Configuración y administración de servidores Web.

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.

1. Funcionamiento de un servidor Web.

Hoy en día el elemento fundamental que usamos cuando utilizamos Internet es 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, el cual 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 de una empresa -por tratarse de una página web interna, de la empresa, no accesible a Internet-, o a todo aquel 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.

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.

También tenemos que pensar que todo puede crecer y lo que ahora es un servidor web que ofrece un número de páginas, mañana puede ofrecer el doble o 10 veces más, con lo cual tenemos que prever la escalabilidad del servidor web, y también la estabilidad.

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.

Aunque son muchos los servidores web que podemos encontrar disponibles en Internet, a la hora de la verdad, entre casi la totalidad de todos los sitios web disponibles hoy en día Internet sólo se están utilizando unos 4 diferentes.

Existen un par de servidores web (Apache y Microsoft IIS) de uso general y muy extendido que se reparten casi la totalidad del mercado. Por otra parte está **nginx**, relativamente reciente, y que ha cogido bastante popularidad en poco tiempo. También aparece alguna solución de Google debido a la popularidad de sus servicios. A partir de ahí, el resto de soluciones, quizás por ser más específicas para productos o tecnologías más concretas y menos extendidas, apenas significan nada en el reparto total. Quizás, junto con el resto de servidores, también puede destacar **lighttpd**, puesto que actualmente se está hablando bastante de él por su bajo consumo de memoria.

Nos centraremos exclusivamente en el servidor web **Apache** por ser el más utilizado actualmente (e históricamente) con bastante diferencia con respecto a los demás.

Netcraft es una compañía de servicios en Internet que lleva más de 20 años realizando un ranking del uso de los diferentes servidores web en Internet. En él se pueden ver la evolución de las diferentes soluciones a lo largo del tiempo y también como quedan, más o menos, a día de hoy

https://news.netcraft.com/archives/2019/10/24/october-2019-web-server-survey.html

1.1. Servicio de ficheros estáticos.

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, debemos diferenciar cuando accedemos a una página web entre código ejecutable en el lado del servidor y en el lado del cliente. 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 es necesaria dicha intervención 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 de 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 copias la página en la ruta correspondiente donde quieres que se visione la página.

En la instalación de Apache se crea una página web en /var/www/html/index.html referenciada a través del archivo default (/etc/apache/sites-available/default), éste contiene la configuración por defecto, generada en la instalación de Apache, para esa página. Si solamente quieres servir una página web la forma más fácil de hacerlo sería sustituyendo la página index.html, referenciada en default, por la página que quieres servir, por ejemplo celia.html.

1.2. Contenido dinámico.

Si al acceder a una página web, dependiendo de si posees una cuenta de usuario o no 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: cuántos más módulos tengamos que soportar, más tendremos que configurar y actualizar. Esto también tendrá una gran repercusión en la seguridad del servidor web: cuántos 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 que es lo que se ejecuta.

1.3. Protocolo HTTP y HTTPS.

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.

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.

El protocolo HTTPS utiliza cifrado sobre SSL/TLS (protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente Internet) 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.

En el protocolo HTTP cuando se escribe una dirección URL en el navegador, por ejemplo http://www.debian.org, antes de ver la página en el navegador existe todo un juego de protocolos, se traduce el dominio DNS por una IP, una vez obtenida se busca en ella si un servidor web aloja la página solicitada en el puerto 80, mediante el protocolo TCP asignado por defecto al protocolo HTTP. Si el servidor web aloja la página ésta será transferida al navegador.

Puerto: número utilizado en las comunicaciones cliente/servidor, en transmisiones TCP o UDP comprendido entre 1 y 65535, que indica por donde tiene lugar la conexión con un servidor. Están estandarizados, esto es, un servidor suele estar activo siempre por definición en un puerto determinado, pero éste puede que sea modificado en la configuración del servidor. Por ejemplo un servidor web espera en el puerto 80/TCP.

TCP: es uno de los protocolos fundamentales en Internet. Garantiza que los datos serán entregados en su destino sin errores y una vez recogidos ponerlos en el mismo orden en que se transmitieron.

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, así se traduce el dominio DNS por una IP, con la IP se busca el servidor web que aloja la página solicitada en 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. Si el certificado está firmado por un Entidad Certificadora de confianza se acepta el certificado y se cifra la comunicación con él, transfiriendo así la página web de forma cifrada.

Puedes hacer que un servidor web para una determinada página espere 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, por ejemplo: **http://www.tupagina.local:80**, de esta forma el servidor web espera la petición de la página www.tupagina.local en el puerto 80; del mismo modo en la dirección URL: **https://www.tupagina.local:443** espera la petición de la página www.tupagina.local en el puerto 443. Hay que tener en cuenta que cualquiera que quisiera acceder a esas páginas debería saber el puerto TCP de la solicitud.

Si no escribes el puerto TCP en las direcciones URL estas se interpretan en el puerto 80 y 443 para el protocolo HTTP y HTTPS respectivamente:

http://www.tupagina.local:80 es igual que http://www.tupagina.local

https://www.tupagina.local:443 es igual que https://www.tupagina.local

En la página https://www.youtube.com/watch?v=2kezQTo57yM puedes encontrar un vídeo sobre el funcionamiento de Internet.

2. Instalación y configuración avanzada del Servidor Web.

2.1. Instalación.

Puesto que en nuestro caso trabajamos con Debian Linux no necesitamos ir al sitio web del Proyecto Apache para descargarnos el software. Simplemente, usando el gestor de paquetes, instalaremos el servidor web, su serie de utilidades y la documentación:

```
$ sudo apt-get install apache2 apache2-utils
```

Una vez instalado todo el software, podremos echar un vistazo a la carpeta /etc/apache2, que es donde se almacena toda la configuración del servidor:

```
$ ls /etc/apache2
```

apache2.conf **conf-enabled** magic **mods-enabled** sites-available

conf-available envvars mods-available **ports.conf** **sites-enabled**

La estructura de directorios queda de la siguiente manera:

```
/etc/apache2/
|-- apache2.conf
|-- ports.conf
|-- mods-enabled
   |-- *.load
   |-- *.conf
|-- conf-enabled
   |-- *.conf
| -- sites-enabled
   |-- *.conf
```

Hay que tener en cuenta que la configuración de Apache está muy estructurada, por lo que es necesario saber en qué fichero/directorio corresponde configurar cada parte. Realmente es mucho más cómodo disponer de varios ficheros de configuración pequeños que de uno muy grande. Eso también hace que sea más modular a la hora de activar o desactivar según que características que iremos viendo más adelante.

- apache2.conf: El fichero de configuración principal de Apache, donde se pueden realizar cambios generales.
- envvars: Contiene la configuración de las variables de entorno.
- ports.conf: Contiene la configuración de los puertos donde Apache escucha.
- magic: aquí se configura los módulos de imagen, vídeo y sonido usados en el servidor.
- conf-available: Contiene ficheros de configuración adicionales para diferentes aspectos de Apache o de aplicaciones web como phpMyAdmin.
- conf-enabled: Contiene una serie de enlaces simbólicos a los ficheros de configuración adicionales para activarlos. Puede activarse o desactivarse con los comandos a2enconf o a2disconf.
- mods-available: Contiene los módulos disponibles para usar con Apache.
- mods-enabled: Contiene enlaces simbólicos a aquellos módulos de Apache que se encuentran activados en este momento. Se crean utilizando los comandos a2enmod y a2dismod que más adelante explicaremos con más detalle.
- sites-available: Contiene los ficheros de configuración de cada uno de los hosts virtuales configurados y disponibles (activos o no).
- sites-enabled: Contiene una serie de enlaces simbólicos a los ficheros de configuración cuyos hosts virtuales se encuentran activos en este momento. Se crean a través de los comandos a2ensite y a2dissite que más adelante explicaremos con más detalle.

2.2. Estado del servicio.

Apache es lo que se conoce como un demonio, gestionado en Debian a través del comando service, de forma que podemos gestionar el servidor en función de las opciones que pasemos.

Si queremos saber cómo se encuentra nuestro servidor Apache, podemos ejecutar el siguiente comando y se mostrará un detalle de cuál es el estado del mismo. Resulta especialmente útil cuando, por alguna razón desconocida, éste está caído y no tenemos claro porqué. Con este comando podemos encontrar, normalmente, una descripción clara del problema.
$ sudo service apache2 status (/etc/init.d/apache2 status)

Podemos detener el servidor, que permanecerá detenido hasta el siguiente reinicio o hasta que sea arrancado manualmente.
$ sudo service apache2 stop (/etc/init.d/apache2 stop)

Aunque se inicia automáticamente en cada arranque del equipo, quizás lo hayamos parado y queramos iniciarlo de nuevo.
$ sudo service apache2 start (/etc/init.d/apache2 start)

Reiniciar el servidor es muy útil si hemos realizado algún cambio en la configuración. Hay que tener en cuenta que Apache lee la configuración una vez en el arranque y no la vuelve a leer hasta la siguiente vez que inicia. Así, para hacer efectivo el más mínimo cambio tendremos que reiniciarlo con este comando.
$ sudo service apache2 restart (/etc/init.d/apache2 restart)

Y por último, si queremos conocer la versión del servidor que tenemos instalada, podemos hacerlo con el siguiente comando.
$ apache2 -v

2.3. Configuración.

apache2.conf es el fichero de configuración principal. Actualmente hay directivas generales de funcionamiento del servidor. Sólo se comentarán las más utilizadas puesto que algunas raras veces se modifican.

En versiones anteriores, este fichero, aparecían opciones interesantes como puertos donde escucha el servidor, configuración del sitio principal, de los módulos habilitados (o no) y los hosts virtuales. Actualmente todas esas configuraciones han sido trasladadas a ficheros de configuración específicos, por lo que en el fichero general quedan pocas opciones interesantes que habitualmente se modifiquen.

```
...
Timeout 300
KeeAlive On/Off
MaxKeepAliveRequests 100
KeepAliveTimeout 5
...
Include ports.conf
...
<Directory /var/www/>
   Options Indexes FollowSymlinks
   AllowOverride None
   Require all granted
</Directory>
...
AccessFileName .htaccess
...
```

ports.conf en este fichero de configuración se establecen los puertos en los que escucha Apache. Puesto que es algo muy estándar no es habitual modificar el fichero.

```
Listen 80
<IfModule ssl_module>
        Listen 443
</IfModule>
 <IfModule mod_gnutls.c>
        Listen 443
</IfModule>
```

sites-available / sites-enabled en estas dos carpetas se almacenan los ficheros de configuración de los hosts virtuales. En sites-available se crean los ficheros de configuración (estén o no activos) y aquellos que queremos que estén activos se vinculan con la carpeta sites-enabled para que se tenga en cuenta su configuración, ya que Apache sólo carga los que se encuentran en esta última carpeta.

```
$ ls /etc/apache2/sites-available

000-default.conf          default-ssl.conf
```

En este caso sólo nos encontramos con la configuración del sitio principal y tal como Apache lo prepara con la instalación por defecto. Más adelante, veremos como configurar el sitio principal y el resto de hosts virtuales cuando lo necesitemos.

000-default.conf

```
<VirtualHost *:80>

#ServerName www.example.com


    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html


    ErrorLog ${APACHE_LOG_DIR}/error.log

    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

En el anterior fichero de configuración del sitio principal se pueden ver una serie de directivas muy importantes:

ServerName: Aunque se encuentra comentada, se utiliza para definir el dominio de Internet para el que se configura este sitio. Puesto que ahora estamos realizando pruebas en nuestro propio equipo no tendría sentido habilitarlo, puesto que no estamos dando servicio a ningún dominio real. Más adelante veremos como engañar al equipo y configuraremos al completo un sitio de Apache para ver cómo funciona totalmente esta directiva

ServerAdmin: Sirve para definir la dirección de correo del administrador del servidor web. En caso de error se mostrará al usuario y será a quién deba dirigirse para notificar cualquier incidencia

DocumentRoot: Es muy importante puesto que define la ruta del equipo donde se encuentran todos los ficheros que se están sirviendo en cada momento a través del sitio web que define este fichero. Todo lo que haya en este directorio y por debajo de él está, en principio, siendo ofrecido a través de la web.

ErrorLog: Define en que fichero (como texto) se van a ir registrando todos los errores que se produzcan.

CustomLog: Define un fichero donde se registra toda la información posible (dependiendo de cómo se configure) sobre las visitas que llegan al sitio web y sus usuarios. Es la información que, más adelante, nos podrá permitir sacar las estadísticas de visitas de nuestra página web

Y aquí podemos ver como la configuración del sitio principal tiene un vínculo en sites-enabled para que Apache cargue la configuración y el sitio esté operativo.

$ ls -la /etc/apache2/sites-enabled

root root 4096 Nov 19 2015 .

root root 4096 Sep 22 10:26 ..

root root 35 Nov 19 2015 000-default.conf -> ../sites-available/000-default.conf

3. Instalación y configuración de módulos.

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 sea uno de los servidores web más manejables y potentes que existen.

En Debian, y derivados, existen dos comandos fundamentales para el funcionamiento de los módulos en el servidor web Apache: a2enmod y a2dismod.

- **a2enmod:** Utilizado para habilitar un módulo de apache. Sin ningún parámetro preguntará que 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 desde /etc/apache2/mods-enabled/

- **a2dismod:** Utilizado para deshabilitar un módulo de Apache. Sin ningún parámetro preguntará que 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/

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.

3.1.- Operaciones sobre módulos.

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.

Operaciones sobre módulos Apache en GNU/Linux Debian	
Instalar un módulo	Ejemplo: Instalar el módulo ModSecurity
apt-get install nombre-modulo	apt-get install libapache2-modsecurity
Desinstalar un módulo	Ejemplo: Desinstalar el módulo ModSecurity
apt-get remove nombre-modulo	apt-get remove libapache2-modsecurity
Habilitar un módulo	Ejemplo: Habilitar el módulo ModSecurity
a2enmod nombre-modulo-apache	a2enmod modsecurity2
Deshabilitar un módulo	Ejemplo: Deshabilitar el módulo ModSecurity
a2dismod nombre-modulo-apache	a2dismod modsecurity2

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.

Para extender la funcionalidad del servidor web Apache añadiendo y configurando algunos módulos que, aunque se deben de instalar manualmente aparte, empiezan a ser parte indispensable de este servidor:

Soporte para PHP

Para instalar el soporte para PHP en Apache, lo primero que tenemos que hacer es instalar los paquetes para darle soporte. Podemos instalar los de la versión 7.

```
$ sudo apt-get install php7.0
```

El paquete para la versión 7 de PHP tiene como dependencia el libapache-mod-php7.0 que instala el módulo de apache para dar soporte a esta tecnología.

Si además queremos añadir soporte para Bases de Datos, tendremos que instalar el SGBD que nos interese. En el caso de PHP el más utilizado es MySQL, por lo que instalaremos éste:

```
$ sudo apt-get install mysql-server
```

Y también el soporte del lenguaje PHP para conectarse con MySQL:

```
$ sudo apt-get install php7.0-mysql
```

Una vez instalado el soporte para PHP, la mejor forma de comprobar que todo funciona correctamente es invocar a la función phpinfo(), que nos proporciona información muy detallada sobre su instalación. Así, podemos preparar un brevísimo script PHP con las siguientes líneas y copiarlo en el directorio del sitio principal (/var/www/html).

```
<?php
        phpinfo();
 ?>
```

Y al cargar dicho script obtendremos una web con varias tablas donde se detalla en profundidad toda la configuración de PHP para nuestro servidor Apache.

Además, en el caso de que hayamos optado por instalar PHP junto con MySQL, puede resultar de mucha utilidad instalar un gestor para el SGBD, en este caso como aplicación web. Se trata de phpMyAdmin, conocida aplicación web para la gestión de Bases de Datos MySQL.

$ sudo apt-get install phpmyadmin

Para poder acceder comprobar que en estos directorios se han creado los siguientes enlaces durante la instalación:

$ sudo ls -la /etc/apache2/conf-enabled
`phpmyadmin.conf -> ../conf-available/phpmyadmin.conf`

$ sudo ls -la /etc/apache2/conf-available
`phpmyadmin.conf -> ../../phpmyadmin/apache.conf`

Si no es así, crear a mano dichos enlaces o únicamente uno:

$ sudo ln -s /usr/share/phpmyadmin /var/www/html/phpmyadmin

Por defecto crea un usuario phpmyadmin con privilegio de uso, para administrar el SGBD:

mysql -u root

```
> CREATE USER 'nombre_usuario'@'localhost' IDENTIFIED BY 'tu_contraseña';
> GRANT ALL PRIVILEGES ON * . * TO 'nombre_usuario'@'localhost';
> FLUSH PRIVILEGES;
```

Limitar el ancho de banda

Otro de los módulos de Apache, (bw, bandwith), permite controlar el ancho de banda de los usuarios que se conectan a nuestro sitio web en función de numerosos parámetros. En este apartado veremos como instalarlo, activarlo y configurar un par de ejemplos:

Para instalarlo, como siempre, con la herramienta apt-get:

$ sudo apt-get install libapache2-mod-bw

Una vez instalalo el módulo tenemos que cargarlo:

$ sudo a2enmod bw

Y ahora vamos a configurarlo para controlar el ancho de banda en todo el sitio, el máximo, el mínimo y el número de conexiones simultáneas por conexión (por usuario)

```
<VirtualHost *:80>

. . .

    # Turn bandwidth limitation on

    BandwidthModule On

    # force limitation on every request

    ForceBandWidthModule On

    # limit to 1000kb/s

    Bandwidth all 1000000

   # limit max connections to 400

   MaxConnection all 400

</VirtualHost>
```

También es posible, por ejemplo, limitar la velocidad en un determinado directorio para determinados ficheros. Por ejemplo, si almacenamos en un directorio específico los ficheros más grandes (videos por ejemplo) y queremos limitar la velocidad sólo en el caso de que la gente se baje esos ficheros.

```
. . .

  <Directory "/var/www/example1.com/public_html/media">

    # Limit files 1 mB to 200 kb/s.

     LargeFileLimit * 1024 200000

  </Directory>

. . .

  <Directory "/var/www/example1.com/public_html/media">

    # Limit avi and mpg extensions to 50kb/s.

    LargeFileLimit .avi 1 50000

    LargeFileLimit .mpg 1 50000

  </Directory>
```

4. Hosts virtuales. Creación, configuración y utilización.

Se pueden alojar múltiples páginas web en el servidor web Apache, pero todas pertenecientes al mismo sitio/dominio, se pueden alojar múltiples sitios mediante la configuración de hosts virtuales o virtualhosts. Estos básicamente lo que hacen es permitir que un mismo servidor web pueda alojar múltiples dominios, así configurando hosts virtuales podemos alojar: empresa1.com, empresa2.com, ..., empresaN.com en el mismo servidor web. Cada empresa tendrá su virtualhost único e independiente de las demás.

Cada virtualhost es único e independiente de los demás, todo aquello que no esté incluido en la definición de cada virtualhost se heredará de la configuración principal: apache2.conf (/etc/apache2/apache2.conf), si quieres definir una directiva común en todos los virtualhost no debes modificar cada uno de los virtualhost introduciendo esa directiva sino que debes definir esa directiva en la configuración principal del servidor web Apache, de tal forma que todos los virtualhost heredarán esa directiva, por ejemplo en apache2.conf puedes encontrar la directiva Timeout 300, que establece la directiva Timeout igual a 300 segundos, esto es, indica el número de segundos antes de que se cancele un conexión por falta de respuesta.

4.1. Cómo crear un hosting compartido

Consiste en el mantenimiento de diferentes sitios web (independientes entre ellos) en el mismo equipo, compartiendo recursos. Es la forma más económica puesto que, al poder compartir los recursos, es posible crear y mantener un gran número de éstos en el mismo equipo.

$ nano /etc/apache2/sites-available/misitio.local.conf

```
<VirtualHost *:80>
   ServerAdmin webmaster@misitio.local
   DocumentRoot "/var/www/html/misitio.local"
   ServerName misitio.local
   ServerAlias www.misitio.local
   ErrorLog "misitio.local-error.log"
   CustomLog "misitio.local-access.log" combined
</VirtualHost>
```

El comando a2ensite (en Debian y derivados) habilita 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 desde /etc/apache2/sites-enabled/

Cuando hayamos terminado de configurar el nuevo host virtual, podemos activarlo utilizando el

comando a2ensite. Automáticamente se creará un enlace simbólico con la configuración del sitio de sites-available en sites-enabled:

```
$ sudo a2ensite misitio.local
Enabling site misitio.local.
To activate the new configuration, you need to run:
  systemctl reload apache2
```

Básicamente podríamos haber hecho lo mismo con el siguiente comando:

```
$ sudo ln -s ../sites-avaiable/misitio.local.conf /etc/apache2/sites-enabled/misitio.local.conf
```

Tendremos que reiniciar el servicio de Apache para que se active la nueva configuración.

También podemos desactivar un host virtual con el comando a2dissite (apache2 disable site). El sitio permanecerá desactivado hasta que lo volvamos a activar con a2ensite.

```
$ a2dissite misitio.local
Site misitio.local disabled.
To activate the new configuration, you need to run:
  systemctl reload apache2
```

En estos casos, donde empezamos a tener numerosos ficheros de configuración, es donde resulta útil el comando service apache2 status ya que, en caso de que tengamos algún error que impida arrancar el servicio, nos dará la información necesaria para encontrarlo. En el siguiente ejemplo un fichero de configuración de un sitio virtual está dando problemas. Al arrancarlo con a2ensite me ha dado un error y con service apache2 status visualizo toda la información y puedo ver cuál es el motivo del error.

```
$ service apache2 status
```

```
apache2.service - The Apache HTTP Server
Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
Active: failed (Result: exit-code) since Sun 2018-11-04 16:54:31 CET; 53s ago
Process: 1148 ExecStop=/usr/sbin/apachectl stop (code=exited, status=1/FAILURE)
Process: 1155 ExecStart=/usr/sbin/apachectl start (code=exited, status=1/FAILURE)
Main PID: 1047 (code=exited, status=0/SUCCESS)
nov 04 16:54:31 despliegue-daw systemd[1]: Starting The Apache HTTP Server...
nov 04 16:54:31 despliegue-daw apachectl[1155]: apache2: Syntax error on line 233 of
/etc/apache2/apache2.conf:Syntax error on line 8 of /etc/apache2/sites-enabled/misitio.local.conf:
</VirtualHost> directive missing closing '>'
nov 04 16:54:31 despliegue-daw apachectl[1155]: Action 'start' failed.
nov 04 16:54:31 despliegue-daw apachectl[1155]: The Apache error log may have more
information.
```

```
nov 04 16:54:31 despliegue-daw systemd[1]: apache2.service: Control process exited, code=exited status=1
nov 04 16:54:31 despliegue-daw systemd[1]: Failed to start The Apache HTTP Server.
nov 04 16:54:31 despliegue-daw systemd[1]: apache2.service: Unit entered failed state.
nov 04 16:54:31 despliegue-daw systemd[1]: apache2.service: Failed with result 'exit-code'.
```

Si finalmente comprobamos que la configuración es correcta, nuestro host virtual estará funcionando correctamente. Si suponemos que nuestro servidor está corriendo en Internet y que además ya somos dueños del dominio (y éste apunta a nuestro equipo), los usuarios ya podrán empezar a visitar nuestra nueva web misitio.com.

En nuestro caso, en clase, vamos a decirle a nuestro ordenador que el dominio misitio.com apunte a nuestro propio equipo, así podremos comprobar que todo funciona. Para ello editaremos el fichero /etc/hosts que funciona como DNS local.

```
$ nano /etc/hosts
```

Y añadimos la IP que queremos asignar al dominio misitio.com (en nuestro caso nuestro propio equipo, 127.0.0.1)

```
127.0.0.1    localhost
127.0.0.1    daw.local
127.0.0.1    misitio.local
```

Simplemente haciendo ping contra el dominio podemos ver que responde nuestro propio equipo, por lo que todo funciona correctamente.

```
$ ping misitio.local
PING misitio.local (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.029 ms
```

Preparamos ahora una web de bienvenida para el nuevo sitio web: index.html

```
<!DOCTYPE>
<html lang="es">
<head>
   <meta charset="UTF-8">
   <title>Bienvenido a misitio</title>
</head>
<body>
   <p>Estás en misitio</p>
</body>
</html>
```

Y así ahora lo visitamos podremos ver que la página que visitamos es diferente en función de si vamos al host virtual misitio.com $ links http://misitio.local

O al sitio principal $ links http://daw.local

Lo puedes probar desde Windows:

C:\Windows\System32\drivers\etc

Ip_servidor daw.local

Ip_servidor misitio.local

5.- Autenticación y control de acceso.

HTTP proporciona un método de autenticación básico de usuarios. Este método ante una petición del cliente (navegador web) al servidor cuando se solicita una URL mostrará un diálogo pidiendo usuario y contraseña. Una vez autenticado el usuario, el cliente volverá a hacer la petición al servidor pero ahora enviando el usuario y contraseña, en texto claro (sin cifrar) proporcionados en el diálogo. Es recomendable entonces si empleas este método que lo hagas combinado con conexión SSL (HTTPS).

La autenticación HTTP es un método por el cual podemos proteger carpetas dentro de nuestro servidor web (en este caso con Apache).

Lo primero que tendremos que tener instalado es el paquete apache2-utils, que contiene una serie de utilidades para, entre otras cosas, manipular archivos de autenticación (con el comando htpasswd) que serán los ficheros a través de los cuales configuraremos el acceso (mediante usuario/contraseña) a esas carpetas protegidas dentro de nuestro servidor web

$ sudo apt-get install apache2-utils

Una vez instalado apache2-utils, estaremos en disposición de ejecutar el comando htpasswd que se utiliza tanto para crear el fichero por primera vez y añadir un usuario a la zona protegida que vamos a crear, como para añadir otros usuarios más adelante.

Creamos primero un usuario manu con su contraseña y almacenamos el fichero en la ruta de configuración de nuestro servidor web. Como el fichero todavía no existe utilizamos la opción -c

```
$ sudo htpasswd -c /etc/apache2/.htpasswd manu
New password:
Re-type new password:
Adding password for user manu
```

Y añadimos un segundo usuario. En este caso el fichero ya existe y no es necesario especificar ninguna opción adicional al ejecutarlo.

```
$ sudo htpasswd /etc/apache2/.htpasswd otro
New password:
Re-type new password:
Adding password for user otro
```

Si echamos un vistazo al fichero, veremos como quedan almacenados usuario / contraseña, y ésta última cifrada.

```
$ cat /etc/apache2/.htpasswd
manu:$apr1$mDMINEK5$34dXLMYERhHz7QEmhtK.x/
otro:$apr1$tYxVwhYM$8guRCfIs5aIJovO8YazTL/
```

A continuación, asignamos la propiedad del fichero y el grupo al usuario y grupo sobre el que se ejecuta Apache en nuestro Linux, www-data

```
$ chown www-data:www-data /etc/apache2/.htpasswd
```

Puesto que teníamos ya creados algunos hosts virtuales, vamos a seleccionar uno de ellos para proteger dentro del mismo una de las carpetas. En este caso teníamos el sitio misitio.com ya creado como aparece aquí debajo:

```
<VirtualHost *:80>
   ServerAdmin webmaster@misitio.com
   DocumentRoot "/var/www/html/misitio.com"
   ServerName misitio.com
   ServerAlias www.misitio.com
   ErrorLog "misitio.com-error.log"
   CustomLog "misitio.com-access.log" combined
   <Directory "/var/www/html/misitio.com/usuarios">
      AuthType Basic
      AuthName "Acceso Restringido a Usuarios"
      AuthUserFile /etc/apache2/.htpasswd
      Require valid-user
   </Directory>

</VirtualHost>
```

Vamos a añadir la directiva Directory como aparece arriba para proteger un directorio al que llamaremos usuarios donde suponemos que hay cierta información a la que no todo el mundo debe acceder. Configuraremos dicha zona para indicar que es una zona restringida y que sólo los usuarios que están especificados en ese fichero pueden entrar.

Donde:

AuthType Basic: Define el tipo de autenticación

AuthName: Define el mensaje que se mostrará al usuario cuando se le solicite el usuario y contraseña para acceder

AuthUserFile: Se indica la ruta al fichero que hemos creado (tal y como se explica más arriba) con los usuarios/contraseña que tienen permitido el acceso

Require: Indica que sólo se podrá acceder con un usuario válido. Si sólo queremos que se pueda acceder con un único usuario podemos indicar Require user manu si sólo queremos que el usuario manu pueda acceder a esa carpeta.

Para terminar, reiniciamos el servicio de Apache: $ sudo service apache2 restart

Ahora podemos probar a acceder a la URL http://misitio.com/usuarios y veremos como aparece una ventana al estilo popup solicitando que nos identificamos mediante un usuario y una contraseña.

Autenticación HTTP con .htaccess

También es posible, y quizás más cómodo, activar y configurar la autenticación HTTP creado un fichero .htaccess de forma que podamos almacenarlo dentro de la carpeta del sitio que queremos proteger.

Antes de nada tenemos que modificar la configuración general de Apache para permitir que las propiedades sobre el directorio raíz puedan ser modificadas. Cambiaremos el valor AllowOverride All para permitir dichas modificaciones. Así, la política sobre dicho directorio podrá ser diferente a la establecida en la carpeta si así se define con algún fichero .htaccess

/etc/apache2/apache2.conf

```
. . .
<Directory /var/www/>
   Options Indexes FollowSymLinks
   AllowOverride All
   Require all granted
</Directory>
. . .
```

A continuación, sobre la carpeta que queremos proteger con contraseña, creamos un fichero .htaccess con el siguiente contenido:

/var/www/html/misitio.com/usuarios/.htaccess

```
AuthType Basic
AuthName "Acceso Restringido a Usuarios"
```

```
AuthUserFile /etc/apache2/.htpasswd
Require valid-user
```

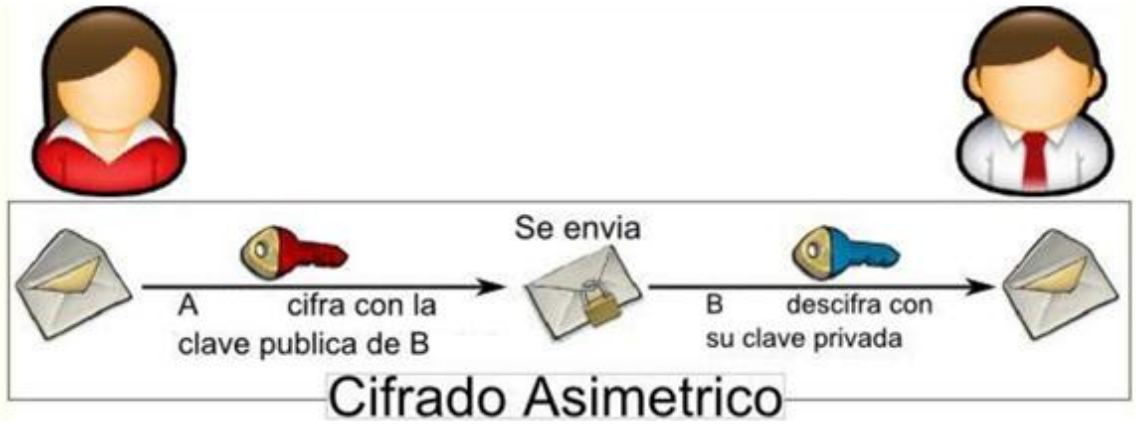
Y, para terminar, tendremos que reiniciar el servicio de Apache: $ sudo service apache2 restart

6.- El protocolo HTTPS.

Existe la posibilidad de asegurar la información sensible que viaja entre el navegador y el servidor, 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 no 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.

http://www.criptored.upm.es/intypedia/video.php?id=criptografia-asimetrica&lang=es



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.

¿Cómo garantizas la autenticidad de B? Pues 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 un entidad certificadora de confianza, el Ministerio del Interior, en Internet existen autoridades de certificación que aseguran la autenticidad del certificado digital, y así la autenticidad de B. 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? Pues, en principio no, por lo que los navegadores avisarán que la página a la cuál 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.

6.1.- 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.

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 quieres 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 TTP, 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 PKIs, 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.

6.2. Seguridad con SSL/TLS en Apache

SSL (Secure Socket Layer) es un protocolo de seguridad que nació con el objetivo de cifrar las comunicaciones entre los servidores web y los navegadores de forma que, si se interceptaba la conexión, nunca se pudiera desvelar el contenido de la misma. Con el paso del tiempo se han ido encontrando diversas vulnerabilidades críticas que han hecho que la recomendación sea usar un nuevo protocolo llamado TLS (Transport Secure Layer).

El primer paso para configurar SSL en Apache será crear el certificado y la clave, que se quedarán almacenados en /etc/apache2/certs. Para eso primero creamos la carpeta y luego el certificado y su correspondiente clave.

Hay que tener en cuenta que estamos creando un certificado autofirmado. Este tipo de certificados sólo se deben usar con el propósito de enseñar o hacer una demostración puesto que en la práctica no son válidos. El navegador no confiará en él porque somos nosotros quienes lo hemos firmado. Los certificados, para que sean válidos, deben ser validados por una entidad certificadora.

```
$ sudo mkdir /etc/apache2/certs

$ sudo openssl req -x509 -sha256 -days 365 -newkey rsa:2048

-keyout /etc/apache2/certs/apache2.key -out /etc/apache2/certs/apache2.crt

Generating a 2048 bit RSA private key

.....+++

..... +++

writing new private key to '/etc/apache2/certs/apache2.key'

Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

-----

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.
```

Country Name (2 letter code) [AU]: SP

State or Province Name (full name) [Some-State]: Almería

Locality Name (eg, city) []:Almería

Organization Name (eg, company) [Internet Widgits Pty Ltd]: IES Celia Viñas

Organizational Unit Name (eg, section) []: 2º DAW

Common Name (e.g. server FQDN or YOUR name) []: daw.org

Email Address []:root@daw.org

Ahora ya tenemos el certificado y su clave. Activamos entonces el módulo SSL de Apache:

$ sudo a2enmod ssl

Considering dependency setenvif for ssl:

Module setenvif already enabled

Considering dependency mime for ssl:

Module mime already enabled

Considering dependency socache_shmcb for ssl:

Enabling module socache_shmcb.

Enabling module ssl.

See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create self-signed certificates.

To activate the new configuration, you need to run:

  systemctl restart apache2

Podemos revisarlo haciendo lo siguiente:

$ sudo openssl x509 -in /etc/apache2/certs/apache2.crt -text -noout

Por lo que tendremos que fijarnos en las líneas:

    Signature Algorithm: sha256WithRSAEncryption

    Public Key Algorithm: rsaEncryption

    RSA Public Key: (2048 bit)

Como se puede ver, se ha usado una encriptación RSA, mediante una clave pública de 2048 y mediante el algoritmo SHA256.

Y ahora configuramos un host virtual para que soporte conexión segura. En este caso he seleccionado el mismo sitio que antes hemos configurado como ejemplo de host virtual y he realizado los cambios necesarios para que ahora soporte conexión segura (HTTPS). Básicamente es cambiar el puerto donde escucha (ahora es 443, donde escuchan las conexiones seguras), activar el soporte para SSL e indicar donde están el certificado y la clave.

```
$ nano /etc/apache2/sites-available/misitio.com.conf

<VirtualHost *:443>

   ServerAdmin webmaster@misitio.com

   DocumentRoot /var/www/html/misitio.com

   ServerName misitio.com

   ServerAlias www.misitio.com

   ErrorLog ${APACHE_LOG_DIR}/misitio.com-error.log

   CustomLog ${APACHE_LOG_DIR}/misitio.com-access.log combined


   SSLEngine On

   SSLCertificateFile /etc/apache2/certs/apache2.crt

   SSLCertificateKeyFile /etc/apache2/certs/apache2.key

   SSLProtocol All -SSLv3

</VirtualHost>
```

Para probarlo, abrimos cualquier navegador e introducimos la dirección https://misitio.com. Automáticamente, al usar HTTPS, se conectará al puerto 443 y el navegador intentará comprobar la validez del certificado. Si fuera válido se establecería la conexión segura y veríamos como el símbolo del candado en el navegador nos indica que la web es segura y podremos navegar sin problemas.

En nuestro caso hemos hecho todos los pasos correctamente pero, como hemos comentado anteriormente, nuestro certificado es autofirmado. El navegador mostrará el correspondiente error diciendo que quién firma el certificado es desconocido y avisará al usuario, quién podrá decidir no visitar la web si no fía o bien añadir a dicho sitio como excepción en el caso de que esté completamente seguro de que, a pesar del certificado, el sitio es totalmente fiable.

Si finalmente aceptamos el certificado no seguro como excepción para el navegador podremos visitar la web utilizando una conexión segura.

Llegados a este punto nos puede interesar que todo el tráfico de la web se vea forzado a utilizar el protocolo seguro HTTPS. Incluso aunque el usuario introduzca la URL directamente y decide navegar utilizando HTTP (http://.....) nosotros podemos redirigirle hacia la opción de utilizar la opción segura.

En ese caso podemos incluso configurar el host virtual no seguro con las opciones mínimas para redirigirlo al seguro. No haría falta ni incluir la opción DocumentRoot.

```
<VirtualHost *:80>
  ServerName www.misitio.com
  Redirect / https://www.misitio.com/
</VirtualHost>
<VirtualHost *:443>
 ServerName misitio.com
 DocumentRoot . . .
 . . .
</VirtualHost>
```

Finalmente podemos optar por una redirección permanente (de esta forma así se notificará a los buscadores) modificando la orden Redirect por la siguiente:

```
 . . .
 Redirect permanent / https://www.misitio.com/
 . . .
```

7. Pruebas de rendimiento.

Webalizer es una aplicación que podemos instalar para procesar el fichero log de Apache y generar un documento HTML con las estadísticas de nuestro sitio web.

Podemos instalarlo utilizando la herramienta apt:

```
$ sudo apt-get install webalizer
```

Y a continuar utilizar el comando webalizer para procesar el fichero log que queramos (en este caso el del sitio raíz) y éste generará una carpeta en /var/www/webalizer con un sitio web donde podremos visitar diferentes estadísticas sobre el uso del servidor.

```
$ sudo webalizer /var/log/apache2/access.log
```

Puesto que ahora por defecto en Debian la carpeta raíz para Apache es /var/www/html, tendremos que hacer un enlace simbólico dentro de dicha carpeta que apunte a la que Webalizer genera para poder visualizar el informe. También podría ser interesante proteger esa carpeta de alguna manera para que no fuera visible para cualquier visitante.

```
$ cd /var/www/html
$ sudo ln -s ../webalizer webalizer
```

Ahora podemos visitar la página web generada donde podremos observar las estadísticas ya preparadas para nuestro sitio web (en este caso en http://ip_server/webalizer, donde la ip_server será la de la máquina donde tenemos desplegado el sitio, o bien el dominio si lo tenemos)

Monitorización

GoAccess es una analizador visual de logs de Apache en tiempo real. De esa manera permite monitorizar el acceso y uso al servidor por parte de los usuarios en cada momento con numerosas métricas.

```
$ sudo apt-get update
$ sudo apt-get install goaccess
```

Como Debian en su repositorio oficial tiene la versión 0.9.4 vamos a instalar la 1.2 del sitio oficial de los creadores de la herramienta, primero instalamos la herramientas y librerías necesarias:

```
$ sudo apt-get update
$ sudo apt-get build-essential
$ sudo apt-get install libncursesw5-dev
$ sudo apt-get install libgeoip-dev libtokyocabinet-dev
```

A continuación descargamos y compilamos los fuentes de la versión 1.2:

```
$ cd /usr/local/src
$ sudo wget http://tar.goaccess.io/goaccess-1.2.tar.gz
$ sudo tar -xzvf goaccess-1.2.tar.gz
$ cd goaccess-1.2/
$ sudo ./configure --enable-utf8 --enable-geoip=legacy
$ sudo make
```

```
$ sudo make install
```

Finalmente copiamos el ejecutable a /usr/bin

```
$ sudo cp  goaccess  /usr/bin
```

```
$ sudo goaccess --version
```

```
GoAccess - 1.2.
```

Podemos usarla de dos maneras, visualizando los resultados por consola o en formato HTML como una web más.

Para visualizarlos desde la consola, basta localizar el fichero log de Apache que queremos monitorizar (en este caso el fichero general, pero podríamos pasar los ficheros log de los diferentes hosts virtuales que hayamos configurado) y ejecutamos la aplicación tal y como se muestra en el siguiente ejemplo:

```
$ sudo goaccess -f /var/log/apache2/access.log -c
```

Y tras elegir el formato de fichero log que usamos, podemos visualizarlo en modo texto.

También podemos pedirle a GoAccess que prepare un documento HTML en tiempo real donde podremos ver las estadísticas en tiempo real desde el navegador.

```
$ sudo goaccess /var/log/apache2/access.log -o /var/www/html/report.html --log-
format=COMBINED --real-time-html
```

```
WebSocket server ready to accept new client connections
```

Ahora al que acceder con el navegador en la dirección http://ip_server/report.html podremos ir monitorizando el uso del servidor web puesto que se irá actualizando constantemente sin necesidad de recargar la página.