicaría a un directorio como /var/www/html/test, y a otros tales como /var/www/html/test o /var/www/html/tesoro, pero no al directorio /var/www/html/mitest. Otro carácter comodín es el interrogante (?), que significaría por un único símbolo alfanumérico.

En el ejemplo anterior se han aplicado además dos directivas bastante comunes a la sección Directory: la directiva Options que se puede aplicar en varias secciones y que permite habilitar o deshabilitar características concretas para una sección; y la directiva AllowOverride, que solo se puede usar dentro de un Directory, y que permite indicar si se pueden sobrescribir algunas directivas a través de archivos .htaccess.

Vamos a detenernos un momento a analizar ese ejemplo de uso de la directiva Options. Al escribir Options +Indexes estamos diciendo a Apache que, si en la URL no se especifica el recurso a servir (por ejemplo, se pone <http://localhost/test/> en vez de <http://localhost/test/imagen1.jpg>), y en dicha carpeta no encuentra un archivo DirectoryIndex (`index.html` por ejemplo), se generará un listado del contenido de la carpeta similar al siguiente:

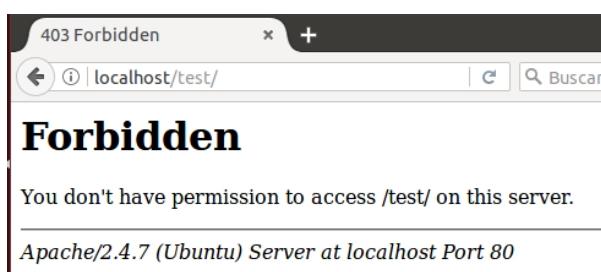
Index of /test

Name	Last modified	Size	Description
 Parent Directory		-	
 Imagen1.jpg	2017-05-26 17:50	0	
 Imagen2.jpg	2017-05-26 17:50	0	

 Apache/2.4.7 (Ubuntu) Server at localhost Port 80

Obra derivada de una captura de pantalla del navegador Firefox, bajo licencia Mozilla Public License 2.0 (MPL).
Captura realizada por Salvador Romero ([Mozilla Public License 2.0](#))

Si por el contrario, la directiva fuera Options -Indexes el resultado sería el contrario, deshabilitaríamos esa característica y obtendríamos el siguiente resultado:

 403 Forbidden

localhost/test/

Forbidden

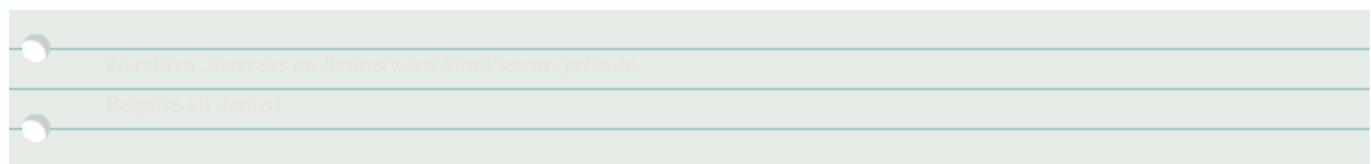
You don't have permission to access /test/ on this server.

Apache/2.4.7 (Ubuntu) Server at localhost Port 80

Obra derivada de una captura de pantalla del navegador Firefox, bajo licencia Mozilla Public License 2.0 (MPL).
Captura realizada por Salvador Romero ([Mozilla Public License 2.0](#))

Existen varias características asociadas a la directiva Options y lo mejor es consultar la documentación para conocerlas, dado que aquí es imposible tratarlas todas. La otra directiva indicada antes, AllowOverride permite escribir directivas en archivos .htaccess. Los archivos .htaccess que se colocan directamente el directorio en el que queremos aplicar dichas directivas, por ejemplo, si queremos aplicar directivas específicas al directorio /home/www/html/tesoro/privado, podríamos hacer lo siguiente:

- ✓ En un archivo de configuración principal (/etc/apache2/apache2.conf por ejemplo), pondríamos AllowOverride dentro de una sección Directory. Recuerda que la sección Directory aplica las directivas al directorio indicado y a los subdirectorios que haya dentro, por lo que el ejemplo anterior: <Directory /home/www/html/te*>; nos serviría.
- ✓ Crearíamos el archivo .htaccess en el directorio en cuestión (/home/www/html/tesoro/privado en este caso), y dentro escribiríamos las directivas a aplicar, por ejemplo:



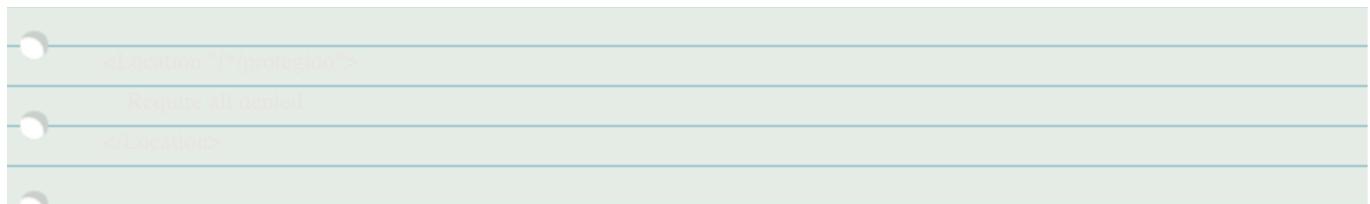
Con el ejemplo anterior estaríamos limitando el acceso a la dirección /home/www/html/tesoro/privado. La ventaja de los archivos .htaccess, es que podemos escribir directivas en ellos y no es necesario reiniciar Apache para que las directivas se apliquen. El inconveniente es que en dichos archivos solo se pueden usar aquellas directivas aplicables en el contexto .htaccess.

El conjunto de directivas que podemos poner en un archivo .htaccess va limitado por la directiva AllowOverride. Veamos diferentes ejemplos:

- ✓ AllowOverride All: Todas las directivas aplicables dentro del contexto .htaccess.
- ✓ AllowOverride AuthConfig: Solo directivas de tipo autenticación y control de acceso, como AuthUserFile o **Require**.
- ✓ AllowOverride AuthConfig Indexes: Se pueden usar directivas de tipo de autenticación y relativas a la indexación de documentos (por ejemplo, para cambiar el DocumentIndex de una carpeta concreta).

El conjunto de variantes completo de AllowOverride es mejor que lo consultes en la documentación.

Otra sección de relevancia es la sección Location que permite aplicar directivas en función de la URL de la petición. Imagina que en un servidor web que tienes configurado estás sirviendo contenido a través de la URL `http://www.midominio.local/contenido` (es decir, un navegador puede acceder al contenido de tu web a través de esa URL anterior), pero quieras limitar el acceso cuando la URL sea `http://www.midominio.local/contenido/protegido`. En ese caso podrías poner directivas que se aplicaran solo cuando una URL tenga una forma específica:



En este caso, cuando el recurso solicitado en la petición HTTP siga el patrón anterior (recuerda que * es un carácter comodín), se denegará el acceso. Aunque la sección Location tiene muchas ventajas (por ejemplo, se aplica aunque se cambie el DocumentRoot) normalmente se prefiere el uso de la directiva Directory.

Paralelamente a las secciones Directory y Location existen las secciones DirectoryMatch y LocationMatch, que permiten poner una expresión regular en vez de un directorio o un camino al recurso, y que son mucho más flexibles que los caracteres comodín.

Es normal encontrar las secciones Directory y Location (y sus compañeros que usan expresiones regulares), dentro de la configuración global, pero también es muy normal encontrarlas dentro lo que llamamos un VirtualHost o host virtual.

Apache permiten servir diferente contenido en función del nombre de dominio, del puerto, o simplemente de la dirección IP. Por ejemplo, un mismo servidor web puede servir contenido diferente para `http://localhost/miweb`, para `http://www.midominio.local/miweb`, y para otros casos que se nos ocurra configurar; y esto se logra gracias a los VirtualHost.

Cuando las secciones Directory y Location se configuran dentro de un host virtual, solo se aplican en el contexto de ese host virtual, y no a nivel global. Por ejemplo:

VirtualHost	www.midominio.local
	Servidor web Apache (localhost)
	Directorio /var/www/html/midominio.local
	Directiva Directory /var/www/html/midominio.local
	Contexto de Directorio: /var/www/html/midominio.local/protegidos
	Requerimiento de autenticación
	Directiva de autorización: /var/www/html/midominio.local
	CustomLog /var/log/apache2/midominio.local.log combined

En este VirtualHost se especifica el contenido a servir cuando se accede al servidor web Apache solicitando el dominio www.midominio.local. En el ejemplo anterior la directiva Directory solo se aplica en el contexto del VirtualHost. Los hosts virtuales se ven con mayor detalle un poco más adelante.

Autoevaluación

Para poder poner archivos .htaccess en un directorio es necesario poner una directiva en los archivos de configuración principal. ¿Cuál es esa directiva?

- Options AllowOverride
- AllowOverride Options
- Options +All
- AllowOverride All

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

4.3.- Directivas de configuración básicas.

Caso práctico

Para poder llevar a buen fin el proyecto, **María** reúne al equipo destinado al mismo, ya que quiere que todo el personal tenga claro los requisitos, entregables y fechas de ejecución del proyecto. Así, en esta reunión informativa para todo el equipo destinado al proyecto, trató los siguientes temas:



- 
 - 1.- Recursos del equipo servidor.
 - 2.- Conectividad del equipo servidor.
 - 3.- Servidor web empleado: el porqué de su elección y funcionamiento.
 - 4.- Posibilidades del servidor web empleado.
 - 5.- Requisitos de las aplicaciones web del proyecto.
 - 6.- Entregables y fechas.

Como ya hemos comentado, el archivo de configuración predeterminado de Apache2 es apache2.conf o httpd.conf, y suele estar en diferentes ubicaciones dependiendo de la instalación (en instalaciones recientes de Ubuntu y Debian suele estar `/etc/apache2/apache2.conf`). Como ya se ha dicho, desde este archivo de configuración se incluye más archivos. En todos ellos se pueden configurar diferentes aspectos de Apache a través de directivas. Por ejemplo:

checkbox (CC BY)

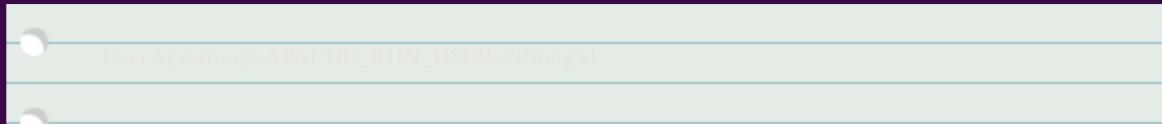
- ✓ ServerRoot "/etc/apache2": generalmente no hay que ponerlo, pero permite indicar cual es el directorio donde estará la configuración de Apache.
- ✓ ServerName www.ejemplo.local: permite indicar cual es el nombre del servidor web configurado (en este caso www.ejemplo.local). Esta directiva podemos encontrarla en el ámbito global o dentro de un VirtualHost (de hecho, lo más normal es encontrarla dentro de este último). En el caso de no tener un dominio registrado, deberíamos configurar el nombre de la máquina a través del archivo /etc/hosts o emplear el nombre localhost.
- ✓ ServerAdmin "webmaster@ejemplo.local": especifica la dirección de correo del administrador del servidor. El valor por omisión es webmaster@localhost.
- ✓ >ErrorLog> >/var/log/httpd/error_log: permite especificar la ubicación del archivo donde se registran los errores que se han producido al procesar una petición y otros errores del servidor. Si nuestro servidor no funciona, el primer sitio donde debemos mirar es en el archivo de log de errores. Esta directiva puede usarse tanto a nivel global, como dentro de un VirtualHost.
- ✓ KeepAlive on: permite especificar si admitirán las conexiones persistentes, es decir, si se dejará abierto el puerto después de una primera petición para recibir más peticiones desde el mismo cliente web. Se puede configurar a nivel global y dentro de un VirtualHost. Se pueden deshabilitar con off.
- ✓ Timeout 60: permite especificar el tiempo máximo que el servidor esperará antes de cerrar una conexión.
- ✓ ServerTokens Full: cuando Apache responde a una solicitud, incluye una cabecera del estilo a Server: Apache/2.4.7 (Ubuntu) que le permite al navegador web conocer información del servidor. Con esta directiva, que solo se puede usar a nivel global, podemos indicar el nivel de detalle que el servidor enviará al cliente web.

Las directivas anteriores son del módulo core. Si deseas configurar algunas de estas directivas, o mirar donde están configuradas, no olvides mirar también en los archivos de configuración que son incluidos desde el principal. Por ejemplo, en Ubuntu y Debian se usa el archivo /etc/apache2/sites-available/000-default.conf, que es donde está la configuración del VirtualHost por defecto. Otras directivas interesantes, de otros módulos, son las siguientes:

- ✓ Alias, es una directiva del módulo base mod_alias que permite direccionar a una carpeta que puede estar fuera del árbol de directorios especificado en DocumentRoot. Se puede poner dentro de varias secciones.
- ✓ UserDir: cuando el módulo mod_userdir está activo, Apache puede servir páginas que están alojadas dentro del directorio principal de cada usuario del sistema (fuera del DocumentRoot), es decir, dentro de su home. Imagina por ejemplo que tienes activo este módulo y que tienes un usuario en el servidor llamado test. Una configuración habitual es que los documentos albergados en /home/test/public_html sean accedidos con la url http://localhost/~test. Esta directiva permite configurar el funcionamiento de este módulo.

Debes conocer

Antes de seguir, has de saber que en la configuración de Apache es posible hacer referencia a variables de entorno del sistema con la siguiente sintaxis:



En el ejemplo anterior, la directiva User usaría el valor de la variable de entorno APACHE_RUN_USER. Las variables de entorno son variables que contienen valores dinámicos accesibles desde los procesos del sistema y que el usuario puede configurar.

En una instalación de un Ubuntu o Debian actual, podemos ver las variables de entorno que usará nuestro Apache en el archivo / etc/apache2/envvars. En el siguiente enlace puedes ver más información sobre las variables de entorno en Apache:

 [Variables de entorno en Apache.](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

4.4.- Contenido dinámico.

Citas para pensar

“

El progreso consiste en el cambio.

Miguel de Unamuno



[David Davies CC BY-SA](#)

Muchas veces seguro que te encuentras visitando una página web y la información te parece tan interesante que procedes y guardas en **Favoritos** la dirección URL para una posterior visión, pero cuando de nuevo deseas ver la página resulta que lo que estás viendo no tiene nada que ver o es distinto de lo que esperabas. ¿Qué ha ocurrido?

Puede que la página haya cambiado su contenido o que la página que visitas posee contenido no estático (dinámico), dependiente del código ejecutado en el servidor o en el cliente al acceder a la página.

Imagínate que accedes a una página web y que dependiendo de si posees una cuenta de usuario u otra el contenido es distinto, o que presionas en una imagen de la página y se produce un efecto en la misma, o que el contenido cambia dependiendo del navegador. De cualquier forma la página ha sido modificada mediante una interacción con el usuario y/o el navegador, por lo tanto nos encontramos con una **página dinámica**.

Como bien puedes pensar, una página dinámica, necesita más recursos del servidor web que una página estática, ya que consume más tiempo de CPU y más memoria que una página estática. Además la configuración y administración del servidor web será más compleja: cuantos más  módulos tengamos que soportar, más tendremos que configurar y actualizar. Esto también tendrá una gran repercusión en la seguridad del servidor web: cuantos más módulos más posibilidades de problemas de seguridad, así si la página web dinámica necesita, para ser ofrecida, de ejecución en el servidor debemos controlar cuidadosamente qué es lo que se ejecuta.

Algunos módulos con los que trabaja el servidor web Apache para poder soportar páginas dinámicas son: mod_actions, mod_cgi, mod_cgid, mod_ext_filter, mod_include, mod_ldap, mod_perl, mod_php5, mod_python.

Para saber más

En el siguiente enlace a la página de Apache puedes ampliar la información que te proporcionamos sobre los módulos.

-  [Página web oficial de documentación sobre módulos para la última versión de Apache.](#)
-  [Página web oficial de documentación de Apache \(apache 2.2\) sobre módulos.](#)

Autoevaluación

Abres el navegador y solicitas una página a un servidor web: ¿cuál de las siguientes acciones indica que la página solicitada no es dinámica?

- La página tiene un panel de control, al cual accedes mediante tu usuario y tu contraseña, los cuales nunca cambias. La página entonces establece comunicación con una base de datos y te permite el acceso a tu perfil, distinto del perfil del administrador de la página.
- Al pasar el puntero por encima de una imagen, ésta se redimensiona y al salir vuelve al tamaño original.
- Cuando visitas la página con distintos navegadores aparece un comentario de alerta indicando el navegador con el cual estás accediendo a la página.
- La página solicitada es un manual sobre el Servidor Apache, y está totalmente escrita en código HTML y CSS.

5.- Hosts virtuales. Creación, configuración y utilización.

Caso práctico



A la empresa **BK Programación** le ha surgido el siguiente proyecto: una empresa con varias sucursales quiere montar una aplicación web por sucursal. La empresa en cuestión consta de 7 sucursales. Todas ellas dedicadas a la misma línea de negocio. Así, las aplicaciones tendrán un frontal similar, pero estarán personalizadas dependiendo de la situación de la sucursal, de tal forma que los banners, logos e imágenes de cada aplicación serán monumentos locales a la zona de la sucursal.

El equipo de trabajo del proyecto está coordinado por **María**, ella es la encargada del montaje, creación y configuración del servidor web donde irán alojadas las aplicaciones web.

La empresa quiere que las sucursales puedan ser localizadas en Internet mediante URL tipo: www.sucursal-zonaX.empresaproyecto.com, donde X puede variar de 1 a 7.

Además quiere que si las páginas se buscan sin www. éstas sigan viéndose, es decir, que sucursal-zonaX.empresaproyecto.com se dirija a la misma página que www.sucursal-zonaX.empresaproyecto.com

La empresa también desea que exista **un único panel de control de usuarios**, en la URL www.empresaproyecto.panel-de-control.com, de tal forma que según el perfil que posea la persona usuaria, podrá ver un contenido u otro. Así, desea que los comerciales tengan la posibilidad de saber qué productos y cantidades de los mismos existen en stock. Al panel de control se accede a través de un enlace configurado en cada aplicación.

María se reúne con **Juan**, el encargado del desarrollo de las aplicaciones web, y con Carlos, que ejerce el rol del usuario destinado a comprobar el buen funcionamiento de las aplicaciones haciendo pruebas con distintos navegadores:

—Pienso —dijo **María**— que la mejor forma de llevar a buen puerto el proyecto se realiza configurando **hosts** virtuales en el servidor web Apache y no solamente colgando las aplicaciones web en un directorio raíz común para luego, cada una, disponer de su espacio en una carpeta independiente.

—Sí, —dijo **Juan**—, además tenemos que tener en cuenta la seguridad del panel de control, deberíamos pensar en el protocolo HTTPS, para asegurarnos que la información vaya cifrada.

—Estoy de acuerdo —afirmó **María**—. Entonces, **Carlos**, deberás hacer las pruebas mediante HTTP y HTTPS.

—Vale, de acuerdo —dijo **Carlos**—.

Ya hemos visto cómo Apache puede servir una o varias páginas web almacenadas en un directorio. El directorio del que se sirven las páginas web se puede configurar en Apache con la directiva DocumentRoot. Hasta ahora dichas páginas web estaban relacionadas con un mismo dominio configurado con la directiva ServerName.

Si nuestra única pretensión es servir un sitio web asociado a un único dominio, entendiendo como sitio web un conjunto de páginas web relacionadas entre sí, podemos integrar su contenido como se ha visto con anterioridad, copiando el contenido de la web en el directorio al que apunta la directiva DocumentRoot y configurando el ServerName en Apache.



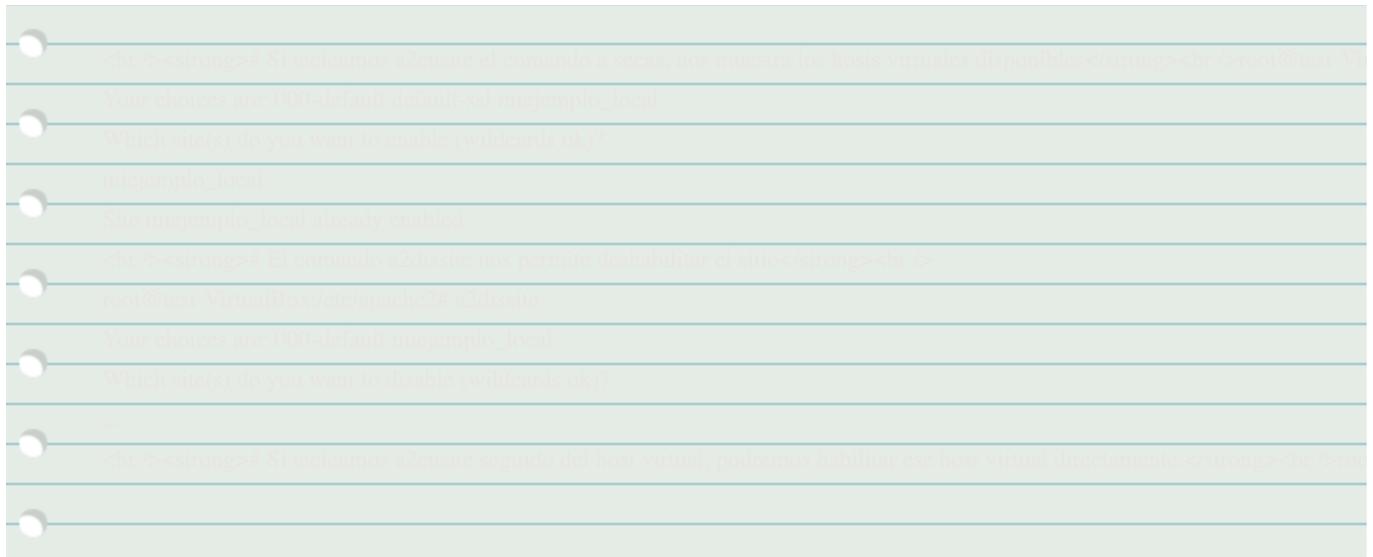
Ivanpw CC BY

- ServerName www.misdominios.local
- DocumentRoot /var/www/html/

Como ya se ha comentado, estas dos directivas pueden encontrarse a nivel global, o configuradas en lo que se llama un VirtualHost por defecto. La configuración de este VirtualHost por defecto puede variar dependiendo del sistema, por ejemplo, en Ubuntu suele estar configurado en el archivo /etc/apache2/sites-available/000-default.conf y suele tener un aspecto como el siguiente (con algunas cosas adicionales):

- <VirtualHost *:80>
- ServerName www.misdominios.local
- DocumentRoot /var/www/html/
- </VirtualHost>

En el ejemplo anterior se ha configurado un dominio llamado un VirtualHost para servir contenido relacionado con el sitio web `www.miejemplo.local`. Ubuntu y Debian proporcionan los comandos `a2ensite` y `a2dissite` para poder hacer más fácil la habilitación o deshabilitación respectivamente de un VirtualHost, veamos un ejemplo:



Aquí es importante que recuerdes que después de cualquier configuración hay que comprobar la configuración y reiniciar Apache. También es importante que recuerdes que `a2ensite` y `a2dissite` SON comandos específicos de Ubuntu y Debian (y otros sistemas operativos derivados), y es posible que no estén en otros sistemas o entornos.

Otro detalle importante es que, a pesar de que cada `<virtualHost>` es único e independiente de los demás, todo aquello que no esté incluido expresamente en la definición de cada `<virtualHost>` se heredará de la configuración principal: `apache2.conf` o `httpd.conf`. Así, que si quieres definir una directiva común en todos los `<virtualHost>` no debes modificar cada uno de ellos introduciendo esa directiva, sino que puedes **definir esa directiva en la configuración principal del servidor web Apache**, de tal forma que todos los `<virtualHost>` heredarán esa directiva. Esto es válido para aquellas directivas que se puedan poner tanto a nivel global, como dentro de un VirtualHost. Por ejemplo, a nivel global puedes encontrar la directiva `Timeout 300`, la cual puede usarse también dentro de un VirtualHost; pero si no se especifica, se usará el valor a nivel global.

Si no tienes configurado un servidor DNS con las entradas de dominio necesarias, puedes generar estas entradas modificando el archivo `/etc/hosts`, añadiéndolas al final del mismo:



Esto permitirá simular que tu ordenador tiene asignado uno o más nombres de máquina o FQDN. Cada campo de cada entrada puede ir separado por espacios o por tabuladores. En cada entrada empezamos escribiendo la dirección IP, y a continuación la lista de nombres que queremos que respondan a dicha dirección IP. En el ejemplo anterior hemos añadido un total de 12 nombres adicionales para este servidor.

IMPORTANTE: Estas entradas solamente serán efectivas en el equipo en el que se modifique el archivo / etc/hosts. Así debes modificar el archivo/etc/hosts en cada equipo que quieras que se resuelvan esas entradas. Ten en cuenta que las direcciones IP que empiezan por 127 son direcciones IP de loopback internas a cada equipo y que no son accesibles desde otro sistema, solo desde nuestro propio equipo.

Veamos ahora como podría ser una secuencia completa para configurar un VirtualHost nuevo en un Ubuntu o Debian actual:

```
apt-get install apache2 -y
root@localhost:~# nano /etc/hosts
root@localhost:~# cat /etc/hosts
127.0.0.1 localhost
127.0.0.1 www.example.local
127.0.0.1 localhost www.example.local

root@localhost:~# nano /etc/apache2/sites-available/www.example.local.conf
root@localhost:~# cat /etc/apache2/sites-available/www.example.local.conf
<VirtualHost *:80>
    ServerName www.example.local
    DocumentRoot /var/www/www.example.com
    <Directory /var/www/www.example.com>
        Options Indexes FollowSymLinks
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>

root@localhost:~# nano /etc/apache2/sites-available/www.example.local.conf
root@localhost:~# cat /etc/apache2/sites-available/www.example.local.conf
<VirtualHost *:80>
    ServerName www.example.local
    DocumentRoot /var/www/www.example.com
    <Directory /var/www/www.example.com>
        Options Indexes FollowSymLinks
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>

root@localhost:~# nano /etc/apache2/sites-available/www.example.local.conf
root@localhost:~# cat /etc/apache2/sites-available/www.example.local.conf
<VirtualHost *:80>
    ServerName www.example.local
    DocumentRoot /var/www/www.example.com
    <Directory /var/www/www.example.com>
        Options Indexes FollowSymLinks
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

000-default.conf
DocumentRoot /var/www/html
ServerName localhost
ServerAdmin webmaster@localhost
ErrorLog /var/www/error.log
LogLevel warn

AddDefaultCharset UTF-8
Content-Type: text/html

AddDefaultCharset UTF-8
Content-Type: text/html

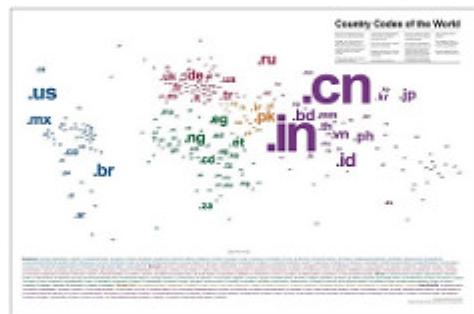
AddDefaultCharset UTF-8
Content-Type: text/html

Recomendación

Crear un nuevo VirtualHost es a veces tedioso. En sistemas operativos como Ubuntu o Debian es más cómodo copiar el VirtualHost por defecto, ubicado en el archivo 000-default.conf a otro archivo .conf, y modificar la copia creada para lograr la configuración deseada.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

5.1.- Virtualhosts basados en nombre.



• Jason Whittaker (CC BY-NC-SA)

- Virtual host existence (VirtualHost directive)
 - Document root (DocumentRoot directive)
 - Directory configuration (Directory directive)
 - Server block configuration (ServerBlock directive)

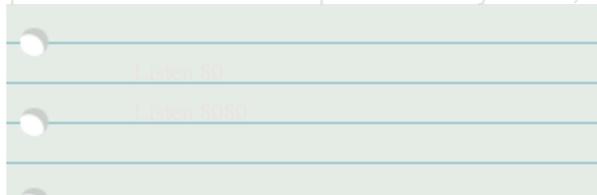
- **Communication** - communication with the customer and other stakeholders
 - **Document** - document management system
 - **Performance** - performance management system
 - **Customer Experience** - customer experience management system

- VirtualHost *:80:
DocumentRoot /var/www/empresa1/
ServerName www.empresa1.com
ServerAlias empresa1.com empresa1.es www.empresa1.es
- VirtualHost *:8080:
DocumentRoot /var/www/empresa2/
ServerName www.empresa2.com
ServerAlias empresa2.com empresa2.es www.empresa2.es

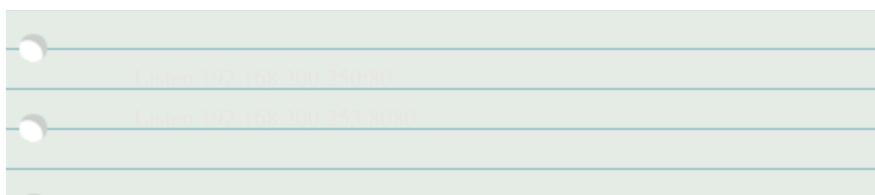
Recuerda que en sistemas operativos como Ubuntu o Debian es común crear los hosts virtuales en la carpeta `/etc/apache2/sites-available`, y no directamente en el archivo de configuración principal (`apache2.conf` o `httpd.conf`), para después crear enlaces simbólicos a la carpeta `/etc/apache2/sites-enabled` de aquellos host virtuales que queramos activar.

Revisemos ahora la configuración anterior directiva a directiva:

- ✓ <VirtualHost IP_Servidor_Web:80>: en el inicio, la etiqueta `VirtualHost` define la IP del servidor web (recuerda que en un mismo servidor podemos tener asignadas varias direcciones IP). También se indica el punto TCP para el protocolo HTTP, que por defecto es el **80**. Recuerda que podemos configurar que puertos escucha nuestro servidor web Apache mediante la directiva `Listen`. Recuerda también que se pueden usar varias directivas `Listen` para especificar varias direcciones y puertos de escucha. El servidor web Apache responderá a peticiones en cualquiera de esas direcciones y puertos. Por ejemplo, para hacer que el servidor acepte conexiones en los puertos 80 y 8080, usa:



Para hacer que el servidor acepte conexiones en dos direcciones IP y puertos diferentes, usa:



- ✓ DocumentRoot /var/www/empresa1/: definición de la ruta donde está alojada la página web en el servidor, en este caso: `/var/www/empresa1/` mediante la directiva `DocumentRoot`.
- ✓ ServerName www.empresa1.com: declaración del nombre DNS para el que se servirán las páginas alojada en la ruta declarada mediante la directiva `DocumentRoot`. Es el nombre que escribes en el navegador para visitar la página.
- ✓ ServerAlias empresa1.com: la directiva `ServerAlias` permite definir otros nombres DNS para la misma página.
- ✓ </VirtualHost>: fin de la sección `VirtualHost`.

Autoevaluación

Deseas que tu servidor web ofrezca en la misma IP las URL:

- ✓ www.sucursal-zona2.empresa-proyecto.com
- ✓ sucursal-zona2.empresa-proyecto.com
- ✓ www.empresa-proyecto.panel-de-control.com.

donde las 2 primeras identifican el mismo sitio web y la última otro totalmente distinto. Entonces, ¿qué podrías utilizar para definir los virtualhosts?

- Un solo fichero.
- Dos ficheros.
- No se pueden utilizar virtualhosts, debido a que los dominios son distintos.
- Incorrecta la pregunta, ya que las URL están mal definidas, no pueden contener el carácter guión.

Caso práctico

En este enlace tienes un caso práctico desarrollado y explicado totalmente para Ubuntu 14.04. Te será muy útil para realizar la tarea.



[¿Cómo configurar Virtual Host de Apache en Ubuntu 14.04 LTS?](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

5.2.- Virtualhosts basados en IP.

En un servidor, y en cualquier ordenador realmente, podemos tener varias interfaces de red (tarjetas  Ethernet por ejemplo), cada una con una dirección IP independiente. Incluso podemos tener una misma tarjeta de red con múltiples direcciones IP.

Partiendo de la idea anterior, podemos decir que los hosts virtuales basados en IP se utilizan para servir diferentes contenidos en función de la dirección IP a través de la cual se recibe la petición HTTP.

A pesar de lo que pueda pensarse, este tipo de configuración es menos común que los host virtuales basados en nombre, porque implica que un mismo servidor tenga varias direcciones IP, e incluso varias tarjetas de red, y cuando hablamos de direcciones IP públicas eso puede conllevar un sobrecoste si el número de nombres de dominio es muy elevado. Además, puede ser más difícil de mantener si las IP del servidor web se modifican con cierta frecuencia.

Partiendo de esa idea, imagina que tienes un servidor con dos interfaces de red, cada una con las siguientes direcciones IP: 192.168.200.251 y 192.168.200.252. Dada esta configuración, podríamos poner una IP diferente para cada Virtualhost, de manera que se sirviera un sitio web específico a través de una de las direcciones IP y otro sitio web diferente a través de la otra.

¿Cómo lo haces? Sigues el mismo procedimiento usado para los VirtualHost basados en nombre, pero especificando una dirección IP diferente para cada VirtualHost, veamos:

VirtualHost *:80	ServerName www.espressol.com
	DocumentRoot /var/www/espressol
	ServerAlias www.espressol.com
	ServerAlias espressol.com
	VirtualHost
	VirtualHost *:80
	ServerName www.espressol.com
	DocumentRoot /var/www/espressol
	ServerAlias www.espressol.com
	ServerAlias espressol.com
	VirtualHost

Recuerda que dependiendo del sistema operativo y del tipo de instalación realizada la configuración de los host virtuales conviene hacerla en un sitio u otro. En Ubuntu y Debian es mejor crear archivos de configuración independientes en la carpeta /etc/apache2/sites-available, dado que es una carpeta pensada para definir hosts virtuales, y después activarlos con el comando a2ensite.

Veamos ahora con un poco más de detalle las secciones y directivas anteriores:



bcostin [CC BY-NC-SA](#)

- ✓ <VirtualHost 192.168.200.251:80>: en el inicio la etiqueta Virtualhost define la IP del servidor web donde se aloja la página de la empresa, en este caso empresa3. El puerto TCP por defecto es el **80**, definido en la configuración principal del servidor, mediante la directiva Listen.
- ✓ DocumentRoot /var/www/empresa3/: definición de la ruta donde está alojada la página web en el servidor, en este caso: /var/www/empresa3/ mediante la directiva DocumentRoot.
- ✓ ServerName www.empres3.com: declaración del nombre DNS para el que se servirán las páginas alojada en la ruta declarada mediante la directiva DocumentRoot. Es el nombre que escribes en el navegador para visitar la página. definición del nombre DNS que buscará la página alojada en la ruta anterior del servidor mediante la directiva ServerName. Es el nombre que escribes en el navegador para visitar la página.
- ✓ ServerAlias empres3.com: la directiva ServerAlias permite definir otros nombres DNS para la misma página.
- ✓ </VirtualHost>: fin de la etiqueta VirtualHost.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

5.3.- Configuraciones mixtas.

Citas para pensar

“

Cuando me preguntan cuándo estará listo un programa, contesto: depende de cuánto trabaje usted en ello.

Richard Stallman

Existen casuísticas que se salen de los dos casos típicos de configuración; VirtualHosts solo basados en nombre o solo basados en IP. Por ejemplo, podemos tener Virtualhosts basados en nombre y VirtualHosts basados en IP en un mismo servidor:

```
<VirtualHost 192.168.200.100>
    DocumentRoot /var/www/expresado/
</VirtualHost>
<VirtualHost 192.168.200.100>
    DocumentRoot /var/www/expresado/
    ServerName www.expresado.com
</VirtualHost>
<VirtualHost 192.168.200.100>
    DocumentRoot /var/www/expresado/
    ServerName www.expresado.com
    ServerPort 8080
</VirtualHost>
<VirtualHost 192.168.200.100>
    DocumentRoot /var/www/expresado/
    ServerName www.expresado.com
    ServerPort 8080
</VirtualHost>
```

También es común tener configuraciones diferentes cuando nuestro servidor escucha varios puertos. Por ejemplo, imagina que tu servidor escucha los puertos 80 y 8080:

```
<VirtualHost 80>
    DocumentRoot /var/www/expresado/
</VirtualHost>
<VirtualHost 8080>
    DocumentRoot /var/www/expresado/
    ServerName www.expresado.com
</VirtualHost>
```

Si queremos que un VirtualHost se aplique solo a un puerto, por ejemplo el puerto 8080, deberíamos especificar el puerto como hemos hecho hasta ahora:

```
<VirtualHost 192.168.200.200:8080>
    DocumentRoot /var/www/empresa1/
    ServerName www.empresa1.com
```

El VirtualHost anterior serviría contenido para la URL `http://www.empresa1.com:8080/`, pero no para la URL `http://www.empresa1.com`, dado que este último caso usaría el puerto 80. Si queremos que ese VirtualHost sirva un contenido diferente dependiendo del puerto (un contenido para el puerto 80 y otro diferente para el puerto 8080), deberíamos crear dos VirtualHost diferentes, cada uno con un puerto diferente:

```
<VirtualHost 192.168.200.200:80>
    DocumentRoot /var/www/empresa1/comodin/
    ServerName www.empresa1.com

<VirtualHost 192.168.200.200:8080>
    DocumentRoot /var/www/empresa1/comodin/
    ServerName www.empresa1.com
```

Fíjate que en el ejemplo anterior hay que poner un DocumentRoot diferente para cada VirtualHost.

Si por el contrario, deseamos que independientemente del puerto se sirva el mismo contenido para un mismo dominio, podemos usar el comodín * en la parte del puerto:

```
<VirtualHost 192.168.200.200:>
    DocumentRoot /var/www/empresa1/
    ServerName www.empresa1.com
    VirtualHost
```

Con la configuración anterior, sea el puerto que sea, se servirá el mismo contenido (ya sea `http://www.empresa1.com` o `http://www.empresa1.com:8080`). El carácter comodín se puede omitir, obteniendo el mismo resultado que en el ejemplo anterior:

```
<VirtualHost *:80>
    DocumentRoot /var/www/empresa1/
    ServerName www.empresa1.com
    VirtualHost
```

Por último, si además de queremos que el VirtualHost se aplique sobre cualquier IP y sobre cualquier puerto, podemos usar las siguientes configuraciones:

- VirtualHost: configuración de VirtualHost
- DocumentRoot: directorio raíz
- ServerName: nombre del sitio web
- VirtualHosts
- VirtualHosts
- VirtualHosts
- VirtualHosts
- VirtualHosts

Para saber más

En el siguiente enlace podrás encontrar ejemplos de configuraciones de VirtualHosts:

 [Ejemplos de configuración de VirtualHosts](#) (en inglés).

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

6.- Módulos.

Caso práctico

Está bien —pensó **María**—. Manos a la obra. Debo **montar un nuevo servidor web Apache** y necesito... veamos:

- ✓ Que **varias aplicaciones web atiendan en el mismo dominio**, tal que:
 - ⇒ sucursal-zonaX.empresa-proyecto.com,
 - ⇒ www.sucursal-zonaX.empresa-proyecto.com,
- ✓ Un **único panel de control de usuarios**, en la URL:
 - ⇒ www.empresa-proyecto.panel-de-control.com,
- ✓ También **soporte SSL para cifrado**.
- ✓ **Soporte para páginas dinámicas** mediante PHP.
- ✓ Y **soporte para control de usuarios LDAP**.



Uhm..., ya lo tengo claro. Tengo que **montar Apache con varios módulos**, así primero instalaré Apache, luego verificaré que módulos vienen instalados por defecto, si me conviene dejarlos instalados o no, igual tengo que desinstalar alguno y tendré que investigar cuáles son los módulos nuevos a instalar.

Muy bien, pues lo dicho, ¡manos a la obra!

La importancia de un servidor web radica en su **estabilidad, disponibilidad y escalabilidad**. Es muy importante poder dotar al servidor web de nuevas funcionalidades de forma sencilla, así como del mismo modo quitárselas.

Es por esto que la posibilidad que nos otorga el servidor web Apache mediante sus módulos hace que sea uno de los servidores web más manejables y potentes que existen:

- ✓ que necesito soporte SSL, me lo da el módulo SSL;
- ✓ que necesito soporte PHP, me lo da el módulo PHP;
- ✓ que necesito soporte LDAP, me lo da el módulo LDAP;
- ✓ que necesito...cualquier otra cosa, habrá un módulo del servidor web Apache que me lo proporcione.



AJC aicann.wordpress.com CC BY-NC-SA

En Apache los módulos se cargan con la directiva `LoadModule`, y, tal y como ya se ha comentado con anterioridad, son una parte del servidor Apache que se puede cargar según nuestras necesidades. Para poder cargarlos, deben estar previamente compilados. Veamos un ejemplo de como se cargaría un módulo:

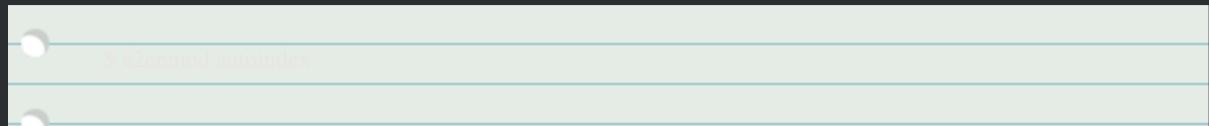


El ejemplo anterior corresponde con un sistema Linux, en otros sistemas esto puede cambiar. Cada módulo aporta un conjunto de directivas y capacidades extra al servidor Apache, con lo que tendremos que consultar la documentación para conocer las directivas proporcionadas por el módulo y los contextos en los que podemos aplicarlas.

En Debian, y derivados (como Ubuntu por ejemplo), cuando se realiza una instalación de Apache desde repositorio tenemos a nuestra disposición utilidades específicas para gestionar los módulos. Existen dos comandos fundamentales para activar y desactivar módulos: `a2enmod` y `a2dismod`.

- ✓ `a2enmod`: utilizado para **habilitar un módulo** de Apache. Sin ningún parámetro preguntará qué módulo se desea habilitar. Los ficheros de configuración de los módulos disponibles están en `/etc/apache2/mods-available/` y al habilitarlos se crea un enlace simbólico en la carpeta `/etc/apache2/mods-enabled/`.
- ✓ `a2dismod`: utilizado para **deshabilitar un módulo** de Apache. Sin ningún parámetro preguntará qué módulo se desea deshabilitar. Los ficheros de configuración de los módulos disponibles están en `/etc/apache2/mods-available/` y al deshabilitarlos se elimina el enlace simbólico desde `/etc/apache2/mods-enabled/`.
- ✓ Si no dispones de esos comandos para poder habilitar y deshabilitar módulos Apache simplemente haces lo que ellos: crear o borrar los enlaces simbólicos correspondientes desde `/etc/apache2/mods-enabled/` hasta `/etc/apache2/mods-available/`.

En la documentación de Apache, y de forma interna a este software, los nombres de los módulos suelen comenzar por mod_ (por ejemplo, mod_autoindex). Pero en distribuciones de Linux tales como Debian o Ubuntu, a dicho módulo se le hace referencia a través de su nombre quitando el mod_ (simplemente autoindex, por ejemplo). Esto tienes que tenerlo en cuenta a la hora de activar o desactivar un módulo. Por ejemplo, para habilitar el módulo mod_autoindex en Ubuntu a través del comando a2enmod lo haríamos como sigue:



Esto es una sola una cuestión de organización interna de Debian y Ubuntu, en otros sistemas operativos puede ser diferente.

Ten en cuenta que la instalación o desinstalación de un módulo no implica la desinstalación de Apache o la nueva instalación de Apache perdiendo la configuración del servidor en el proceso, simplemente implica la posibilidad de poder trabajar en Apache con un nuevo módulo o no.

Recuerda que a2ensite es un comando (en Debian y derivados) para habilitar configuraciones de "sitios web" en Apache2. Los ficheros de configuración de los "sitios web" disponibles (normalmente son configuraciones de hosts virtuales) están en /etc/apache2/sites-available/ y al habilitarlos se crea un enlace simbólico ~~sites-available~~ sites-enabled/. No confundas el comando a2ensite con el comando a2enmod.

Debes conocer

Puedes consultar más información en la documentación de Apache sobre módulos.



[Módulos](#)

Y en el siguiente enlace encontrarás más información sobre como administrar el servidor web Apache en una distribución Debian y derivadas (como Ubuntu), así como el uso de los comandos explicados anteriormente:



[Manual del administrador de Debian, sección servidor web.](#)

6.1.- Operaciones sobre módulos.

Citas para pensar

“

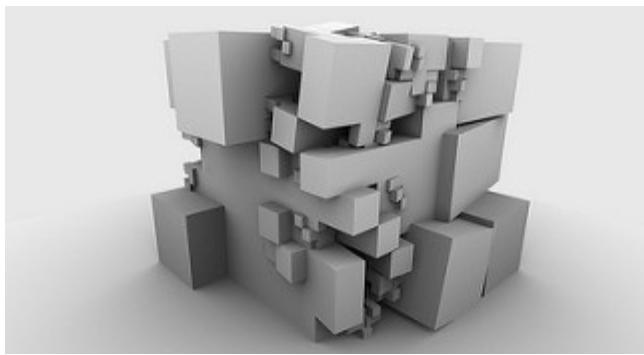
Me lo contaron y lo olvidé. Lo vi y lo entendí. Lo hice y lo aprendí.

Confucio

En esta sección vamos a ver como trabajar de forma cómoda con módulos de Apache en sistemas basados en Debian y Ubuntu. Cada sistema operativo y entorno tiene sus propias peculiaridades y es imposible cubrirlos todos, por lo que aquí se explica como trabajar con módulos partiendo de una instalación de Apache desde repositorio en estos sistemas.

Los módulos de Apache puedes instalarlos, desinstalarlos, habilitarlos o deshabilitarlos, así, puedes tener un módulo instalado pero no habilitado. Esto quiere decir que aunque instales módulos hasta que los habilites no funcionarán.

En la tabla siguiente encontrarás un resumen de operaciones, ejemplos y comandos necesarios que se le pueden realizar a los módulos:



Syntopia CC BY

Operaciones sobre módulos Apache en un GNU/Linux Debian

Operación sobre módulos	Forma genérica del código para nombre-modulo	Ejemplo de forma concreta del código para el módulo ssl
Instalar un módulo	apt-get install nombre-modulo	apt-get install libapache2-mod-gnutls
Desinstalar un módulo	apt-get remove nombre-modulo	apt-get remove libapache2-mod-gnutls
Habilitar un módulo	a2enmod nombre-modulo-apache	a2enmod ssl
Deshabilitar un módulo	a2dismod nombre-modulo-apache	a2dismod ssl

Para **habilitar un módulo** Apache, en Debian, también puedes ejecutar el comando a2enmod sin parámetros. La ejecución de esté comando ofrecerá una lista de módulos a habilitar, escribes el módulo en cuestión y el módulo se habilitará. Del mismo modo para **deshabilitar un módulo** Apache, en Debian, puedes ejecutar el comando a2dismod sin parámetros. La ejecución de esté comando ofrecerá una lista de módulos a deshabilitar, escribes el módulo en cuestión y el módulo se deshabilitará.

Una vez habilitados o deshabilitados los módulos Apache sólo reconocerá estos cambios cuando recargues su configuración, con lo cual debes ejecutar el comando: /etc/init.d/apache2 restart

Si la configuración es correcta y no quieras reiniciar Apache puedes recargar la configuración mediante el comando: /etc/init.d/apache2 reload.

Si no dispones de los comandos `a2enmod` y `a2dismod` puedes habilitar y deshabilitar módulos Apache creando los enlaces simbólicos correspondientes desde `/etc/apache2/mods-enabled/` hasta `/etc/apache2/mods-available/`, por ejemplo si quisieras **habilitar** el módulo `ssl`:

1.- Te sitúas en el directorio `/etc/apache2/mods-available`:

```
root@localhost:~# cd /etc/apache2/mods-available
```

2.- Verificas que el módulo aparece en esta ruta y por lo tanto está instalado:

```
root@localhost:~# ls -l
```

Este comando debe listar dos ficheros: `ssl.conf` (la configuración genérica del módulo) y `ssl.load` (la librería que contiene el módulo a cargar).

3.- Creas el enlace simbólico para habilitar el módulo:

```
root@localhost:~# ln -s /etc/apache2/mods-available/ssl.conf /etc/apache2/mods-enabled/ssl.conf  
root@localhost:~# ln -s /etc/apache2/mods-available/ssl.load /etc/apache2/mods-enabled/ssl.load
```

Estos comandos crean los enlaces `/etc/apache2/mods-enabled/ssl.conf` y `/etc/apache2/mods-enabled/ssl.load` que apuntan a los ficheros `/etc/apache2/mods-available/ssl.conf` y `/etc/apache2/mods-available/ssl.load` respectivamente.

4.- Recargas la configuración de Apache:

```
root@localhost:~# apache2ctl configtest
```

5.- El módulo `ssl` ya está habilitado.

Y si quisieras **deshabilitarlo**, simplemente eliminas en `/etc/apache2/mods-enabled` los enlaces simbólicos creados, así si quisieras deshabilitar el módulo `ssl` ejecutarías el siguiente comando:

```
root@localhost:~# rm /etc/apache2/mods-enabled/ssl
```

Por último, **no te olvides recargar la configuración de Apache**:`/etc/init.d/apache2 restart`

7.- Tipos MIME.

Reflexiona

¿Cómo se transmite un vídeo por Internet, con qué codificación?

¿Cómo sabe un navegador al seguir un enlace de vídeo, el programa que debe utilizar para reproducirlo?

El estándar Extensiones Multipropósito de Correo de Internet o MIME (Multipurpose Internet Mail Extensions), especifica cómo un programa debe transferir archivos de texto, imagen, audio, vídeo o cualquier archivo que no esté codificado en US-ASCII. **MIME** está especificado en seis  RFC (Request for Comments):

- ✓  [RFC 2045](#)
- ✓  [RFC 2046](#)
- ✓  [RFC 2047](#)
- ✓  [RFC 4288](#)
- ✓  [RFC 4289](#)
- ✓  [RFC 2077](#)



[Steve Rhode CC BY-NC-ND](#)

¿Cómo funciona? Imagínate el siguiente ejemplo: transferencia de una página web.

Cuando un navegador intenta abrir un archivo el estándar MIME le permite saber con qué tipo de archivo está trabajando para que el programa asociado pueda abrirlo correctamente. Si el archivo no tiene un tipo MIME especificado el programa asociado puede suponer el tipo de archivo mediante la extensión del mismo, por ejemplo: un archivo con extensión .txt supone contener un archivo de texto.

Bien, pero **¿cómo lo hace?**

El navegador solicita la página web y el servidor antes de transferirla confirma que la petición requerida existe y el tipo de datos que contiene. Esto último, mediante referencia al tipo MIME al que corresponde. Este diálogo, oculto al usuario, es parte de las  cabeceras HTTP, protocolo que se sigue en la web.

En ese diálogo, en las cabeceras respuestas del servidor existe el campo Content-Type, donde el servidor avisa del tipo MIME de la página. Con esta información, el navegador sabe cómo debe presentar los datos que recibe. Por ejemplo cuando visitas <http://www.debian.org/index.es.html> puedes ver como respuesta en la  cabecera del servidor (txt - 0.38 KB) el campo Content-Type: text/html , indicando que el contenido de la página web es tipo text/html.

Cada identificador de tipo MIME consta de dos partes. La primera parte indica la categoría general a la que pertenece el archivo como, por ejemplo, "text". La segunda parte del identificador detalla el tipo de archivo específico como, por ejemplo, "html". Un identificador de tipo MIME "text/html", por ejemplo, indica que el archivo es una página web estándar.

Los tipos MIME pueden indicarse en tres lugares distintos: el servidor web, la propia página web y el navegador.

- ✓ El servidor debe estar capacitado y habilitado para manejar diversos tipos MIME.
- ✓ En el código de la página web se mencionan tipos MIME constantemente en etiquetas `link`, `script`, `object`, `form`, `meta`, así por ejemplo:

- ◆ El enlace a un archivo hoja de estilo CSS:

```
<link href="estilos.css" rel="stylesheet" type="text/css">
```

- ◆ El enlace a un archivo código javascript:

```
<script language="JavaScript" type="text/javascript" src="script.js"></script>
```

- ◆ Con las etiquetas meta podemos hacer que la página participe en el diálogo servidor-cliente, especificando datos MIME:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

- ✓ El navegador del cliente también participa; además de estar capacitado para interpretar el tipo concreto MIME que el servidor le envía, también puede, en el diálogo previo al envío de datos, informar qué tipos MIME puede aceptar la cabecera HTTP Accept. Así por ejemplo, una cabecera Accept típica de un navegador sería:

```
Accept: application/xhtml+xml, application/xml, */*
```

El valor /* significa que el navegador aceptará cualquier tipo MIME.

Para saber más

Complementos del navegador Firefox para ver cabeceras HTTP/HTTPS:



7.1.- Configurar el servidor para enviar los tipos MIME correctos.

En el servidor web Apache podemos especificar el tipo MIME para aquellos archivos que el servidor no pueda identificar automáticamente como pertenecientes a un tipo concreto, esto es, para aquellos que no se resuelven según su extensión.

Pero, ¿cómo consigue Apache identificar automáticamente el tipo MIME de un archivo? Pues a partir de su extensión. Si el archivo tiene como extensión jpg, el tipo usado por Apache será image/jpeg. Esto se logra gracias al módulo de Apache mod_mime. Ten en cuenta que esto es válido para contenido estático, dado que en una aplicación web (hecha con PHP por ejemplo), se puede cambiar la cabecera Content-Type según las necesidades.

En el caso de una instalación de Apache desde repositorio en Debian y Ubuntu, el módulo mod_mime suele estar activo por defecto. El archivo /etc/apache2/mods-available/mime.conf está destinado a configurar este módulo, y el archivo  [/etc/mime.types](#) (zip - 6.42 KB), contiene la lista de tipos MIME reconocidos por el servidor.

Este módulo aporta varias directivas interesantes para configurar los tipos MIME: **AddType**, **ForceType** y la obsoleta **DefaultType**.

- ✓ DefaultType sirve para establecer la cabecera Content-Type a cualquier archivo cuyo MIME no pueda determinarse desde la extensión del archivo. Esta directiva está obsoleta desde la versión 2.2.7 de Apache y no debería usarse.
- ✓ ForceType hace que todos los archivos dentro de un directorio sean servidos con el tipo MIME que se establezca (en la cabecera Content-Type se pondrá el tipo MIME especificado en dicha directiva). Solo se podrá usar dentro de un contexto de directorio (secciones Directory, Files y Location, por ejemplo) y archivos .htaccess.
- ✓ AddType permite indicar que tipo MIME debe usarse para una o más extensiones concretas. Suele usarse para indicar el tipo MIME de extensiones no reconocidas por Apache o el sistema operativo. Por ejemplo, si en nuestro servidor las imágenes en formato PNG están almacenadas con extensión .imagen, con esta directiva podemos indicar que los archivos con extensión .imagen tienen el tipo MIME image/png.

Veamos algunos ejemplos. Imaginemos que queremos que cualquier archivo dentro de una carpeta concreta (/var/www/html/misimagenes por ejemplo), independientemente de la extensión que tenga, se sirva como una imagen PNG. Podríamos usar la siguiente configuración:



Scott MacLeod Liddle CC BY-NC-ND

- `ForceType image/png`
- `Content-Type: image/png`

Esto hará que cuando se sirva contenido de dicha carpeta, el navegador siempre pondrá la cabecera Content-Type: image/png, sea cual sea la extensión, por ejemplo:

```
HTTP/1.1 200 OK
Date: Friday, 20-Nov-2015 16:29:28 GMT
Server: Apache/2.4.17 (Ubuntu)
Last-Modified: Mon, 15 Jun 2015 16:29:28 GMT
Content-Type: image/png
Content-Length: 0
Connection: close
Content-Type: image/png
```

En el ejemplo anterior, aunque el archivo solicitado tiene la extensión .html el Content-Type indicado es el de una imagen. Hay que tener cuidado con la directiva ForceType porque puede provocar que el navegador no llegue a mostrar contenido si el tipo MIME no corresponde con el contenido real.

Imaginemos ahora que queremos que nuestro servidor web sirva cualquier archivo con extensión image fuera una imagen en formato PNG, en este caso podríamos usar la directiva AddType de la siguiente forma:

```
AddType image/png image
```

Esta directiva podemos usarla en muchos contextos: a nivel global, dentro de un host virtual, dentro del contexto directorio y en un archivo .htaccess.

Debes conocer

En los siguientes enlaces puedes encontrar más información sobre la directiva AddType.

 [Directiva AddType \(documentación versión actual del servidor HTTP de Apache, en inglés\)](#)

 [¿Qué es la directiva AddType de Apache?](#)

Para saber más

Puedes consultar más información en la documentación de Apache sobre directivas.

 [Directivas](#)

 [Guía rápida de referencia de directivas](#)

Para saber más

En el siguiente enlace encontrarás la lista oficial de los tipos MIME.

 [Lista oficial de los tipos MIME](#)

Autoevaluación

Abres el navegador y solicitas una página web que contiene un vídeo con la extensión .flv a un servidor web Apache. ¿Cuáles de las siguientes afirmaciones son correctas teniendo en cuenta que el vídeo puede reproducirse y visualizarse sin problemas?

- El servidor web no identifica el tipo MIME pero la extensión .flv es reconocida por el navegador, es por esto que el navegador asocia el programa correspondiente al vídeo y se reproduce sin problemas.
- El archivo no es reconocido por el servidor web, por lo que el servidor web envía al navegador otro tipo MIME, compatible con el esperado y el vídeo se reproduce sin problemas.
- Si la extensión .flv no es reconocida por el navegador ni por el servidor web es debido a que el tipo MIME es reconocido por cómo está programada la página web.
- El servidor web no identifica el tipo MIME pero como el servidor web reconoce la extensión .flv modifica la programación de la página web incorporando el código necesario para la reproducción del vídeo.

[Mostrar retroalimentación](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

8.- Autenticación y control de acceso.

Caso práctico

La reunión tuvo lugar. El equipo de **BK Programación** destinado al proyecto de aplicaciones web para varias sucursales de una empresa llegó a un acuerdo para la **autenticación y el control de acceso sobre la aplicación de panel de control**. Se barajaron varias alternativas: usuarios del sistema, ficheros de usuarios, base de datos SQL y LDAP.



Al final se decantaron por dos opciones: **ficheros de usuarios** para el estado de pruebas y **LDAP** para la aplicación definitiva, con lo cual establecieron el siguiente protocolo de actuación:

- 1.- En la aplicación de desarrollo montada por **María** se realizarán las pruebas, siendo los encargados de las mismas **Ana** y **Carlos**.
- 2.- El diseño web de la aplicación recaerá en **Ana**: banners, logos ...
- 3.- **Juan** se dedicará a la programación del panel de control: autenticación por medio de LDAP.
- 4.- La encargada de montar el servicio LDAP, integrarlo en Apache y conseguir el control de acceso fue **María**.

Mientras se espera que **María** instale y configure Apache con LDAP, no se puede probar la autenticación por LDAP, por lo que **María** crea un fichero de usuarios para autenticarse en la aplicación y todos empiezan a trabajar en el resto de las cosas.

Puede que interese impedir el acceso a determinadas páginas ofrecidas por el servidor web, por ejemplo: ¿crees que a una empresa le interesaría que cualquiera tuviera acceso a determinada información confidencial? O puede que interese controlar el acceso hacia un servicio a través de la web, como el correo electrónico. Para este tipo de casos tenemos que pensar en **la autenticación y el control de acceso**.

Cuando nos autenticamos en una web suele transferirse la información de autenticación a una base de datos, que puede existir en la misma máquina que el servidor web o en otra totalmente diferente. Suelen emplearse bases de datos SQL o LDAP para la autenticación de usuarios, siendo  [OpenLDAP](http://www.OpenLDAP.org) una de las alternativas más empleadas.



[OpenLDAP Foundation](#) (Cita: Todos los derechos reservados)

Para saber más

Puedes visitar el enlace de wikipedia [AAA](#) donde encontrarás más información referente a la autenticación:



HTTP proporciona varios esquemas de autenticación, de manera que solo se conceda acceso a un recurso a un conjunto de usuarios que previamente se ha autenticado. Los métodos más usados son:

- ✓ Esquema **Basic**. En este esquema el usuario o identificador, y la clave van codificados en  BASE64. Con operaciones muy simples se puede averiguar la clave si se interceptan los paquetes que viajan entre el cliente y el servidor, por lo que es recomendable que si empleas este método lo hagas combinado con una conexión SSL (HTTPS).
- ✓ Esquema **Digest**. En este esquema la clave no se envía tal cual, como en el caso anterior. En este caso, se utilizan funciones  hash para enviar los datos de autenticación o credenciales.

El funcionamiento de la autenticación vía HTTP se podría resumir a muy groso modo así:

- 1.- El servidor web está configurado de manera que cuando se intente acceder a un recurso se le pedirá al cliente web (navegador generalmente) que indique las credenciales (usuario y contraseña) usando un esquema concreto (Basic, Digest u otro).
- 2.- El cliente web, el navegador generalmente, al recibir la petición de credenciales, pedirá al usuario que proporcione los datos de acceso (generalmente, un identificador y una contraseña).
- 3.- Una vez que el usuario ha indicado sus datos de acceso, el cliente web envía dichos datos al servidor, el cuál, si los datos son correctos, responderá enviando el recurso solicitado.

En el proceso anterior, el navegador preguntará los datos de acceso al usuario generalmente una sola vez. Si el servidor vuelve a pedir los datos de acceso para el mismo recurso otra vez, el navegador enviará los datos que ya había rellenado el usuario anteriormente.

La autenticación en Apache depende de varios módulos:

- ✓ La autenticación Basic depende del módulo mod_auth_basic. Permitirá que el servidor web solicite al cliente web credenciales usando el método Basic.
- ✓ La autenticación Digest depende del módulo mod_auth_digest. Permitirá que el servidor web solicite al cliente web credenciales usando el método Digest.

Esto tenemos que tenerlo en cuenta a la hora de configurar la autenticación. Además, tendremos que prever donde se almacenarán los datos de los usuarios (es decir, que usuarios podrán acceder al recurso y sus respectivas contraseñas). Como se ha dicho antes, los datos de los usuarios se podrán almacenar en diferentes sitios, es lo que se llama el "proveedor" de autenticación:

- ✓ En una base de datos, que dependerá de los módulos mod_authn_dbd y mod_dbd.
- ✓ En un servicio de directorio LDAP, que dependerá del módulo mod_authnz_ldapd.
- ✓ En un simple archivo de disco, que contendrá la lista de usuarios y sus respectivas contraseñas. Este mecanismo dependerá del módulo mod_authn_file.

Por último, es necesario establecer en qué forma se van a limitar los recursos, si se van a limitar los recursos para usuarios concretos, para grupos de usuarios, etc. Por ejemplo:

- ✓ Si vamos a permitir o denegar el acceso a uno o varios usuarios concretos, deberemos usar el módulo mod_authz_user.
- ✓ Si vamos a permitir o denegar el acceso a un grupo de usuarios, deberíamos, por nuestra comodidad, usar el módulo mod_authz_groupfile.

Vamos a ver un ejemplo de autenticación tipo Basic para entender la dinámica de funcionamiento. Usaremos, por simplicidad, un simple archivo de texto como proveedor de autenticación. Para poder hacer la configuración, los módulos mod_auth_basic, mod_authn_file y mod_authz_user deberán estar activos.

En primer lugar, deberemos crear un archivo con los usuarios y contraseñas. Para ello Apache proporciona una utilidad llamada htpasswd, veamos ejemplos de su uso:

- ✓ htpasswd -c /etc/apache2/lista_de_usuarios unusuario unacontraseña: con la opción -c se indica que el archivo de contraseñas se va a crear (esto se hace solo la primera vez, cuando el archivo no existe), después indicamos el archivo donde se guardarán las contraseñas (en este caso /etc/apache2/lista_de_usuarios), después el nombre del usuario (en este caso unusuario) y por último la contraseña (en este caso unacontraseña). No es obligatorio escribir la contraseña, de hecho, si no se escribe, la utilidad nos la preguntará.
- ✓ htpasswd /etc/apache2/lista_de_usuarios otrosusuario otracontraseña: este ejemplo es igual que el anterior, pero sin la opción -c, lo cual implica que el archivo debe estar creado con anterioridad.

Dado que este archivo contiene información sensible, es altamente recomendable ponerlo fuera de la carpeta donde hay contenido web, es decir, fuera del DocumentRoot.

Después de crear el archivo con los usuarios y las contraseñas, debemos configurar Apache. Y aquí tenemos dos opciones, indicar la configuración directamente en los archivos de configuración principal de Apache dentro de una sección Directory o en un archivo .htaccess (lo cual es muy típico). Si se desea poner en la configuración principal, podríamos hacerlo de la siguiente forma:

- Indicamos el directorio donde se efectuará el control de acceso con `<Directory /var/www/html/>`
- Se pone `AuthType Basic`: Añadimos el mecanismo que se le pone a la página.
- Se pone `AuthName "Acceso a contenido protegido"`: Es el nombre que aparecerá en la pantalla para solicitar el acceso.
- Se pone `Require valid-user`: Indicamos que solo los usuarios válidos podrán acceder.
- Puedes ver el resultado ejecutando `curl -v http://127.0.0.1:8080/`

Si deseamos indicar que todos los usuarios de la lista de usuarios pueden acceder al recurso (si se introduce la contraseña bien como es lógico), podemos poner la directiva `Require` de la siguiente forma:

- `Require valid-user`

En este tipo de autenticación es muy típico utilizar archivos `.htaccess` en los directorios que queremos controlar el acceso. Para ello, dentro de la sección `Directory` usamos la directiva `AllowOverride` para indicar que se pueden crear archivos `.htaccess` en los que se puede escribir la configuración de autenticación:

- Indicamos el directorio donde se efectuará el control de acceso
- Se pone `AllowOverride All`: Permite que se puedan escribir los archivos `.htaccess`.

- `AllowOverride AuthConfig`

- `AuthType Basic`

A partir de ese momento, la configuración del control de acceso puedes ponerla en un archivo `.htaccess`. El archivo deberá estar en la carpeta en la que quieras limitar el acceso. Un ejemplo de contenido podría ser el siguiente:

- Contenido de un hipotético archivo `.htaccess`
- `AuthType Basic`
- `AuthName "Debe introducir usuario y contraseña para seguir"`
- `AuthUserFile /var/www/.htpasswd`
- `AuthUserFile /var/www/.htpasswd_realm`
- `AuthBasicProvider file`

A continuación tienes un ejemplo adicional sobre el uso de la autenticación `Basic` sobre HTTPS y usando archivos `.htaccess`:

- ✓ Configuración de un `VirtualHost` seguro:  [virtualhost-ssl-basic](#) (zip - 0.56 KB)
- ✓ Ejemplo de control de acceso usando un archivo `.htaccess`:  [htaccess](#) (zip - 0.26 KB).

Recuerda que para usar archivos `.htaccess`, necesitas tener una configuración en el servidor que permita poner directivas de autenticación en estos archivos, mediante la directiva `AllowOverride`, así: `AllowOverride AuthConfig`.

Para saber más

Puedes visitar el siguiente enlace donde encontrarás más información referente a la autenticación HTTP Basic:

 [Autentificación, autorización y control de acceso \(versión 2.0\)](#)

Y en el siguiente enlace encontrarás dicha información, en inglés, para la última versión del servidor web de Apache:

 [Autentificación, autorización y control de acceso \(última versión, en inglés\)](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

8.2.- Autenticar usuarios en apache mediante LDAP.

Se ha comentado en el apartado anterior que el servidor web Apache permite la autenticación de usuarios mediante LDAP, es decir, utilizando como proveedor de autenticación el servicio de directorio LDAP. En este apartado vamos a ver como se podría configurar una autenticación basada en LDAP en un servidor web Apache usando Ubuntu o Debian.

¿Qué módulos lo hacen posible? Se consigue mediante los módulos `mod_ldap` y `mod_authnz_ldap`. Recuerda que para activar dichos módulos en una distribución de Debian y Ubuntu actual puedes usar los siguientes comandos:

```
$ sudo a2enmod ldap  
$ sudo a2enmod authnz_ldap
```

Debes conocer

Es conveniente que visites el siguiente enlace que contiene información sobre cómo instalar y configurar un servidor OpenLDAP en un GNU/Linux basado en Debian, con el que Apache puede realizar la autenticación.

 [Instalación y configuración del servidor OpenLDAP](#)

Para una instalación de OpenLDAP en Debian 6 o superior visita el enlace al siguiente documento:

 [Instalación y configuración del servidor OpenLDAP en Debian 6 \(zip - 2.2 KB\)](#)

Para saber más

En el siguiente enlace encontrarás más información sobre la autenticación LDAP para el servidor web Apache.

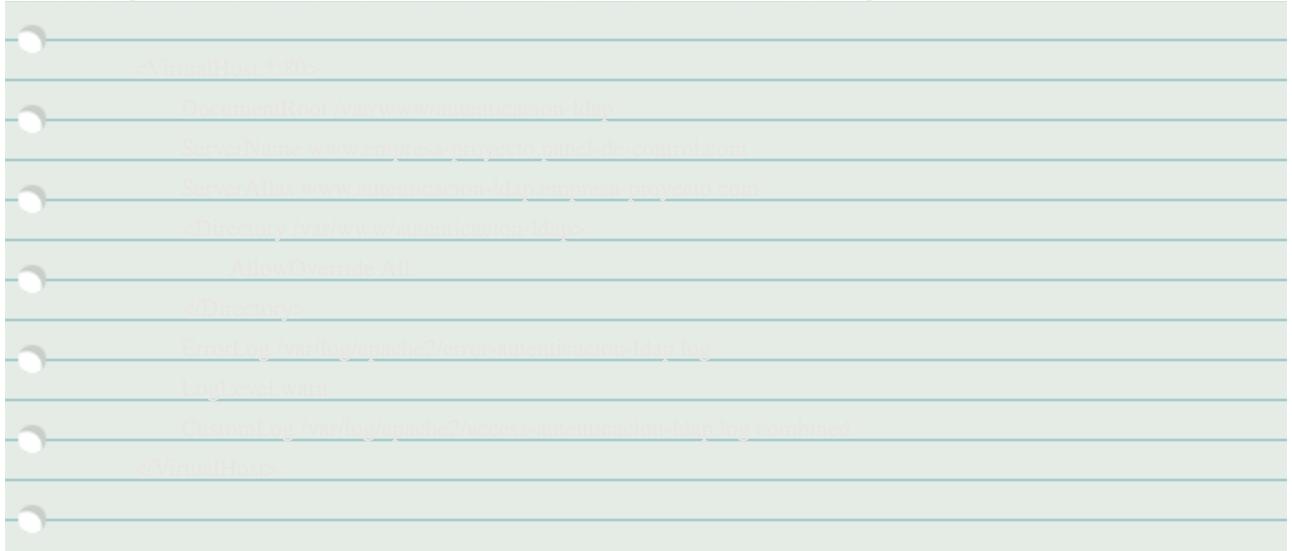
-  [Autenticación LDAP en Apache mediante el módulo authnz_ldap.](#)
-  [Autenticación LDAP en Apache mediante el módulo authnz_ldap en la última versión.](#)

Para el buen funcionamiento de lo expuesto, a continuación se asume que tanto Apache2 como OpenLDAP están instalados y configurados:

1.- Habilita el soporte LDAP para Apache2:



2.- Configura el virtualhost autenticacion-ldap-apache como sigue:



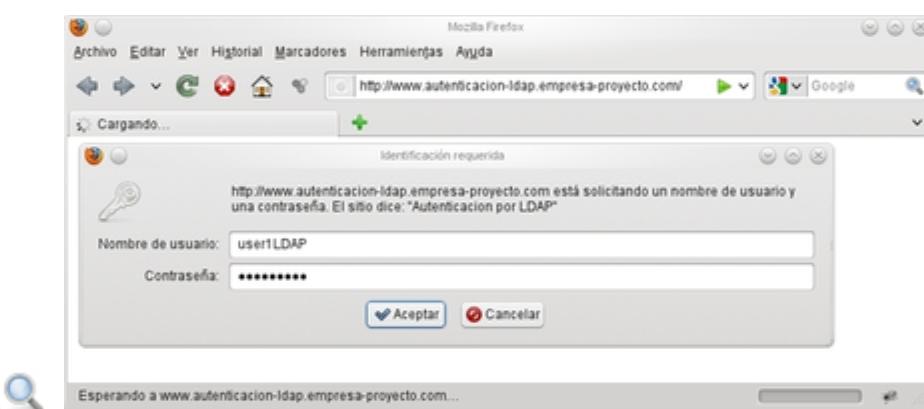
La directiva `AllowOverride All` es necesaria para habilitar ficheros .htaccess

3.- Crea el fichero / var/www/autenticacion-ldap/.htaccess que permite configurar la autenticación LDAP para el virtualhost anterior:



La directiva `Require ldap-user user1LDAP` permite la autenticación al usuario `user1LDAP`, todos los demás usuarios tienen el acceso denegado.

4.- Accede a la URL: www.empresaproyecto.panel-de-control.com o www.autenticacion-ldap.empresaproyecto.com



Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

8.3.- Otras formas de limitar el acceso a recursos.

Existen situaciones en las que tenemos que limitar el acceso a recursos en base a otros parámetros diferentes a usuario y contraseña. Un caso habitual es simplemente denegar el acceso a todo el mundo a una carpeta concreta. Un caso típico es una carpeta que contiene información de configuración de una aplicación web. Esto podemos lograrlo con la directiva `Require`:

- Directiva `Require` con la opción `all denied`
- Ejemplo: `Require all denied`
- Dirección:



[Wikipedia/Espumakid CC BY-SA](#)

En el ejemplo anterior la directriz `Require all denied`, limitará el acceso bajo cualquier condición a un recurso concreto (en este caso el `recurso/www/html/directorioconaccesoprohibido`). En contraposición, la directriz `Require all granted` permitiría el acceso a un recurso bajo cualquier condición.

Otro caso habitual es limitar el acceso a un directorio o VirtualHost en función de la dirección IP del cliente web. Si la dirección IP es una concreta podemos permitir el acceso de la siguiente forma:

- Directiva `Require ip` para limitar el acceso por IP
- Ejemplo: `Require ip 10.20.30.40`
- Dirección:

El módulo de Apache que nos permite el control de acceso basado en IP anterior es el módulo `mod_authz_host`. En el ejemplo anterior la directiva `Require` limitará el acceso a aquellos clientes web cuya dirección IP sea diferente a la especificada. También podemos indicar lo contrario, por ejemplo, permitir el acceso a todos aquellos cuya dirección IP sea diferente a una dada, por ejemplo:

- Directiva `Require not ip` para denegar el acceso por IP
- Ejemplo: `Require not ip 10.20.30.40`
- Dirección:

Se pueden especificar más de una IP o un rango de direcciones IP. Por ejemplo, en la siguiente configuración se permite el acceso a un recurso para todas aquellas direcciones IP que comiencen por 10.20.:

• `<VirtualHost *:80>`

`ServerName www.miejemplo.local`

`ServerAlias localhost`

En todos los casos anteriores se devolverá un código de estado 403 (acceso no permitido o prohibido).

En algunos casos, puede interesar enviar al usuario a otra página URL cuando se solicite un recurso concreto (por ejemplo, si el recurso ha cambiado de lugar). Para eso podemos usar la directiva `Redirect`. No vamos a profundizar en el uso de esta directiva, pero si vamos a ver un pequeño ejemplo. Supongamos que tienes configurado un host virtual de la siguiente forma:

• `<VirtualHost *:80>`

`ServerName www.miejemplo.local`

`ServerAlias localhost`

`Redirect permanent / http://www.miejemplo.local/recursoantiguo`

• `</VirtualHost>`

En esta configuración cuando el cliente accede a la URL a `http://www.miejemplo.local/recursoantiguo` es redirigido a `http://www.miejemplo.local/recursonuevo` de forma transparente al usuario. La directiva `Redirect` depende del módulo `mod_alias`.

Para saber más

En el siguiente enlace encontrarás información extra sobre como redirecciones temporales y permanentes en Apache usando la directiva `Redirect`:

 [Crear redirecciones temporales y permanentes en Apache y NGINX \(en inglés\)](#)

En el siguiente enlace encontrarás información sobre la directiva `RewriteRule`, que te puede evitar tener que utilizar la directiva `ServerAlias`, pues te permite reescribir las direcciones URL.

 [Directiva RewriteRule](#)

9.- Acceso a carpetas seguras.

Caso práctico



En el transcurso del proyecto sobre aplicaciones web de varias sucursales para una empresa en las oficinas de la empresa **BK Programación** tuvo lugar la siguiente charla:

— Bien —le dijo **Ana** a **Ada** —, ya tenemos casi configurado el servidor web Apache.

— ¿Entonces? — preguntó **Ada**.

— Nos falta la configuración de la navegación de forma segura, para que la comunicación viaje cifrada.

— ¿Os llevará mucho tiempo?

— Bueno, **María** nos ha dicho a mi y a Carlos lo que tenemos que ir haciendo, pero ya sabes, estamos en prácticas, necesitaremos un poco de ayuda y supervisión.

— Eso lo puedes dar por supuesto. ¡Manos a la obra! — concluyó **Ada**.

Citas para pensar

“ *Depende qué tan hombre eres.*

Miguel de Icaza

¿Todas las páginas web que están alojadas en un sitio deben ser accesibles por cualquier usuario?

¿Todas las accesibles deben enviar la información sin cifrar, en texto claro?

¿Es necesario que todo el trasiego de información navegador-servidor viaje cifrado?

Existe la posibilidad de asegurar la información sensible que viaja entre el navegador y el servidor, pero esto repercutirá en un mayor consumo de recursos del servidor, puesto que asegurar la información implica en que ésta debe ser cifrada, lo que significa computación algorítmica.

El cifrado al que nos referimos es el cifrado de clave pública o asimétrico: **clave pública (kpub)** y **clave privada (kpriv)**. La **kpub** interesa publicarla para que llegue a ser conocida por cualquiera, la **kpriv** interesa que nadie la posea, solo el propietario de la misma. Ambas son necesarias para que la comunicación sea posible, una sin la otra no tiene sentido, así una información cifrada mediante la **kpub** solamente puede ser descifrada mediante la **kpriv** y una información cifrada mediante la **kpriv** solo puede ser descifrada mediante la **kpub**.

Cifrado de clave pública o asimétrico Resumen textual alternativo

En el cifrado asimétrico podemos estar hablando de individuos o de máquinas, en nuestro caso hablamos de máquinas y de flujo de información entre el **navegador (A)** y el **servidor web (B)**. Mira el siguiente ejemplo de funcionamiento del cifrado asimétrico. Amplía las imágenes para verlas mejor, la segunda es un gif animado:



$$\begin{aligned} A(\text{inf}) &\rightarrow \text{inf cifrada} \rightarrow B \text{ [descifrar inf]} \rightarrow B(\text{inf}) = A(\text{inf}) \\ A(\text{inf}) &\rightarrow \text{inf cifrada} = [\text{inf}]kpub_B \rightarrow [B \text{ inf cifrada}]kpriv_B \rightarrow B(\text{inf}) = A(\text{inf}) \end{aligned}$$

Leyenda o identificación para el ejemplo

Escritura en el ejemplo	Significado
A	Navegador web.
$\text{inf cifrada} = \underline{[(\text{inf})]kpubB}$	Información cifrada mediante la clave pública de B obtenida a través de un certificado digital.
$\underline{[\text{inf cifrada}]kprivB}$	Información descifrada mediante la clave privada de B.
B	Servidor web

Como ves, **A** envía la información cifrada mediante la **kpubB** y **B** la descifra mediante su clave privada (**kprivB**), por lo que se garantiza la confidencialidad de la información.

Pero, ¿estás seguro que B es quién dice que es? ¿Es quién debe ser? ¿Cómo garantizas la autenticidad de B?

Ya que supones que B es quien dice ser mediante un certificado digital, debes confiar en ese certificado, así ¿quién emite certificados digitales de confianza?

Igual que el **DNI** es emitido por una entidad certificadora de confianza, el Ministerio del Interior, en Internet existen autoridades de certificación (**CA** ó **AC**) que aseguran la autenticidad del certificado digital, y así la autenticidad de B, como:  [Verisign](#) y  [Thawte](#)

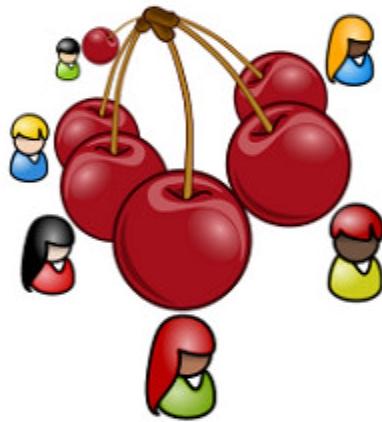
Pero, como ya hemos comentado el Servidor Web Apache permite ser **CA**, por lo que tienes la posibilidad de crear tus propios certificados digitales, ahora bien, ¿el navegador web (A) confiará en estos certificados?

En principio no, por lo que los navegadores avisarán que la página a la cual intentas acceder en el servidor web representa un peligro de seguridad, ya que no existe en su lista de autoridades certificadoras de confianza. En determinados casos, por imagen, puede ser un problema, pero si la empresa posee una entidad de importancia reconocida o el sitio es privado y no público en Internet o sabes el riesgo que corres puedes aceptar la comunicación y el flujo de información viajará cifrado.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

9.1.- Certificados digitales, AC y PKI.

Un certificado digital es un documento electrónico que asocia una clave pública con la identidad de su propietario, individuo o máquina, por ejemplo un servidor web, y es emitido por autoridades en las que pueden confiar los usuarios. Éstas certifican el documento de asociación entre clave pública e identidad de un individuo o máquina (servidor web) firmando dicho documento con su clave privada, esto es, mediante firma digital.



[rfc \(CC BY-NC-SA\)](#)

La idea consiste en que los dos extremos de una comunicación, por ejemplo cliente (navegador web) y servidor (servidor web Apache) puedan confiar directamente entre sí, si ambos tienen relación con una tercera parte, que da fe de la fiabilidad de los dos, aunque en la práctica te suele interesar solamente la fiabilidad del servidor, para saber que te conectas con

el servidor que quieras y no con otro servidor -supuestamente cuando tú te conectas con el navegador al servidor eres tú y no otra persona la que establece la conexión-. Así la necesidad de una Tercera Parte Confiable (TPC ó TPP, Trusted Third Party) es fundamental en cualquier entorno de clave pública.

La forma en que esa tercera parte avalará que el certificado es de fiar es mediante su firma digital sobre el certificado. Por tanto, podremos confiar en cualquier certificado digital firmado por una tercera parte en la que confiamos. La **TPC** que se encarga de la firma digital de los certificados de los usuarios de un entorno de clave pública se conoce con el nombre de **Autoridad de Certificación (AC)**.

El modelo de confianza basado en **Terceras Partes Confiables** es la base de la definición de las **Infraestructuras de Clave Pública (ICP o PKI, Public Key Infrastructures)**. Una Infraestructura de Clave Pública es un conjunto de protocolos, servicios y estándares que soportan aplicaciones basadas en criptografía de clave pública.

Algunos de los servicios ofrecidos por una **ICP (PKI)** son los siguientes:

- ✓ **Registro de claves:** emisión de un nuevo certificado para una clave pública.
- ✓ **Revocación de certificados:** cancelación de un certificado previamente emitido.
- ✓ **Selección de claves:** publicación de la clave pública de los usuarios.
- ✓ **Evaluación de la confianza:** determinación sobre si un certificado es válido y qué operaciones están permitidas para dicho certificado.
- ✓ **Recuperación de claves:** posibilidad de recuperar las claves de un usuario.

Las **ICP (PKI)** están compuestas por:

- ✓ **Autoridad de Certificación (AC):** realiza la firma de los certificados con su clave privada y gestiona la lista de certificados revocados.
- ✓ **Autoridad de Registro (AR):** es la interfaz hacia el mundo exterior. Recibe las solicitudes de los certificados y revocaciones, comprueba los datos de los sujetos que hacen las peticiones y traslada los certificados y revocaciones a la **AC** para que los firme.

Existen varios formatos para certificados digitales, pero los más comúnmente empleados se rigen por el estándar  [UIT-T](#)  [X.509](#). El certificado X.509 contiene los siguientes campos:

- ✓ versión;
- ✓ nº de serie del certificado;
- ✓ identificador del algoritmo de firmado;
- ✓ nombre del emisor;
- ✓ periodo de validez;
- ✓ nombre del sujeto;
- ✓ información de clave pública del sujeto;
- ✓ identificador único del emisor;
- ✓ identificador único del sujeto;
- ✓ extensiones.



Ministerio Educación (captura pantalla sobre Apache) [Licencia Apache 2.0](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

9.2.- Módulo ssl para apache.

Reflexiona

Todos los días los bancos efectúan transferencias bancarias, así como también aceptan conexiones a sus páginas web para ofrecer su servicio online. ¿Qué pasaría si cualquiera pudiese interceptar una comunicación bancaria de ese tipo? ¿Sería interesante cifrar la información a enviar antes y durante la conexión bancaria?

El método de cifrado **SSL/TLS** utiliza un método de cifrado de clave pública (cifrado asimétrico) para la autenticación del servidor.

En Apache, el **módulo mod_ssl** es quien permite cifrar la información entre navegador y servidor web usando dicho cifrado.

Este módulo no suele ir activado por defecto. De hecho, cuando instalamos, por ejemplo, Apache en Ubuntu o Debian desde repositorio este módulo no viene activado, así que se debe ejecutar el siguiente comando para poder activarlo:



DaveBleasdale CC BY

Este módulo proporciona SSL v2/v3 (dependiendo de la versión) y TLS v1 para el Servidor Apache HTTP; y se basa en  OpenSSL para proporcionar el motor de la criptografía.

Debes conocer

En el siguiente enlace puedes encontrar más información sobre el módulo ssl:

-  [Módulo ssl \(versión 2.2\)](#)
-  [Módulo ssl de la versión actual del servidor HTTP de Apache](#)

En la última versión del módulo ssl no está soportado ssl v2.

Ejercicio resuelto

¿Cómo harías, en Debian o Ubuntu, para deshabilitar el módulo ssl de Apache?

[Mostrar retroalimentación](#)

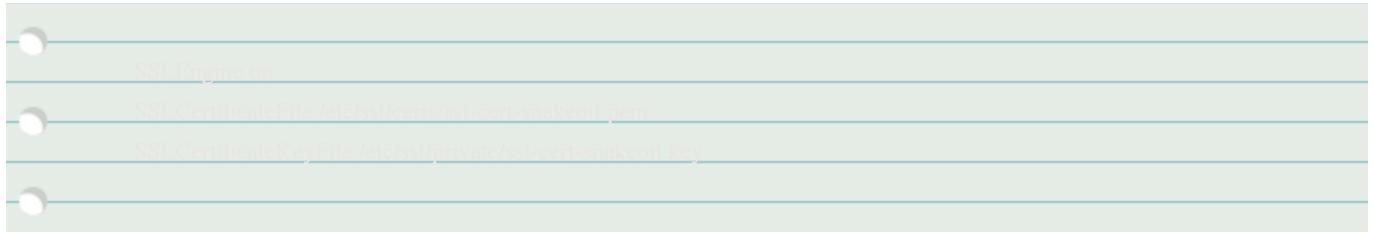
¿Y cómo lo harías si no dispones del comando para Debian o Ubuntu?

[Mostrar retroalimentación](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

9.3.- Crear un servidor virtual seguro en Apache (I).

Cuando instalamos Apache desde repositorio en Debian y Ubuntu, Apache posee por defecto en su instalación el fichero  [/etc/apache2/sites-available/default-ssl.conf](#) (zip - 2.7 kB), que contiene la configuración por defecto de SSL. En su contenido podemos ver las siguientes líneas:



donde,

- ✓ **SSLEngine on** : activa o desactiva SSL a nivel global, o como en este caso, dentro de un host virtual. Para desactivarlo usaríamos **SSLEngine off**. Aunque por defecto está desactivado.
- ✓ **SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem** : certificado digital del propio servidor Apache
- ✓ **SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key**: clave privada del servidor Apache.

Esas líneas lo que quieren decir es que Apache permite conexiones SSL y posee un certificado digital autofirmado por sí mismo, ya que Apache actúa como entidad certificadora.

Cuando activaste el módulo **mod_ssl**, mediante el comando **a2enmod ssl** permitiste que Apache atienda el protocolo SSL. Así que, si ahora lanzas un navegador como **Firefox** con la dirección de tu servidor web Apache mediante el protocolo HTTPS, verás una imagen similar a la siguiente:



Captura pantalla sobre Firefox. [CC BY-SA](#)

Lo que indica que el certificado digital del servidor no viene firmado por una **AC** contenida en la lista que posee el navegador, sino por el mismo Apache. Si lo compruebas haciendo clic en **Detalles Técnicos** verás algo similar a:

- 192.168.200.250 no tiene certificado de seguridad válido.
- No se confía en el certificado porque está caducado.
- El certificado solo es válido para dominio `www`.
- Certificado emitido por `Cloudflare`.

Ahora tienes dos opciones: Confiar en el certificado o no.

- ✓ Si confías haces clic en "**Entiendo los riesgos**" y "**Añadir excepción**". Una vez que confías puedes, antes de "**Confirmar excepción de seguridad**", ver el contenido del certificado. Si estás de acuerdo la comunicación se establece y la información viaja cifrada.
- ✓ Si no confías haces clic en "**¡Sácame de aquí!**"

¿Pero...? Como eres AC puedes firmar certificados e incluso puedes generar también tu propio certificado autofirmado similar al que viene por defecto en Apache.

Hay que tener en cuenta que la negociación SSL es dependiente totalmente de la IP, no del nombre del sitio web, así no puedes servir distintos certificados en una misma IP.

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

9.3.1.- Crear un servidor virtual seguro en Apache (II).

Generar un certificado digital autofirmado no es algo tan complejo. ¿Quieres ver cuál sería, en una distribución Debian o Ubuntu, el procedimiento para generar un certificado digital?

Sería el siguiente:

1.- Instalación del paquete openssl

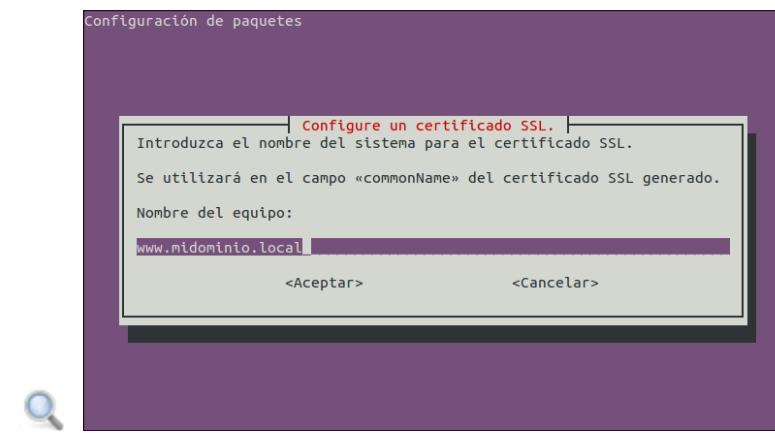


2.- Crear un certificado autofirmado para el servidor web.



En el ejemplo anterior el certificado se creará en el directorio /etc/apache2/tus-ssl/, aunque podría ser otro. Sea cual sea el lugar donde pongas el certificado, no olvides que bajo ningún concepto puede estar en el DocumentRoot (ni en ningún subdirectorio del mismo), dado que representaría un problema grave de seguridad.

3.- En el proceso de creación del certificado, se solicitarán varios datos. Cuando se solicite el nombre del servidor HTTP o nombre del sistema, se debe indicar el nombre DNS que corresponda a la IP del certificado, por ejemplo: autofirmado.ssl.empresaproyecto.com.



Captura de pantalla de Ubuntu 14.04 ejecutando make-ssl-cert (GNU GPL)

Captura realizada por Salvador Romero [GNU GPL](#)

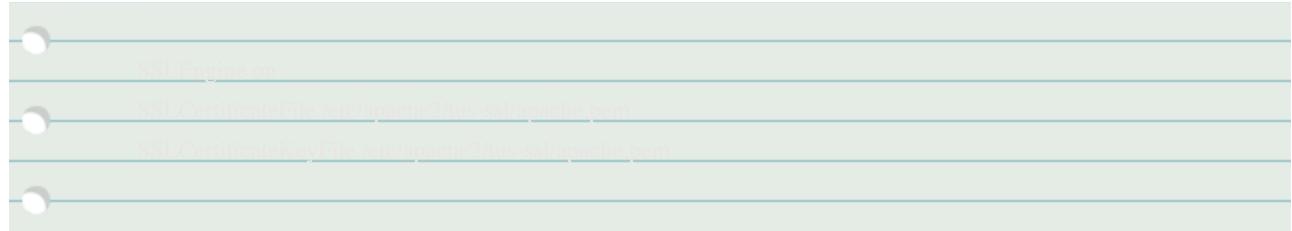
El fichero generado /etc/apache2/tus-ssl/apache.pem contiene tanto el certificado del servidor como la clave privada asociada al mismo.

El comando, de Debian y Ubuntu, make-ssl-cert permite generar certificados autofirmados para pruebas. Los datos de configuración del certificado a generar se indican en /usr/share/ssl-cert/ssleay.cnf. Internamente hace uso de las utilidades de la librería **openssl**.

El nombre de dominio, como `autofirmado.ssl.empresaproyecto.com` del ejemplo anterior, debe resolverse a una IP mediante un servidor DNS o en su defecto mediante el fichero `/etc/hosts`.

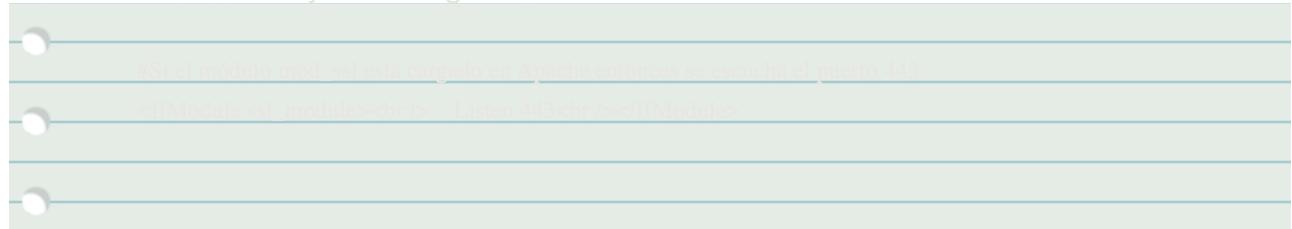
Ahora que hemos creado nuestro certificado autofirmado, podemos usarlo en nuestro servidor Apache. Veamos de forma resumida como podríamos usar dicho certificado. Nuevamente vamos a configurarlo en Ubuntu o Debian partiendo de una instalación de Apache desde repositorio:

1.- Podemos usarlo directamente en la configuración SSL por defecto, que encontraremos en el archivo `/etc/apache2/sites-available/default-ssl.conf`. Para indicar a Apache cual es el nuevo certificado a utilizar y cual es la nueva clave privada podemos usar las siguientes directivas:

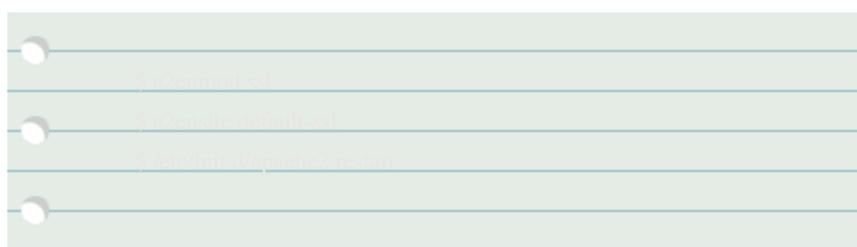


Recuerda que la directiva `SSLEngine` la usamos para activar SSL a nivel global o dentro de un `VirtualHost`.

2.- Asegúrate de que el fichero `/etc/apache2/ports.conf` incluya el valor `Listen 443`. Normalmente se incluye de la siguiente forma:



3.- Habilita el soporte SSL en Apache y habilita la configuración SSL por defecto:



Para verificar que está operativo y bien configurado podríamos hacerlo de varias formas. La más sencilla es simplemente usar un navegador en un equipo cliente:

1.- Se abre el navegador en el equipo cliente.

2.- Se pone la URL en la barra de direcciones el navegador: `https://autofirmado.ssl.empresaproyecto.com`. Nótese que dicha URL usa el esquema HTTPS y el nombre de dominio previamente configurado.

3.- El navegador avisará de que la AC que firma el certificado del servidor no está reconocida. Debemos añadir la correspondiente excepción de seguridad y permitir la descarga y aceptación del certificado. Antes de aceptarlo puedes ver el contenido del certificado:



Captura de pantalla sobre Apache.
[Licencia Apache 2.0](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

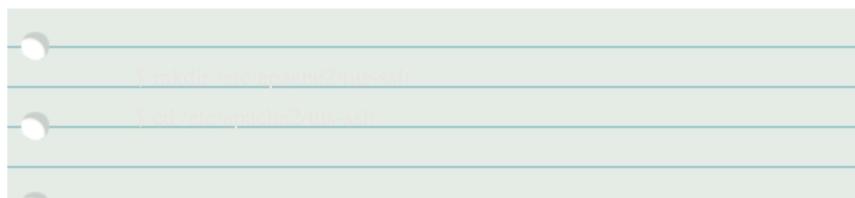
9.3.2.- Crear un servidor virtual seguro en Apache (III).

De forma genérica, por si no posees el comando make-ssl-cert, puedes emplear el software openssl para generar los certificados. Supongamos nuevamente que estamos usando Ubuntu o Debian. ¿Cómo podríamos usarlo? Veamos:

1. Instalación del paquete openssl.



2. Generación del certificado y de la clave privada de tu autoridad de certificación (AC). En primer lugar creamos la carpeta donde se generará el certificado y accedemos a ella:



Después ejecutamos el siguiente comando:



Este comando genera dos archivos:

- ✓ El archivo de la clave privada, con el que firmarás tus futuros certificados: tupaginaweb.key
 - ✓ El archivo del certificado con la clave pública de la AC: tupaginaweb.csr
- Este comando pedirá algunos datos: nombre de empresa, país, contraseña... La contraseña, puedes omitirla, pero por seguridad es conveniente crearla para utilizarla cuando firmes un certificado SSL.

3. Autofirma el certificado. Puedes hacerlo porque eres una AC, de tal forma que el primer certificado que firmas es el de tu propia AC.



El campo **days 3650** significa que el certificado de tu AC tardará 10 años en caducar.

Puedes ver el resultado de la ejecución de los dos comandos anteriores en el archivo  [openssl autofirmado.txt](#) (zip - 0.82 KB) .

4. Editar la configuración SSL por defecto en el archivo /etc/apache2/sites-available/default-ssl.conf para indicar el certificado del servidor y su respectiva clave privada asignando los siguientes valores a los parámetros:

- SSL_Priority
- SSL_CertificateFile /etc/apache2/certs/ssl-cert-snakeoil.pem
- SSL_CertificateKeyFile /etc/apache2/certs/ssl-cert-snakeoil.key

5. Asegúrate que el fichero /etc/apache2/ports.conf incluya el valor Listen 443

6. Habilita el módulo mod_ssl y la configuración SSL por defecto.

- a2enmod ssl
- a2enconf default-ssl
- service apache2 restart

En el archivo /usr/share/doc/apache2.2-common/README.Debian.gz (o /usr/share/doc/apache2/README.Debian.gz dependiendo de la versión) encontrarás información sobre cómo configurar SSL y crear certificados autofirmados. Puedes verlo con el comando siguiente:

```
root@raspberrypi:~# dpkg -r apache2.2-common
```

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

9.4.- Comprobar el acceso seguro al servidor.

Citas para pensar

“

La manera de estar seguro es no sentirse nunca seguro.

Proverbio español

A continuación una serie de actuaciones que te servirán para comprobar que el acceso seguro que estableces con el servidor seguro es el esperado:

- ✓ Siempre que te conectes mediante SSL a un página web y el certificado no sea admitido, debes ver los campos descriptivos del certificado antes de generar la excepción que te permita visitar la página.
- ✓ Debes comprobar en el certificado si la página a la que intentas acceder es la misma que dice el certificado.
- ✓ Típicamente en los navegadores, si no está configurado lo contrario, cuando accedes mediante cifrado SSL a una página web puedes ver en algún lugar del mismo un icono: un candado, por lo cual debes verificar su existencia para asegurarte que estás accediendo por https.

Incluso si el certificado pertenece a alguna AC que el navegador posee en su lista de AC puedes ver en la barra de direcciones indicaciones del tipo de certificado con el que se cifra la comunicación.

- ✓ Revisar la lista de certificados y autoridades certificadoras admitidas que posee tu navegador. En **Firefox**, versión > 3.x , donde x es el número de revisión de la versión 3, puedes verlas dirigiéndote por las pestañas a:

Editar → Preferencias → Avanzado → Certificados → Ver certificados

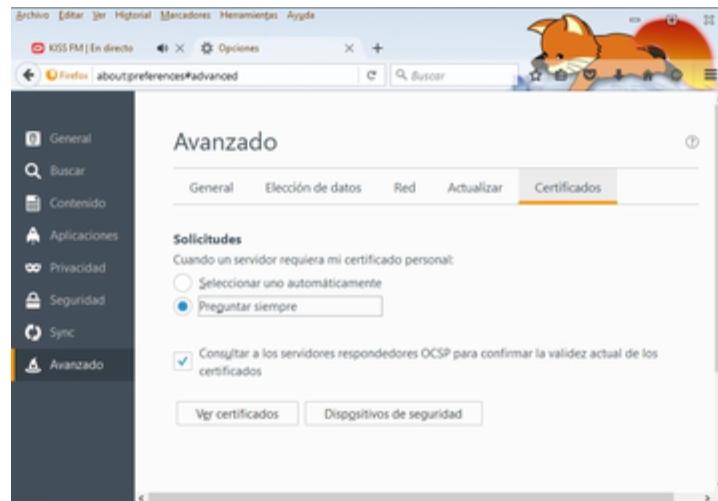
En versiones más recientes de Firefox, puedes acceder usando el menú:

Herramientas → Opciones → Avanzado → Certificados → Ver certificados → Autoridades



ME. Captura de pantalla de Firefox..

[Licencia MPL](#)



NJT [CCO](#)

Puedes Importar/Exportar certificados en los navegadores, con lo cual los puedes llevar a cualquier máquina. Esto es muy útil cuando necesitas un certificado personal en máquinas distintas.

Para saber más

En el siguiente enlace encontrarás información muy interesante, amena y explicativa sobre la seguridad de la información y el cifrado.

 [Enciclopedia de la seguridad de la información.](#)

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

10.- Monitorización del acceso: archivos de registro (logs).

Caso práctico

¿Qué, quién, dónde, cuándo, por qué ha pasado?

—Eso es lo que queremos saber en todo momento —comentó María—. Recordad que es necesario guardar los archivos de registro al menos durante 1 año según la **LSSI/CE**. Tenemos que estar preparados ante cualquier petición de los **logs** (requerimiento judicial) por parte de las administraciones. Es por esto que tú, **Carlos**, vas a realizar una batería de pruebas:

- ✓ accesos a páginas existentes y no existentes;
- ✓ búsqueda de listado de ficheros y no solamente el index.html;
- ✓ accesos no permitidos a bases de datos;
- ✓ accesos controlados por IP;
- ✓ accesos controlados por usuario;
- ✓ etc.

—Muy bien, eso está hecho —dijo **Carlos**.



Tan importante como configurar un servidor web lo es mantener y comprobar su correcto funcionamiento, y para ello debes ayudarte de los **logs** o archivos de registro que te permiten revisar y estudiar su funcionamiento

Apache permite mediante diversas directivas crear archivos de registro que guardarán la información correspondiente a las conexiones con el servidor. Esta información es guardada en formato **CLF (Common Logon Format)** por defecto. Ésta es una especificación utilizada por los servidores web para hacer que el análisis de registro entre servidores sea mucho más sencillo, de tal forma que independientemente del servidor web utilizado podamos emplear el mismo método de análisis de registro, ya sea mediante lectura, mediante programas ejecutables (**scripts**) o mediante programas propios de análisis de registro.

Veamos un ejemplo de log de Apache en formato CLF:



[geralt CC0](#)

host	192.168.200.100
date	[05/May/2011:17:19:18 +0200]
request	/index.html
status	200
Bytes	20

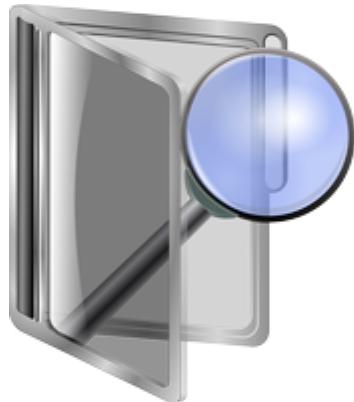
En un archivo de registro en formato CLF cada línea identifica una solicitud al servidor web. Esta línea contiene varios campos separados con espacios. Cada campo sin valor es identificado con un guión (-). Los campos empleados en una configuración por defecto de Apache2 son los definidos en la siguiente tabla:

Campos (especificadores)	Definición	Ejemplo
host (%h)	Identifica el equipo cliente que solicita la información en el navegador.	192.168.200.100
ident (%l)	Información del cliente cuando la máquina de éste ejecuta identd y la directiva IdentityCheck está activada.	
authuser (%u)	Nombre de usuario en caso que la URL solicitada requiera autenticación <u>HTTP</u> .	
date (%t)	Fecha y hora en el que se produce la solicitud al servidor. Va encerrado entre corchetes. Este campo tiene su propio formato: [dia/mes/año:hora:minuto:segundo zona]	[05/May/2011:17:19:18 +0200]
request (%r)	Petición del cliente, esto es, la página web que está solicitando. En el ejemplo /index.html, esto es, dentro de la raíz del dominio que se visite la página	/index.html
status (%s o %>s)	Identifica el código de estado <u>HTTP</u> de tres dígitos que se devuelve al cliente.	200
Bytes (%b)	Sin tener en cuenta las cabeceras <u>HTTP</u> el número de bytes devueltos al cliente.	20

Cada campo tiene su **especificador**, el cual se emplea en las directivas de Apache para indicar que campo queremos registrar.

10.1.- Directivas para archivos de registro.

Vamos a ver cuales son las directivas principales que nos permiten configurar los logs en Apache. Estas directivas pueden emplazarse en dos contextos: de forma global o dentro de hosts virtuales. Las directivas son las siguientes.



[OpenClipartVectors CC0](#)

Directivas para archivos de registro.

Directivas	Definición
<code>LogFormat	Directiva que permite indicar el formato que tendrá el archivo de registro (log). Gracias a esta directiva podemos definir que campos relacionados con la petición <u>HTTP</u> queremos incluir en cada entrada del archivo de log. Suele usarse especialmente con la directiva TransferLog y CustomLog . Además con esta directiva, podemos crear varios formatos de log y darles un nombre al formato en sí (alias).
<code>CustomLog	Directiva que permite indicar el nombre del archivo de registro (log) o programa al que se enviará la información de registro de acceso. Esto quiere decir que cada petición <u>HTTP</u> realizada al servidor web será registrada en el archivo indicado o se enviará a un programa concreto. Dependiendo de donde se configure, la información registrada será las peticiones a todo el servidor web (configuración a nivel global), o a un host virtual concreto (configuración dentro de un host virtual). Es una directiva similar a la directiva TransferLog, pero con la particularidad de que permite personalizar el formato de registro empleando los especificadores anteriormente vistos, o bien indicando un formato de log declarado con LogFormat.
TransferLog	Es similar a la directiva CustomLog, salvo que no permite especificar el formato de log que se usará. Si en algún momento se especifica el formato de log con LogFormat, CustomLog usará dicho formato (el más reciente o cercano si se especifican varios formatos con LogFormat). Si no se especifica, utilizará lo que se llama el formato <u>CLF</u> visto antes.
ErrorLog	Directiva que permite registrar todos los errores que encuentre Apache. Permite guardar la información en un archivo de registro o bien en syslog. Los errores aquí registrados se pueden usar para identificar errores internos y configuraciones erróneas de Apache.

La tabla siguiente muestra la sintaxis y el uso de las anteriores directivas:

Sintaxis y uso de directivas para archivos de registro

	Sintaxis	TransferLog nombre_fichero_archivo_registro tubería_para_enviar_al_programa_la_información_de_registro
Directiva TransferLog	Uso	<p>Normalmente se indica primero el formato de log a usar:</p> <pre>LogFormat %>s %b \"%{Referer}\\" \"%{User-agent}\\" "<span</pre> <p>Y después el archivo donde se guardarán los logs:</p> <pre>TransferLog logs/acceso_a_empresa1.log</pre>
	Sintaxis	<pre>LogFormat format alias_logformat [alias_logformat]</pre> <p>Permite definir un formato que se usará en los archivos referidos por TransferLog y por CustomLog. Cuando tenemos varios formatos, incluso ponerle un nombre a un formato para ser usado por TransferLog (alias_logformat).</p>
Directiva LogFormat	Uso	<pre>LogFormat "%h %l %u %t \"%r\" %>s %O" common
 \"%r\" %>s %O" sería el formato del log y common sería el nombre del formato.</pre> <p>Los nicknames o alias definidos con LogFormat de forma global no son visibles dentro de un virtual host, por lo que es habitual definirlos solo una vez a nivel global, y usarlos luego a conveniencia de cada virtual host.</p>
Directiva ErrorLog	Sintaxis	ErrorLog nombre_fichero_archivo_registro
	Uso	ErrorLog logs/acceso_a_empresa1.log
	Sintaxis	<pre>CustomLog nombre_fichero_archivo_registro tubería_para_enviar_al_programa_la_información_de_registro formato nickname [variable_de_entorno_opcional]</pre>
Directiva CustomLog	Uso	<p>Podemos indicar el formato de log directamente:</p> <pre>CustomLog logs/acceso_a_empresa1.log %>s %b</pre> <p>Y también podemos hacer referencia a un formato de log previamente definido con la directiva LogFormat, a través de su nickname o alias:</p> <pre>CustomLog logs/acceso_a_empresa1.log %>s %b</pre>

En GNU/Linux puedes comprobar en tiempo real desde un terminal en qué equipo se guardan los logs, que puede ser el propio equipo servidor web. Eso lo consigues accediendo a una página web observando el contenido de los archivos de registro mediante el comando: tail -f nombre_archivo_de_registro.log

Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

10.2.- Rotación de los archivos de registro (I).

Como los archivos de registro a medida que pasa el tiempo van incrementando su tamaño, debe existir una política de mantenimiento de registros para que éstos no consuman demasiados recursos en el servidor, así es conveniente **rotar los archivos de registro**, esto es, hay que depurarlos, comprimirlos y guardarlos. Básicamente tienes dos opciones para rotar tus registros:

- ✓ rotatelogs un programa proporcionado por Apache;
- ✓ logrotate, una utilidad presente en la mayoría de los sistemas GNU/Linux.

No debes olvidar que la información recopilada en los **ficheros log** se debe conservar al menos durante 1 año por eventuales necesidades legales, de este modo, además de rotarlos se opta habitualmente por **comprimir logs**.



El comando rotatelogs suele usarse con la directiva **CustomLog** y **ErrorLog**. ¿Recuerdas que a dichas directivas se le podía indicar un programa al que se le enviaría la información de registro de acceso? Pues este es un ejemplo. Veamos como se usaría:

```
CustomLog "|ruta_rotatelogs ruta_log_a_rotar numero_segundos|tamaño_máximoMB"  
alias_logformat
```

La cadena "**|ruta_rotatelogs ruta_log_a_rotar numero_segundos|tamaño_máximoMB**" indica a **CustomLog** que debe ejecutarse un programa en vez de guardarse en un archivo. Veamos algunos ejemplos concretos:

Ejemplos de uso de rotatelogs en combinación con CustomLog.

Ejemplo	Comando rotatelogs para implementar el ejemplo
Rotar el archivo de registro access.log cada 24 horas	CustomLog " /usr/bin/rotatelogs /var/log/apache2/access.log 86400" common
Rotar el archivo de registro access.log cada vez que alcanza un tamaño de 5 MB	CustomLog " /usr/bin/rotatelogs /var/log/apache2/access.log 5M" common
Rotar el archivo de registro error.log cada vez que alcanza un tamaño de 5 megabytes y el archivo se guardará con el sufijo de formato: YYYY-mm-dd-HH_MM_SS (Año-Mes-Día-Hora_Minutos_Segundos)	ErrorLog " /usr/bin/rotatelogs /var/log/errorlog.%Y-%m-%d-%H_%M_%S 5M" common
Rotar el archivo de registro access.log cada vez que alcanza un tamaño de 5 MB manteniendo un histórico de 10 archivos (cuando haya más de 10 archivos, se irán borrando los más antiguos). La opción -n solo está disponible a partir de la versión 2.4.5 de Apache.	CustomLog " /usr/bin/rotatelogs -n 10 /var/log/apache2/access.log 5M" common

Los ficheros rotados por intervalo de tiempo, lo harán siempre y cuando en el intervalo de tiempo definido existan nuevos datos.

Por defecto, si no se define formato mediante ningún modificador % para guardar los archivos de registro, el sufijo nnnnnnnnnn (10 cifras) se agrega automáticamente y es el tiempo en segundos traspasados desde las 24 horas (medianocche).

Como ya se explico antes, el alias_logformat usado en la directiva LogFormat permite definir un grupo de modificadores en una palabra, de tal forma que incorporando esa palabra en la directiva log correspondiente estás activando todo un grupo de modificadores. Si hemos instalado Apache en Ubuntu o Debian desde repositorio, podemos ver que en el archivo /etc/apache2/apache2.conf existen los siguientes alias o nicknames ya definidos: vhost_combined, combined, common, referer y agent, que puedes ver a continuación:

- ✓ `LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\" vhost_combined`
- ✓ `LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\" combined`
- ✓ `LogFormat "%h %l %u %t \"%r\" %>s %O" common`
- ✓ `LogFormat "%{Referer}i -> %U" referer`
- ✓ `LogFormat "%{User-agent}i" agent`

Si instalas Apache 2.4 o superior desde repositorio en una versión reciente de Ubuntu o Debian puedes usar el comando rotatelogs también de la siguiente forma:



Las otras formas vistas antes también funcionan, pero de esta manera no es necesario conocer la ruta entera del comando.

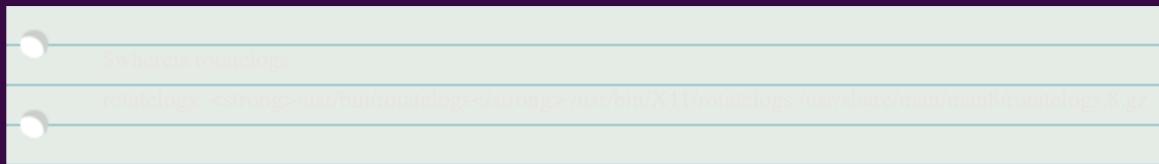
Debes conocer

Es conveniente que le des una visita al manual de rotatelogs: `man rotatelogs`. También puedes encontrar más información sobre el comando en la página oficial de Apache:

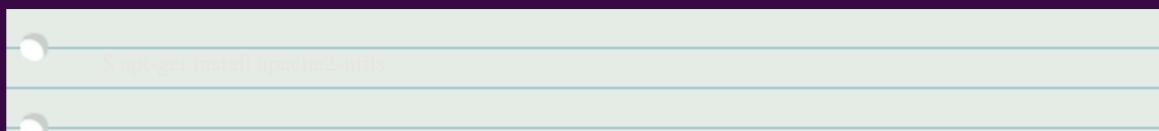


[Documentación sobre el comando rotatelogs \(en inglés\).](#)

Ten en cuenta que la ubicación del comando rotatelogs puede cambiar dependiendo del sistema Linux y de la versión. Puedes encontrar su ubicación exacta en cualquier sistema Linux con el comando siguiente:



Es muy probable que dependiendo de la versión de Debian y Ubuntu tengas que instalar un paquete adicional para poder usar el comando rotatelogs:



Obra publicada con [Licencia Creative Commons Reconocimiento 4.0](#)

10.2.1.- Rotación de los archivos de registro (II).



ME. [CC BY-NC-SA](#)

El programa logrotate rota, comprime y envía archivos de registro a diario, semanalmente, mensualmente o según el tamaño del archivo. Suele emplearse en una tarea diaria del  **cron**. Es un programa que se puede encontrar de forma habitual en cualquier sistema Linux.

En Debian y Ubuntu puedes encontrar los siguientes archivos de configuración para logrotate:

- ✓ /etc/logrotate.conf : define los parámetros globales, esto es, los parámetros por defecto de logrotate. Puedes encontrar un archivo tipo en el siguiente enlace:  [logrotate.conf](#) (conf - 0.56 KB)
- ✓ /etc/logrotate.d/apache2 : define como será el rotado de logs para Apache, todos aquellos parámetros que no se encuentren aquí recogen su valor por omisión del archivo /etc/logrotate.conf. Puedes encontrar un archivo tipo en el siguiente enlace:  [logrotate.d/apache2](#) (d_apache2 - 0.54 KB)

Uso de logrotate

Uso de logrotate	Código concreto a ejecutar
Comprobar la correcta configuración de la rotación de un log	<pre>logrotate -d /etc/logrotate.conf</pre>
Forzar la ejecución de logrotate	<pre>logrotate -f /etc/logrotate.conf</pre>
Script básico para ejecutar logrotate diariamente en el cron. Podemos encontrar un ejemplo de este código /etc/cron.daily/logrotate.	<pre>#!/bin/sh # Logrotate script # # /etc/cron.daily/logrotate # Logrotate configuration file /etc/logrotate.conf</pre>
Ejemplo en el que se ha añadido directamente en archivo /etc/crontab el trabajo a ejecutar por cron (para editar este archivo es mejor usar el comando crontab -e).	<pre>#!/bin/sh # Logrotate configuration file /etc/logrotate.conf # Run logrotate daily 0 0 * * * root /usr/sbin/logrotate /etc/logrotate.conf > /dev/null 2>&1</pre>

Recomendación

El **rotado de logs** descrito anteriormente lo podemos aplicar a cualquier otra herramienta del sistema. Es conveniente que le des una visita al manual de logrotate: man logrotate.

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



MINISTERIO
DE EDUCACIÓN
Y FORMACIÓN PROFESIONAL



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 03.00.01

Fecha de actualización: 21/09/21

Actualización de materiales y correcciones menores.

Versión: 03.00.00

Fecha de actualización: 16/06/17

Autoría: Salvador Romero Villegas

Ubicación: Varios apartados

Mejora (tipo 1): Reestructurar la unidad.

Ubicación: Apartados 1 y 2

Mejora (tipo 3): Propuestas de mejora para la DAW01:

Agregar contenido sobre protocolos de red implicados en la web. Protocolo HTTP. Métodos GET/POST/PUT/DELETE/HEAD. Cabeceras HTTP y respuestas (20X, 30X, 40X).

Introducción rápida a la resolución de nombres. Conceptos de FQDN y URL.

Explicar también que es la Web Social y la Web Semántica.

Actualizar la unidad para hacer una instalación más “générica” en cualquier Debian/Ubuntu actual y en otros sistemas como Windows.

Adaptar los apuntes a la versión 2.4 de apache, porque se hace referencias a cosas de la versión 2.0 y 2.2 que en algunos casos están obsoletas (como la directiva DefaultType).

Explicar que es una directiva de apache y una sección. Que tipos de secciones hay. Explicar que es el contexto de una directiva y módulo que la proporciona.

Explicar secciones tipo “file”, “directory” o “location”, en especial la directory, dado que es importante usarlas junto con las directivas Options y AllowOverride.

Explicar como limitar el acceso a recursos (respuesta 401 u otra) cuando hay por ejemplo una carpeta con información de configuración para una aplicación web php, a través de la directiva redirect. Y no estaría mal tratar superficialmente la reescritura de URL con mod_rewrite.

Cambiar el apartado que trata los Virtualhosts basados en nombre, dado que pone una ip, y se supone que el contenido que se sirve se determina en base al FQDN. Tratar los comandos “a2ensite” y “a2dissite” en el apartado 2.2

Mejorar el tratamiento de la limitación de acceso por IP dado que “allow” y “deny” están deprecated en favor de “require ip ...”.

Mejorar el tratamiento del control de acceso por usuario comentando al menos el método Digest de autenticación, y el uso de grupos con la autenticación basada en archivos. Mejorar el tratamiento de la autenticación basada en archivos en general.

Tratar, aunque sea someramente, la existencia de IIS, NGINX y LightHTTPD entre otros, y sus características.

Versión: 02.00.00

Fecha de actualización: 05/07/16

Autoría: Julio Gómez López

Ubicación: No especificada.

Mejora (tipo 1): Materiales realizados por Narciso Jáimez Toro

Ubicación: todo el tema

Mejora (tipo 3): Unificar la unidad 1 y 2. Quitar el punto 3 (Despliegue de aplicaciones) y pasarlo a la unidad 2

Versión: 01.00.00

Fecha de actualización: 30/10/13

Versión inicial de los materiales.