UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Integración de un proxy inverso NGINX con un panel de control Virtualmin para crear una plataforma de servicios de hospedaje Web

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Borja Molina Coronado

Tutor: Julio Pons Terol

2014/2015

Resumen

La utilización de un proxy inverso es una técnica que permite mejorar prestaciones y reducir la carga de los servidores mientras que se mantiene compatibilidad con los servidores web más populares, como por ejemplo Apache. Virtualmin GPL es un panel de control y gestor de plataformas web, open source, que permite crear y configurar de forma automática todos los recursos asociados con un hospedaje web basado en el servidor Apache. El objeto de este proyecto es instalar este software en un servidor e integrarlo con NGINX en su modalidad de proxy inverso, de manera que se pueda gestionar de forma transparente desde el panel de control Virtualmin. El desarrollo se ha realizado en una máquina virtual para tener mayor flexibilidad. Inicialmente se han analizado las herramientas y diseñado los scripts de automatización que realizan todas las configuraciones. Todo ello, bajo el S.O. Linux con distribución CentOS.

Palabras clave: NGINX, Apache, servidor virtual, Python, Virtualmin, WebMin, integración, proxy inverso, Linux, CENTOS.

Abstract

The use of a reverse proxy is a technique that allows you to improve the performance and decrease the servers load while the compatibility with the most popular web servers like Apache is kept. Virtualmin GPL is an open source control panel and hosting manager that allows you to create and configure automatically all resources associated with an Apache based hosting web. The purpose of this project is to install this software on a server and integrate it with NGINX in reverse proxy mode, so it can be managed in a transparent way from Virtualmin. The development has been made on a virtual machine for greater flexibility. Initially the tools have been analyzed and then the automatization scripts that make all the configurations have been designed. All of this was done using CentOS Linux Operating System distribution.

Keywords : NGINX, Apache, virtual server, Python, Virtualmin, WebMin, integration, reverse proxy, Linux, CENTOS.

Tabla de contenidos

1. Introducción

Hoy en día, el campo de las tecnologías de la información (IT por sus siglas en inglés) pone de manifiesto su importancia en el ámbito de los negocios, facilitando aspectos tales como: ganar visibilidad en el mercado, mejorar la eficiencia y proporcionar soluciones tecnológicas en base a las necesidades especificas de cada uno de ellos.

Para cubrir algunas de estas necesidades han surgido en los últimos años distintas alternativas a las que ya se encontraban implantadas. Este auge de soluciones tecnológicas ha provocado que estas herramientas ya arraigadas tengan que trabajar en armonía con las nuevas aplicaciones que se han venido desarrollando a lo largo del tiempo. Además, se han ido implantando de forma paralela paneles software como VirtualMin —a un nivel más alto— facilitando la configuración y automatizando muchas de las tareas necesarias para desplegar dichas soluciones en los centros de datos. Este es uno de los motivos por los que la integración de aplicaciones software que utilizan tecnologías distintas se ha convertido en una de las responsabilidades mas destacadas dentro del campo de la informática e Internet.

Esa torre de babel de tecnologías y protocolos de transmisión de datos, que llamamos Internet, donde la lengua que mayoritariamente se habla desde 1990 es el protocolo de transferencia de hipertexto (HTTP), desarrollado por el consorcio *World Wide Web (*W3C de sus siglas en inglés) y estandarizado por la *Internet Engineering Task Force* (IETF), es el que se utiliza en la web siguiendo un modelo de arquitectura cliente-servidor, en el que el programa servidor es quien recibe las peticiones de los usuarios, que serán los que den inicio a la conexión. Puesto que cada cliente puede abrir una o varias conexiones sobre el mismo servidor para recibir la información solicitada, uno de los grandes inconvenientes de esta arquitectura es la centralización de la información.

La utilización de un proxy HTTP inverso es una técnica transparente para los clientes que realicen peticiones al servicio, proporcionando mejoras en el rendimiento y reducción en la carga de los servidores a la vez que se mantiene compatibilidad con las aplicaciones de servidor web más extendidas, como por ejemplo Apache. Esta es una de las principales razones por las que este tipo de despliegues comienza a tomar un papel importante dentro de las arquitecturas de red de las empresas de servicios de Internet, que comienzan a ver negocio en las posibilidades que ofrece, comercializando soluciones seguras, distribuidas y eficientes; aunque, pueden llegar a ser complejas de gestionar.

1.1. Objeto

Debido a la expansión dentro del campo de las tecnologías de la información que esta experimentando el servidor NGINX, aumenta a su vez la necesidad de proporcionar herramientas que faciliten el manejo y la gestión de este producto. Gracias a esta fama que el servidor ha alcanzado —sobretodo en su modalidad como proxy inverso—, y debido a la complejidad de integración con Apache y la ausencia de soluciones desarrolladas —ya que no existen módulos que cubran estas tareas para los paneles de gestión opensource como son Webmin o VirtualMin—,

surge la necesidad de implementar una serie de herramientas que cumplan con este cometido, evitando al administrador la laboriosidad de realizar a mano todas y cada una de la operaciones requeridas.

La finalidad de este proyecto es integrar NGINX de forma que trabaje como servidor proxy inverso conjuntamente con Apache, y más concretamente con los distintos servidores virtuales gestionados desde un panel de control VirtualMin. Para ello, se desarrollará un módulo con una interfaz intuitiva y que será capaz de realizar todas las tareas necesarias para desplegar dicha configuración en el servidor sin interferir con los distintos servicios que pueda haber en funcionamiento en el sistema, proporcionando transparencia y agilidad en su gestión.

El desarrollo se realizará usando una maquina virtual, esto es, una aplicación en un sistema informático que emula un ordenador físico, proporcionando agilidad y aislamiento al desarrollo. Este emulador dispone de una instalación del sistema operativo de código abierto Linux CentOS en su versión 6.6. Este sistema operativo, además de las herramientas y tecnologías utilizadas, serán tratados con mas detalle en el punto 2, donde se explica el contexto tecnológico del presente trabajo para facilitar al lector su comprensión.

Además se detalla el procedimiento llevado a cabo para la creación del módulo. Desde los *scripts* de automatización de las tareas habituales de instalación del servidor NGINX y las herramientas necesarias para que funcione junto con Apache, hasta la instalación del módulo creado para el panel de control, pasando por las configuraciones necesarias para que ambos servicios trabajen en sinfonía.

2. Contexto tecnológico

2.1. VirtualMin

Basado en el panel web de código abierto para administración de sistemas Webmin, Virtualmin ha conseguido posicionarse como una alternativa libre y fiable a paneles de control de pago como cPanel o Plesk para la gestión de múltiples servidores web Apache. Existen dos modalidades de la aplicación: una gratuita y otra comercial, siendo la misma herramienta pero con soporte ofrecido por la empresa VirtualMin LLC.

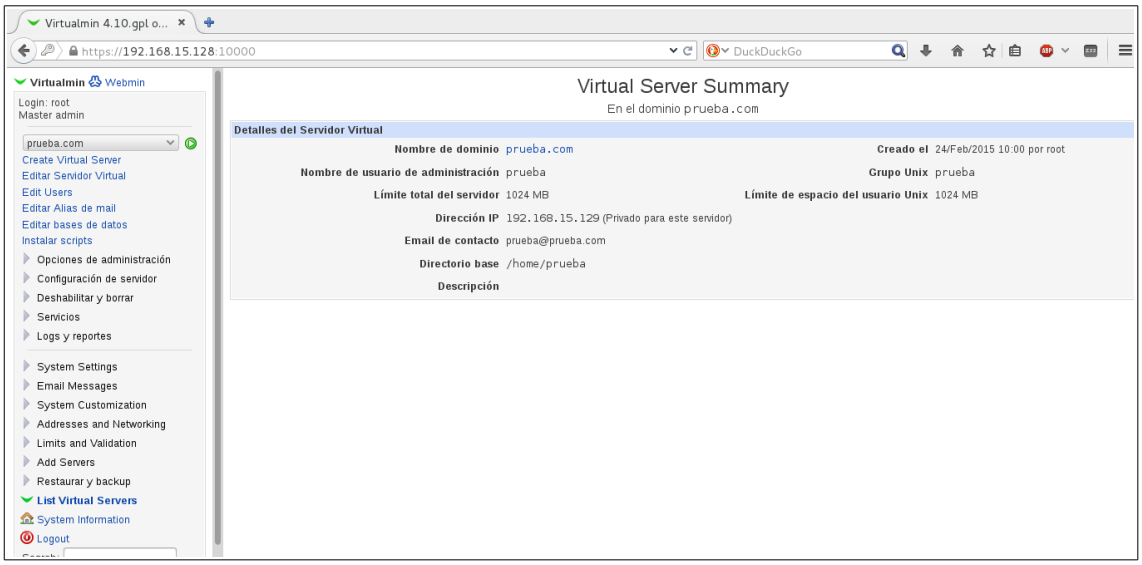

Figura 1 – WebMin GUI para configuración del Vserver prueba.com

Concebido para facilitar la gestión de tareas de *hosting*, centralizando en una pagina web con una interfaz gráfica de usuario (GUI por sus siglas en Inglés) amigable la administración de todos los sitios web alojados por un servidor —también llamados *Virtual Servers*—. Un ejemplo de su interfaz puede observarse en la *figura 1*, en la que se muestra la pantalla de configuración de un servidor virtual de Apache.

Como se puede ver en la imagen, proporciona métodos sencillos para la gestión de servidores virtuales dentro de una misma máquina mediante la visualización de menús y formularios en los que se pueden indicar los distintos ajustes, encargándose por sí mismo de forma transparente para el administrador, de todas las tareas necesarias para su creación y configuración.

2.2. NGINX

Originalmente desarrollado por Igor Sysoev, un ingeniero informático Kazajo, para soportar el alto trafico del segundo sitio web mas visitado de Rusia, *Rambler.ru*, quien decidió liberar su código bajo licencia BSD simplificada[1].

Tal y como se explica en la propia documentación del proyecto, NGINX es un servidor web y proxy inverso liviano de alto rendimiento, es decir, potente y de bajo consumo de recursos, y que además también puede ser empleado como proxy de correo electrónico.

Su gran eficiencia, demostrada durante los mas de dos años y medio que este servidor ha estado desplegado en producción para el sitio web ruso, ha propiciado que otras empresas como VK, la red social por excelencia en Rusia; o Wordpress.com, una de las plataformas de blog mas utilizadas en Internet; decidieran migrar sus sistemas a NGINX, convirtiéndolo según un estudio de Netcraft en febrero de 2015 [16] en el segundo servidor web con mayor cuota de mercado para los sitios web más ocupados del mundo justo por detrás de Apache. Además, según este mismo estudio, es el tercer servidor del mercado con mas sitios web activos —por debajo de Apache que encabeza la lista—, mientras que en segundo lugar se encuentra *Microsoft Internet Information Services*. Véase *figura 2*.

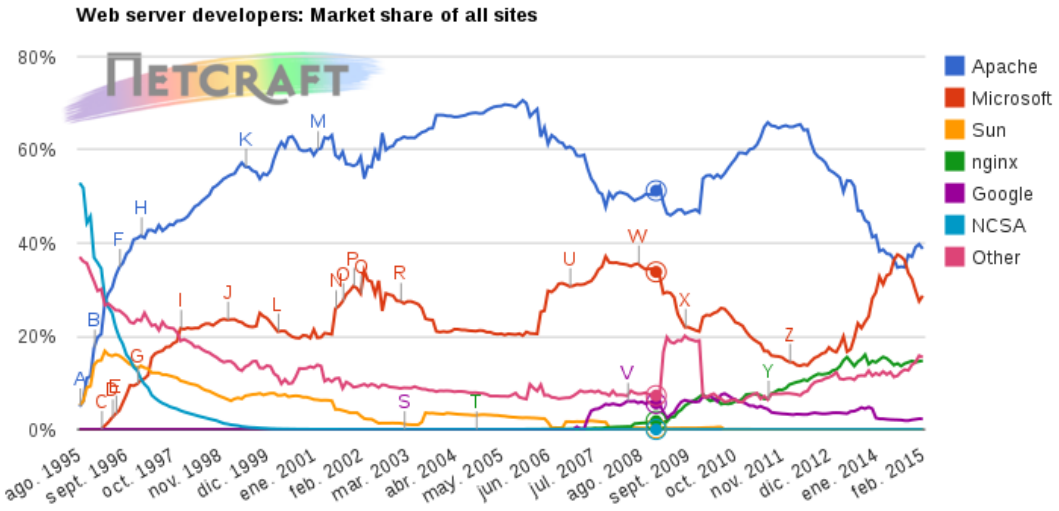

Figura 2 – Gráfica Netcraft febrero 2015

Otra de las ventajas de NGINX es el modo en que puede ser configurado. Los ficheros de configuración son simples archivos de texto divididos en bloques de directivas, con campos de valores que pueden ser cambiados por el administrador, y que serán los que definan el comportamiento del servidor. Los bloques de directivas vienen determinados por los módulos del programa que tenemos activos, es decir, si activamos un nuevo módulo en NGINX dispondremos de un juego específico de directivas para él, que podremos indicar en la configuración. Además, es posible anidar bloques de directivas para indicar una configuración especifica en caso de ser

1 Información acerca de la licencia BSD simplificada puede encontrarse en [27]

necesario. Esto nos da una idea del alto grado de flexibilidad que nos ofrece el servidor con respecto a sus capacidades de configuración.

Cada directiva será un campo al que se asigna un valor y que se aplicará dentro del bloque en que se encuentra especificado en el fichero, por lo que es posible indicar distintos valores para un mismo parámetro que se encuentre en bloques de directivas distintos. Sin embargo, todas y cada una de las directivas tienen asignadas un valor por defecto, que será aplicado por NGINX en el arranque del programa si no se ha indicado un valor para ese parámetro en la configuración.

```
server {
    listen          443 ssl;
    server_name     www.example.com;
    ssl_certificate www.example.com.crt;
    ...
}
```

Figura 3 – Ejemplo sintaxis de configuración NGINX

Como se muestra en la *figura 3*, la sintaxis de los bloques viene definida por llave de apertura `{`, al inicio del bloque y precedido del nombre del módulo. Las directivas, que son los campos clave-valor delimitados al final por `;`. Y llave de cierre `}` indicando el final del bloque.

2.3. Apache

El servidor HTTP Apache viene desarrollándose desde 1995 y pese a que comenzó siendo una ampliación/mejora del servidor NCSA Httpd, durante meses fue re-diseñado y programado por completo por *webmasters* e ingenieros del *National Center for Supercomputing Applications* (NCSA)[30] —creadores de la fundación Apache que lo mantiene— siendo publicada esta primera versión en diciembre de ese mismo año.

Tan solo unos meses mas tarde, en abril de 1996, Apache ya era el servidor mas utilizado de Internet [16]. Desde entonces, y tan solo habiendo perdido esa posición durante un corto periodo de tiempo, mantiene su posición destacada frente a todas las alternativas que han ido surgiendo tal y como puede verse en la *figura 2,* extraída del estudio de Netcraft de Febrero de 2015. Entre sus características podemos destacar:

- Multiplataforma. Disponibilidad de versiones para todos los sistemas operativos comunes, Windows, Linux, BSD, MacOS...
- Estructura modular y API de desarrollo de módulos para hacerlo extensible, lo que permite aumentar sus funcionalidades mediante la adicción de complementos. Para ello proporciona una librería de funciones (kit de desarrollo) que simplifica el proceso de creación de extensiones.
- Soporte. Debido a su gran divulgación a lo largo de los años, pueden encontrarse gran cantidad de documentación en linea acerca de cualquiera de sus componentes, además de una gran cantidad de usuarios con conocimientos técnicos, foros especializados, libros, etc.

- Código abierto. Se distribuye bajo licencia Apache 2.0, dando la libertad al usuario para modificar, distribuir y vender las modificaciones que se hagan (con pequeñas restricciones como informar que el nuevo desarrollo es un proyecto basado en el código de Apache y adjuntando el aviso de licencia original).
- Utiliza la última versión estandarizada del protocolo HTTP por lo que asegura la compatibilidad y el funcionamiento con todas las aplicaciones que hagan uso del mismo.
- Configuración sencilla gracias a la sintaxis y el formato de los ficheros de texto.

Cabe mencionar que Apache es utilizado para servir tanto contenido estático como contenido dinámico generado por CGIs (*Common Gateway Interfaces* por sus siglas en Inglés), estos son, documentos web formados por código que se ejecuta en el servidor en el momento de ser invocados y cuyo resultado se envía como respuesta al cliente. Pese a esto, NGINX es mucho mas eficiente sirviendo con un menor consumo de memoria documentos estáticos almacenados en disco.

Por otro lado, en contraste con Apache, el servidor NGINX no dispone de soporte para ficheros de tipo `.htaccess` —un fichero que contiene distintas directivas de configuración para cada directorio en particular dentro del servidor web, evitando la edición del fichero de políticas principal de la aplicación[35]—. Sin embargo, esto no significa que no exista la posibilidad de realizar este procedimiento en NGINX, que puede ser llevado a cabo añadiendo estos parámetros en el fichero de configuración principal del servicio.

En cuanto a la versión de Apache, cabe destacar que pese a estar disponible una versión mas reciente del servidor —nos referimos a la versión 2.4—, la utilizada en éste trabajo es anterior. Esto es debido a que Virtualmin no es totalmente compatible con versiones de Apache posteriores a la 2.2, obligandonos a utilizar ésta. Además, es el panel el encargado de instalar los servicios, siendo la 2.2 la versión predeterminada.

2.4. CentOS

La plataforma CentOS (*Community Enterprise Operating System* por sus siglas en inglés), es una distribución Linux originada a partir del código fuente del sistema operativo de ámbito empresarial —con el que mantiene compatibilidad funcional— Red Hat Enterprise Linux, propiedad de la corporación Red Hat Inc y para el que es necesaria una suscripción de pago.

Nacida como una alternativa para el ámbito profesional libre y gratuita, se encuentra sustentada por una comunidad de usuarios fuerte formada por administradores de sistemas, desarrolladores, administradores de red y entusiastas a lo largo de todo el mundo. Entre sus ventajas destaca su gran periodo de soporte debido a que cada una de sus versiones es mantenida durante 10 años.

La version 7.1 del sistema operativo, cuyo periodo de soporte esta fechado hasta el 30 de Junio de 2024 y que incluye varios cambios y mejoras respecto a las anteriores como soporte de nuevo hardware, el la última lanzada. Sin embargo, no ha sido posible ser usada en este proyecto debido a que el panel de control VirtualMin todavía no ha sido migrado del todo. Es por éste motivo que

la version del sistema operativo empleado en este proyecto inferior. En concreto, la versión 6.6 —utilizada en la máquina servidora— tiene una fecha límite de soporte hasta el 30 de Noviembre de 2020 [1], proporcionando actualizaciones durante ese periodo de tiempo y preservando el sistema seguro y fiable.

2.5. Python

Tal y como reza en su propia página web: *Python is a programming language that lets you work quickly and integrate systems more effectively* [28], Python es un lenguaje de programación de código abierto que se viene desarrollando desde 1994. Inicialmente ideado por Guido van Rosum, pronto fue ganando adeptos que fueron aportando mejoras y parches a la versión inicial hasta convertirse en un lenguaje característico por su sencillez y flexibilidad.

Es un lenguaje multiparadigma, es decir, soporta varios estilos de programación como pueden ser imperativo, funcional, estructurado, orientado a objetos... Además es interpretado, lo que agiliza los desarrollos —en realidad, la compilación (transformación a código maquina), se hace en tiempo de ejecución según se van ejecutando cada una de las lineas que componen el programa— facilitando la depuración del código.

Otras ventajas de Python son: el tipado dinámico, lo que implica que no es necesario indicar el tipo de datos que contendrán las variables del programa ya que el propio interprete del lenguaje detecta de forma automática la clase de datos que contiene cada una de ellas, además de la capacidad de ser ejecutado en cualquier maquina que disponga de un interprete del lenguaje independientemente del sistema operativo sobre el que se ejecute (multiplataforma).

La versión más reciente del lenguaje, lanzada en 2014, es la 3.4.3. Pese a esto, en este trabajo para todos los desarrollos realizados con este lenguaje se ha empleado la versión 2.7, ya que todas las versiones 3.x son incompatibles con las anteriores, por lo que es recomendable usar la ultima versión 2.x (2.7) en caso de ser necesario desarrollar para entornos que no controlamos o en caso de utilizar librerías que aún no están soportadas en las versiones posteriores como son las 3.x [13]. Esta es la causa principal por la que la mayoría de distribuciones Linux, como es el caso de CentOS (utilizada en el desarrollo), únicamente incorporan la versión 2.x por compatibilidad.

Pese a que los módulos y librerías de desarrollo de Virtualmin están escritos en Perl, han sido semi-portadas también a Python lo que lo convierte en un lenguaje recomendable para programar nuevas funcionalidades para el panel. Sin embargo, hemos prescindido del uso de éstas librerías puesto que nos era más sencillo programar directamente código HTML que aprendernos el amplio rango de funciones que realicen éstas tareas.

3. Descripción del problema

Tal y como se ha explicado en el apartado 1.1 de este mismo documento, este proyecto trata de cubrir una necesidad surgida debido a la expansión que lleva experimentando NGINX a lo largo de los últimos años, desarrollando una herramienta que facilite al administrador las operaciones de integración del servidor con las configuraciones de servidores virtuales Apache que pueden ser gestionadas en VirtualMin.

De esta forma, resulta de vital importancia que el desarrollo sea capaz de interoperar con estas herramientas y sus configuraciones, resultando en una herramienta flexible, es decir, en la que el administrador pueda ser capaz de configurar los parámetros básicos dependientes del sistema en que se ejecute (rutas de instalación, rutas hacia ficheros de configuración...) y de fácil manejo.

3.1. Esquema inicial del servidor.

Antes de comenzar con el desarrollo, vamos a describir la configuración disponible en el servidor de la maquina virtual para entender las modificaciones que se detallan mas adelante con el objetivo de lograr la configuración deseada. Así, el servidor, disponible en la dirección IP `192.168.15.128`, se compone de:

- Un servicio web Apache instalado y configurado en el puerto 80.
- Panel de control Webmin instalado y accesible desde el puerto 10000.
- Panel/Módulo VirtualMin instalado y accesible desde Webmin.

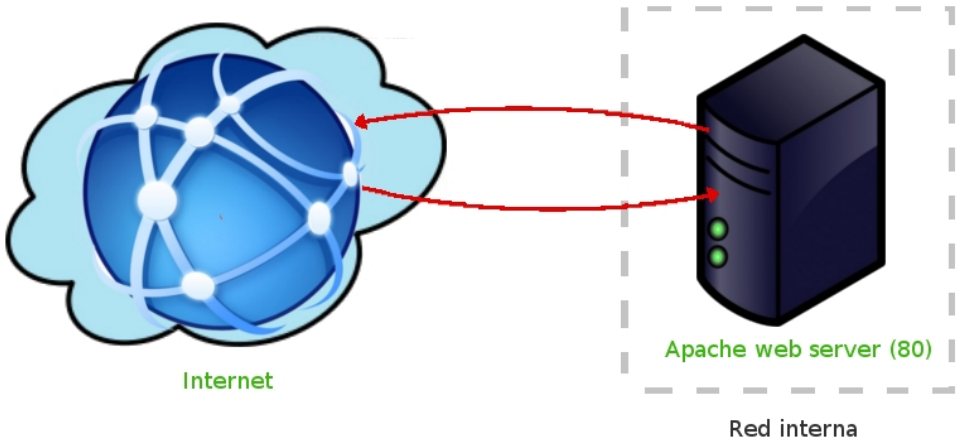

Figura 4. Esquema inicial maquina virtual

La *figura 4* muestra el diseño a partir del cual se parte en este desarrollo. Esto es, una configuración típica —y sencilla— para cualquier servidor web que disponga de un panel de

administración remota. En esta implementación, es el servidor quien se encarga directamente de servir todas las peticiones HTTP que le llegan desde internet con destino a su IP.

3.2. Esquema con proxy inverso.

Una vez conocemos la configuración típica de un servidor web como la que disponemos para el desarrollo, es necesario explicar de qué se compone su variante con proxy inverso. Pero para ello, se hace necesario explicar cómo funcionan este tipo de servidores.

Un proxy es un elemento intermedio entre los clientes y el servidor que actúa como servidor para el cliente —la dirección destino en la cabecera IP será la del proxy, y la de origen, la del cliente original—; esto es, como destino de todas las peticiones que el cliente envía hacia el servidor web, y como cliente para el servidor web, es decir, como origen de todas las peticiones que éste recibe —la dirección origen en la cabecera IP es la del proxy y la de destino la del servidor web, en nuestro caso Apache—, siendo este proceso totalmente transparente para los usuarios [31][44].

Además, este tipo de servicios poseen la capacidad de almacenar objetos en memoria, lo que evita realizar peticiones para esos recursos al servidor. Este tipo de diseños hacen las funciones de cache web, proporcionando mejoras en el rendimiento aligerando la carga de un servidor web debido al gran número de peticiones que puede recibir.

A diferencia de una configuración proxy normal, los *surrogates* o proxy inversos se sitúan en las redes internas de los servidores web y no en las redes de los clientes. Esta diferencia radica en el propósito del proxy, ya que en la red del cliente se utilizan principalmente para filtrar y realizar funciones de monitorización de tráfico, y en cambio, un proxy inverso es una forma más eficaz de aligerar la carga de los servidores web acelerando los tiempos de respuesta.

En comparación con el esquema inicial mostrado en la *figura 4*, este diseño le otorga un grado de complejidad tal y como puede verse en la *figura 5*.

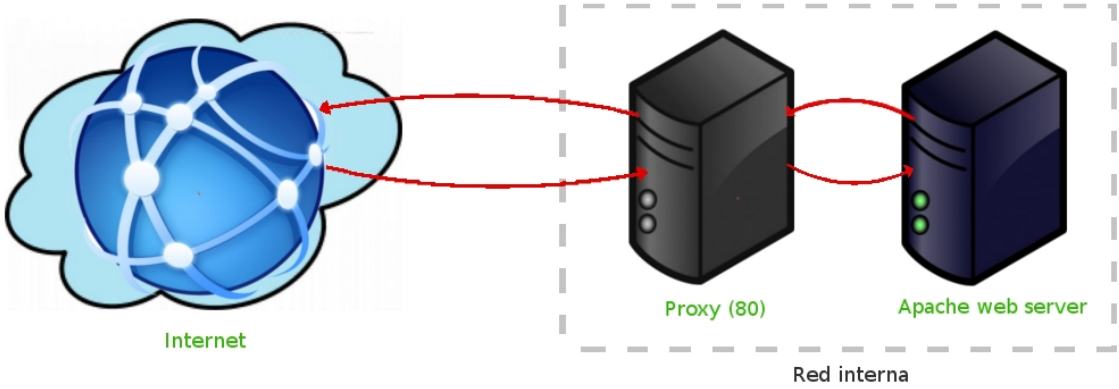

Figura 5. Diseño red con Proxy

De esta forma, el *gateway* será el que recibe toda la carga que va destinada al servidor Apache (el servidor web) y responderá a todas la peticiones —capacidad por la cual también son llamados aceleradores web o *surrogates*—. En caso de que el proxy no disponga de ninguna copia en

memoria cache del objeto solicitado o la copia que tiene no esté actualizada, envía una solicitud al servidor original para obtener el recurso. Una vez el *surrogate* ha recibido la respuesta de Apache, éste almacena una copia en memoria para que disponga de ella en la siguiente ocasión, y finalmente, lo reenvía al cliente que lo solicitó originalmente. Además, estas copias no son permanentes ya que tienen un periodo de validez tras el cuál son borradas, evitando así la mala gestión por desperdicio de memoria.

Así, el mecanismo de proxy inverso principalmente resuelve un inconveniente, el alto tiempo medio de respuesta de Apache frente a NGINX debido al gran número de componentes que Apache soporta y que carga en memoria, lo que ve afectado su rendimiento, consumiendo mas CPU y memoria RAM por cada petición que recibe.

3.3. Esquema objetivo.

Otra variante (véase *figura 6*), que surge como resultado de una mezcla entre los dos esquemas anteriores y en la que podemos disponer de ambos servicios (*surrogate* y servidor web) instalados en el mismo equipo, es la que se desarrolla a lo largo de este trabajo. Este tipo de diseño elimina la necesidad de habilitar dos máquinas distintas para albergar cada uno de los servicios, simplificando el esquema —aunque no la configuración—.

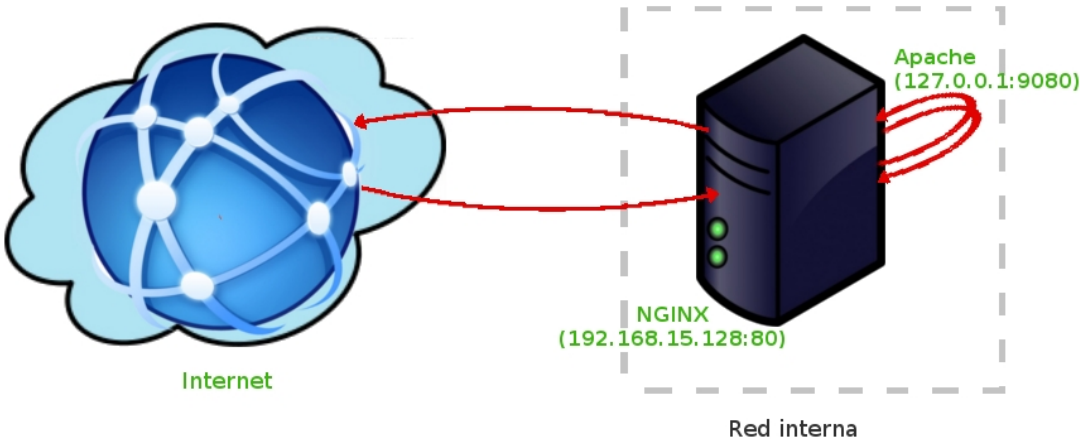

Figura 6. Esquema con servidor proxy y web en la misma máquina

En este diseño el *surrogate*, NGINX en nuestro caso, se encuentra instalado en el mismo equipo que el servidor web Apache, con la característica fundamental de que el servicio que ha de permanecer a la espera de peticiones HTTP en el puerto 80 es el proxy —en realidad esto no es obligatorio ya que se pueden utilizar técnicas como el *Port Forwarding* (redirección de puertos) siempre y cuando se redireccione desde el puerto 80 (por defecto HTTP) hacia el puerto en que se encuentra el proxy—, y el servidor web (Apache) se configura en cualquier otro puerto alternativo en el cuál recibe las peticiones procedentes del *surrogate*.

Así, a modo de resumen, la configuración del servidor queda de la siguiente forma:

- NGINX configurado para atender peticiones HTTP en el puerto 80 de la IP `192.168.15.128`
- Apache configurado a la espera de solicitudes HTTP en el puerto 9080 de la IP en la que se encuentren configurados los servidores virtuales, que son mencionados más adelante (por defecto `192.168.15.128`).
- Webmin con el módulo VirtualMin instalado y preparado en el puerto 10000 de la IP `192.168.15.128`

El objetivo de este esquema es que NGINX sea el que responda todas las peticiones sirviendo el contenido estático, ya sean imágenes o documentos HTML (HiperText Markup Language)[2], mientras que todo el contenido que sea generado de forma dependiente de los datos de la petición por medio de CGIs (dinámico), serán solicitadas al servidor Apache desde el proxy.

Así, del párrafo anterior se puede deducir que NGINX funcionará como un servidor web normal —al margen de Apache— para todo el contenido estático, esto es posible ya que el servicio se encuentra en el mismo equipo que los recursos que conforman las webs, permitiendo al *surrogate* acceder al disco duro para su lectura de una forma mas eficiente, debido al uso de funciones nativas del sistema operativo, y posterior envío hacia el cliente. Además, NGINX será el encargado de cifrar y comprimir todas las comunicaciones pues, en estas tareas, es tambien más eficaz que Apache.

Por otro lado, Apache —que tambien se encuentra instalado en el mismo servidor—, será el encargado de recibir desde NGINX las peticiones de los clientes para todo el contenido dinámico, es decir, aquel contenido que se genera en base a alguno de los valores incluidos en la petición, ya sea en las cabeceras o en el cuerpo de la misma. De esta forma, Apache sólo se ocupará de una parte mínima de todo el contenido que compone las webs alojadas en el servidor, y NGINX almacenará en su memoria cache aquellos recursos previamente generados por Apache, que se encargará de recuperar y transmitir a los clientes en caso de que éstos vuelvan a ser solicitados, actuando así, como un proxy o acelerador web.

2 Lenguaje de marcas para crear contenido web. Mas información en [42]

4. Implementación

En primer lugar, se ha estudiado el funcionamiento interno de VirtualMin, así como sus ficheros de configuración y ejecutables, con el objetivo de conocer qué ficheros son necesarios para su lectura o modificación por parte de la herramienta desarrollada. Del mismo modo en que se ha estudiado el funcionamiento del panel, también se han analizado y comprendido los ficheros de configuración de Apache, que el propio VirtualMin modifica en función de las acciones que el administrador realice desde la interfaz del panel.

Partiendo del análisis previo, se ha procedido a desarrollar los scripts de automatización en Python que realizan la integración, y posteriormente, se ha hecho un simple análisis estadístico, mediante el uso de una sencilla aplicación desarrollada en Java, para deducir con qué valores configurar NGINX de la forma más eficiente posible.

Por último, haciendo uso de todo lo desarrollado de forma previa, se ha creado un módulo para el panel de control Webmin que posibilite de forma transparente, desde la interfaz, la integración de NGINX en modo proxy inverso con Apache para los servidores virtuales o webs gestionados desde VirtualMin.

4.1. Análisis de VirtualMin

El primer paso en el desarrollo ha sido investigar el funcionamiento interno de VirtualMin, los ficheros que emplea, en los cuáles se almacenan los datos de configuraciones de los servidores virtuales que pueden ser gestionados desde el panel y los ejecutables que se lanzan cuando el administrador realiza una acción en el mismo. De esta forma y conociendo el funcionamiento de las herramientas en las que debíamos integrar nuestro desarrollo, nuestra aplicación es capaz interaccionar con VirtualMin modificando el contenido de los ficheros necesarios sin interrumpir el correcto funcionamiento del panel de control.

El directorio de instalación de VirtualMin en CentOS es `/usr/libexec/webmin/virtual-server/`. Esto se debe a que, en realidad, VirtualMin no es mas que un módulo de Webmin llamado `virtual-server`, que aumenta las capacidades del panel, lo que hace necesario que éste último se encuentre instalado en el sistema para que VirtualMin funcione.

En el interior del directorio de instalación de VirtualMin encontramos los archivos ejecutables que el panel lanza cuando el administrador realiza alguna acción desde la interfaz web, esto son: los CGI, los archivos con las configuraciones de los CGI, y los ficheros necesarios para que Webmin interprete correctamente el módulo.

De la misma forma que el directorio con los ejecutables (CGI) se encuentra bajo el directorio de Webmin, dentro de `/etc/webmin/` se localilzan los directorios con ficheros de configuración de los módulos instalados en dicho panel y, por lo tanto, también se encuentra el directorio correspondiente para VirtualMin, cuya ruta es `/etc/webmin/virtual-server/`.

Dentro de este directorio, el archivo mas importante es el fichero `config`, cuyo contenido es un atributo de configuración con su valor asignado en cada linea, *véase la figura 7*. Este fichero es cargado en una variable de tipo *hash*[3] (diccionario) cuando cualquier CGI del módulo realiza una llamada a la función `init_config()`, y además, es utilizado por el panel de control para permitir que los ajustes de configuración sean editables por el administrador, como son las rutas hacia otros ficheros de configuración u otras opciones para controlar el comportamiento del módulo.

Es en este fichero en el que se encuentra la configuración de la plantilla por defecto que VirtualMin asigna en la creación de los distintos servidores virtuales para Apache, por lo que nuestros scripts deben modificar estos valores conforme a los requerimientos del administrador.

```
backup_feature_logrotate=1
mysql_chgrp=1
logrotate_config=rotate 5 weekly compress postrotate [ ! -f
/var/run/nginx.pid ] || kill -USR1 `cat /var/run/nginx.pid`
/etc/rc.d/init.d/httpd graceful ; sleep 5 endscript sharedscripts
show_sysinfo=2
avail_mailboxes=1
defforceunder=0
alias_types=1,2,5,6,7,8,9,10,11,12,13
spam_lock=0
key_size=2048
webmin_ssl=0
```

Figura 7. Extracto del fichero config de VirtualMin modificado

Ya analizados cuáles son los lugares en que se encuentran los ficheros de instalación y configuración del panel de control, es importante realizar una breve descripción de cuál es el comportamiento general del panel para entender mejor su funcionamiento.

Una vez el equipo con el panel es puesto en marcha, tenemos una web de gestión disponible desde la que podremos utilizarlo, indicando en el navegador la dirección IP del servidor y el puerto en que se encuentre funcionando el servicio, que es el encargado de mostrar el resultado de los CGI que conforman el panel —pues estos, al fin y al cabo son ejecutables—. Esto es, el puerto 10.000 por defecto, quedando la url en nuestro servidor de pruebas como sigue https://192.168.15.128:10000. *Véase figura 8*.

Figura 8. Acceso de Webmin en el servidor

3 Estructura de datos eficiente —también llamada array asociativo— en la que puede accederse a su contenido directamente a través de una clave "única" que tiene asociado un valor.

Es preciso señalar que cuando iniciamos sesión en el panel se realiza el mismo proceso que cuando hacemos login en el equipo, es decir, se consultan los usuarios registrados en el servidor (contenidos por el fichero `/etc/shadow`) para permitirnos controlar el equipo.

A continuación de que hemos iniciado la sesión en el equipo, el servidor web propio de Webmin situado en el puerto 10.000, será el encargado de leer del disco duro el CGI de inicio del panel, ejecutarlo y devolvernos el resultado que el navegador es el encargado de representar en pantalla. Este proceso se repite para todos los programas ejecutables que conforman el panel y que sean demandados por el administrador desde el navegador mediante los clics que éste realice en la interfaz.

Cada uno de estos ejecutables esta compuesto por órdenes escritas en un lenguaje de programación concreto, que el servidor interpreta, recogiendo el resultado y retornándolo al origen de la petición. Por defecto, todos los CGI de VirtualMin están implementados en Perl, pero se otorga al desarrollador la libertad de elegir cualquier otro lenguaje, siempre y cuando el equipo disponga de un interprete instalado que sea capaz de entenderlo.

4.2. Análisis de Apache

Una vez tenemos una idea mas clara del funcionamiento de VirtualMin, es necesario aclarar cómo funciona y se configura el servidor web Apache, sobretodo lo relativo a los servidores virtuales, ya que como hemos dicho antes, estos son con los que el panel trabaja.

Partiendo de la configuración inicial de nuestro servidor de pruebas, Apache se encuentra instalado en `/etc/httpd/` en cuyo interior se hayan dos directorios importantes `conf` y `conf.d`. El primero de ellos alberga los ficheros de configuración del servicio web, mientras que el segundo contiene los ficheros de configuración de los módulos que Apache tiene instalado.

Dentro de `/etc/httpd/conf/` el fichero mas importante, llamado `httpd.conf`, es el de configuración del servicio. Es en este fichero en el que podemos indicar por ejemplo el puerto o puertos en el que queremos que Apache se encuentre esperando peticiones web —Apache tiene la capacidad de poder escuchar simultaneamente peticiones en distintos puertos y direcciones IP—, a que rutas irá a buscar los documentos solicitados por las peticiones y, lo mas importante en este trabajo, los servidores virtuales.

El formato de todos los ficheros de configuración de Apache consta de una directiva por cada linea, y por cada una de las directivas, sus argumentos han de estar separados por un espacio en blanco. En caso de que el valor de una directiva contenga espacios, este debe ir entre comillas (“”). Todas las directivas se aplican al servidor. Sin embargo, el alcance de las directivas puede limitarse agrupándolas en bloques, en cuyo caso, éstas solo se aplican a una parte del servidor[37].

Cada sección va delimitada al principio por el nombre de la sección encerrado entre `<>`, y al final, por `</>`. Podemos encontrar bloques de los siguientes tipos: `<Directory>`, `<DirectoryMatch>`, `<Files>`, `<FilesMatch>`, `<Location>`, `<LocationMatch>`, y `<VirtualHost>`[37]. A continuación, en la *figura 9*, se muestra un extracto correspondiente al fichero de configuración de Apache en el servidor relativo a gestión de procesos, lo que nos da un ejemplo de cómo se organiza la información en éste tipo de archivos.

```
Listen 80

NameVirtualHost 172.20.30.40

<VirtualHost 172.20.30.40>
DocumentRoot /www/example1
ServerName www.example.com
</VirtualHost>

<VirtualHost 172.20.30.40>
DocumentRoot /www/example2
ServerName www.example.org
</VirtualHost>

<VirtualHost 172.20.30.40>
DocumentRoot /www/example3
ServerName www.example3.net
</VirtualHost>

# IP-based
<VirtualHost 172.20.30.50>
DocumentRoot /www/example4
ServerName www.example4.edu
</VirtualHost>

<VirtualHost 172.20.30.60>
DocumentRoot /www/example5
ServerName www.example5.gov
</VirtualHost>
```

Figura 9. Extracto httpd.conf para configuración de VirtualHost

De todos los bloques que contiene el fichero de configuración, para este trabajo los mas importantes son los correspondientes a los servidores virtuales, es decir, las secciones `<VirtualHost>`. Un servidor virtual es la capacidad de Apache para servir diferentes sitios web ubicados en un mismo equipo, diferenciándolos por su nombre (*name-based*) —indicado por la directiva `ServerName`— o, puerto o dirección IP destino de las peticiones (*IP-based*)[36].

En caso de servidores virtuales *name-based*, las directivas tipo `NameVirtualHost` indican a Apache que debe diferenciar por el nombre los servidores configurados en la dirección IP y el puerto que acompañan al parámetro. De esta forma cuando reciba peticiones web a través de la interfaz de red que tenga dicha dirección IP, Apache comprueba que el nombre de la web (`ServerName`) coincida con el valor indicado en el campo `Host` de la cabecera HTTP. Véase la *figura 9* extraída de la documentación de Apache [33].

Por otra parte, en el directorio `/etc/httpd/conf.d/` se ubican todos los ficheros de configuración necesarios para los distintos módulos de Apache que tengamos instalados en el servidor, como por ejemplo los ficheros de configuración del módulo de soporte para PHP (`/etc/httpd/conf.d/php.conf`).

4.3. Desarrollo

La idea básica del desarrollo consta de la realización de los scripts que realizan el proceso de integración de NGINX para las configuraciones del servidor web Apache desplegadas en Virtualmin. Posteriormente, se detalla el proceso de desarrollo de un módulo para el panel

Webmin que haga uso de los ejecutables previamente desarrollados para su facil ejecución desde el panel.

Después de analizar las herramientas con las que debemos realizar la integración, el siguiente paso es detallar el contenido de los *scripts* de integración y el desarrollo del módulo para Webmin desde el que se puedan gestionar de forma fácil.

4.3.1 conf.file

Debido a la flexibilidad que se ha querido otorgar al desarrollo, los *scripts* se han ideado con la propiedad característica de que puedan ser configurados de forma fácil indicando valores a los campos de un fichero de configuración. Este fichero contiene los parámetros importantes —tales como rutas de instalación por defecto, que pueden variar en función de la distribución Linux— que el administrador puede especificar para asegurar el correcto funcionamiento de la integración en su sistema. Además, también se almacenan en este archivo los datos de los ficheros de configuración de los servicios, que se leen durante la ejecución y que son necesarios para su posterior uso. El contenido de `conf.file` es leído al inicio de cada *script* por la función `cargaConf()`, perteneciente al módulo `nginxReverselib`, y cargado en una variable de tipo diccionario llamada `config`, que será accesible durante la ejecución de los mismos.

Así, el fichero `conf.file` contiene los siguientes parámetros (véase *figura 10*):

- Número de puerto HTTP alternativo para Apache. Puerto hacia el que se redirigirán las peticiones HTTP desde el proxy.

  Sintaxis `nPuertoAp: numero`
  Valor por defecto: `9080`

- Numero de puerto SSL alternativo para Apache. Puerto en el que se configurarán los VirtualHost para atender peticiones SSL provenientes del proxy. Por defecto, la configuración de Apache atiende sólamente peticiones a través de HTTP. Aun así, es posible habilitar ésta característica si se quiere que la comunicación entre NGINX y Apache sea cifrada (HTTPS).

  Sintaxis `nPuertoApSSL: numero`
  Valor por defecto: `9443`

- Ruta hacia el fichero de configuración de Apache.

  Sintaxis `nfConfApache: ruta`
  Valor por defecto: `/etc/httpd/conf/httpd.conf`

- Dirección IP del servidor donde NGINX ha de atender peticiones para los servidores virtuales de Apache.

  Sintaxis `IP: ip`
  Valor por defecto: `sin valor`

- Ruta hacia el fichero que contiene la plantilla por defecto de VirtualMin. Éste parámetro solo es necesario que sea modificado en caso de que el panel se encuentre instalado en otro lugar.

  Sintaxis `nfPlantilla: ruta`
  Valor por defecto: `/etc/webmin/virtual-server/config`

- Directorio de instalación por defecto de NGINX. En caso de haber instalado NGINX en otro lugar ha de indicarse la ruta en este campo.

  Sintaxis `rutaNginx: ruta`
  Valor por defecto: `/etc/nginx/`

- Ruta hacia el fichero de configuración del módulo SSL de Apache.

  Sintaxis `nfConfSSLApache: ruta`
  Valor por defecto `/etc/httpd/conf.d/ssl.conf`

- Ruta hacia el fichero de configuración del módulo RPAF. En caso de ya encontrarse instalado, ruta hacia el fichero. En caso contrario, lugar donde se creará.

  Sintaxis `nfConfRPAF: ruta`
  Valor por defecto: `/etc/httpd/conf.d/mod_rpaf.conf`

- Ruta por defecto en el sistema donde se encuentra el fichero de configuración de NGINX en caso de encontrarse instalado. Ruta donde comprobará si está instalado.

  Sintaxis `nfConfNGINX: ruta`
  Valor por defecto: `/etc/nginx/nginx.conf`

```
nPuertoAp: 9080
nPuertoApSSL: 9443
IP: 192.168.15.128
nfPlantilla: /etc/webmin/virtual-server/config
rutaNginx: /etc/nginx/
nfConfNginx: /etc/nginx/nginx.conf
nfConfApache: /etc/httpd/conf/httpd.conf
nfConfSSLApache: /etc/httpd/conf.d/ssl.conf
nfConfRPAF: /etc/httpd/conf.d/mod_rpaf.conf
```

Figura 10. Contenido del fichero conf.file

4.3.2 Libreria nginxReverselib.py

En primer lugar, todos los métodos comunes que los scripts de integración utilizan durante su ejecución se han incluido en un fichero llamado `nginxReverselib.py`, que hace las funciones de librería. De esta forma, los scripts de integración incluyen este fichero como referencia —mediante la orden `import nginxReverselib`—, para mantener acceso a las funciones, las cuáles son llamadas de la siguiente manera: `modulo.funcion(parametros)`.

Así, la librería de los scripts de integración contiene las siguientes funciones:

- `cargaConf()`

  Esta función realiza la lectura del fichero `conf.file` y carga su contenido en la variable de tipo diccionario `config`.

- `GuardaConf()`

  Almacena el contenido de la variable `config` en el fichero `conf.file` de forma que los cambios realizados en la configuración por los scripts sean salvados para la siguiente ejecución.

- `imprimeConf()`

  Imprime el contenido del fichero `conf.file`

- `leeFichero(nombre)`

  Realiza la lectura del fichero que se encuentra en la ruta indicada por el valor del parámetro `nombre`, devolviendo una variable tipo lista con el contenido leído.

- `escribeFichero(fich, conf)`

  Escribe en el fichero indicado por el valor de la variable `fich` el contenido de la varable tipo lista `conf`.

- `lsDir()`

  Lista el contenido del directorio que recibe como parámetro. En caso de no recibir ningun parámetro, por defecto devuelve el listado del contenido del directorio actual (`./`).

- `log()`

  Escribe en el fichero de registro `nginxReverse.log` el contenido de la variable que recibe como parámetro. De esta forma y, haciendo uso de esta función, se van registrando todas las operaciones que realiza el código de los scripts facilitando las tareas de depuración de errores o mala configuración de los servicios.

A modo de aclaración, el contenido del fichero librería puede observarse en la *figura 11*.

```
config={};

################################################################################################
# Funcion que guarda en el fichero de conf.file la configuracion de los script contenida en el
diccionario global "config"
################################################################################################
def guardaConf ( ):
    f=open("conf.file","w")
    for conf in config: # recorre todas las entradas en config
        if conf=="":
            continue
        linea=conf+": "
        for val in config[conf]: # recorre todas las entradas en un parametro de config
            if re.match("\s+",val): # omite espacios vacios
                continue
            linea+=val+" "
        f.write(linea+"\n")
    f.close()

################################################################################################
# Funcion que lee el fichero conf.file para cargar la configuracion de los script y la guarda en el
diccionario global "config"
################################################################################################
def cargaConf ( ):
    f=open("conf.file","r")
    lineas=f.readlines()
    f.close()
    valores=[]
    for linea in lineas:
        fin=linea.find(":")
        valores=re.split("(\s)+",linea) #guarda parametros separadas por espacio(incluido)
        valores=valores[1:] # quita el primer espacio
        valores=valores[1::2] # quita el resto de espacios. omite la posicion par de la lista
        config[linea[0:fin]]=valores #asigna la lista resultado a la lista de conf

################################################################################################
# Funcion que imprime la configuracion del modulo
################################################################################################
def imprimeConf ( ):
    for conf in config:
        print conf+" --- "
        for valor in config[conf]:
            print valor

################################################################################################
# Funcion que escribe en el fichero de log el registro de todos los cambios en configuraciones que
realiza el script
################################################################################################
def log ( cadenaLog ):
    f=open("nginxReverse.log","a")
    f.write("["+time.ctime()+"] -"+cadenaLog)
    f.close()

################################################################################################
# Funcion que escribe el fichero que se pasa por parametro "fich" la configuracion o el contenido de
"config"
################################################################################################
def escribeFichero ( fich, conf ):
    log(" INFO -> Escribiendo el fichero "+fich+"\n")
    f=open(fich,"w")
    f.writelines(''.join(conf))
    f.close()

################################################################################################
# Funcion que abre un fichero y devuelve el contenido leido
################################################################################################
def leeFichero ( nombre ):
    log(" INFO -> Leyendo el fichero "+nombre+"\n")
    f=open(nombre,"r")
    contenido=f.readlines()
    f.close()
    return contenido

################################################################################################
# Funcion que devuelve una lista con los nombres de ficheros que contiene un directorio
################################################################################################
def lsDir ( directorio="./" ):
    log(" INFO -> Listando el directorio "+directorio+"\n")
    files=os.listdir(directorio)
    return files
```

Figura 11. Contenido nginxReverselib.py

4.3.3 Script ApaConf.py

Partiendo del estado inicial de la máquina en que solamente se encontraba instalado el panel de control y el servidor Apache, se ha decidido comprobar desde los scripts si los elementos necesarios para el funcionamiento del proyecto se encontraban instalados, y en caso negativo instalarlos.

Uno de los módulos necesarios para que Apache funcione correctamente junto con un proxy es RPAF. Éste complemento sirve para obtener la dirección IP que realmente es origen de las peticiones que recibe Apache desde un proxy. Esto es debido al funcionamiento de los servicios proxy, ya que quién realiza realmente las peticiones hacia el servidor es el *surrogate* —actuando como cliente para Apache— y no el cliente original, de forma que la dirección de red origen (campo *source address* de cabecera) que figurará en los datagramas IP será la del elemento que generó la petición, es decir, el proxy.

Esto impide que Apache sepa quién es realmente el origen de la petición, por lo que los proxys incluyen un campo en la cabecera del protocolo HTTP para indicar a los servidores web la dirección IP origen real. `X-Forwarded-For` es el campo que el *surrogate* agrega a la cabecera HTTP y cuyo valor contiene la dirección IP del cliente que generó originalmente la petición hacia el proxy. De esta forma, y gracias a RPAF, Apache es capaz de interpretar este nuevo campo y gestionar de forma correcta las estadísticas, que en caso contrario serían erróneas debido a que todas las peticiones que Apache recibiese tendrían el mismo origen.

El código encargado de comprobar si el módulo RPAF se encuentra instalado consulta la ruta indicada en el fichero de configuración de los scripts para el campo `nfConfRPAF` (véase la *figura 12*), de forma que si existe el fichero de configuración de RPAF, no instala el módulo, y si no existe, realiza la instalación y configura el complemento.

```
##############################################################################################
# Funcion que comprueba si el modulo RPAF esta instalado basandose en la existencia de su fichero de
configuracion
##############################################################################################
def comprobarRPAF ( ):
    print "<br>"+"Comprobando si el modulo RPAF esta instalado"
    try:
        f=open(nginxReverselib.config["nfConfRPAF"][0],"r")
        f.close()
    except:
        return False
    return True

##############################################################################################
# Funcion que configura el modulo RPAF para que Apache pueda trabajar con un proxy
##############################################################################################
def configuraRPAF ( ):
    nginxReverselib.log("Configurando RPAF en"+nginxReverselib.config["nfConfRPAF"][0]+"\n")
    f=open(nginxReverselib.config["nfConfRPAF"][0],"w")
    lineas=["LoadModule rpaf_module modules/mod_rpaf-2.0.so","# mod_rpaf Configuration","RPAFenable
On","RPAFsethostname On","RPAFproxy_ips 127.0.0.1 "+apaIP,"RPAFheader X-Forwarded-For"]
    f.writelines( "\n".join(lineas) )
    f.close( )

##############################################################################################
# Funcion que instala el modulo RPAF para que Apache pueda trabajar con un proxy
##############################################################################################
def instalarRPAF ( ):
    print "<br>"+"Instalando y configurando el modulo RPAF"
    nginxReverselib.log("yum -y install httpd-devel\n")
    os.system("yum -y install httpd-devel")
    nginxReverselib.log("Descargando mod_rpaf-0.6 en /usr/local/src/\n")
    os.system("wget -P /usr/local/src/ http://mirror.trouble-free.net/sources/mod_rpaf-0.6.tar.gz")
    nginxReverselib.log("Desempaquetando mod_rpaf-0.6 en /usr/local/src/\n")
    os.system("tar zxvf /usr/local/src/mod_rpaf-0.6.tar.gz")
    nginxReverselib.log("Compilando /usr/local/src/mod_rpaf-0.6\n")
    os.system("apxs -i -c -n /usr/local/src/mod_rpaf-0.6/mod_rpaf-2.0.so /usr/local/src/mod_rpaf-0.6/
mod_rpaf-2.0.c")
    configuraRPAF()
```

Figura 12. Extracto script ApaConf.py. Gestión RPAF

Otro componente necesario para el funcionamiento del sistema, y que obligatoriamente ha de estar instalado, es NGINX. El código que aparece en la *figura 13* sirve para comprobar si el servicio se encuentra instalado en el sistema. Para ello se consulta la ruta indicada en el fichero de configuración de los scripts para el campo `nfConfNginx`, de forma que si existe el fichero de configuración de NGINX no instala el servidor proxy, y si no existe, realiza la instalación.

```
###############################################################################################
# Comprueba si NGINX esta instalado en el sistema
###############################################################################################
def comprobarNGINX ( ):
    print "<br>"+"Comprobando si NGINX esta instalado"
    try:
        f=open(nginxReverselib.config["nfConfNginx"][0],"r")
        f.close()
    except:
        return False
    return True

###############################################################################################
# Funcion que instala NGINX en el sistema. Primero se descarga el repo de NGINX y lo pone en el
sistema y despues instala NGINX
###############################################################################################
def instalarNGINX ( ):
    print "<br>"+"Instalando y configurando NGINX"
    nginxReverselib.log("wget http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-
centos-6-0.el6.ngx.noarch.rpm\n")
    os.system("wget http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-
centos-6-0.el6.ngx.noarch.rpm")
    nginxReverselib.log("rpm -ivh nginx-release-centos-6-0.el6.ngx.noarch.rpm\n")
    os.system("rpm -ivh nginx-release-centos-6-0.el6.ngx.noarch.rpm")
    nginxReverselib.log("yum -y install nginx\n")
    os.system("yum -y install nginx")
    # chkconfig nginx on
    nginxReverselib.log("chkconfig nginx on\n")
    os.system("chkconfig nginx on")
```

Figura 13. Extracto script ApaConf.py. Gestión NGINX

Una función también importante de este ejecutable es la modificación de la configuración de Apache y los atributos de la plantilla de VirtualMin conforme a las indicaciones del administrador para trabajar junto con NGINX, asignándole a los parámetros correspondientes los valores del fichero `conf.file`.

Debido a que pueden existir servidores virtuales o webs —hablando desde el punto de vista de las tareas de hosting— creados anteriormente a la ejecución del script, éste se encarga de leer el fichero de configuración de Apache en busca de las entradas `VirtualHost`, que contienen la dirección IP y el puerto en que estará disponible cada web; `NameVirtualHost`, que contienen la configuración de IP y puerto en que Apache escuchará peticiones para servidores virtuales basados en nombre; y `Listen`, que indica el puerto en que se esperan las peticiones de los clientes para sustituirlas por las correspondientes a los valores asignados por el administrador. Los valores por los que se sustituirá la configuración previa de estas directivas son los asignados en el fichero `conf.file` para los atributos `nPuertoAp`, `nPuertoApSSL` e `IP`. Véase *figura 14*.

Además, para el caso de los servidores virtuales configurados para utilizar cifrado (SSL), el script tambien se encarga de modificar las lineas `Listen` del fichero de configuración de Apache para SSL, cuya ruta se encuentra almacenada en el parámetro `nfConfSSLApache` del fichero de configuración del módulo.

```
##############################################################################################
# Funcion que comprueba la configuracion de Apache para realizar los cambios en los puertos por
defecto para el reenvio del proxy
##############################################################################################
def ConfApache ( conf ):
    print "<br>"+"Comprobando configuracion de Apache..."
    mod=False
    isconf=False
    try :
        apaIP=nginxReverselib.config["ApacheIP"][0] #Coge IP de una conf Anterior
        c=1
    except (KeyError):
        c=0 #Coge IP de la configuracion de APACHE
        nginxReverselib.config["ApacheIP"]=[]
    for n in range(0,len(conf)):
        linea=conf[n]
        if re.match("NameVirtualHost \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\:\d{2,4}\n", linea):
            print "<br>"+linea
            ini=linea.find(" ")
            fin=linea.find(":")
            apaIP=linea[ini+1:fin]
            print "<br>IP"+apaIP
            if c==0 : ## Coge la IP de conf de apache solo una vez
                c=1
                nginxReverselib.config["ApacheIP"].append(apaIP)
            apaPuerto=linea[fin+1:len(linea)-1] ## Leemos el puerto en el que esta configurado
            print "<br>"+apaPuerto # Puerto actual
            if apaPuerto=="80": #NO Esta Configurado HTTP
                mod=True
                conf[n]="NameVirtualHost "+nginxReverselib.config["ApacheIP"][0]
+":"+nginxReverselib.config["nPuertoAp"][0]+"\n" #Configuramos
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfConfApache"][0]+": "+linea)
                isconf=True #Configurado
            if apaPuerto=="443": #NO Esta Configurado HTTP+SSL
                mod=True
                conf[n]="NameVirtualHost "+nginxReverselib.config["ApacheIP"][0]
+":"+nginxReverselib.config["nPuertoApSSL"][0]+"\n" #Configuramos
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfConfApache"][0]+": "+linea)
                isconf=True #Configurado
        elif re.match("Listen (\*\:)?\d{2,4}", linea):
            print "<br>"+linea
            ini=linea.find(":")
            if ini==-1:
                ini=linea.find(" ")
            apaPuerto=linea[ini+1:len(linea)-1] # Puerto actual
            print "<br>"+apaPuerto
            if apaPuerto=="80": #NO Esta Configurado en HTTP
                mod=True
                conf[n]="Listen "+nginxReverselib.config["nPuertoAp"][0]+"\n" #Guardamos la nueva conf
para la linea coincidente
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfConfApache"][0]+": "+linea)
                isconf=True
            if apaPuerto=="443": #NO Esta Configurado HTTP+SSL
                mod=True
                conf[n]="Listen "+nginxReverselib.config["nPuertoApSSL"][0]+"\n" #Configuramos
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfConfApache"][0]+": "+linea)
                isconf=True
        elif re.match("<VirtualHost ((\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})|(.+))\:\d{2,4}>", linea ):
#Comprobamos que no haya en 80 ni 443 Vhost
            nginxReverselib.log(" INFO -> Cambiando puerto VirtualHost "+linea)
            ini=linea.find(":")
            apaPuerto=linea[ini+1:len(linea)-2] ## Leemos el puerto en el que esta configurado el
VirtualHost
            if apaPuerto=="80": #Cambiamos los Vhost en otro puerto HTTP
                linea=linea[:ini+1]+nginxReverselib.config["nPuertoAp"][0]+">\n" #Configuracion
                conf[n]=linea
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfConfApache"][0]+": "+linea)
                mod=True
            if apaPuerto=="443": #Cambiamos los Vhost en otro puerto HTTP+SSL
                linea=linea[:ini+1]+nginxReverselib.config["nPuertoApSSL"][0]+">\n" # Conf SSL
                conf[n]=linea
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfConfApache"][0]+": "+linea)
                mod=True
    return conf,mod
```

Figura 14. Extracto script ApaConf.py. Configuración de Apache

Una vez se han cambiado las configuraciones para las webs que pudiesen estar ya en funcionamiento, y como se puede ver en la *figura 15*, se modifica el fichero en que se encuentra almacenada la plantilla de configuración de VirtualMin. Esta plantilla se aplica por defecto en caso de que el administrador no elija cualquier otra configuración durante la creación de un nuevo Virtual Host, por lo que, debemos cambiar los parámetros con los que serán creados los nuevos servidores virtuales, obligando a que la configuración deseada por el administrador sea la que se aplique por defecto.

```
###############################################################################################
# Funcion que cambia los valores para los puertos http (web_port) y https (web_sslport) ademas de la
configuracion de rotacion de logs en la plantilla por defecto de virtualmin
# (logrotate_config y logrotate_files)
###############################################################################################
def ConfPlantilla ( conf ):
    mod=False
    for n in range(0,len(conf)):
        linea=conf[n]
        if re.match("web_port=\d{2,4}\n", linea):
            ##print "<br>"+linea
            ini=linea.find("=")
            vActual=linea[ini+1:len(linea)-1] ## Guardamos el valor del puerto actual de Apache
            if vActual!=nginxReverselib.config["nPuertoAp"][0]:
                linea="web_port="+nginxReverselib.config["nPuertoAp"][0]+"\n"
                conf[n]=linea
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfPlantilla"][0]+": "+linea)
                mod=True
        elif re.match("web_sslport=\d{2,4}\n", linea):
            ##print "<br>"+linea
            ini=linea.find("=")
            vActual=linea[ini+1:len(linea)-1] ## Guardamos el valor del puerto SSL actual de Apache
            if vActual!=nginxReverselib.config["nPuertoAp"][0]:
                linea="web_sslport="+nginxReverselib.config["nPuertoApSSL"][0]+"\n"
                conf[n]=linea
                nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config
["nfPlantilla"][0]+": "+linea)
                mod=True
        elif re.match("logrotate_config=.*\n", linea):
            #nginxReverselib.log("Cambiando la linea -> "+linea)
            ##print "<br>"+linea
            linea="logrotate_config="+logRConf+"\n"
            conf[n]=linea
            nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config["nfPlantilla"][0]
+": "+linea)
            mod=True
        elif re.match("logrotate_files=.*\n", linea):
            ##print "<br>"+linea
            linea="logrotate_files="+logFiles+"\n"
            conf[n]=linea
            nginxReverselib.log("INFO -> Linea modificada en "+nginxReverselib.config["nfPlantilla"][0]
+": "+linea)
            mod=True
    return (conf,mod)

###############################################################################################
# Funcion que modifica la plantilla por defecto utilizada en la creacion de nuevos servidores
virtuales en Virtalmin. Configuracion de plantilla
# por defecto en fichero de configuracion del modulo virtual-server (/etc/webmin/virtual-server/config)
###############################################################################################
def modPlantillaVmin ( ):
    conf=nginxReverselib.leeFichero(nginxReverselib.config["nfPlantilla"][0])
    (conf,mod)=ConfPlantilla(conf)
    if mod : nginxReverselib.escribeFichero(nginxReverselib.config["nfPlantilla"][0],conf)
    print "<br>"+"Plantilla CORRECTA"
```

Figura 15. Extracto script ApaConf.py. Gestión plantilla

Específicamente, los parámetros que se modifican durante la ejecución del script para las plantillas son los relativos a los puertos por defecto —tanto HTTP como HTTPS (HTTP+SSL/TLS)— en que se mantienen a la espera de peticiones los distintos servidores virtuales, así como, las directivas de rotación de logs de VirtualMin para las webs. De esta forma, se aplica la gestión de log para los ficheros de registro de acceso y error, que crean tanto NGINX como Apache, permitiendo que cualquier gestor de estadísticas del panel muestre datos correctos de las mismas.

Cabe destacar, que para facilitar las tareas de depuración de errores en caso de haber indicado algún parámetro de forma errónea en el fichero `conf.file`, todas las operaciones de los scripts, así como, de los componentes del módulo para Webmin, se van registrando en un fichero de log llamado `nginxReverse.log` que se ubica en la raíz del directorio en que se encuentran los ejecutables.

4.3.4 Script Ap2Nginx.py

El segundo de los scripts de integración tiene como fin realizar la configuración de los distintos servidores virtuales de Apache en NGINX, de forma que sean accesibles desde el proxy, además de configurar NGINX para realizar las peticiones de elementos dinámicos a Apache, para que sean realizadas al servidor web mientras que los ficheros estáticos son leídos del disco duro.

En primer lugar, se crea el directorio del disco duro donde se almacenan los elementos que son solicitados, esto es la cache, para evitar realizar peticiones al servidor web cada vez que se requiere el mismo contenido, evitando de esta forma, la redundancia de peticiones a Apache. Dicho directorio se situa en `/var/nginx/cache/`. Además, la política de cache viene definida por las directivas de configuración de NGINX que se detallan en el siguiente punto de este capítulo. Después, se procede a la lectura del fichero de configuración de Apache, detectando las entradas `<VirtualHost>`, y para cada servidor virtual encontrado, se crea su equivalente para NGINX.

Tal y como se puede ver en la *figura 16*, se realiza la lectura del fichero de ajustes de Apache y se obtiene el bloque entero, limitado por `<VirtualHost>` y `</VitualHost>`. Estos bloques contienen la configuración para cada uno de los servidores virtuales. Se procede a leer su configuración, almacenando tanto la IP como el puerto en que se encuentra el servidor virtual.

```
##############################################################################################
# Funcion que recorre el fichero de configuracion de Apache en busca de etiquetas <VirtualHost></
VirtualHost>, que marcaran bloques de
# configuracion de cada servidor virtual, y llama a la funcion confVirtualHost pasandole el bloque
encontrado junto con la IP y el puerto.
# Despues recibe la respuesta y si el metodo anterior ha hecho algo, guarda el bloque con la nueva
configuracion en el fichero de configuracion
# de Apache.
##############################################################################################
def leeVirtualServers():
    inibloque=0
    finbloque=0
    bloque=[]
    lineas=nginxReverselib.leeFichero(nginxReverselib.config["nfConfApache"][0])
    nginxReverselib.log(" INFO -> Leyendo configuracion VirtualHost de Apache\n")
    for n in range(0,len(lineas)):
        linea=lineas[n]
        if re.match("<VirtualHost ((\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})|(.+))\:\d{2,4}>", linea ):
            nginxReverselib.log(" INFO -> Encontrado VirtualHost "+linea)
            inibloque=n
            ## Cogemos IP y puerto para ese VirtualHost
            print "<br>"+linea
            ini=linea.find(" ")
            fin=linea.find(":")
            IP=linea[ini+1:fin] #IP de VHost Apache
            puerto=linea[fin+1:len(linea)-2] ## Leemos el puerto en el que esta configurado
            print "<br>"+IP+":"+puerto
            ##
            continue
        if re.match("</VirtualHost>", linea):
            finbloque=n
            if inibloque!=0:
                nginxReverselib.log(" INFO -> Obteniendo bloque ["+str(inibloque)+":"+str(finbloque)+"]
\n")
                bloque=lineas[inibloque:finbloque]
                (nbloque,isConf)=confVirtualHost(bloque, IP, puerto)
                if not isConf :
                    lineas[inibloque:finbloque]=nbloque
                inibloque=0
    nginxReverselib.escribeFichero(nginxReverselib.config["nfConfApache"][0],lineas)
    #nginxReverselib.log(" INFO -> Guardadas nuevas configuraciones para VirtualHost de Apache\n")
```

Figura 16. Extracto Ap2Nginx.py. Lectura bloques VirtualHost

Una vez hecho esto, se pasa el bloque entero, junto con la IP y el puerto, y se comprueba si ha sido ya procesado —no se encuentra configurado para NGINX— o si no ha sido excluido en la configuración de los *scripts* (marcado para no configurar). Realizada esta comprobación, se procede a crear el fichero que contendrá las directivas relativas a cada web en el proxy. Cabe destacar, que para cada servidor virtual se crea un fichero de configuración independiente, facilitando las tareas de administración y proporcionando flexibilidad en las configuraciones (véase la *figura 17 y figura 18*).

```
##############################################################################################
# Funcion que recibe un bloque de configuracion VirtualHost encontrado en el fichero de configuracion
de Apache y comprueba que no este ya
# configurado en NGINX. Para ello mira en las variables de los scripts. Si ha sido configurado no hace
nada y si no esta aun
# configurado, cambia la configuracion de log para Apache y procede a crear la configuracion en Nginx.
Devuelve si ha sido configurado y la
# nueva configuracion para Apache
##############################################################################################
def confVirtualHost ( bloque, IP, puerto ):
    serverName=""
    accessLog=""
    isConf=False
    print "<br>"+"Leyendo bloque VirtualHost"
    # Recorremos el bloque para averiguar los datos de conf del VServer
    for n in range(0,len(bloque)):
        linea=bloque[n]
        if re.match("(\s)*ServerName (\S)+\n",linea):
            nginxReverselib.log(" INFO -> Encontrado ServerName : "+linea)
            ini=linea.find("e ")
            fin=len(linea)-1 #Tambien quitamos el \n
            serverName=linea[ini+2:fin]#Guardamos el nombre del servVirt
            print "<br>"+"Encontrado "+serverName+""
            #
            # Comprobamos que no este en excepciones
            try:
                k=nginxReverselib.config["vs-exception"].index(serverName) # Si existe no lanza EXC
                nginxReverselib.log(" INFO -> EXCEPCION : "+serverName+"\n")
                print "<br><b>"+serverName+"</b> Excepcion para no configurar en NGINX"
                isConf=True
                break #BLOQUE EN EXCEPCION DE CONFIGURACION
            except (KeyError):
                #SIN CONFIGURACION DE EXCEPCIONES EN EL MODULO
                nginxReverselib.config["vs-exception"]=[]
                nginxReverselib.log(" INFO -> "+serverName+" NO EXCEPCION\n")
            except (ValueError):
                #NO EXCEPCION
                nginxReverselib.log(" INFO -> "+serverName+" NO EXCEPCION\n")
            ###
            #
            # Comprobamos que no este ya configurado
            try:
                k=nginxReverselib.config["v-servers"].index(serverName) # Si existe no lanza EXC
                nginxReverselib.log(" INFO -> YA CONFIGURADO : "+serverName+"\n")
                print "<br><b>"+serverName+"</b> Ya configurado en NGINX"
                isConf=True
                break #BLOQUE YA CONFIGURADO
            except (KeyError):
                #SIN CONFIGURACION EN EL MODULO PARA VSERVERS CONFIGURADOS
                nginxReverselib.config["v-servers"]=[]
                nginxReverselib.log(" INFO -> "+serverName+" NO CONFIGURADO\n")
            except (ValueError):
                #SIN CONFIGURACION EN NGINX
                nginxReverselib.log(" INFO -> "+serverName+" NO CONFIGURADO\n")
            ###
            continue
        if re.match("(\s)*CustomLog (/?\S+)+ (\S)+\n", linea):
            print "<br>"+"Encontrada entrada CustomLog"
            if serverName!="" and not isConf : ## Si tenemos nombre de servidor y no esta configurado
                bloque[n]=" CustomLog /var/log/virtualmin/"+serverName+"_apache_access_log combined\n"
                nginxReverselib.log(" INFO -> Cambiando CustomLog en ApacheCONF para "+serverName+"
ANTIGUO - "+linea+" -\n")
                creaVSconfNGINX(serverName, IP, puerto) ##Procedemos a crear su fichero de
configuracion en NGINX
                break
    return (bloque,isConf)
```

Figura 17. Extracto Ap2Nginx.py. Comprueba configuración VirtualHost.

```
##############################################################################################
# Funcion que crea el fichero de configuracion de Nginx para los parametros VirtualServer que recibe
como parametro
##############################################################################################
def creaVSconfNGINX (serverName, IPAp, puerto):
    pos=serverName.find(".")
    if pos==-1:
        pos=len(serverName)
    userName=serverName[:pos]
    ip=nginxReverselib.config["IP"][0] # IP configurada por el usuario para escuchar del exterior
    apaIP=nginxReverselib.config["ApacheIP"][0] #IP en la que escuchaba Apache del exterior
    if ip=="" :
        ip=apaIP
    conf=["server {\n listen "+ip+":80;\n server_name www."+serverName+" "+serverName+";\n",
    "\n # Strict Transport Security\n add_header Strict-Transport-Security max-age=2592000;\n",
    " return 301 https://www."+serverName+"$request_uri; \n}\n",
    "\nserver {\n # server port and name\n listen "+ip+":443;\n",
    " server_name "+serverName+" www."+serverName+";\n root /home/"+userName+"/public_html;\n",
    "\n # ssl certificate files;\n ssl on;\n ssl_certificate /home/"+userName+"/ssl.cert;\n
ssl_certificate_key /home/"+userName+"/ssl.key;\n",
    "\n # add ssl specific settings\n keepalive_timeout 60;\n ssl_session_cache shared:SSL:10M;\n
ssl_session_timeout 10m;\n\n # limit ciphers\n ssl_ciphers HIGH:!aNULL:!MD5;\n",
    " ssl_protocols TLSv1 TLSv1.1 TLSv1.2;\n ssl_prefer_server_ciphers on;\n","\n access_log /var/log/
virtualmin/"+serverName+"_access_log;\n error_log /var/log/virtualmin/"+serverName+"_nginx_error_log;\n
\n",
    " location / {\n proxy_pass http://"+IPAp+":"+puerto+";\n include /etc/nginx/proxy.conf;\n }\n",
    "\n location = /robots.txt {\n allow all;\n log_not_found off;\n"," access_log off;\n break;\n }
\n",
    "\n location ~* ^.+\\.(js|jpeg|jpg|gif|png|ico|css|zip|tgz|gz|rar|bz2|doc|xls|exe|pdf|ppt|txt|tar|
mp3|htm|html|xml)$ {\n",
    " expires 30d;\n gzip_static on;\n try_files $uri $uri/ =404;\n }\n\n location ~ /\\.ht {\n deny
all;\n }\n}\n"]
    rutaF=nginxReverselib.config["rutaNginx"][0]+"conf.d/"+serverName+".conf" #Ruta ficheros de
configuracion Servidores Virtuales NGINX
    nginxReverselib.log("INFO -> Configuracion "+serverName+" NGINX creada\n")
    nginxReverselib.escribeFichero(rutaF, conf)
    if "v-servers" not in nginxReverselib.config:
        nginxReverselib.config["v-servers"]=[]
    nginxReverselib.config["v-servers"].append(serverName)
    print "<br>Agregado a v-servers <b>"+serverName+"</b>"
```

Figura 18. Extracto Ap2Nginx.py. Configuración para NGINX

Como apunte importante, es necesario remarcar que la configuración por defecto establecida en éstos archivos requiere que se incluyan en el directorio raíz de las webs —ubicado dentro del directorio `/home/` del sistema— los ficheros relacionados con las conexiones HTTPS, es decir, tanto el fichero que contiene el certificado, llamado `ssl.cert`, y que se envía al usuario para cifrar las conexiones; como el que contiene la clave privada del servidor virtual, llamado `ssl.key`, utilizada para descifrar la información.

4.3.5 Configuración de Virtual Servers en NGINX

Una vez hemos visto qué operaciones realizan los scripts para realizar la configuración como proxy inverso de Apache, detallaremos el contenido de los ficheros que se han creado en NGINX para arrojar luz sobre los ajustes implementados.

En primer lugar y como elemento guía de nuestras configuraciones, se ha llevado a cabo un estudio estadístico que analiza el tamaño de los recursos para varias webs. De esta forma tenemos una visión mas clara acerca de que parámetros otorgar al *surrogate*. Véase el Anexo 1.

Volviendo a los ficheros de configuración para NGINX. El contenido del archivo principal de configuración del servicio, llamado `nginx.conf`, alberga las directivas que son comunes y se aplican a todos los servidores virtuales que contiene el *surrogate*. Dentro de este fichero, las directivas empleadas, distribuidas por el contexto al que se aplican, son:

- `user apache;`

Se usa para indicar el usuario con el que se ejecutan los *workers*, es decir, los procesos que atienden las peticiones. En este caso, `apache` es el usuario por defecto que tiene acceso al contenido de los servidores virtuales ya que VirtualMin utiliza éste usuario en su creación. Contexto: main

- `worker_processes auto;`

  Número de procesos *worker* que atienden peticiones y gestionan las conexiones. NGINX adecuará éste valor en función del número de CPUs que contenga el servidor. Contexto: main

- `worker_connections 2048;`

  Número máximo de conexiones simultaneas que puede atender un *worker*. Contexto: events

- `use epoll;`

  Método de procesado de conexiones. `epoll` es un método eficiente usado en los kernels de Linux a partir de la versión 2.6. Contexto: events

- `log_not_found on;`

  Habilita el registro en el error log para ficheros no encontrados. De esta forma, NGINX incluira aquellas peticiones que obtengan un codigo de respuesta 404. Contexto: http

- `include /etc/nginx/mime.types;`

  Agrega el contenido del fichero `mime.types` situado en `/etc/nginx` a la configuración. Este fichero contiene directivas para soportar envío y recepción de contenido codificados con estos tipos de codificación estándar. Contexto: http

- `proxy_cache_path /var/nginx/cache levels=1:2 keys_zone=zonacache:30M max_size=100M;`

  Esta directiva indica al proxy la configuración relativa a la cache. En este caso, el lugar en disco en que se almacenan los objetos cacheados es `/var/nginx/cache`. Esta ruta de cache contendrá 2 niveles de subdirectorios que se crearán en función del hash obtenido a partir de los datos del objeto. Las claves se almacenan en una parte de la memoria compartida que tiene el nombre de zonacache y un máximo de 30MB.

  Según la documentación de NGINX, 1MB de memoria puede llegar a albergar cerca de 8000 claves, por lo que en caso de que el numero de claves sea tan alto que los objetos sobrepasen los 100MB en disco, el proceso de gestión borrará las entradas menos usadas. Contexto: http

- `sendfile on;`

  Permite hacer uso de la funcion del kernel `sendfile`, mucho mas efectiva para realizar lecturas del disco, acelerando este proceso. Contexto: http

- `tcp_nodelay on;`

  Permite ignorar el mecanismo implementado por el algoritmo de Nagle[4], que obliga a que todos los paquetes estén completos —tamaño igual al MTU[5] de la red— antes de ser enviados. La activación de la directiva `tcp_nodelay` permite que los paquetes sean enviados tan pronto como sea posible, evitando así, retardos en las respuestas. Contexto: http

- `tcp_nopush on;`

  Combinado con `sendfile`, la activación de esta directiva hace que NGINX espere a que el contenido de la respuesta alcance el MTU (Maximum Transmission Unit) de la red, evitando la sobrecarga de la misma. Cuando una lectura con `sendfile` no llega al MTU se deshabilita esta opción y entra en funcionamiento `tcp_nodelay`. Contexto: http

- `gzip on;`

  Junto con las opciones siguientes del fichero, habilita la compresión del contenido de las respuestas para todos los tipos MIME indicados en `gzip_types`, siempre y cuando el navegador del cliente no sea Microsoft Internet Explorer 6 (`gzip_disable`) —en el que no funciona bien la descompresión— y el tamaño de la carga sea superior a 1100Bytes (`gzip_min_length`). El nivel de compresión, además, viene determinado por las características del compresor, que a partir del nivel 2 reduce poco el tamaño del fichero comprimido en proporción al uso de CPU (véase el análisis de Gzip desarrollado en el Anexo 2). Contexto: http

- `keepalive_timeout 10s;`

  Tiempo máximo que espera NGINX antes de cerrar una conexión en que no recibe ninguna petición. En el valor indicado en la configuración del servicio, se establece un tiempo pequeño para evitar la infrautilización del servidor teniendo conexiones inactivas. Contexto: http

- `client_header_buffer_size 2k;` y `large_client_header_buffers 4 2k;`

  Tamaño de los buffers de recepción de cabeceras HTTP por cada conexión para las peticiones. En caso de que el tamaño de cabeceras de una petición sea mayor de 2KBytes se aplica el parámetro `large_client_header_buffers`. Contexto: http

- `client_body_buffer_size 64k;` y `client_max_body_size 100M;`

  64KBytes es el tamaño del buffer que almacena el contenido (*payload*) de cada petición. Además, el tamaño máximo de envío de datos del cliente por petición POST

4 Más información sobre el funcionamiento del algoritmo de Jonh Nagle en [14]

5 Tamaño máximo de *payload* (carga útil) junto con las cabeceras para un datagrama TCP. En Ethernet 1500Bytes.

es de 100MBytes. Este último parámetro ha de ser adecuado a las necesidades de las webs ya que puede ser un problema de seguridad habilitar un tamaño excesivo debido al consumo de ancho de banda necesario para enviar al servidor tales cantidades de datos. Contexto: http

- `send_timeout 10s;`

  En el caso de que transcurran 10 segundos entre dos escrituras para un mismo cliente en una misma respuesta, NGINX cierra la conexión para evitar el despilfarro de recursos. Contexto: http

- `output_buffers 4 32k;`

  Tamaño y número de buffers para leer una respuesta almacenada en el disco duro, en nuestro caso un objeto estático. Basándonos en el análisis estadístico anterior (Anexo 1), podemos apreciar que el 97% de los elementos de las webs analizadas tenían un tamaño igual o inferior a 128KBytes, que es el máximo fijado para los buffers de lectura de disco (4 x 32KB = 128KB). Contexto: http

- `include /etc/nginx/conf.d/*.conf;`

  NGINX incluirá los ficheros de configuración —aquellos con extensión `.conf`— que se encuentran en el directorio `/etc/nginx/conf.d/`. Si nos fijamos, es en éste directorio en que el script `Ap2Nginx.py` ubica los ficheros de configuración para NGINX de los servidores virtuales detectados en Apache, de esta forma es como el proxy carga las distintas configuraciones para cada uno de ellos. Contexto: http

Así, el contenido de este fichero es el que puede apreciarse en la *figura 19*.

```
user apache;
worker_processes auto;

error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
 worker_connections 2048;
 use epoll;
}

http {
 log_not_found on;
 include /etc/nginx/mime.types;

 proxy_cache_path /var/nginx/cache/ levels=1:2 keys_zone=zonacache:30M max_size=100M;

 default_type application/octet-stream;

 log_format main '$remote_addr - $remote_user [$time_local] "$request" '
 '$status $body_bytes_sent "$http_referer" '
 '"$http_user_agent" "$http_x_forwarded_for"';

 access_log /var/log/nginx/access.log main;

 sendfile on;
 tcp_nopush on;
 tcp_nodelay on;

 gzip on;
 gzip_http_version 1.1;
 gzip_vary on;
 gzip_disable "MSIE [1-6]\.(?!.*SV1)";
 gzip_comp_level 2;
 gzip_proxied any;
 gzip_min_length 1100;
 gzip_buffers 16 16k;
 gzip_types text/plain text/css application/json application/javascript application/x-javascript text/
javascript text/xml application/xml application/rss+xml application/atom+xml application/rdf+xml

 keepalive_timeout 10s; #Timeout de una conexion keepalive

 client_header_buffer_size 2k; #tamaño del bufer recepcion de headers para una peticion del cliente
 large_client_header_buffers 4 2k; #tamaño de los buffers de recepcion de headers para peticiones con
cabeceras grandes,  se usa en GSM
 client_body_buffer_size 64k; #tamaño bufer recepcion payload del cliente para una peticion
 client_max_body_size 100M; #Maxio tamaño del payload de un cliente para una peticion
 client_body_timeout 10s; #Tiempo de timeout sin recibir algo de body de un cliente, tiempo entre dos
lecturas
 client_header_timeout 10s; #Tiempo de timeout para recibir un header entero del cliente
 send_timeout 10s; #Tiempo que se envia al cliente para que espere entre dos recepciones de una misma
respuesta
 reset_timedout_connection on; #Cierre abrupto de una conexion con tiempo de espera agotado
 output_buffers 4 32k; #Numero y tamaño de buffers para leer una respuesta del disco 128k

 include /etc/nginx/conf.d/*.conf;
}
```

Figura 19. Contenido de nginx.conf

Por otro lado, las configuraciones especificas a cada una de las webs se distribuyen en ficheros separados tal y como se ha tratado en el punto 4.3.3, donde se explica el funcionamiento del *script* `Ap2Nginx.py`. Estos ficheros contienen las directivas que se aplican por separado a cada servidor virtual que, en NGINX, vienen determinados por el bloque `server`, en cuyo interior se encuentran. Ademas, es necesario aclarar que cada fichero de configuración para un servidor virtual contiene dos bloques de directivas `server`.

El primero de estos bloques sirve para indicar al cliente que debe usar el protocolo HTTP con cifrado, o lo que es lo mismo, HTTPS, comunicando a todas las peticiones que se reciben por el puerto 80 (HTTP) que deben hacerlo al 443 (HTTPS). Para ello, el bloque `server` ubicado en el pueto 80 envía una respuesta con un código de estado 301 Moved Permanently que obliga al navegador a realizar todas las peticiones posteriores usando la dirección que lo acompaña, y además, agrega la cabecera Strict-Transport-Security —habilitando el mecanismo HSTS— que indica al navegador del cliente que sólo debe comunicarse por HTTPS con el servidor[7].

De esta forma, el navegador del cliente realizará las conexiones sucesivas al puerto 443. Es el segundo bloque de directivas el que contiene la configuración HTTPS para el servidor virtual. Así, los parámetros que el script `Ap2Nginx.py` otorga por defecto durante su ejecución para este bloque son los siguientes:

- `listen 192.168.15.128:443;`

  La directiva `listen` se utiliza para indicar la dirección IP y el puerto en el que se atienden peticiones para ese servidor virtual. Por defecto, el valor que se aplica para la dirección es el que obtiene en la lectura del fichero de configuración de Apache para los VirtualHost.

- `server_name prueba.com www.prueba.com;`

  La directiva `server_name` indica los nombres de un servidor virtual. Estos valores se comparan con el del campo de cabecera `Host` de la petición HTTP para aplicar la configuración específica para cada web.

- `root /home/prueba/public_html;`

  Directorio en el que se encuentran los recursos que conforman la web almacenados en el disco duro, al cuál se accede para su lectura y posterior envío cuando son solicitados.

- `ssl on;`

  Habilita el cifrado de las comunicaciones HTTP.

- `ssl_certificate /home/prueba/ssl.cert;`

  Certificado digital con información que identifica el propietario del sitio web y que contiene la clave pública (cifrado asimétrico[6]), que los clientes usarán para cifrar el intercambio de clave[7]. Por defecto, el nombre asignado es `ssl.cert` y se ubica en la raíz del directorio del usuario propietario del servidor virtual.

  Éste certificado puede ser obtenido de una entidad certificadora, de forma que constate la información del servidor, o puede ser creado por nosotros mismos y estar autofirmado. El proceso de creación de la solicitud de firma de certificado (necesaria para el método con autoridad certificadora) y de autofirmado se ha detallado en el Anexo 3.

- `ssl_certificate_key /home/prueba/ssl.key;`

  Ruta hacia el fichero que contiene la clave privada con la que se ha firmado digitalmente el certificado, además, esta clave se utiliza para descifrar los datos que se reciben del cliente —previamente cifrados con nuestra clave pública— durante el proceso de negociación de la conexión segura. Por defecto el nombre asignado es

6 Más información acerca del cifrado asimétrico puede encontrarse en [38]

7 Más información acerca del cifrado híbrido puede encontrarse en [38]

`ssl.key` y se ubica en la raíz del directorio del usuario propietario del servidor virtual.

Esta clave es necesario que sea creada por nosotros mismos para obtener el certificado digital independientemente de cualquiera de las formas mencionadas en el punto anterior, ya sea a través de una CA (Autoridad Certificadora) o autofirmado. El proceso de obtención de la clave privada se encuentra detallado en el Anexo 3.

- `ssl_ciphers HIGH:!aNULL:!MD5;`

  Esta directiva indica a NGINX que sólo debe usar cifrados fuertes. En este caso rechazamos el uso de los algoritmos MD5 y aNULL (ADH y AECDH), todos ellos vulnerables.

- `ssl_protocols TLSv1 TLSv1.1 TLSv1.2;`

  Al igual que la directiva anterior, sólo usamos protocolos criptográficos fuertes, como es TLS[8].

- `ssl_prefer_server_ciphers on;`

  Se indica a NGINX que deben prevalecer en la negociación los cifrados ofrecidos por el servidor en vez de los del cliente.

- `ssl_session_cache shared:SSL:10m;` y `ssl_session_timeout 10m;`

  Establecemos una memoria compartida por todos los workers en la que se almacenan durante 10 minutos (`ssl_session_timeout`) los parámetros de las sesiones SSL, para evitar su renegociación —que es la parte más costosa del proceso— en caso de un mismo cliente.

- `access_log /var/log/virtualmin/prueba.com_access_log;` y `error_log /var/log/virtualmin/prueba.com_nginx_error_log;`

  VirtualMin por defecto establece que los ficheros de log de los servidores virtuales en Apache registren los fallos y accesos en los ficheros anteriores. Puesto que ya no es Apache sino NGINX el encargado de recibir todo el trafico, éstos se establecen con el mismo nombre que les otorga VirtualMin, de forma que éste sea capaz de interpretarlos.

- `location = /robots.txt { … }`

  Cada vez que se recibe una petición, NGINX busca el bloque de directivas `location` que más se ajusta a la URI que el cliente ha solicitado para aplicarle las políticas que en su interior se hallan. Cuando el recurso solicitado es `/robots.txt`, se aplican las directivas de permitir acceso a todo el mundo sin mantener un registro de accesos para el recurso.

8 Transport Layer Security (TLS) es un protocolo que proporciona comunicaciones privadas en Internet [2].

- `location ~* ^.+\.(js|jpeg|jpg|gif|png|ico|css|zip|tgz|gz|rar|bz2|doc|xls|exe|pdf|ppt|txt|tar|mp3|htm|html|xml)$ { … }`

  Al igual que el bloque anterior, cuando la URI solicitada por el cliente encaja con la expresión regular que lo acompaña —tiene una extensión de entre las que se encuentran entre paréntesis— , será NGINX el que accede al directorio indicado por la directiva `root` para leer el recurso y servirlo (`try_files $uri $uri/ =404;`). En caso de que no encuentre el recurso, devolverá un codigo de respuesta 404-Not found. Además, la directiva `gzip_static on;` indica al *surrogate* que siempre debe comprimir las respuestas de contenido estático si el cliente lo soporta.

- `location ~ /\.ht { … }`

  Por seguridad, se deniega el acceso a todos los ficheros cuyo nombre empieza por `.ht`. Recordemos que los ficheros `.htaccess` de Apache contienen información importante acerca de las políticas del servidor para cada directorio. Esta directiva impide (`deny all;`) el acceso a este tipo de ficheros.

- `location / { … }`

  Este bloque de directivas, es el que establece que, en caso de no cumplirse ninguno de los bloques `location` explicados anteriormente, NGINX realice la petición a Apache (`proxy_pass`) aplicándole las directivas de configuración del fichero `/etc/nginx/proxy.conf`, es decir, hace que Apache sirva todo el contenido que no sea estático.

A modo de ejemplo, en la *figura 20* puede verse el contenido del fichero de configuración para la web `prueba.com` ubicada en el servidor de pruebas.

```
server {
 listen 192.168.15.128:80;
 server_name www.prueba.com prueba.com;

 # Strict Transport Security
 add_header Strict-Transport-Security max-age=2592000;
 return 301 https://www.prueba.com$request_uri;
}

server {
 # server port and name
 listen 192.168.15.128:443;
 server_name prueba.com www.prueba.com;
 root /home/prueba/public_html;

 # ssl certificate files;
 ssl on;
 ssl_certificate /home/prueba/ssl.cert;
 ssl_certificate_key /home/prueba/ssl.key;

 # add ssl specific settings
 keepalive_timeout 60;
 ssl_session_cache shared:SSL:10M;
 ssl_session_timeout 10m;

 # limit ciphers
 ssl_ciphers HIGH:!aNULL:!MD5;
 ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
 ssl_prefer_server_ciphers on;

 access_log /var/log/virtualmin/prueba.com_access_log;
 error_log /var/log/virtualmin/prueba.com_nginx_error_log;

 location / {
 proxy_pass http://192.168.15.128:9080;
 include /etc/nginx/proxy.conf;
 }

 location = /robots.txt {
 allow all;
 log_not_found off;
 access_log off;
 break;
 }

 location ~* ^.+\.(js|jpeg|jpg|gif|png|ico|css|zip|tgz|gz|rar|bz2|doc|xls|exe|pdf|ppt|txt|tar|mp3|htm|
html|xml)$ {
 expires 30d;
 gzip_static on;
 try_files $uri $uri/ =404;
 }

 location ~ /\.ht {
 deny all;
 }
}
```

Figura 20. Contenido del fichero prueba.com.conf

Una vez hemos visto las políticas con que son configurados por defecto los servidores virtuales en NGINX, el comportamiento como proxy del *surrogate* viene definido en el fichero al que se ha hecho referencia anteriormente —recordemos que para cualquier recurso no estático NGINX realiza las peticiones a Apache—, este es, `/etc/nginx/proxy.conf`. A continuación se detalla el contenido de éste fichero:

- `proxy_set_header Host $host;`

  Establece la cabecera Host con el valor que se recibe en la petición por parte del cliente, si no existe este campo de cabecera (HTTP/1.0), se sustituye por el nombre del servidor NGINX.

- `proxy_set_header X-Real-IP $remote_addr;`

  Se añade una cabecera para que Apache conozca la dirección del cliente que realizó originalmente la petición a fin de que pueda llevar un registro de sucesos correcto.

- `proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;`

  La cabecera X-Forwarded-For lleva como valor la dirección del proxy, en este caso NGINX, a través del cual se envía a Apache la petición. Al igual que el parámetro anterior, esto se lleva a cabo con el fin de que Apache sepa realmente de quién recibe peticiones.

- `proxy_connect_timeout 5s;`

  Esta directiva, establece un tiempo de vida de la conexión con el servidor Apache de 5 segundos. Transcurrido este periodo de tiempo, en que la conexión esté inactiva, la conexión se dará por concluida.

- `proxy_buffering off;`

  Deshabilitando la capacidad de *buffering*[9] de NGINX, logramos que la respuesta sea pasada directamente al cliente desde el instante en que se va recibiendo de Apache, eliminando así, los retardos causados por éste mecanismo.

- `proxy_buffers 4 64k;`

  Esta directiva indica el número total de buffers y su tamaño que NGINX dispondrá para leer una respuesta para una sola conexión. En nuestro caso, la capacidad de *buffering* de NGINX para cada una de ellas se ha fijado en 256KB, más que suficiente según nuestro análisis (véase Anexo 1).

- `proxy_buffer_size 128k;`

  La directiva `proxy_buffer_size` establece el tamaño de espacio en memoria que utiliza NGINX para recibir la respuesta de Apache y enviarla inmediatamente. Este tamaño se ha ajustado a 128KB de forma que se pueda recibir todo el contenido de las respuestas de una sola vez, englobando así, y según nuestro análisis (véase Anexo 1) cerca del 100% de las mismas.

- `proxy_busy_buffers_size 128k;`

  Esta directiva indica el tamaño de `proxy_buffers` que se usa para responder al cliente y, por tanto, permanece bloqueado frente a escrituras mientras la respuesta todavia no ha sido leida totalmente. Asi, se usan 128KB para leer desde Apache y 128KB para responder al cliente.

- `proxy_pass_header Set-Cookie;`

  Permitimos que el servidor Apache gestione las sesiones de los clientes incluyendo la cabecera `Set-Cookie` de las peticiones HTTP que NGINX recibe. Esto se debe a que es Apache quien responde las solicitudes de contenido dinámico, el cuál puede estar condicionado por los datos de las *cookies*.

9 Se llama *buffer* a la memoria que se utiliza temporalmente mientras se realiza un flujo de salida o de entrada de datos. Por lo tanto, *buffering* es la capacidad de un sistema de utilizar parte de su memoria para este cometido.

- `proxy_cache zonacache;`

  Se establece la zona de memoria compartida de cache como `zonacache`. Esta zona ha sido definida en el fichero `/etc/nginx/nginx.conf` ,y tal y cómo se ha nombrado antes, es en esta zona de memoria donde se almacenan las claves activas —referencia a los datos almacenados en disco resultado de aplicar una suma md5[10] al nombre del recurso—.

- `proxy_cache_methods GET HEAD;`

  La directiva `proxy_cache_methods` establece que solamente sean tratadas por el módulo de cache de NGINX aquellas peticiones cuyo método HTTP sea `GET` o `HEAD`.

- `proxy_cache_min_uses 2;`

  Número mínimo de veces que se debe solicitar un recurso para que sea almacenado en cache en disco.

- `proxy_cache_valid 200 2m;`

  Tiempo que permanece como válida la clave de un recurso en `zonacache`. Transcurrido este tiempo (2 minutos), el proxy volverá a solicitar el recurso a Apache cuando sea solicitado por algún cliente. Además, el primer argumento indica que sólo se almacenaran en cache las respuestas cuyo código de estado sea `200OK`., esta condición se une a las dos directivas anteriores como requisito para ser almacenado en cache.

- `proxy_cache_bypass $cookie_nocache;` y `proxy_no_cache $cookie_nocache;`

  Éstas dos directivas indican a NGINX que no debe almacenar en cache, es decir, que ignore aquellas respuestas que contengan el campo de cabecera `Cache-Control: no-cache`.

- `proxy_cache_key $scheme$host$request_method$request_uri;`

  La directiva `proxy_cache_key` establece qué información es utilizada para calcular la suma md5 que sirve como referencia en memoria compartida hacia un recurso almacenado en disco duro. Además, el nombre de los ficheros en disco duro que contienen los datos se obtiene de esta suma. Para nuestra configuración se han elegido: `$scheme`, que puede ser HTTP o HTTPS; `$host`, el nombre de la web para la que se solicita el recurso; `$request_method`, contiene el método HTTP de la petición; `$request_uri`, dirección completa de la petición (incluidos el host y los parámetros).

10 Función hash (resumen) que calcula un identificador único (suma) de una longitud fija a partir de los datos que componen el objeto. Es utilizado como comprobación de integridad ya que un fichero con la misma información da como resultado la misma suma.

Una vez explicadas las directivas utilizadas, el fichero que contiene las políticas de configuración para el módulo proxy de NGINX puede verse en la *figura 21*.

```
####Configuracion de PROXY

proxy_set_header Host $host; # Anyade la cabecera host a la peticion que reenvia al servidor
proxy_set_header X-Real-IP $remote_addr; # Anyade la cabecera con la direccion IP del cliente
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; # Contiene la client request-header "X-
Forwarded-For" o remote address
proxy_pass_header Set-Cookie; ## que no elimine la cabecera Set-Cookie de las peticiones

#Timeouts
proxy_connect_timeout 5s; #Timeout maximo para conectar con el servidor Apache
proxy_send_timeout 20s; #Timeout entre escrituras sucesivas para una misma peticion hacia Apache
proxy_read_timeout 20s; #Timeout entre lecturas sucesivas para una misma peticion desde Apache

#Buffering
proxy_buffering off; # NGINX responde al cliente tan pronto como recibe algo de Apache
proxy_buffers 4 64k; # Tamaño de los buffers de lectura de una respuesta para una conexion (256k total)
proxy_buffer_size 128k; # Máximo que puede recibir NGINX de Apache en una respuesta de una sola vez
proxy_busy_buffers_size 128k; #Tamaño del buffer que se usa para responder al cliente

####Configuracion de PROXY CACHE

proxy_cache zonacache; ## Nombre de la zona de memoria compartida donde buscará o almacenará las
entradas

proxy_cache_min_uses 2; ## Minimo de ocurrencias para que una misma peticion sea almacenada en cache
proxy_cache_valid 200 2m; ## Solo guardar en cache las respuestas 200 OK durante 2m (las que hayan
ocurrido 2 veces además)

proxy_cache_bypass $cookie_nocache;
proxy_no_cache $cookie_nocache;

proxy_cache_key $scheme$host$request_method$request_uri; ## Clave identificativa de cada peticion que
se almacena en cache

proxy_cache_methods GET HEAD; ## Solo guardar en cache las peticiones get y head
```

Figura 21. Contenido del fichero proxy.conf

A modo de resumen, el comportamiento del *surrogate* cuando reciba una petición por parte de un cliente dependerá de muchos factores. El más importante es el tipo de recurso solicitado. En caso de que ese recurso sea un objeto estático NGINX accederá para leerlo del disco duro y enviarlo al cliente.

En cambio, si la petición se realiza para un objeto dinámico, el proxy calculará la suma md5 a partir de la información de la misma y comprobará en la zona de memoria compartida `zonacache` si ese recurso ya ha sido solicitado de forma previa —al menos en 2 ocasiones en los últimos 2 minutos siempre y cuando el método sea `GET` o `HEAD`—, si es así, lo recupera del directorio dentro del disco duro donde se almacena la cache y lo envía al cliente; si no, NGINX requerirá el recurso a Apache, y cuando lo comience a recibir irá respondiendo al cliente.

Una vez concluido el desarrollo de los scripts de integración de NGINX en el sistema, el siguiente paso era hacer la ejecución de los mismos de forma más fácil e intuitiva. Para ello se ha desarrollado un módulo para el panel de control Webmin desde el que podemos configurar nuestros scripts y lanzarlos, además de brindarnos la posibilidad de modificar los ficheros de configuración que se crean para NGINX por defecto.

4.3.6 Módulo para Webmin

Con el fin de facilitar las tareas de gestión de los scripts de integración, se ha implementado un módulo para el panel de control Webmin desde el que se pueden realizar todas las tareas para los que éstos han sido diseñados. Este panel de control ofrece la posibilidad al administrador de

modificar los parámetros de los scripts en base a la configuración de su sistema o sus necesidades, y ejecutar los programas de integración de forma fácil y sencilla.

Un módulo o complemento de Webmin es simplemente un directorio empaquetado, con una extensión .wbm (un fichero .tar al que cambiamos la extensión a mano), en cuyo interior se encuentran los CGI que ejcutará el servidor cuando el cliente lo solicite (y los ficheros necesarios para su ejecución), es decir, los ficheros relacionados con la lógica del módulo, y por otra parte, los ficheros pertenecientes a la lógica del panel, necesarios para que el complemento sea interpretado correctamente.

Para la parte de la lógica del panel son necesarios ha sido necesario incluir los siguientes ficheros al módulo:

- `install_check.pl`

  Este programa en Perl realiza las comprobaciones necesarias para asegurar que el módulo ha sido instalado correctamente en el panel de forma que se asegura su correcto funcionamiento. Si es así, devuelve 0. En nuestro caso, este fichero contiene básicamente la orden `return 0;`. Recordemos que el fichero .wbm contiene todos los componentes del módulo en su interior, de forma que Webmin únicamente descomprime durante la instalación el módulo empaquetado.

- `module.info`

  Contiene meta-información, es decir, campos clave-valor, que Webmin utiliza para gestión del módulo en el panel. Los campos requeridos son: desc, que contiene una descripción de la funcionalidad del módulo (este texto es el que aparece en el menú de Webmin), en nuestro caso “Nginx como Proxy Inverso de Apache”; os_support, una lista con los sistemas operativos soportados por el módulo, en nuestro caso, cualquier distribución Linux (*-linux); y category, que será la sección del menú de Webmin donde se ubicará el módulo. Para nuestro módulo, la sección del menú en que se sitúa es “Servidores”.

- `lang/en`

  En este directorio se encuentran los ficheros de lenguaje que contienen los campos clave-valor que el panel utiliza para mostrar la información del módulo en el idioma indicado por el nombre del fichero según la norma ISO_639-1[11]. Puede haber tantos ficheros como idiomas disponibles. Sin embargo, y por sencillez, en nuestro caso sólamente disponemos del fichero en inglés cuyo contenido es el nombre de nuestro módulo. Esto se debe a que no hacemos uso de la funciones de las librerias de desarrollo que los utilizan, por lo que, en éste desarrollo, los valores de texto se encuentran en el código.

Respecto a los ficheros pertenecientes a la lógica del módulo que se ha desarrollado, podemos encontrar los CGI y los archivos que éstos mismos utilizan, es decir, tanto las imágenes que se muestran en los menús y que se encuentran dentro del directorio `images/` como los scripts de integración detallados en los puntos anteriores.

11 Códigos de dos letras que identifican los distintos idiomas del mundo [8]

Estos CGI, desarrollados en Python, generan código HTML a partir del resultado que generan. Dentro del módulo, realizan las siguientes acciones:

- `index.cgi`

  Genera el menú principal del módulo de integración desde el que se accede a todas las acciones se pueden realizar. Entre ellas, es posible reiniciar o parar el servidor NGINX, configurar los parámetros del módulo de forma previa a la realización de la integración, realizar la integración con sólo hacer clic en la opcion “Instalar nginx como proxy inverso” del menú, editar los ficheros de configuración del proxy (opcion “Editar archivos de configuración”) y listar los servidores virtuales ya integrados. Véase la *figura 22*.

Figura 22. Página index.cgi

- `install.cgi`

  Este fichero, ejecuta el contenido de los scripts mencionados anteriormente, es decir, realiza la integración de todos los servidores virtuales configurados en Apache para funcionar con NGINX, mostrando los mensajes de estado que los scripts van generando. Este CGI es ejecutado cuando se hace clic en la opcion “Instalar nginx como proxy inverso” del menú. Véase la *figura 23*.

```
#!/usr/bin/env python

##############################################################################################
#
# CGI del modulo NGINX como proxy inverso de Apache para Webmin
#
# Fecha: 17/03/2015
# Autor: Borja Molina Coronado (bormoco@gmail.com)
##############################################################################################
#install.cgi
#installs all necesary tools for the module can works and configure all Apache VirtualServers on NGINX

import nginxReverselib #importamos la libreria del modulo
import cgi
import sys
import subprocess
import ApaConf # Modulo que instala NGINX y RPAF, configura Apache para trabajar con NGINX y cambia
las plantillas de VirtualMIN
import Ap2Nginx # Modulo que lee todos los VSERVERS de Apache y los configura en NGINX para ser
servidos por el.

print "Pragma: no-cache"
print "Content-Type: text/html\n"
sys.stdout.flush()
sys.stderr.flush()
sys.stderr = sys.stdout


print "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\"\n\"http://www.w3.org/TR/REC-
html40/loose.dtd\">"
print "<html><head><link rel='stylesheet' type='text/css' href='/unauthenticated/reset-fonts-grids-
base.css'>"
print "<link rel='stylesheet' type='text/css' href='/unauthenticated/virtual-server-style.css' />"
print "<title>Servidor NGINX como Proxy inverso de Apache</title></head><body>"
ApaConf.main()#lanzamos el script para instalar NGINX como proxy de apache
Ap2Nginx.main()#lanzamos el script para configurar los VServers de Apache en NGINX

print "<br>Reiniciando VirtualMin(Webmin)"
print "<br><br>"
print "<a href='/nginxReverse/index.cgi'>Indice del Modulo</a>"
print "</body></html>"
sys.stdout.flush()
nginxReverselib.log(" INFO - Reiniciando WebMin\n")
subprocess.call(["service","webmin","restart"])
```

Figura 23. Código de install.cgi

- `config.cgi y config_save.cgi`

  `config.cgi`, accesible desde la opcion “Configuración del módulo” del menú principal, genera un formulario desde el que al administrador puede modificar los parámetros de configuración de los scripts de integración, es decir, los atributos del fichero `conf.file`. Véase la *figura 24*.

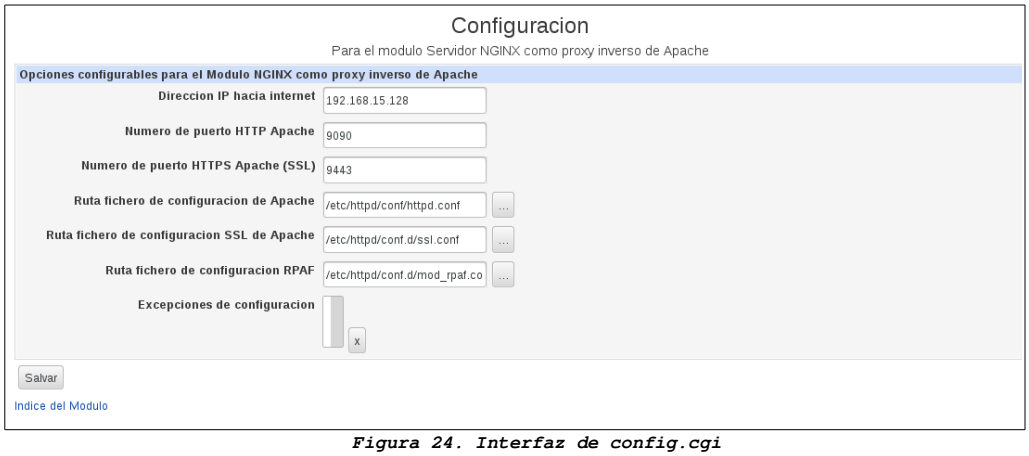

Figura 24. Interfaz de config.cgi

En la *figura 24* se muestra la interfaz generada por el codigo que contiene el CGI de configuración del módulo, y además, pueden verse los campos que el administrador tiene la opción de modificar. Cabe destacar que estos atributos son necesarios para el

correcto funcionamiento de los scripts de integración por lo que habrán de ser cambiados conforme a las características del sistema en que se ejecute el módulo.

Por otro lado, en la *figura 25* se puede ver el código encargado de recibir y almacenar en el fichero de configuración del módulo los cambios realizados, que son enviados por `config.cgi` una vez el administrador hace clic en el botón "Salvar" del formulario.

```
import sys
import os
import cgi
import re # modulo para expresiones regulares
import nginxReverselib

print "Pragma: no-cache"
print "Content-Type: text/html\n"
sys.stdout.flush()
sys.stderr.flush()
sys.stderr = sys.stdout

nginxReverselib.cargaConf()

form = cgi.FieldStorage() # Cargamos los campos que recibe como parametro
if not form["nfConfApache"].value=="" : ##Si no recibimos valor par el campo no hacemos nada
    nginxReverselib.config["nfConfApache"][0]=form["nfConfApache"].value
if not form["nfConfRPAF"].value=="" : ##Si no recibimos valor par el campo no hacemos nada
    nginxReverselib.config["nfConfRPAF"][0]=form["nfConfRPAF"].value
if not form["nfConfSSLApache"].value=="" : ##Si no recibimos valor par el campo no hacemos nada
    nginxReverselib.config["nfConfSSLApache"][0]=form["nfConfSSLApache"].value
if form["nPuertoApSSL"].value!="" and form["nPuertoApSSL"].value!="443" : ##Si no recibimos valor par
el campo o si el valor recibido es donde estara NGINX no hacemos nada
    nginxReverselib.config["nPuertoApSSL"][0]=form["nPuertoApSSL"].value
if form["nPuertoAp"].value!="" and form["nPuertoAp"].value!="80" : ##Si no recibimos valor par el
campo o si el valor recibido es donde esta NGINX no hacemos nada
    nginxReverselib.config["nPuertoAp"][0]=form["nPuertoAp"].value
if form["IP"].value!="" : ##Si no recibimos valor par el campo, no hacemos nada
    nginxReverselib.config["IP"][0]=form["IP"].value
#### Servidores virtuales excluidos que han sido eliminados de la exclusion
if "exvs-exc" in form and form["exvs-exc"].value!="": ##Si no recibimos valor par el campo no hacemos
nada
    valor=form["exvs-exc"].value
    valores=valores=re.split("-",valor) ## separa por los guiones
    valores=valores[0::2] ## quita los guiones
    print valores
    for n in valores:
        nginxReverselib.config["vs-exception"].pop(int(n) ) # borramos de la lista el vs

nginxReverselib.guardaConf()

print "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\"\n\"http://www.w3.org/TR/REC-
html40/loose.dtd\">"
print "<html><head><link rel='stylesheet' type='text/css' href='/unauthenticated/reset-fonts-grids-
base.css'>"
print "<link rel='stylesheet' type='text/css' href='/unauthenticated/virtual-server-style.css' />"
print "<title>Servidor NGINX como Proxy inverso de Apache</title></head><body>"
print "<p>Configuracion del modulo guardada</p>"
print "<p><a href='/nginxReverse/index.cgi'>Indice del Modulo</a></p>"
print "</body></html>"
```

Figura 25. Código de config_save.cgi

- `allmaual_form.cgi` y `allmanual_save.cgi`

  El primero de ellos, muestra al administrador el contenido del fichero de configuración de NGINX que previamente haya sido seleccionado de una lista desplegable. El segundo, por otro lado, almacena los cambios que se hayan realizado en estos ficheros cuando el administrador pulsa el botón "Salvar". Véanse las *figuras 26 y 27*.

Figura 26. Interfaz de allmanual_form.cgi

Mientras que la *figura 26* muestra la interfaz desde la que se pueden realizar los cambios en el fichero seleccionado de la lista desplegable ubicada en la parte superior de la pantalla, en la *figura 27* se puede apreciar el código encargado de almacenar los cambios en el fichero correspondiente.

```
import nginxReverselib #Modulo con las funciones propias que son de utilidad
import subprocess
import cgi
import sys



form = cgi.FieldStorage() # Cargamos los campos que recibe como parametro
if "file" not in form and "data" not in form : ##Si no recibimos nombre de fichero ni datos del
fichero no hacemos nada
    exit()
else :
    print "Pragma: no-cache"
    print "Content-Type: text/html; Charset=UTF-8\n"
    sys.stdout.flush()
    sys.stderr.flush()
    sys.stderr = sys.stdout
    print "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\"\n\"http://www.w3.org/TR/REC-
html40/loose.dtd\">"
    print "<html><head><link rel='stylesheet' type='text/css' href='/unauthenticated/reset-fonts-grids-
base.css'>"
    print "<link rel='stylesheet' type='text/css' href='/unauthenticated/virtual-server-style.css' />"
    print "<title>Servidor NGINX como Proxy inverso de Apache</title></head><body>"
    print "Guardando configuracion en "+ form["file"].value+"</br>"
    datos=form["data"].value ## leemos los datos que se le han mostrado al usuario
    ###datos=datos.replace("</br>","\n") ##cambiamos los saltos de linea de html a \n
    nginxReverselib.escribeFichero(form["file"].value,datos) #escribimos los cambios
    print "</br>"
    print "Guardado</br>"
    print "</br>"
    print "Reiniciando NGINX...</br>"
    nginxReverselib.log(" INFO -> Reinicio NGINX desde WEBMIN\n")
    subprocess.call(["service","nginx","restart"])
    print "Reiniciado</br>"
    print "<a href='/nginxReverse/index.cgi'>Indice del Modulo</a>"
    print "</body></html>"
```

Figura 27. Código allmanual_save.cgi

- `list_vs.cgi`, `edit_vs.cgi` y `drop_vs.cgi`

  `list_vs.cgi` genera un listado que contiene los servidores virtuales configurados en NGINX. Para ello, lee el directorio donde se almacenan los ficheros de los servidores virtuales del proxy.

  Si el administrador hace clic en el nombre de dominio de uno de ellos, `edit_vs.cgi` muestra la información basica de esa web y solicita la confirmación

para deshabilitar el servidor virtual en el *surrogate*. En caso afirmativo, `drop_vs.cgi` es el encargado de borrar el fichero de configuración de esa web en NGINX y por lo tanto, inhabilitándolo para recibir peticiones en el proxy.

Cabe destacar que este proceso sólamente borra el fichero y añade el nombre de dominio a una lista de excepciones de configuración, para que, en caso de que se vuelva a realizar la integración, no sea configurado. De forma que si se desea reconfigurar cualquier web que se encuentre en la lista de excepciones, basta con eliminarla desde el formulario de configuración del módulo. Sin embargo, la configuración para ese servidor virtual en Apache no se verá alterada y permanecerá en el puerto alternativo para no entrar en conflicto con el *surrogate*. El contenido de éstos CGI puede verse en las *figuras 28, y 29*.

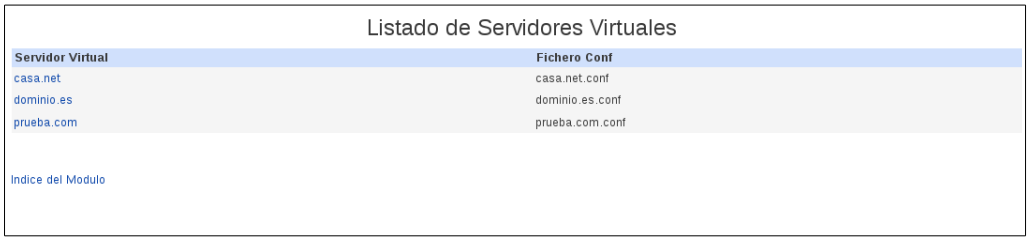

Figura 28. Interfaz de list_vs.cgi

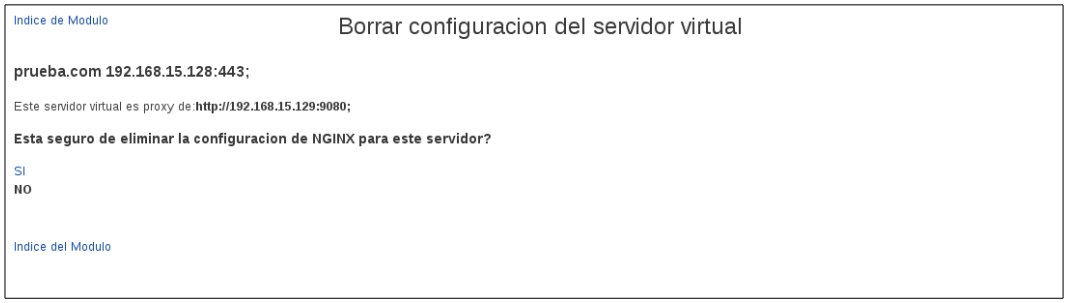

Figura 29. Interfaz de edit_vs.cgi

Este módulo, un empaquetado en `.tar` de todos los elementos que lo conforman, renombrado con formato `.wbm`, puede ser instalado fácilmente en Webmin desde el propio panel. Para ello, en el apartado `Webmin/Configuracion de Webmin/Modulos de Webmin` disponemos de un formulario desde el que seleccionamos el módulo a cargar. Tal y como puede verse en la *figura 30*, basta con pulsar sobre el botón `Instalar módulo` de la interfaz para que nuestro desarrollo quede incluido en el panel.

En nuestro caso, el módulo desarrollado, queda incluido en la sección “servidores” del panel con el nombre de NGINX como proxy inverso de Apache.

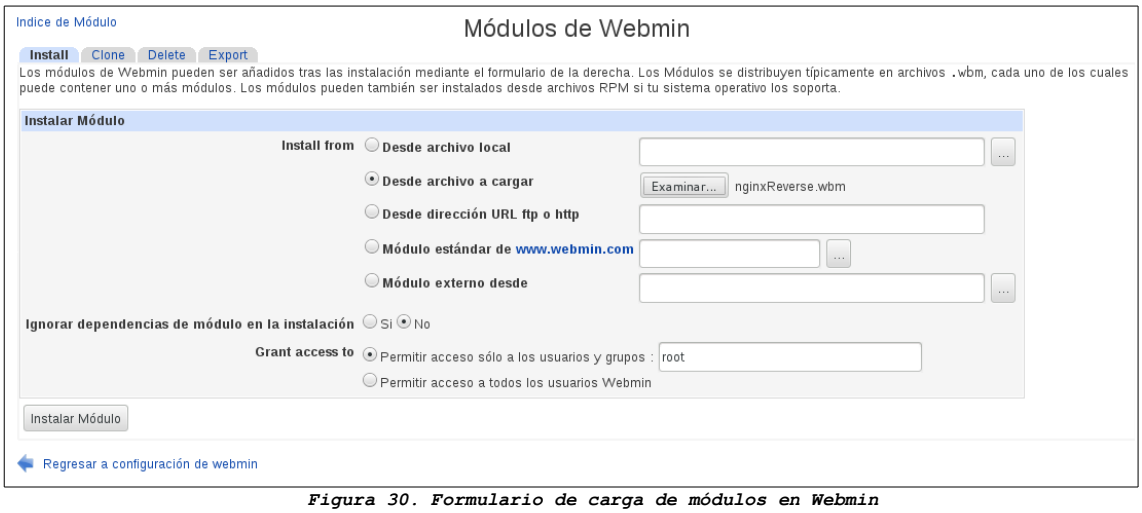

Figura 30. Formulario de carga de módulos en Webmin

5. Pruebas

5.1. Sistema proxy inverso

Una vez finalizado el desarrollo de los scripts y el módulo para el panel Webmin, la siguiente fase era la comprobación del correcto funcionamiento y configuraciones que éstos mismos establecen. De esta forma, la primera parte de las pruebas consistía en la comprobación de que las configuraciones establecidas por los módulos desarrollados eran las deseadas, logrando que el sistema funcionase de la forma adecuada.

Para ello se han realizado peticiones HTTP desde la maquina host —que actuaba como cliente— y observado las cabeceras del protocolo en la respuesta. En concreto, se utilizaba la orden `curl` para realizar ésta función indicándole mediante el parámetro `-H`, el valor de la cabecera `Host:` el servidor virtual en concreto para el que se deseaba comprobar su funcionamiento, en nuestro caso `prueba.com`. Así, la orden, junto con su resultado, pueden verse en la *figura 31*.

```
root@lynx:~# curl -vk -H "Host: prueba.com" http://192.168.15.128/index.html
* Hostname was NOT found in DNS cache
*   Trying 192.168.15.128...
* Connected to 192.168.15.128 (192.168.15.128) port 80 (#0)
> GET /index.html HTTP/1.1
> User-Agent: curl/7.38.0
> Accept: */*
> Host: prueba.com
>
< HTTP/1.1 301 Moved Permanently
* Server nginx/1.6.2 is not blacklisted
< Server: nginx/1.6.2
< Date: Mon, 22 Jun 2015 01:44:05 GMT
< Content-Type: text/html
< Content-Length: 184
< Connection: keep-alive
< Location: https://www.prueba.com/index.html
< Strict-Transport-Security: max-age=2592000
<
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.6.2</center>
</body>
</html>
* Connection #0 to host 192.168.15.128 left intact
root@lynx:~#
```

Figura 31. Forzar HTTPS

Tal y como puede verse en la imagen anterior, el servidor que responde a nuestra petición es NGINX, que atiende las peticiones en el puerto HTTP (80) donde lo habíamos configurado. Además, y debido a la configuración por defecto que aplicamos a todos los servidores virtuales (véase el apartado 4.3.5) para habilitar el mecanismo HSTS —que fuerza conexiones seguras bajo HTTPS—, el código de respuesta que NGINX nos remite es el 301, indicándonos una redirección

hacia la página indicada por el campo de cabecera `Location`; que como se puede apreciar es una dirección bajo HTTPS (`Location: https://www.prueba.com/`).

Siguiendo con las pruebas, y con la intención de comprobar que NGINX redirige las peticiones hacia Apache, es decir, que el mecanismo de proxy inverso se encontrara correctamente desplegado, se ha repetido la orden curl cambiando la dirección por la que NGINX nos había recomendado en la respuesta anterior. De ésta manera, el resultado obtenido puede verse en la *figura 32*.

```
mobylette@lynx:~$ curl -vk -H "Host: prueba.com" https://192.168.15.128/index.html
* Hostname was NOT found in DNS cache
*   Trying 192.168.15.128...
* Connected to 192.168.15.128 (192.168.15.128) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Server key exchange (12):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* Server certificate:
*        subject: C=ES; ST=Valencia; L=Valencia; O=Default Company Ltd; OU=IR; CN=basucon; emailAddress=as
*        start date: 2015-02-18 09:04:48 GMT
*        expire date: 2016-02-18 09:04:48 GMT
*        issuer: C=ES; ST=Valencia; L=Valencia; O=Default Company Ltd; OU=IR; CN=basucon; emailAddress=as
*        SSL certificate verify result: self signed certificate (18), continuing anyway.
> GET /index.html HTTP/1.1
> User-Agent: curl/7.38.0
> Accept: */*
> Host: prueba.com
>
< HTTP/1.1 200 OK
* Server nginx/1.6.2 is not blacklisted
< Server: nginx/1.6.2
< Date: Mon, 22 Jun 2015 14:05:41 GMT
< Content-Type: text/html
< Content-Length: 11
< Last-Modified: Thu, 02 Apr 2015 15:25:14 GMT
< Connection: keep-alive
< ETag: "551d5f5a-b"
< Expires: Wed, 22 Jul 2015 14:05:41 GMT
< Cache-Control: max-age=2592000
< Accept-Ranges: bytes
<
prueba.com
* Connection #0 to host 192.168.15.128 left intact
mobylette@lynx:~$
```

Figura 32. Petición HTTPS a prueba.com

En la imagen anterior podemos apreciar en primer lugar cómo se realiza la fase de negociación de la conexión segura en la que se pactan los parámetros de la conexión, entre ellos, el protocolo de cifrado y el intercambio de claves. Si nos fijamos en la respuesta de la petición, observamos que quien responde a nuestra petición es NGINX (cabecera `Server`), y además, nos envía una cabecera de tipo ETag[12] para control de versiones de cache. Por último, al final de las lineas de cabecera de la respuesta se muestra el contenido del recurso solicitado, en este caso, el texto “prueba.com”.

Con el fin de asegurarnos que para ésta prueba es NGINX el realiza las funciones de servidor web, esto es, quien lee del disco duro el fichero index.html —cuyo contenido es el mostrado en la

12 The entity tag MAY be used for comparison with other entities from the same resource [3].

respuesta de la *figura 30*— para servirlo al cliente, se ha comprobado el fichero de log de acceso que NGINX mantiene y en el que se va añadiendo una linea por cada petición que éste recibe.

Así, en caso de que NGINX fuera efectivamente el que realizaba el procedimiento de leer un recurso estático solicitado y responderlo al cliente sin intervención de Apache, encontraríamos en su fichero de log `/var/log/virtualmin/prueba.com_access_log` la entrada correspondiente a dicha petición, y sin embargo, en el fichero de registro de acceso de Apache `/var/log/virtualmin/prueba.com_apache_access_log` no existiría su equivalente.

En la *figura 33*, se puede apreciar cómo la fecha y hora (GMT) de la petición mostrada por la *figura 32* coincide con la última entrada del fichero de registro de acceso para NGINX (GMT+2), mientras que en el log de Apache no aparece, corroborando el hecho anterior.

```
[root@mv203 ~]# tail -n 2 /var/log/virtualmin/prueba.com_access_log
192.168.15.1 - - [22/Jun/2015:15:22:00 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
192.168.15.1 - - [22/Jun/2015:16:05:41 +0200] "GET /index.html HTTP/1.1" 200 11 "-" "curl/7.38.0"
[root@mv203 ~]# tail -n 2 /var/log/virtualmin/prueba.com_apache_access_log
192.168.15.1 - - [22/Jun/2015:14:50:18 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
192.168.15.1 - - [22/Jun/2015:15:22:00 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
[root@mv203 ~]#
```

Figura 33. Últimas dos entradas en logs de NGINX y Apache

Por último, se ha comprobado el comportamiento de NGINX en caso de solicitarse un recurso dinámico al servidor, en cuyo caso, debía reenviar las peticiones hacia Apache, esperar la respuesta del mismo, y devolverla al cliente. Esto es, comportarse como un proxy inverso.

El procedimiento llevado a cabo es similar al anterior, con la diferencia que, en éste caso la petición debía registrarse en ambos ficheros de registro de acceso. El CGI solicitado, llamado ejemplo.php, se encargaba de generar código HTML de forma dinámica mediante la ejecución de la función `phpinfo()` desde el servidor Apache.

En la *figura 34* se puede ver la petición HTTPS realizada con el comando `curl` (la respuesta ha sido cortada debido a su extensión).

```
mobylette@lynx:~$ curl -vk -H "Host: prueba.com" https://192.168.15.128/ejemplo.php
* Hostname was NOT found in DNS cache
*   Trying 192.168.15.128...
* Connected to 192.168.15.128 (192.168.15.128) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Server key exchange (12):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* Server certificate:
*        subject: C=ES; ST=Valencia; L=Valencia; O=Default Company Ltd; OU=IR; CN=basucon; emailAddress=as
*        start date: 2015-02-18 09:04:48 GMT
*        expire date: 2016-02-18 09:04:48 GMT
*        issuer: C=ES; ST=Valencia; L=Valencia; O=Default Company Ltd; OU=IR; CN=basucon; emailAddress=as
*        SSL certificate verify result: self signed certificate (18), continuing anyway.
> GET /ejemplo.php HTTP/1.1
> User-Agent: curl/7.38.0
> Accept: */*
> Host: prueba.com
>
< HTTP/1.1 200 OK
* Server nginx/1.6.2 is not blacklisted
< Server: nginx/1.6.2
< Date: Mon, 22 Jun 2015 12:47:22 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept-Encoding
< X-Powered-By: PHP/5.3.3
<
{ [data not shown]
100 58869    0 58869    0     0  2207k      0 --:--:-- --:--:-- --:--:-- 2299k
* Connection #0 to host 192.168.15.128 left intact
mobylette@lynx:~$
```

Figura 34. Petición HTTPS ejemplo.php

Así, en la *figura 35* se puede apreciar el contenido de los ficheros de registro después de realizar la petición anterior.

```
[root@mv203 ~]# tail -n 2 /var/log/virtualmin/prueba.com_access_log
192.168.15.1 - - [22/Jun/2015:04:59:04 +0200] "GET /ejemplo.php HTTP/1.1" 200 58
890 "-" "curl/7.38.0"
192.168.15.1 - - [22/Jun/2015:14:47:23 +0200] "GET /ejemplo.php HTTP/1.1" 200 58
882 "-" "curl/7.38.0"
[root@mv203 ~]# tail -n 2 /var/log/virtualmin/prueba.com_apache_access_log
192.168.15.1 - - [22/Jun/2015:04:59:04 +0200] "GET /ejemplo.php HTTP/1.0" 200 58
869 "-" "curl/7.38.0"
192.168.15.1 - - [22/Jun/2015:14:47:22 +0200] "GET /ejemplo.php HTTP/1.0" 200 58
869 "-" "curl/7.38.0"
[root@mv203 ~]#
```

Figura 35. Contenido logs NGINX y Apache

Tal y como cabe esperar, ambos servicios registran la petición, lo que indica el correcto funcionamiento del diseño mostrado en el apartado 3.3.

5.2. Compresión de recursos

El cometido principal de la realización de estas pruebas era la comprobación y demostración del funcionamiento de las configuraciones aplicadas relativas a la compresión de los recursos mediante Gzip. Las directivas detalladas en el apartado 4.3.5 indican a NGINX que debe comprimir el contenido de las respuestas siempre que se cumplan los requisitos especificados en la configuración y el cliente haya indicado que soporta la compresión.

Para ello, el cliente indica al servidor que puede recibir el contenido de las respuestas comprimido en este formato mediante la inclusión de la cabecera HTTP `Accept-Encoding: gzip`. Un ejemplo de una petición realizada utilizando este método puede verse en la *figura 36*, en la que se solicita —haciendo uso de la orden de Linux `curl` e indicando que solo queremos visualizar las cabeceras de la respuesta— el recurso `corto.txt` de la web `prueba.com` albergada en nuestro servidor.

```
mobylette@lynx:~$ curl -IkH "Accept-Encoding: gzip" https://prueba.com/corto.txt
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 30 Jun 2015 05:50:17 GMT
Content-Type: text/plain
Content-Length: 28
Last-Modified: Tue, 30 Jun 2015 05:45:19 GMT
Connection: keep-alive
ETag: "55922cef-1c"
Expires: Thu, 30 Jul 2015 05:50:17 GMT
Cache-Control: max-age=2592000
Accept-Ranges: bytes

mobylette@lynx:~$
```

Figura 36. Petición indicando soporte gzip

Tal y como se aprecia en la imagen anterior, pese a que indicamos a NGINX que soportamos la compresión del contenido de las respuestas, el tipo de datos de la respuesta remitida por el servidor, indicado por la cabecera `Content-Type`, es texto plano (`text/plain`) sin ningún tipo de codificación especial. Esto se debe a que en la configuración del servicio, uno de los requisitos para comprimir los recursos es que su tamaño sea mayor de 1100Bytes, y sin embargo, en este caso el tamaño del recurso solicitado es de 28Bytes —indicado por el valor de la cabecera `Content-Length`—.

Con el fin de probar lo anterior, se ha realizado el mismo procedimiento que en el caso del recurso `corto.txt` para un fichero de texto cuyo tamaño fuese superior al mínimo indicado en la configuración del *surrogate*. En esta ocasión, el contenido de la respuesta obtenida para la petición del fichero `largo.txt`, cuya longitud es de 16383Bytes, es enviada por NGINX comprimida, tal y como indica la cabecera `Content-Encoding: gzip` de la respuesta mostrada en la *figura 37*.

```
mobylette@lynx:~$ curl -IkH "Accept-Encoding: gzip" https://prueba.com/largo.txt
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 30 Jun 2015 06:06:29 GMT
Content-Type: text/plain
Last-Modified: Tue, 30 Jun 2015 05:44:30 GMT
Connection: keep-alive
Vary: Accept-Encoding
Expires: Thu, 30 Jul 2015 06:06:29 GMT
Cache-Control: max-age=2592000
Content-Encoding: gzip

mobylette@lynx:~$
```

Figura 37. Cabeceras de respuesta comprimida en gzip

Cabe destacar que las pruebas anteriores también son válidas para los recursos generados dinámicamente por los CGI, los cuales son solicitados a Apache desde el proxy que, se encarga de comprimirlos y enviarlos al cliente. Véase la *figura 38* en la que se muestran las cabeceras de la respuesta para el recurso `ejemplo.php`.

```
mobylette@lynx:~$ curl -IkH "Accept-Encoding: gzip" https://prueba.com/ejemplo.php
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 30 Jun 2015 06:24:47 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.3.3
Content-Encoding: gzip

mobylette@lynx:~$
```

Figura 38. Cabeceras de respuesta comprimida para ejemplo.php

5.3. Cache NGINX

El siguiente grupo de pruebas realizadas estaban encaminadas a probar el comportamiento de la cache de NGINX. Para ello, se han realizado dos solicitudes (*figuras 39 y 40*) para cada recurso, que era el mínimo exigido en la configuración, detallada en el apartado 4.3.5, para almacenar durante 2 minutos una entrada en la zona de cache del proxy.

```
> GET /ejemplo.php HTTP/1.1
> User-Agent: curl/7.38.0
> Host: prueba.com
> Accept: */*
>
< HTTP/1.1 200 OK
* Server nginx/1.8.0 is not blacklisted
< Server: nginx/1.8.0
< Date: Tue, 30 Jun 2015 22:50:19 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept-Encoding
< X-Powered-By: PHP/5.3.3
< X-Cache: MISS
<
{ [data not shown]
```

Figura 39. Petición 1 para ejemplo.php y fallo cache

```
> GET /ejemplo.php HTTP/1.1
> User-Agent: curl/7.38.0
> Host: prueba.com
> Accept: */*
>
< HTTP/1.1 200 OK
* Server nginx/1.8.0 is not blacklisted
< Server: nginx/1.8.0
< Date: Tue, 30 Jun 2015 22:50:27 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept-Encoding
< X-Powered-By: PHP/5.3.3
< X-Cache: MISS
<
{ [data not shown]
```

Figura 40. Petición 2 para ejemplo.php y fallo cache

Tal y como vemos en las imágenes anteriores, la cabecera `X-Cache: MISS` nos indica que la respuesta no ha sido obtenida de cache. Esta cabecera se ha añadido para hacer mas visible el correcto funcionamiento de la cache de NGINX en las pruebas realizadas.

Una vez llevadas a cabo éstas solicitudes —que han sido registradas en el fichero de log de Apache y NGINX—, procedemos a observar el contenido del directorio del disco duro en el que NGINX gestiona el almacenamiento de recursos de forma temporal. Véase la *figura 41*.

```
[root@mv203 ~]# cat /var/log/virtualmin/prueba.com_apache_access_log

192.168.15.1 - - [01/Jul/2015:00:50:19 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:50:27 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
[root@mv203 ~]# cat /var/log/virtualmin/prueba.com_access_log

192.168.15.1 - - [01/Jul/2015:00:50:19 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:50:27 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
[root@mv203 ~]# ls -R /var/nginx/cache/
/var/nginx/cache/:
5b

/var/nginx/cache/5b:
e0

/var/nginx/cache/5b/e0:
26a9fd980f6f5ea0171b8164860c
[root@mv203 ~]#
```

Figura 41. Logs de Apache y NGINX y directorio de cache

Después de comprobar que en la segunda petición, la respuesta es almacenada por NGINX en su directorio de cache, realizamos una tercera petición para el mismo recurso. Esta vez, la solicitud sólo se ha visto reflejada en el log del *surrogate*, ya que tiene una copia en su zona de memoria temporal. Véase la *figura 42*. Además, tal y como puede verse en la *figura 43*, la cabecera `X-Cache` nos indica efectivamente que ha habido un acierto y se ha encontrado el recurso en memoria temporal.

```
[root@mv203 ~]# cat /var/log/virtualmin/prueba.com_access_log

192.168.15.1 - - [01/Jul/2015:00:50:19 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:50:27 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:51:11 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
[root@mv203 ~]# cat /var/log/virtualmin/prueba.com_apache_access_log

192.168.15.1 - - [01/Jul/2015:00:50:19 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:50:27 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
[root@mv203 ~]#
```

Figura 42. Logs de NGINX y Apache para acierto de cache

```
> GET /ejemplo.php HTTP/1.1
> User-Agent: curl/7.38.0
> Host: prueba.com
> Accept: */*
>
< HTTP/1.1 200 OK
* Server nginx/1.8.0 is not blacklisted
< Server: nginx/1.8.0
< Date: Tue, 30 Jun 2015 22:51:11 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept-Encoding
< X-Powered-By: PHP/5.3.3
< X-Cache: HIT
<
{ [data not shown]
100 58869    0 58869    0     0  2814k      0 --:--:-- --:--:-- --:--:-- 2874k
* Connection #0 to host prueba.com left intact
mobylette@lynx:~$
```

Figura 43. Petición con acierto de cache para ejemplo.php

Transcurridos los dos minutos tras los que un recurso caduca en memoria temporal si no es requerido en este periodo de tiempo, volvemos a realizar la misma petición y observamos los ficheros de registro de acceso de ambos servicios, corroborando que ésta vez y como puede verse en la *figura 44* —y puesto que la copia en cache del *surrogate* ha caducado— la petición si se ve reflejada en ambos ficheros.

```
[root@mv203 ~]# cat /var/log/virtualmin/prueba.com_access_log

192.168.15.1 - - [01/Jul/2015:00:50:19 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:50:27 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:51:11 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:01:33:10 +0200] "GET /ejemplo.php HTTP/1.1" 200 58882 "-" "curl/7.38.0"
[root@mv203 ~]# cat /var/log/virtualmin/prueba.com_apache_access_log

192.168.15.1 - - [01/Jul/2015:00:50:19 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:00:50:27 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
192.168.15.1 - - [01/Jul/2015:01:33:10 +0200] "GET /ejemplo.php HTTP/1.0" 200 58869 "-" "curl/7.38.0"
[root@mv203 ~]#
```

Figura 44. Logs tras petición una vez caducada la copia en cache

6. Conclusiones

Este proyecto es un ejemplo de cómo la evolución tecnológica obliga a mejorar o adaptar las soluciones previas para cubrir las necesidades actuales, realizando nuevas partes para estos programas que interaccionen con múltiples tecnologías diferentes. La finalidad de este trabajo de final de grado, era justamente esa misma, cubrir una necesidad surgida con la evolución de las tecnologías que no se encontraba suplida por ninguna otra herramienta o módulo para el panel de control Webmin.

Se ha logrado crear un módulo de integración para Webmin, capaz de interaccionar con Apache, VirtualMin y NGINX, para mejorar las prestaciones de las web alojadas en un servidor gestionadas por VirtualMin, desplegando NGINX como proxy inverso del servidor web Apache y manteniendo la compatibilidad con el panel.

Tareas cómo analizar Apache, NGINX o Webmin, estudiando uno a uno los ficheros de VirtualMin y en los que almacena sus configuraciones, leer códigos de sus CGI a fín de desgranar el sistema y averiguar qué ficheros debíamos modificar para realizar nuestro cometido sin alterar el funcionamiento original del panel, han sido clave para la realización de este proyecto, topándonos con tecnologías y lenguajes de distinta índole como son Perl o HTML.

Esto me ha proporcionado una buena base de conocimiento a nivel académico, haciéndome enfrentar al mundo real, dónde no hay una guía o patrón concreto de actuación como en las practicas de las asignaturas, creciendo como profesional del ámbito de la informática y aprendiendo un nuevo lenguaje de programación como Python, el cuál no he tratado a lo largo de mi educación superior en esta escuela. Además, he ampliado mis conocimientos en administración de servidores web, ya sea Apache, como NGINX, hasta ahora desconocido para mi.

Por otro lado, las tareas de análisis del panel VirtualMin para comprender su funcionamiento han supuesto un reto a la hora de dar luz sobre el funcionamiento interno de un sistema complejo —no desarrollado por nosotros y no muy documentado— con la unica ayuda de la documentación para desarrollar nuevos módulos.

Un trabajo cuyo tema considero de actualidad, realista y que puede ser útil para otras personas, lo que, personalmente, me llena como profesional y persona. Pese a esto, y debido al tiempo limitado del que se dispone, queda mucho por implementar, pero he aquí mi pequeño grano de arena al mundo opensource.

6.1. Trabajo futuro

Las posibles ampliaciones que pueden ser desarrolladas de manera posterior al trabajo realizado en este proyecto engloban aspectos como la ampliación del módulo para Webmin, haciendo posible que desde un formulario se puedan modificar los parámetros de configuración específicos para cada uno de los servidores virtuales en el proxy, ya que, actualmente esta funcionalidad se ha de realizar editando directamente el fichero de configuración para cada web.

Además, otra funcionalidad que podría resultar interesante agregar, es la posibilidad de incluir plantillas de configuración por defecto, es decir, que el administrador pueda elegir en el momento de la integración entre distintos modelos de configuración para los servidores virtuales en base al tipo de web que contiene cada uno, optimizando su funcionamiento, puesto que, actualmente, los servidores virtuales se crean con una configuración genérica estándar partiendo del análisis detallado en el Anexo 1.

Otra característica importante con respecto al trabajo futuro que se desee aportar, es la adaptación del módulo a las nuevas versiones que vayan surgiendo tanto del sistema operativo CentOS, como los servidores Apache o NGINX, o el propio panel de control VirtualMin. Estas versiones futuras, en muchos casos ya se encuentran en desarrollo, como por ejemplo la versión de VirtualMin totalmente compatible con Apache HTTP Server 2.4.

7. Referencias

[1] Centos.org. CentOS wiki. <http://wiki.centos.org/es/Download>[Consultado:24/06/2015]

[2] Dierks, T. y Allen, C. The TLS Protocol Version 1.0. <http://www.ietf.org/rfc/rfc2246.txt>[Consulta: 04/04/2015]

[3] Fielding, R. ; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P. y Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. <http://tools.ietf.org/html/rfc2616>[Consulta: 27/03/2015]

[4] Fjordvald, Martin (2015). Optimizing Nginx for High Traffic Loads. <https://blog.martinfjordvald.com/2011/04/optimizing-nginx-for-high-traffic-loads/>[Consulta: 2 de Junio de 2015]

[5] Gite, Vivek (2010). Top 20 Nginx WebServer Best Security Practices. <http://www.cyberciti.biz/tips/linux-unix-bsd-nginx-webserver-security.html> [Consulta: 3 de Junio de 2015]

[6] Gite, Vivek (2013). HowTo: Create a Self-Signed SSL Certificate on Nginx For CentOS / RHEL. <http://www.cyberciti.biz/faq/nginx-self-signed-certificate-tutorial-on-centos-redhat-linux/>[Consulta: 29 de Junio de 2015]

[7] Hodges, J.; Jackson, C. y Barth, A. (2012). HTTP Strict Transport Security. <http://tools.ietf.org/html/rfc6797>[Consulta: 1 de Junio de 2015]

[8] International Organization for Standardization. Language codes - ISO 639. <http://www.iso.org/iso/home/standards/language_codes.htm>[Consulta: 8 de Junio de 2015]

[9] Kavon, Alex (2014). How To Optimize Nginx Configuration. <https://www.digitalocean.com/community/tutorials/how-to-optimize-nginx-configuration>[Consulta: 2 de Junio de 2015]

[10] Kumar Pandey, Atul. Nginx vs Apache : Selection of Web Servers. <http://atulhost.com/nginx-vs-apache>[Consulta: 26 de Marzo de 2015]

[11] Kurose, James F. y Ross, Keith w (2010). Redes de computadoras. Un enfoque descendente (5a ed). Madrid: Pearson.

[12] Lemburg, Marc-André. The Python Wiki. <https://wiki.python.org/moin/>[Consulta: 27 de Marzo de 2015]

[13] Lemburg, Marc-André. Python2orPython3. <https://wiki.python.org/moin/Python2orPython3>[Consulta: 27 de Marzo de 2015]

[14] Nagle, John (1984). Congestion Control in IP/TCP Internetworks. <http://tools.ietf.org/html/rfc896>[Consulta: 2 de Junio de 2015]

[15] Nedelcu, Clément (2013). Nginx HTTP Server (2nd Edition). Birmingham: Packt Publishing.

[16] Netcraft. February 2015 Web Server Survey. <http://news.netcraft.com/archives/2015/02/24/february-2015-web-server-survey.html>[Consulta: 26 de Marzo de 2015]

[17] NGINX, Inc. NGINX Reverse Proxy. <http://nginx.com/resources/admin-guide/reverse-proxy/>[Consulta: 2 de Junio de 2015]

[18] NGINX, Inc. NGINX Content Caching. <http://nginx.com/resources/admin-guide/caching/>[Consulta: 2 de Junio de 2015]

[19] Nginx.org. Configuration file measurement units. <http://nginx.org/en/docs/syntax.html>[Consulta: 26 de Marzo de 2015]

[20] Nginx.org. Core functionality. <http://nginx.org/en/docs/ngx_core_module.html>[Consulta: 30 de Mayo de 2015]

[21] Nginx.org. Nginx HTTP Core Module. <http://nginx.org/en/docs/http/ngx_http_core_module.html>[Consulta: 30 de Mayo de 2015]

[22] Nginx.org. Nginx HTTP Proxy Module. <http://nginx.org/en/docs/http/ngx_http_proxy_module.html>[Consulta: 2 de Junio de 2015]

[23] Nginx.org. Nginx HTTP SSL Module. <http://nginx.org/en/docs/http/ngx_http_ssl_module.html>[Consulta: 2 de Junio de 2015]

[24] Nginx.org. Configuring HTTPS servers. <http://nginx.org/en/docs/http/configuring_https_servers.html>[Consulta: 2 de Junio de 2015]

[25] Nginx.org. Nginx about page. <http://nginx.org/en/>[Consulta: 26 de Marzo de 2015]

[26] Nginx.org. Nginx spanish wiki. <http://wiki.nginx.org/NginxEs>[Consulta: 26 de Marzo de 2015]

[27] Open Source Initiative. The BSD 2-Clause License. <http://opensource.org/licenses/BSD-2-Clause>[Consulta: 27 de Marzo de 2015]

[28] Python Software Foundation. General Python FAQ. <https://docs.python.org/2/faq/general.html#what-is-python>[Consulta: 27 de Marzo de 2015]

[29] Rivest, R. (1992). The MD5 Message-Digest Algorithm. <http://tools.ietf.org/html/rfc1321>[Consulta: 26 de Marzo de 2015]

[30] SSLSHOPER.COM (2010). How to Create and Install an Apache Self Signed Certificate <https://www.sslshopper.com/article-how-to-create-and-install-an-apache-self-signed-certificate.html>[Consulta: 29/06/2015]

[31] STALLINGS, WILLIAM (2007). Data and computer communications (8ed). USA: Pearson Education.

[32] THE APACHE SOFTWARE FOUNDATION. Licenses. <http://www.apache.org/licenses/>[Consulta: 23 de Marzo de 2015]

[33] THE APACHE SOFTWARE FOUNDATION. Name-based Virtual Host Support. <http://httpd.apache.org/docs/2.2/vhosts/name-based.html>[Consulta: 5 de Junio de 2015]

[34] THE APACHE SOFTWARE FOUNDATION. About the Apache HTTP Server Project. <http://httpd.apache.org/ABOUT_APACHE.html>[Consulta: 23 de Marzo de 2015]

[35] THE APACHE SOFTWARE FOUNDATION. Apache HTTP Server Tutorial: .htaccess files. <http://httpd.apache.org/docs/2.2/howto/htaccess.html>[Consulta: 26 de Marzo de 2015]

[36] THE APACHE SOFTWARE FOUNDATION. Apache Virtual Host documentation. <http://httpd.apache.org/docs/2.2/vhosts/>[Consulta: 5 de Junio de 2015]

[37] THE APACHE SOFTWARE FOUNDATION. Configuration Files – Apache HTTP Server Version 2.4. <https://httpd.apache.org/docs/current/configuring.html>[Consulta: 18 de Mayo de 2015]

[38] THE FREE SOFTWARE FOUNDATION (1999). Guía de "Gnu Privacy Guard". <https://www.gnupg.org/gph/es/manual/book1.html>[Consulta: 5 de Junio de 2015]

[39] VILLAMIL, FREDERIC DE (2014). Nginx Optimization: understanding sendfile, tcp_nodelay and tcp_nopush. <https://t37.net/nginx-optimization-understanding-sendfile-tcp_nodelay-and-tcp_nopush.html>[Consulta: 2 de Junio de 2015]

[40] VIRTUALMIN, INC. Open Source Web Hosting and Cloud Control Panels | Virtualmin. <http://www.virtualmin.com/>[Consulta: 27 de Marzo de 2015]

[41] W3C CONSORTIUM. CGI - Common Gateway Interface. <http://www.w3.org/CGI/>[Consulta: 26 de Marzo de 2015]

[42] W3SCHOOLS.IN. What is HTML. <http://www.w3schools.in/html/intro/>[Consulta: 28 de Marzo de 2015]

[43] WEBMIN WIKI PAGE. Webmin Module Development. <http://doxfer.webmin.com/Webmin/Module_Development>[Consulta: 27 de Mayo de 2015]

[44] WESSELS, DUANE (2001). Web caching. Beijing: O'Reilly

Anexos

Anexo 1: Análisis estadístico

Con el fin de encontrar unos parámetros de configuración mas óptimos para NGINX, se ha diseñado una aplicación de consola en Java que analice el tamaño de los objetos que conforman las webs que lee como entrada desde un fichero de texto llamado `Webs.txt`. Para cada una de estas 15 webs que conforman la población de estudio, la aplicación se encargaba de realizar peticiones HTTP —como si de un navegador se tratase— a todos los objetos enlazados en la primera página de cada uno de los sitios, almacenando para cada petición el tamaño de la respuesta.

Estas webs fueron elegidas en función del propósito de la misma, desde webs gubernamentales, centros educativos o blogs, hasta medios de información digitales. La lista de sitios de los que se realizó el estudio es la siguiente:

- http://www.upv.es – Web de la Universidad Politécnica de Valencia.
- http://www.uclm.es – Web de la Universidad de Castilla-La Mancha
- http://www.ugr.es/ – Web de la Universidad de Granada.
- http://www.aemet.es/es/portada – Web de la Agencia Española de Meteorología.
- http://www.csic.es/ – Web del Centro Superior de Investigaciones Científicas.
- http://www.europarl.europa.eu/portal/es – Web del Parlamento Europeo.
- http://www.eldiario.es/ – Web del periódico digital eldiario.es.
- http://www.lemonde.fr – Web del periódico digital francés LEMONDE.
- http://rt.com/ – Web del canal de noticias de la Federación de Rusia.
- http://nginx.com/ – Web del software NGINX.
- http://www.w3schools.com/ – Web de educación del consorcio World Wide Web.
- http://www.debian.org/ – Web del sistema operativo de código abierto Debian.
- http://www.securityartwork.es/ – Blog de seguridad TI de la empresa S2 Grupo.
- http://securityblog.redhat.com/ – Blog de seguridad TI de los desarrolladores de RedHat Enterprise Linux.
- http://www.omicrono.com/ – Blog de tecnología.

Basándonos en los resultados de este análisis estadístico para todos los objetos analizados (en total 1511), obtuvimos la proporción de todos los recursos que se encontraban dentro de cada rango de tamaños:

Tamaño	Porcentaje	Acumulado
Hasta 4kB	0.19258769	0.19258769
De 4kB a 8kB	0.07478491	0.2673726
De 8kB a 16kB	0.112508275	0.37988088
De 16kB a 32kB	0.13765718	0.5175381
De 32kB a 64kB	0.22964925	0.7471873
De 64kB a 128kB	0.22964925	0.9768365
Mas de 128kB	0.023163468	1.0

Como se puede apreciar en la tabla de resultados anterior, la mayoría de los objetos que contienen las webs analizadas tienen un tamaño mayor de 16kB, aproximadamente el 60% de ellas, por lo que éste ha sido un dato importante en las configuraciones que siguen.

Así, el código Java que ha sido desarrollado para permitir el análisis anterior es el siguiente:

Clase Main.java

En primer lugar, la clase `main.java` se encarga de leer el fichero de texto llamado `webs.txt` que contiene las URL de los sitios web señalados anteriormente, almacenandolos en una lista. Una vez leido, se lanzan tantos hilos simultaneos de tipo `DescargaWeb` como indique la variable `numPetPa` –por defecto 7-- para descargar todo el contenido enlazado en la primera página de los sitios web (el contenido de la misma es examinado por `DescargaWeb`). Estas conexiones almacenan todos los datos en la lista llamada `resultados` para su posterior análisis estadístico, cuyos resultados serán almacenados en un fichero de texto.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;


public class Main {
	private static List<String> listawebs;
	static List<Peticion> resultados;
	static int numPetPa=7;

	//Leemos el fichero que contiene las webs a analizar
	private static void leeWebs ()
	{
		BufferedReader lector;
```

```
            File f;
            String linea="";
            FileReader fRead;
            listawebs=new ArrayList<String>();

            try {
                  f=new File("webs.txt");
                  fRead = new FileReader(f);
                  lector=new BufferedReader(fRead);
                  while ( (linea=lector.readLine())!=null ){
                        listawebs.add(linea);
                          // Realiza tantas peticiones como enlaces
tiene la web
                  }
                  lector.close();
                  fRead.close();
            } catch (Exception e) {
                  System.out.println("leeWebs - Error al leer el
fichero"+ e);
                  e.printStackTrace();
            }
      }

      private static void escribeRes ()
      {
            FileWriter fRes;
            PrintWriter escritor;
            Peticion res;
            int n4k=0, n8k=0, n16k=0, n32k=0, n64k=0, n128k=0,
nmas128k=0;
            float hasta4k=0, hasta8k=0, hasta16k=0, hasta32k=0,
hasta64k=0, hasta128k=0, masde128k=0;
            int suma=0, maximo=0, talla=0;
            try {
                  fRes = new FileWriter("Resultados.txt");
                  escritor=new PrintWriter(fRes);
                  for ( int n=0; n<resultados.size(); n++ ){
                        res=resultados.get(n);
                        talla=res.getSize();
                        if ( talla>maximo ) maximo=talla;
                        if ( talla<=4096 ) n4k++;
                        if ( talla>4096 && talla<=8192 ) n8k++;
                        if ( talla>8192 && talla<=16384 ) n16k++;
                        if ( talla>16384 && talla<=32768 ) n32k++;
                        if ( talla>32768 && talla<=65536 ) n64k++;
                        if ( talla>65536 && talla<=65536*2 ) n128k++;
                        if ( talla>65536*2 ) nmas128k++;
                        suma+=talla;
                        //escritor.println(res.getName()+"          --
"+talla);
                  }
                  hasta4k=((float)n4k/resultados.size());
                  hasta8k=((float)n8k/resultados.size());
                  hasta16k=((float)n16k/resultados.size());
                  hasta32k=((float)n32k/resultados.size());
                  hasta64k=((float)n64k/resultados.size());
```

```
                hasta128k=((float)n128k/resultados.size());
                escritor.println("Objetos
TOTALES="+resultados.size());
                escritor.println("MEDIA -> "+
(suma/resultados.size()));

                escritor.println("MAXIMO -> "+maximo);
                escritor.println("Hasta       4k="+hasta4k+"       -
Acumulado="+hasta4k);
                escritor.println("De  4k  a  8k  ->  "+hasta8k+"  -
Acumulado="+((float)(n8k+n4k)/resultados.size()));
                escritor.println("De  8k  a  16k  ->  "+hasta16k+"  -
Acumulado="+((float)(n8k+n4k+n16k)/resultados.size()));
                escritor.println("De  16k  a  32k  ->  "+hasta32k+"  -
Acumulado="+((float)(n32k+n8k+n4k+n16k)/resultados.size()));
                escritor.println("De  32k  a  64k  ->  "+hasta64k+"  -
Acumulado="+((float)(n64k+n32k+n8k+n4k+n16k)/resultados.size()));
                escritor.println("De  64k  a  128k  ->  "+hasta128k+"  -
Acumulado="+((float)
(n128k+n64k+n32k+n8k+n4k+n16k)/resultados.size()));
                escritor.println("Mas  de  128k  ->  "+masde128k+"  -
Acumulado="+((float)
(nmas128k+n128k+n64k+n32k+n8k+n4k+n16k)/resultados.size()));
                escritor.close();
                fRes.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
      }

      private static void lanzaPeticiones ()
      {
            DescargaWeb pet=null;
            DescargaWeb []peticiones=new DescargaWeb[numPetPa];
            int n=0, k=0;
            for ( n=0, k=0; n<listawebs.size(); n++, k++){
                if ( k==numPetPa ){
                      for ( k=k-1; k>=0; k--){
                            try {
                                  peticiones[k].join();  //Esperamos
a que terminen las webs de descargarse
                            } catch (InterruptedException e) {
                                  e.printStackTrace();
                            }
                      }
                }
                else {
                      pet=new DescargaWeb(listawebs.get(n));
                      pet.start(); // Lanzamos las webs de forma
paralela maximo numPetPa
                      peticiones[k]=pet;
                }
            }
            for ( k=k-1; k>=0; k--){
                try {
```

```
                        peticiones[k].join();    //Esperamos    a    que
terminen las webs restantes
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
        }

        public static void main(String[] args) {
                resultados=new ArrayList<Peticion>();
                leeWebs();
                lanzaPeticiones();
                if ( resultados.size()>0 ) escribeRes();
                System.out.println("Fin del programa");
                return;
        }
}
```

Clase main.java

Clase DescargaWeb.java

`DescargaWeb.java` es la clase encargada de crear objetos de tipo `TCPCon` para descargar los recursos que contienen cada una de las paginas leídas del fichero. Para ello, en primer lugar descarga la pagina principal de las webs y analiza su codigo en busca del contenido enlazado en el mismo, almacenando las direcciones en una lista. Posteriormente, realiza conexiones simultaneas (`TCPCon`) para descargar todos los objetos enlazados encontrados de 5 en 5 y espera a que terminen para lanzar otras nuevas.

```
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;

public class DescargaWeb extends Thread {

        TCPCon conexion=null;
        Hashtable<String,String> hash=new Hashtable<String,String>();
        List<String> lista;
        String servidor="";

        public DescargaWeb (String url)
        {
                conexion=new TCPCon(url);
                servidor=conexion.host;
        }

        public void run ()
        {
                int k=0, talla=0, n=0;
                conexion.start(); // Lanzamos la peticion para index
                examinaCodigo(); // Examinamos el codigo de index para
extraer los enlaces
                talla=lista.size();
                TCPCon hilos[]=new TCPCon[5];
                System.out.println("Hay  "+lista.size()+"  elementos  en
"+servidor);
                for(  n=0,  k=0;  n<talla;  n++,k++  )  {  //  Realizamos
peticiones para todos los enlaces
                    if (k==5){
```

```
                        System.out.println("Lanzados 5 Threads para
"+servidor);
                        for ( k=0; k<5; k++){ //Esperamos a que los 5
hilos hayan acabado
                              try {
                                    hilos[k].join();
                              } catch (InterruptedException e) {
                                    e.printStackTrace();
                              }
                        }
                        k=-1;                     // Ponemos a 0 el
contador de hilos lanzados
                  }
                  else { //lanzamos hilos mientras no llevemos 5
                        hilos[k]=new   TCPCon(lista.get(n).toString(),
false);
                        hilos[k].start(); //Lanzamos 5 hilos
                  }
            }
            System.out.println("Lanzados     "+k+"     Threads     para
"+servidor);
            for (int t=0 ; t<k; t++){ //Esperamos a los que hayan
quedado lanzados al final pero no sean 5 justos
                  try {
                        hilos[t].join();
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }
            System.out.println("Lanzados todos los Threads para
"+servidor);
      }

      // Obtiene las url de cod coincidentes con el el contenido de
busca
      private void dirObjetos(String cod, String busca) {
        String aux2 = "", ruta = "";
        String objeto = "", enlace = "";
        while (true) {
            if (cod.indexOf(busca) < 0) {
                break;
            }
             cod = cod.substring(cod.indexOf(busca) + busca.length(),
cod.length());
            if ( cod.startsWith("https://")) continue;
             aux2 = cod.substring(0, cod.indexOf("\"")); // Extraemos
enlace a enlace
            aux2=aux2.replace("#","");
            if ( aux2.contains("instagram") ) continue;
                 if (aux2.startsWith("http://")) { // No hace falta
cambiar el enlace
                  ruta=aux2;
            }
             else { // Hay que añadirle http y el servidor en el que
esta
                if ( aux2.startsWith("/") ) objeto=aux2;
                else objeto = "/" + aux2;
                ruta = "http://"+servidor+objeto;
            }
              //Almacenamos en una lista los pares de direccion y lo
que se necesita
            if ( hash.get(ruta)==null ){
                  hash.put(ruta,"");
                  lista.add(ruta);
            }
            cod = cod.substring(cod.indexOf("\"")+1);
        }
```

```
    }

    public void examinaCodigo() {
      try {
                  conexion.join();
            } catch (InterruptedException e) {
                  System.out.println("fallo en la espera");
                  e.printStackTrace();
            }
      lista=new ArrayList<String>();
      String codigo=conexion.getRes();
        dirObjetos(codigo, "src=\"");
        dirObjetos(codigo, "SRC=\"");
        dirObjetos(codigo, "rel=\"stylesheet\" href=\"");
        dirObjetos(codigo, "REL=\"stylesheet\" HREF=\"");
        dirObjetos(codigo, "href=\"");
        dirObjetos(codigo, "HREF=\"");
        hash.clear();
    }
}
```

Clase DescargaWeb.java

Clase TCPCon.java

Por otro lado, la clase `TCPCon.java` realiza las conexiones al servidor, descarga los recursos y calcula su tamaño. Además, almacena en una lista de objetos de la clase `Peticion` toda esta información para su posterior análisis estadístico en la clase `Main.java`.

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.List;

import javax.sound.sampled.AudioFormat;

public class TCPCon extends Thread {

    Socket con;
    BufferedInputStream leeSock;
    PrintWriter escSock;
    int port;
    String url="";
    String host; //host que alberga el objeto
    String objeto; //objeto
    int tamCabecera=0, tamBody=0, tamTotal=0;
    String res="";
    boolean quieroCodigo=false, disponible=false, espera=false;

    public TCPCon(String host, int puerto, String loc) {
        port = puerto;
        this.host = host;
        this.objeto = "/";
    }

    public TCPCon(String host, int puerto, String dir, String loc) {
        port = puerto;
        this.host = host;
        this.objeto = dir;
```

```
    }

    public String getRes ()
    {
      return res;
    }

    public TCPCon(String url) {
        getValuesURL(url);
        quieroCodigo=true;
    }

    public TCPCon(String url, boolean quieroCodigo) {
      //System.out.println(url);
        getValuesURL(url);
        this.quieroCodigo=quieroCodigo;
    }

    private void getValuesURL(String url) {
        int indice = -1;
        this.url=new String(url);
        url=url.replaceAll("http://", "");
        url=url.replaceAll(" ", "%20");
        indice = url.indexOf("/");
        if (indice > 0) {
              host = url.substring(0, indice); // obtiene el host al
que se va a contectar
              objeto = url.substring(indice); // obtiene el objeto a
pedir
            if (objeto.equals("")) {
                objeto = "/"; // pagina index
            }
        }
        else {
            host=url;
            objeto="/";
        }
        port=80;
    }

    public boolean bajaCodigo() throws Exception
    {
        byte[] cbuf = new byte[512];
        int leidos=0;
        String aux;
        disponible=false;
        //System.out.println("Accediendo a -> "+host+" - "+objeto);
        escSock.print("GET " + objeto + " HTTP/1.1\n");
        escSock.print("Host:"+host+"\n");
        escSock.print("Connection: close\n");
        escSock.print("User-Agent: BrowserAnalizer/1.0\n\n");
        escSock.flush();
        tamTotal=0;
        do {
            leidos=leeSock.read(cbuf,0,512);
            tamTotal+=leidos;
            if ( leidos==-1 ) break;
            if (quieroCodigo){ // Si en el constructor le hemos dicho
que leeremos codigo, lo pasa a string
                  aux=new String(cbuf,0,leidos);
                  res+=""+aux;
            }

        } while ( true );
        leeSock.close();
```

```
            escSock.close();
            return true;
        }

        public void conecta() {
            try {
                con = new Socket(host, port);
                leeSock = new BufferedInputStream(con.getInputStream());
                escSock = new PrintWriter(con.getOutputStream());
            }
            catch (Exception e) {
                System.err.println("No se ha podido conectar con el Host.
Compruebe su conexion.");
            }
        }

        public void desconecta() {
            try {
                con.close();
            }
            catch (Exception e) {
                System.err.println("No hay conexion");
            }
        }

        private synchronized void agregaStats ()
        {
          Main.resultados.add(new Peticion(url, tamTotal));
        }

        @Override
        public void run() {
            try {
                conecta();
                if (bajaCodigo()) {
                    agregaStats(); // agregamos el resultado de descarga
                }
                desconecta();
            } catch (Exception ex) {
                System.err.println(ex+"->"+host+"->"+objeto);
            }
        }
}
```

Clase TCPCon.java

Clase Peticion.java

Por último, la clase `Peticion.java` es utilizada para almacenar la URL y el tamaño de cada uno de los objetos que se descargan para cada una de las webs.

```
public class Peticion {
	String name;
	int size=0;

	public Peticion(String url, int tamTotal) {
		name=url;
		size=tamTotal;
	}

	public void set (String name, int size)
	{
		this.name=name;
		this.size=size;
	}

	public int getSize ()
	{
		return size;
	}

	public String getName ()
	{
		return name;
	}

}
```

Clase Peticion.java

Anexo 2: Análisis compresion GZIP

El motivo de éste análisis era optimizar el proceso de compresión del contenido de las respuestas por parte de NGINX de forma que se obtuviese un equilibrio entre consumo de recursos del servidor —principalmente CPU— y el tamaño del resultado. Para ello, se ha realizado la compresión de un mismo fichero[13] con los distintos niveles de compresión del programa GZIP.

Así, el código del script de shell que realizaba este proceso, es el siguiente:

```
#!/bin/bash

for n in 1 2 3 4 5 6 7 8 9
do
      echo "$n" >> res
      { time gzip -$n -k rfc.txt ; } &>> res
      echo `ls -l rfc.txt.gz` >> res
      echo "---------------------------------" >> res
      rm -r rfc.txt.gz
done
```

A partir del cual se han extraído los siguientes resultados:

Nivel de compresión	Tamaño (Bytes)	Tiempo (ms)
0 – Sin compresión	422.279	0
1	144.176	14
2	137.416	15
3	131.639	22
4	124.022	17
5	119.345	24
6	117.010	33
7	116.613	39
8	116.309	58
9	116.331	75

Como se puede ver en la tabla anterior, a partir del nivel 3 la compresión no es muy grande con respecto al tiempo de ejecución que conlleva el proceso, por lo que, un nivel 2 o 3 es aceptable.

13 Fichero utilizado (RFC2616) https://www.rfc-editor.org/rfc/rfc2616.txt

Anexo 3. Generación de clave RSA y certificado autofirmado.

Tal y como se ha mencionado de forma previa en este trabajo, para poder hacer uso del mecanismo HSTS, se hace necesario disponer de un certificado digital que autentifique la identidad del servidor y que nos sirva para intercambiar las claves de la sesión SSL de forma segura. Para ello, haciendo uso de las utilidades que nos proporciona la suite de cifrado `OpenSSL`, disponemos de dos formas para conseguir este objetivo.

Mediante el primer método, generaremos nuestro propio certificado digital autofirmado de manera simple con la introducción del siguiente comando en un interprete de nuestro servidor. Este comando genera una clave RSA privada de 2048bits de forma aleatoria —y sin cifrar— con la que realiza la firma del certificado que, tendrá un tiempo de validez de un año y contendrá la información solicitada durante su ejecución.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
mysitename.key -out mysitename.cert
```

La ejecución de la orden anterior, tal y como se puede ver en la *figura A3-1*, nos solicitará la introducción de datos acerca de nuestro sitio web, los cuales estarán contenidos en el certificado y aportarán información al usuario en el momento de la aceptación del certificado.

```
[root@mv203 ~]# openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout ssl.key -out ssl.crt
Generating a 2048 bit RSA private key
.....................+++
.....................................................................................+++
writing new private key to 'ssl.key'
-----
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) [XX]:ES
State or Province Name (full name) []:Valencia
Locality Name (eg, city) [Default City]:Valencia
Organization Name (eg, company) [Default Company Ltd]:prueba
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:prueba.com
Email Address []:admin@prueba.com
[root@mv203 ~]#
```

Figura A3-1. Generación certificado autofirmado sin clave cifrada

En cambio, haciendo uso del segundo método, en caso de desear obtener una clave cifrada, evitando así, la lectura de la clave privada desde el fichero `.key` en texto plano, obtenemos nuestra clave privada RSA cifrada con la que firmamos el certificado. Este metodo se diferencia del anterior en que es necesario descifrar la clave privada cada vez que el servicio que hace uso de ella arranca —en este caso NGINX—, tal y como puede verse en la *figura A3-2*. El comando de OpenSSL necesario para generar esta clave de 2048bits es el siguiente:

```
openssl genrsa -des3 -out ssl.key 2048
```

```
[root@mv203 cache]# service nginx restart
Enter PEM pass phrase:
nginx: [warn] 2048 worker_connections exceed open file resource limit: 1024
Parando nginx:                                             [  OK  ]
Iniciando nginx: Enter PEM pass phrase:
nginx: [warn] 2048 worker_connections exceed open file resource limit: 1024
                                                           [  OK  ]
[root@mv203 cache]#
```

Figura A3-2. Solicitud de contraseña al arrancar NGINX

El siguiente paso despues de generar la clave privada es crear la petición de firma de certificado con extensión `.csr` (de las siglas en ingles de *Certificate Signing Request)*. Para realizar ésto, basta con introducir el siguiente comando:

```
openssl req -new -key ssl.key -out ssl.csr
```

Durante la ejecución de la orde anterior, nos serán solicitados los datos relativos a nuestro sitio web, que se incluirán en el certificado permitiendo que el usario o cualquier entidad pueda comprobar a quién pertenece el mismo.

Una vez hecho esto, el siguiente paso es firmar el `.csr`, por lo que, obtenemos el certificado firmado `.cert`. Basta con ejecutar la siguiente orden durante la cuál se nos solicita la contraseña de cifrado de la clave privada RSA:

```
openssl x509 -req -days 365 -in ssl.csr -signkey ssl.key -out
ssl.cert
```

Todo el proceso anterior puede verse de manera resumida en la figura A3-3.

```
[root@mv203 cache]# openssl genrsa -des3 -out ssl.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
.......+++
e is 65537 (0x10001)
Enter pass phrase for ssl.key:
Verifying - Enter pass phrase for ssl.key:
[root@mv203 cache]# openssl req -new -key ssl.key -out ssl.csr
Enter pass phrase for ssl.key:
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) [XX]:ES
State or Province Name (full name) []:Valencia
Locality Name (eg, city) [Default City]:Valencia
Organization Name (eg, company) [Default Company Ltd]:prueba.com
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:prueba.com
Email Address []:admin@prueba.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@mv203 cache]# openssl x509 -req -days 365 -in ssl.csr -signkey ssl.key -out ssl.cert
Signature ok
subject=/C=ES/ST=Valencia/L=Valencia/O=prueba.com/OU=IT/CN=prueba.com/emailAddress=admin@prueba.com
Getting Private key
Enter pass phrase for ssl.key:
[root@mv203 cache]# ls
ssl.cert  ssl.csr  ssl.key
```

Figura A3-3. Proceso creación clave cifrada y certificado

Cabe destacar que en caso de que no se desee indicar la clave cada vez que el servicio que hace uso del certificado lo requiere, es posible realizar la firma del certificado despues de generar el fichero que contenga la clave privada descifrada. Esto se realiza ejecutando el siguiente comando:

```
openssl rsa -in sslcifrado.key -out ssl.key
```

Una vez obtenidos tanto el certificado como la clave, basta con mover los ficheros al directorio raiz del servidor virtual para el que se generan —por defecto `/home/nombreservidor/`— y reiniciar NGINX.