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 Varnish Cache con un panel de control Virtualmin para crear una plataforma de servicios de hospedaje Web.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pablo Álvarez Baeza

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 webs, *opensource*, 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 el proxy inverso Varnish Cache de forma que se pueda gestionar desde el panel de control Virtualmin. El desarrollo se ha realizado en máquinas virtuales para tener mayor flexibilidad. Inicialmente, se han realizado configuraciones manuales, se han afinado los parámetros de configuración y posteriormente se han diseñado scripts de automatización. Se ha utilizado el S.O. Linux con distribución CENTOS.

Palabras clave: proxy inverso, Varnish Cache, panel de control, Virtualmin, hospedaje Web, Apache, CentOS.

Abstract

The usage of an inverse proxy on a server, usually leads to an improvement on the server performance and a reduction of the server load, while keeping system's compatibility with the most popular web serves such as Apache. Virtualmin GPL is a web hosting, opensource, control panel for Linux and UNIX systems, which allows to automatically configure every web hosting-related resource on a system based on Apache web server. The objective of this project is to install this software on a server, and integrate it with Varnish Cache reverse proxy, giving the administrator the tools for controlling everything directly from Virtualmin. We will be using virtual machines in order to improve flexibility in the development process. This project will be developed in CentOS Linux distribution.

Keywords : Varnish cache, Apache, virtual server, Python, Virtualmin, Webmin, integration, reverse proxy.

Índice

1. Introducción

De un sitio web en 1991, a mil millones de webs activas. Es la cifra máxima que, según NetCraft[1], se alcanzó en octubre del pasado año 2014. Desde entonces la cifra se ha mantenido estable, bajando la cuenta a más de 950 millones debido a las diferentes fluctuaciones de las estadísticas producidas por sitios web inactivos. Sin embargo se estima que la cifra vuelva a alcanzar ese pico y lo supere de manera estable a lo largo del próximo año.

Y es que las páginas web, desde su aparición a los principios de los años 90, han conseguido entrar de lleno en nuestro día a día ofreciendo soluciones en prácticamente todos los campos imaginables. Hoy en día es realmente complicado encontrar a un ciudadano medio que no sea capaz de citar una web de noticias, de música, de interacción social o de una empresa, incluso aunque el campo de negocio de ésta sea ajeno al mundo de las tecnologías de la información.

Y detrás de cada sitio web hay uno o varios servidores web ofreciendo todo tipo de contenidos las 24 horas al día los 365 días al año. Estos servidores web ejecutan una aplicación también denominada servidor web que se encargan de atender y procesar las peticiones que le llegan desde Internet a su dirección IP.

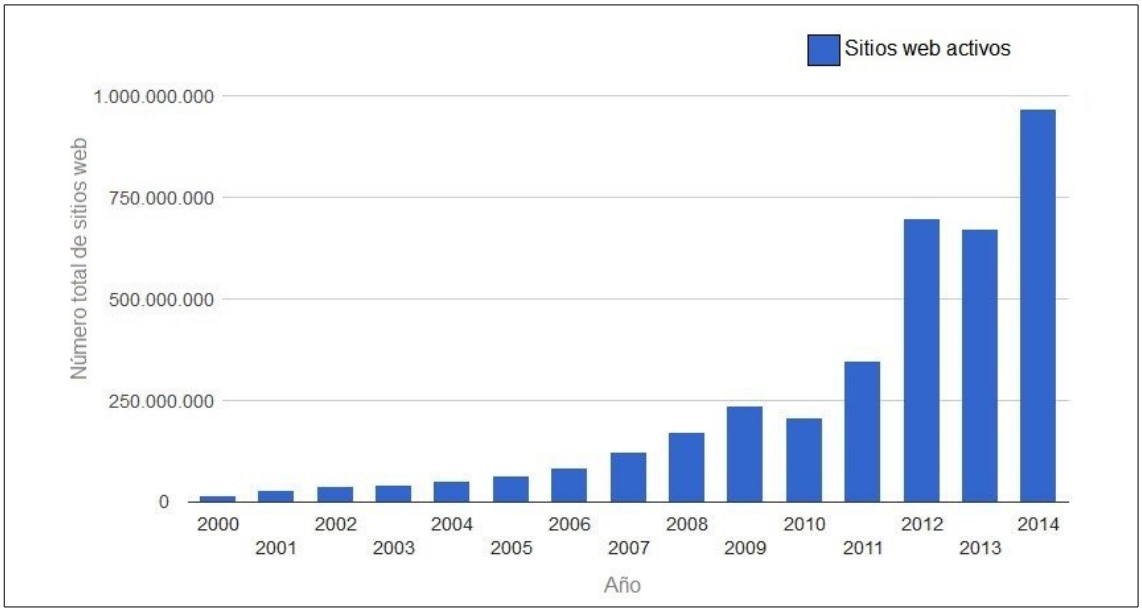

Ilustración 1: Gráfica de sitios web en el mundo según Netcraft.

El trabajo de estas aplicaciones a grandes rasgos es entender la petición, buscar en el disco el objeto que se está solicitando y enviarlo a través de la red. Es en este punto donde aparecen los programas conocidos como proxies inversos. Estos programas se colocan entre el servidor web e Internet, recibiendo todas las peticiones dirigidas al servidor. Almacenan la petición y la reenvían al servidor web, el cual le remitirá la respuesta correspondiente. Entonces, el proxy inverso almacena el par petición-respuesta y mantiene la información en cache. De esta forma, si vuelve a llegar al servidor la misma petición, en cuanto la intercepte el proxy, será él el que

sirva la respuesta desde la cache, consiguiendo un menor tiempo de respuesta y descargando al servidor web de tener que procesar todas las respuestas iguales.

En el mercado pueden encontrarse gran variedad de proxys inversos, siendo los más populares Nginx y Varnish Cache. Es este último sobre el cual trabajamos en este proyecto.

También existen numerosas soluciones software para servidores web, pero sin duda la más popular es Apache. Según la encuesta mensual realizada por Netcraft[2], en junio de 2015, aproximadamente un 39% de los servidores del mundo ejecutan Apache. Por ese motivo hemos basado el desarrollo de este proyecto en esa plataforma.

A su vez, para permitir a los administradores de los servidores web las diferentes tareas de gestión, control y administración de sus máquinas, se han desarrollado herramientas que facilitan la realización de estas tareas a distancia, utilizando para ello una interfaz web gráfica e intuitiva. Estas herramientas se denominan comúnmente "paneles de control" y su uso está muy extendido entre los administradores de servidores web. Existen numerosos paneles de control en el mercado, pero el que más nos interesa es Virtualmin, que es un panel desarrollado bajo licencia GPL (*General Public License).*

Sin embargo, todos los paneles de control flaquean cuando se intenta buscar una solución total que consiga integrar, de forma sencilla para el administrador, todas las herramientas descritas previamente. Ésta es la necesidad a la que hemos dado solución en este proyecto.

1.1. Objetivo

El objetivo de este proyecto es desarrollar un modulo para el panel de control Virtualmin, que automatice la integración del proxy inverso Varnish Cache sobre el servidor web Apache. Esto debe lograrse sin necesidad de que el administrador tenga que modificar previamente ninguna configuración de su sistema actual. La integración de toda la solución software debe realizarse de manera que no interfiera con los diferentes servicios que ya estén en ejecución en el servidor.

El módulo desarrollado debe permitir también una administración sencilla del proxy inverso Varnish, para poder configurarlo según las necesidades concretas de cada sistema sobre el que se monte.

Este desarrollo busca ofrecer una solución lo más genérica posible, aplicando una configuración por defecto, que resulte en una mejora de rendimiento y de tiempos de respuesta del servidor web para la mayoría de sistemas actuales sin necesidad de que el administrador tenga que afinar a mano configuraciones más personalizadas. Es decir, todo debe funcionar correctamente "*out of the box"* en la mayoría de sistemas que están funcionando actualmente y que usan el software concreto mencionado.

Esta generalidad de la solución desarrollada es un arma de doble filo. Por un lado, al desarrollar una solución lo más genérica posible, no es necesario afinar al máximo todas las configuraciones de software posibles en un sistema concreto, lo cual elimina algo de complejidad al desarrollo. Pero por otro lado, hemos tenido que decidir cuales son las configuraciones que abarcan el mayor numero de sistemas actuales, quedando excluidos algunos casos concretos de servidores web que no sufrirían mejora alguna al integrar nuestra solución.

2. Entorno de trabajo

2.1. Sistema operativo CentOS

Nacido en 2004, el sistema operativo CentOS (*Community Enterprise Operating System)*, es una distribución Linux creada a partir del código fuente del sistema operativo de pago *Red Hat Enterprise Linux (*RHEL en adelante), propiedad de la empresa *Red Hat Inc.* La diferencia principal está en los paquetes de software propietario que se sustituyen por alternativas *opensource.* Por este motivo, CentOS es funcionalmente compatible con la distribución RHEL, gratis y con libre redistribución.

El ciclo de vida de las distribuciones de CentOS va a la par que el ciclo de vida de las distribuciones de RHEL. De esta forma, la última versión disponible, la versión 7, tendrá soporte hasta 2024. Sin embargo, ésta aún es una versión muy joven y todavía tiene algunas incompatibilidades.

Por ese motivo, para el desarrollo de este trabajo se ha elegido utilizar la versión 6.6 de CentOS, que es una versión más madura. Tuvo su lanzamiento en julio de 2011, y oficialmente mantendrá ese soporte hasta Noviembre de 2020. De esta forma, durante los próximos 5 años tendremos un sistema operativo seguro y robusto, ampliamente usado en servidores web, con lo que nos aseguramos que el módulo desarrollado puede llegar a la mayor cantidad de sistemas con garantías de compatibilidad.

Tal y como se ha mencionado anteriormente, este sistema operativo será desplegado en una máquina virtual, lo que nos ofrecerá agilidad y flexibilidad de trabajo.

2.2. Lenguaje Python

La elección de Python como lenguaje de programación del módulo para Virtualmin se ha basado en el análisis de los diferentes módulos desarrollados previamente por la comunidad. La mayoría estaban desarrollados en Perl, pero una parte importante habían sido desarrollados en Python. La curva de aprendizaje de Python lo convierte en nuestra preferencia, pues aprender Perl nos consumiría más tiempo aportándonos los mismos o similares resultados.

Python es un lenguaje de programación interpretado, lo que significa que los fuentes son traducidos a lenguaje máquina en tiempo de ejecución, en lugar de compilarse y generar un ejecutable. Esta característica ofrece a Python la cualidad de tener una ejecución correcta independientemente del procesador en el que se ejecute, ya que no hay que crear código para una máquina en concreto, si no que funcionará igual en cualquier ordenador que pueda interpretar el lenguaje.

Otra de las características de Python que que es un lenguaje multiparadigma, soportando programación imperativa, programación orientada a objetos e incluso programación funcional. Es un lenguaje con tipado dinámico, lo que le confiere más flexibilidad a la hora de crear programas que la que ofrecería un lenguaje con tipado estático.

La historia del lenguaje de programación Python se remonta a finales de los años 80, y su implementación comenzó en diciembre de 1989 cuando en navidad, *Guido Van Rossum* que trabajaba en un centro de investigación holandés que, entre otras cosas, actualmente alberga la oficina central del W3C, decidió empezar el proyecto como un pasatiempo dándole continuidad al lenguaje de programación ABC de cuyo desarrollo había formado parte. Dicho lenguaje se enfocaba en ser fácil de usar y de aprender, consiguiendo al mismo tiempo un rendimiento ejemplar. Pero el hardware disponible en la época de su creación hacía difícil su uso y el proyecto no trascendió como se esperaba. *Van Rossum* es por tanto el autor principal de Python, y continúa ejerciendo un papel central, encabezando la junta directiva del lenguaje, en la comunidad de Python.

La versión más reciente del lenguaje, lanzada en 2014, es la 3.4.3. Sin embargo, para todos los desarrollos realizados en este lenguaje se ha utilizado la versión 2.7. Esto es debido a que la distribución CentOS solo ofrece soporte oficial para Python en sus versiones *2.x*. Además, las versiones de Python *3.x* no tienen garantías de cumplir retro-compatibilidad con las versiones 2.x, lo que hace que no se pueda asegurar que un programa desarrollado en Python 2, funcione correctamente al ejecutarlo en un interprete de la última versión.

2.3. Servidor web Apache

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.12 y el concepto de servidor virtual. Es capaz de servir tanto páginas estáticas HTML como también generar contenido dinámico, por ejemplo una página con código PHP, suponiendo este último caso una carga extra para la máquina que lo ejecuta.

Cuando comenzó su desarrollo en 1995, se basó inicialmente en el código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo. El servidor se desarrolla dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation.

Apache goza de una amplia aceptación en la red. Desde 1996, Apache, es el servidor HTTP más usado, alcanzando su máxima cuota de mercado en 2005, siendo el servidor empleado en el 70% de los sitios web en el mundo, sin embargo ha sufrido un descenso en su cuota de mercado en los últimos años.

A pesar de ese descenso, según los datos recogidos por *BuitWith*[30] y tal y como puede apreciarse en la figura 2, Apache sigue siendo líder, ejecutándose en un 36% de todos los servidores web del mundo.

El servidor web Apache está desarrollado con una arquitectura modular, lo que permite al administrador adaptarlo a sus necesidades concretas de una forma sencilla añadiendo, eliminando o incluso creando los diferentes módulos que considere, pues ofrece un kit de desarrollo con el que simplifica la tarea de programar alguna funcionalidad concreta que sea necesaria.

Gracias a esta modularidad y a su extensa aceptación en los servidores web del mundo, Apache goza de una comunidad muy activa que corrige continuamente los diferentes bugs o agujeros de seguridad que se van descubriendo

La licencia de software bajo la cual el software de la fundación Apache es distribuido es una parte distintiva de la historia de *Apache HTTP Server* y de la comunidad de código abierto. La Licencia Apache permite la distribución de derivados de código abierto y cerrado a partir de su código fuente original.

En febrero del año 2012, se lanzó al publico la versión la versión 2.4 del servidor, sin embargo, la versión del software que hemos utilizado para desarrollar este proyecto es la 2.2 por cuestiones compatibilidad. Más en concreto, es el panel Virtualmin el que no soporta Apache en su última versión.

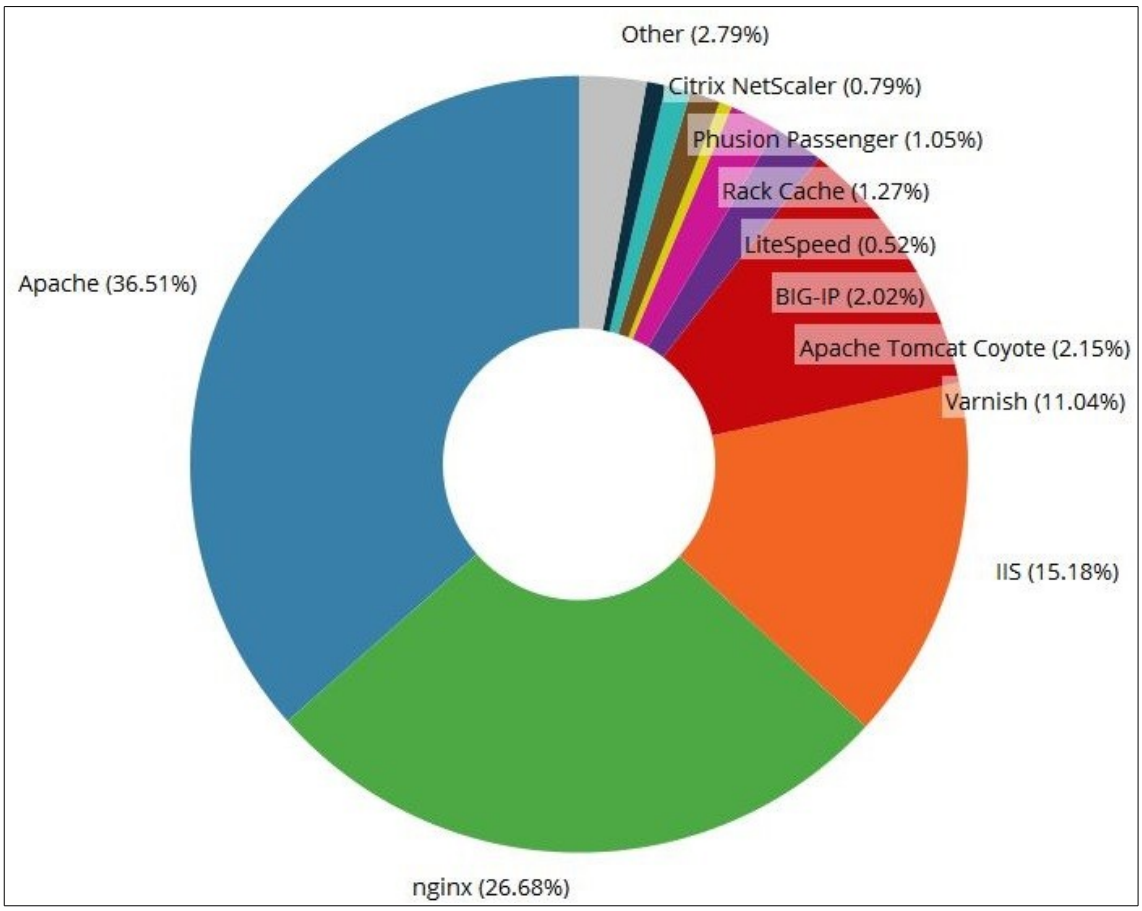

Ilustración 2: Gráfico que ilustra la distribución del mercado de servidores web según BuitWith.

Es de obligada mención el fichero de configuración de Apache, pues es ampliamente usado durante el desarrollo del proyecto. Este fichero se encenta en */etc/httpd/conf/httpd.conf* y contiene toda la información de directivas de configuración del servidor y sus servicios, así como la configuración de los servidores virtuales que tuviera el sistema.

2.4. Panel Virtualmin

Virtualmin es un panel de gestión de hosting web para sistemas UNIX. Está basado en el panel opensource de control de sistemas, Webmin. Añade un amplio abanico de funcionalidades, entre las que destacan la administración de hosts virtuales, gestión de bandejas de entrada de correo, control de bases de datos, aplicaciones web y en general permite realizar las tareas de administración de servidores desde una interfaz gráfica intuitiva.

Existe una versión gratuita y otra de pago denominada Virtualmin Proffessional. Las diferencias entre ambas versiones no son significativas para el usuario medio, siendo la más significativa que la versión de pago tiene soporte continuo al cliente mediante un sistema de tickets.[11]

Webmin es una herramienta con licencia BSD que permite gestionar máquinas basadas en unix. Con él se pueden configurar aspectos internos de muchos sistemas operativos, como usuarios, cuotas de espacio, servicios, archivos de configuración, apagado del equipo, etcétera, así como modificar y controlar muchas aplicaciones libres, como el servidor web Apache, PHP, MySQL, DNS, Samba, DHCP, entre otros.

Todas estas utilidades que cubre el panel se ofrecen al administrador de forma remota a través de una web. Gracias a esto, la persona encargada de gestionar la máquina podrá prescindir de la linea de comandos en la mayoría de escenarios típicos de administración, utilizando una interfaz gráfica intuitiva.

Webmin está escrito en Perl, versión 5, ejecutándose como su propio proceso y servidor web. Por defecto se comunica mediante TCP a través del puerto 10000, y puede ser configurado para usar SSL si OpenSSL está instalado con módulos de Perl adicionales requeridos.

Está construido a partir de módulos, los cuales tienen una interfaz a los archivos de configuración y el servidor Webmin. Esto hace fácil la adición de nuevas funcionalidades sin mucho esfuerzo. Debido al diseño modular de Webmin, es posible para cualquier interesado escribir extensiones para configuración de escritorio.

2.5. Varnish Cache

Varnish Cache es un acelerador de aplicaciones web, también conocido proxy HTTP inverso. Se instala delante de cualquier servidor HTTP y se configura para almacenar en la cache del servidor una copia del recurso solicitado. Está ideado para aumentar el rendimiento de aplicaciones web con contenidos pesados y APIs o recursos altamente consumidos.

Es muy importante destacar que Varnish no es capaz de trabajar con HTTPS, por eso debemos tener este hecho en cuenta a la hora de diseñar la integración, para ser capaces de dejar intacto cualquier servicio que estuviera corriendo previamente por el puerto de HTTPS, es decir, el 443.

La forma de conseguir ese aumento de rendimiento, a grandes rasgos, es almacenando un par clave-valor en memoria que relaciona la petición atendida con el objeto de respuesta. Los objetos de almacenan en memoria, y las referencias estos objetos se guardan en un árbol hash. También es destacable la característica multi-hilo del programa, que ejecuta un proceso por cada petición de forma independiente, de este modo tiene una gran escalabilidad en sistemas multi-nucleo.

Actualmente se utiliza en una gran cantidad de sitios web de alta demanda y trafico como *Wikipedia, The New York Times, The Guardian, The Hindu* y redes sociales como *Facebook, Twitter, Vimeo, Tumblr,* entre otros.

El proxy inverso Varnish es un proyecto *opensource* y gratuito. El proceso de desarrollo es publico, y cualquiera puede subir parches o estudiar el código sí hay alguna duda de cómo trabaja cualquier aspecto concreto del software. También cabe destacar que se desarrolla y se prueba en GNU/Linux y FreeBSD. Su código base se desarrolla con la intención de mantenerlo lo más autocontenido posible para minimizar el uso de librerías externas.

Los principios de desarrollo de Varnish, según su documentación oficial, son el rendimiento y la flexibilidad y por eso se han tenido que sacrificar otros aspectos de compatibilidad. De esta forma, Varnish ha sido desarrollado para demostrar todo su potencial en arquitecturas de 64 bits y el rendimiento escalará de forma directamente proporcional al numero de núcleos de CPU que hayan disponibles en el sistema.

Sí se decide instalar Varnish en un entorno con arquitectura de 32 bits, el límite de memoria que Varnish puede direccionar se reduce a 3GB, por lo que también se reducirán el numero de hilos simultáneos que se pueden lanzar y al mismo tiempo se reducirá el tamaño máximo de la cache.

Otro aspecto que confiere a Varnish una mejora de rendimiento es que no decide por si mismo si la cache HTTP que reserva se encuentra en memoria principal o en disco, delegando en el kernel del sistema operativo la gestión de la RAM, pues se considera que es un mejor gestor de memoria y de *swapping,* de esta forma el sistema operativo nunca sufrirá inanición de memoria principal por culpa de la cache de Varnish.

Por último, la forma en la que el programa trata las peticiones es completamente configurable por el usuario, utilizando un lenguaje propio de configuración denominado *Varnish Configuration Languaje* (VCL en adelante). Todo el código debe escribirse en un fichero con extensión *.VCL* e indicar al programa que utilice ese fichero al aceptar peticiones HTTP. El código se traduce a lenguaje C, se compila con un compilador de C y se enlaza dinámicamente a los hilos de Varnish en tiempo de ejecución. El hecho de que el código se ejecute como C, otorga al programa un añadido de rendimiento, además de permitir escribir directamente utilizando C en lugar de VCL.

Este aspecto nos confiere varias ventajas, de las cuales la más práctica es la libertad que tiene un administrador de servidores para decidir cómo quiere que Varnish se comporte ante las peticiones, pudiendo hilar hasta el grano más fino.

Para el desarrollo de este proyecto, hemos utilizado la versión 3 de Varnish a pesar de estar disponible la 4 debido a problemas de compatibilidad entre éste y el sistema operativo CentOS 6. Nuestro sistema operativo solo soporta una versión beta de Varnish 4, mientras que tiene soporte completo para la última versión estable de Varnish 3.

3. Planificación de la solución

Lo más importante para resolver un problema es hacer un correcto planteamiento previo, tanto del problema a resolver como de la solución que se va a desarrollar. Remitiéndonos al apartado de objetivo de esta memoria procedemos a recuperar la información de la necesidad para detallarla de una forma más concreta.

El objetivo del proyecto es conseguir integrar Varnish para que funcione correctamente con un servidor web apache, creando un módulo para Virtualmin que permita al administrador web configurar de una forma cómoda los parámetros básicos de la integración.

3.1. Situación de partida

El sistema operativo de la máquina virtual cuenta con un servidor web Apache ya instalado, así como con un panel de control Virtualmin.

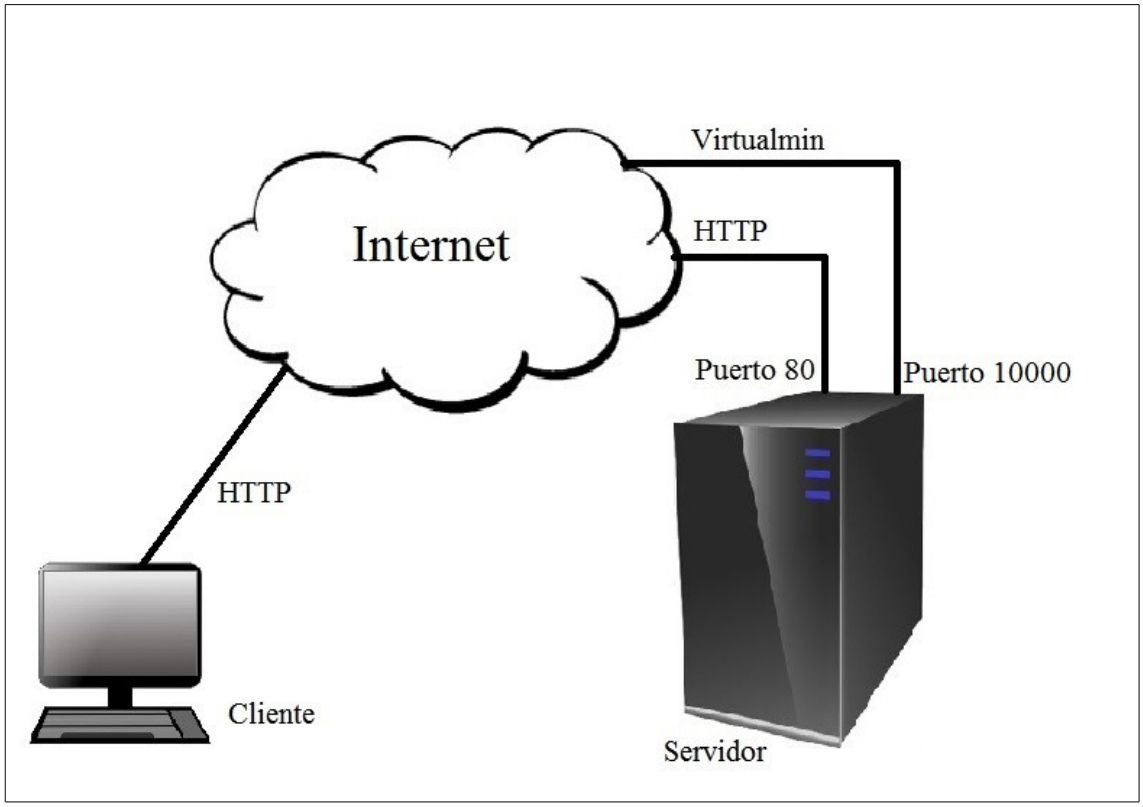

Ilustración 3: Esquema del estado inicial de nuestro sistema.

El servidor web Apache se encuentra activo, escuchando las peticiones HTTP que le lleguen por el puerto 80 a la dirección IP de la máquina, que en nuestro caso es la dirección IP 192.168.0.103.

El panel Virtualmin se encuentra también activo, siendo accesible por el puerto 10000 del servidor. Para conectarse hay que acceder con un navegador web con soporte para HTTPS y autenticarse como usuario válido del sistema CentOS. Se usará *root* para tener los permisos administrativos necesarios.

Se trata de una configuración típica y sencilla de un servidor web genérico, sin ningún proxy instalado. Por este motivo, en este caso es Apache el que recibe, procesa y responde todas las peticiones web que le lleguen por el puerto estándar de HTTP, el puerto 80.

3.2. Situación objetivo

A continuación vamos a explicar cómo deberá de quedar el sistema una vez se ejecute nuestra solución de integración de manera correcta. Para ello, antes de nada, explicaremos a groso qué es y como funciona tanto un proxy como un proxy inverso.

3.2.1 Proxy

Un proxy, o servidor proxy, en una red informática, es un servidor (un programa o sistema informático), que sirve de intermediario en las peticiones de recursos que realiza un cliente a un servidor (nuestra máquina con Apache). Por ejemplo, si una hipotética máquina A solicita un recurso a C, lo hará mediante una petición a B (el proxy), que a su vez trasladará la petición a C; de esta forma C no sabrá que la petición procedió originalmente de A. Esta situación estratégica de punto intermedio suele ser aprovechada para soportar una serie de funcionalidades: control de acceso, registro del tráfico, prohibir cierto tipo de tráfico, mejorar el rendimiento, mantener el anonimato, proporcionar cache web, etc; este último sirve para acelerar y mejorar la experiencia del usuario mediante elementos de la web que guardará el proxy, esto se debe a que la próxima vez que se visiten las páginas web no se extraerá información directamente del servidor web, si no que se recuperara información de la cache.

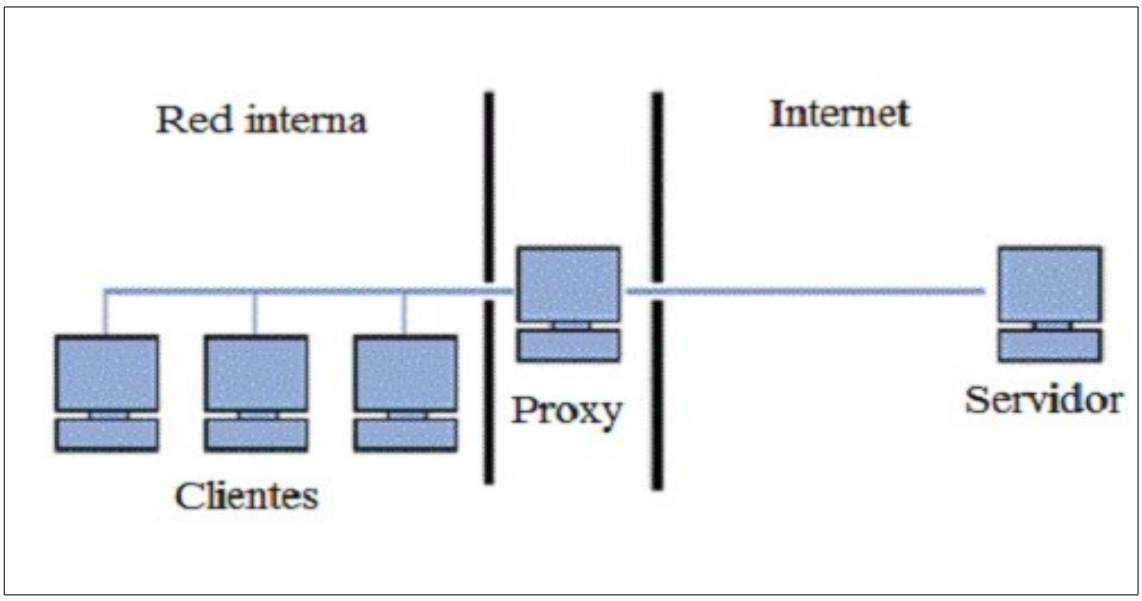

Ilustración 4: Esquema de un proxy situado a la salida de una intranet.

La situación más típica para un proxy es a la salida de una red interna hacía Internet, proporcionando a los usuarios de esta intranet una puerta de salida hacía el exterior. De esta forma se consigue controlar todo el tráfico de salida si se desea, pues debe de pasar por el proxy. Así también se pueden reducir tiempos de navegación, pues sí el proxy tiene almacenado en cache el objeto solicitado desde la red interna, éste lo servirá rápidamente, sin necesidad de enviar la petición al servidor externo y esperar su respuesta.

3.2.2 Proxy inverso

Cuando hablamos de un proxy inverso, nos referimos a un proxy que se sitúa entre uno o varios servidores web e Internet. Gracias a esta arquitectura de red, los servidores quedan protegidos de ataques externos, pues es el proxy el que puede controlar y filtrar todas las peticiones entrantes. Sí además añadimos la característica de proxy cache inverso, conseguiremos librar a los servidores de la carga que supondría el procesado de todas las peticiones que ahora son respondidas desde este intermediario. De cara al cliente, todo este proceso es transparente, y no podrá distinguir si el que le sirve la petición es el servidor o el proxy.

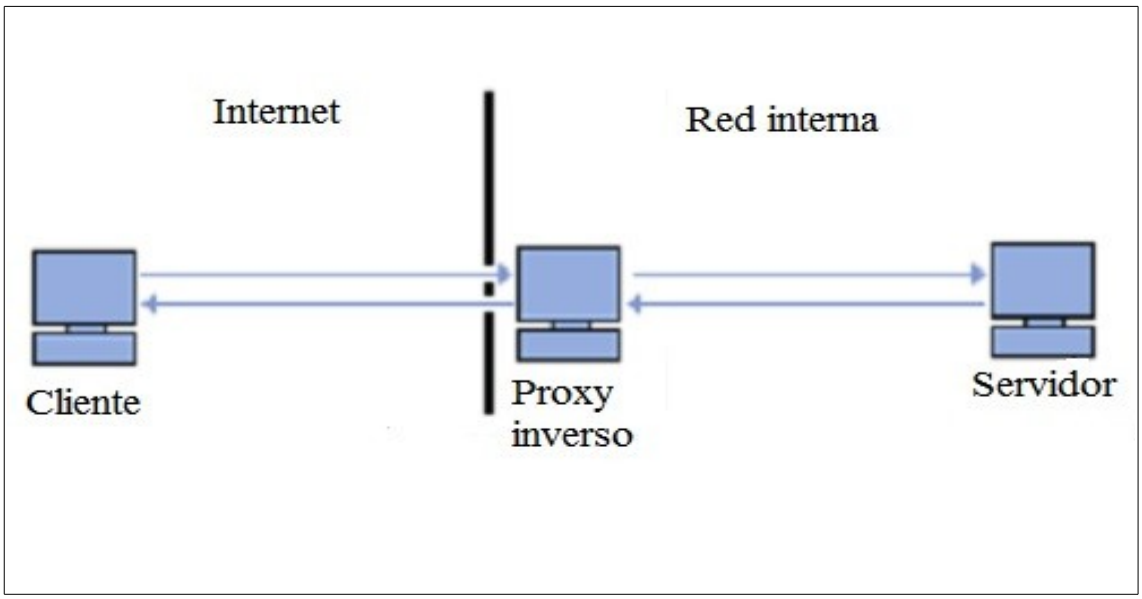

Ilustración 5: Esquema de un proxy inverso.

Con los conceptos anteriores claros, a continuación concretamos cómo queda nuestra máquina y sus conexiones una vez realizada la integración de Apache con Varnish mediante Virtualmin.

Al ser el proxy inverso el que debe recibir las peticiones HTTP, deberá escuchar hacía Internet en el puerto 80. Por consiguiente, dado que ejecutaremos en la misma máquina los servicios de Apache y de Varnish, deberemos mover Apache de ese puerto y colocarlo en otro en el que no tenga conflicto con otros servicios. Típicamente y como buena práctica, apache se suele colocar en el puerto 8080 y en la dirección local *loopback* o *localhost,* ésta es la 127.0.0.l. De esta forma evitaremos que nadie pueda atacar el servidor web directamente desde el exterior de la máquina.

Por último deberemos configurar Varnish para que sepa en qué dirección y puerto está Apache, para así poder retransmitirle las peticiones cuando sea necesario.

De esta forma, en la situación final, ante una petición HTTP entrante, Varnish será el encargado de recibirla y procesarla. Entonces reenviará la susodicha petición procesada a Apache. El servidor entonces responderá a la petición con el o los objetos necesarios que serán recibidos por Varnish. En este momento el proxy almacenará en cache el par petición/respuesta al mismo tiempo que envía los objetos a la dirección que originó la petición.

Ahora, ante una petición entrante que tenga que recibir la misma respuesta, será Varnish el que la responda directamente con los objetos que tenga en cache sin necesidad de que Apache reciba ni procese nada, alcanzando así la consiguiente reducción de carga al sistema y evitando los tiempos de procesado de peticiones de Apache, que son superiores a los de Varnish.

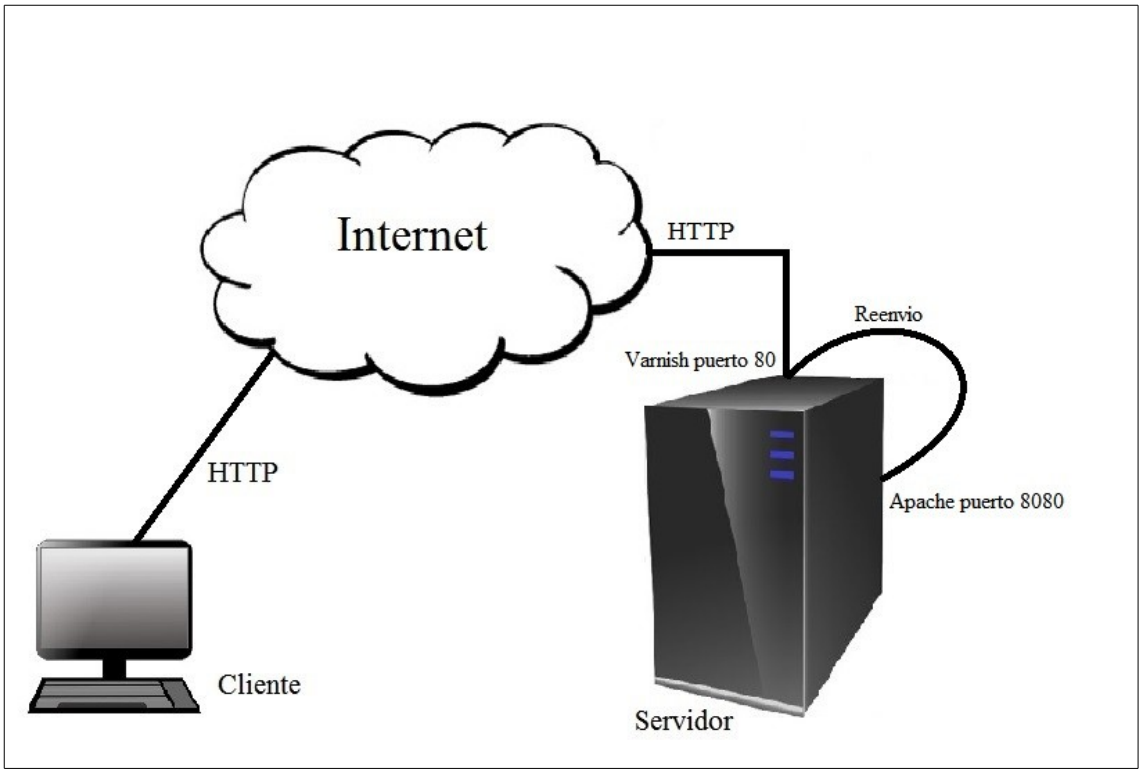

Ilustración 6: Esquema de la situación de nuestro sistema una vez completado el objetivo.

4. Conceptos previos al desarrollo

Una vez planificada la solución que se busca, se ha procedido a estudiar de forma exhaustiva los diferentes integrantes software del proyecto, con el fin de conocer sus formas de funcionamiento, ficheros de configuración, lógicas de trabajo y demás factores clave que hay que conocer para poder abordar el desarrollo de forma eficaz..

4.1. Panel Virtualmin

En primer lugar vamos a explicar cómo se puede acceder al panel de control desde una máquina remota. Es el paso esencial para poder utilizarlo como administradores. Aunque pueda resultar algo trivial, la máquina debe estar arrancada y funcionando, así como conectada a la red con una configuración válida. Entonces, sí en el equipo se encuentra instalado el panel Virtualmin y se encuentra lanzado el servicio (cosa que se hace por defecto al arranque del sistema), desde un navegador externo deberemos introducir la dirección IP de nuestro servidor e indicar que la conexión se realice por el puerto 10000. También se debe de usar una conexión con SSL, es decir, habrá que introducir la siguiente dirección en el navegador:

- https://192.168.0.103:10000

Una vez lancemos la petición a esa dirección, nos aparecerá una ventana de *login* a través de la cual podremos acceder al panel y a todas sus funciones ingresando nuestras credenciales, como la que se muestra en la figura 7.

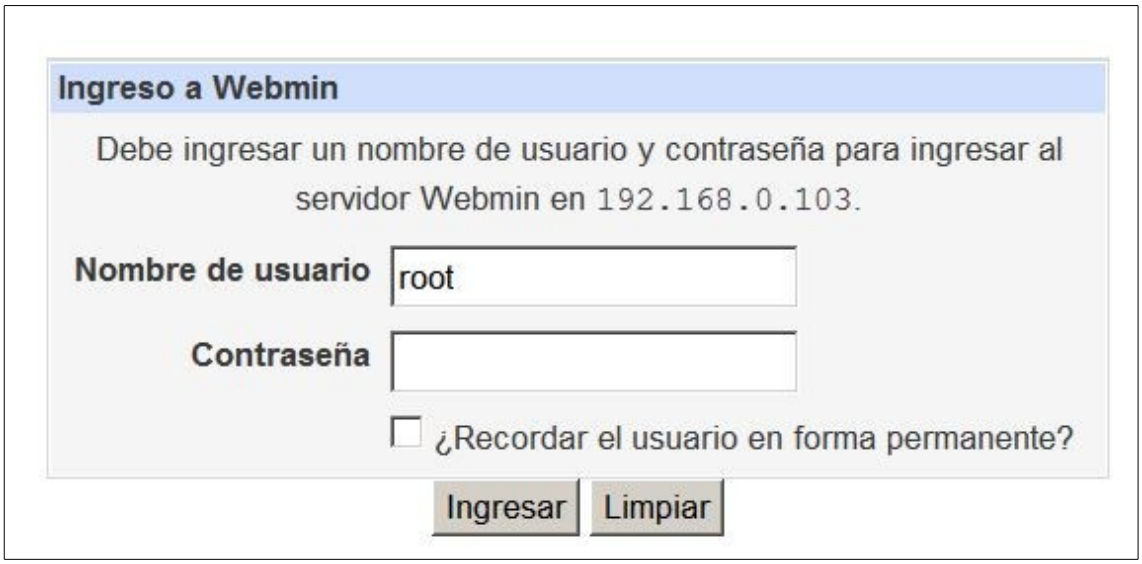

Ilustración 7: Pantalla de acceso al panel webmin.

Estas credenciales son las mismas que se utilizan para iniciar sesión en el sistema operativo, por lo que nuestro usuario y contraseña deberán estar dados de alta en nuestro CentOS, y por ende, al ser una distribución Linux, deberán existir en el fichero */etc/passwd* donde se almacenan todos los usuarios creados y sus propiedades.

Una vez autenticados en la ventana de *login*, en la máquina se lanza el ejecutable encargado de leer la configuración del panel y representar la información en una página web html. Remarquemos que cuando un ejecutable trabaja de esta forma, es decir, procesando información en el servidor y generando al vuelo código HTML (o cualquier otro objeto MIME) para ser representado en un navegador, se le denomina CGI (*Common Gateway Interface*).

Esta es la forma general de trabajar del panel de control en todas las situaciones de interacción. Cuando el administrador hace clic sobre los elementos de la página del panel, el sistema no hace más que lanzar a ejecutar los CGIs asociados a esas acciones que se encargan de procesar la información en el servidor y generar el código de la página web correspondiente que se muestra en el navegador.

Estos ejecutables de los paneles se componen de instrucciones escritas en Perl, un lenguaje de programación interpretado, que el servidor traduce a código máquina y ejecuta, y obteniendo como resultado una página de código HTML que el sistema envía al navegador origen de la petición. Aunque la gran mayoría de los CGIs de Virtualmin estén escritos en Perl, si nos interesa desarrollar a nosotros mismos algún módulo o función extra, podremos hacerlo en cualquier lenguaje interpretado, siempre y cuando la máquina en la que se ejecute disponga del intérprete correspondiente para el lenguaje escogido.

En segundo lugar, se ha analizado y estudiado la herramienta Virtualmin. El objetivo es conocer a fondo los diferentes ficheros que emplea para almacenar toda la información de configuración de los diferentes servidores virtuales que administra. También debemos conocer los diversos ejecutables que se lanzan cuando el administrador ejecuta acciones sobre la interfaz web del panel de control. Gracias a estos conocimientos, hemos podido diseñar nuestra solución para que sea capaz de interoperar con Virtualmin sin perjudicar el correcto funcionamiento tanto del panel como de los servicios que tenga activos el propio servidor en la situación previa a la integración.

El primer paso es localizar el directorio raíz del panel, para así poder analizar posteriormente su contenido. Recordamos que Virtualmin es una extensión del panel de control Webmin, al que le añade la capacidad de gestionar servidores virtuales. Por lo tanto parece razonable buscar nuestro objetivo dentro de la familia de directorios de Webmin. De esta forma el directorio donde se instala Virtualmin en nuestro sistema CentOS es:

- `/usr/libexec/webmin/virtual-server/`

Como curiosidad, podemos apreciar que el propio nombre del directorio unicamente hace referencia a la funcionalidad que el panel añade a Webmin.

En el interior de este directorio raíz de Virtualmin se encuentran los archivos ejecutables que se lanzan cada vez que el administrador realiza alguna acción en el panel. También se localizan aquí los diferentes ficheros de configuración de esos ejecutables así como diversos ficheros necesarios para que Virtualmin funcione en sinfonía con Webmin.

De entre todos los ficheros, el más importante para el desarrollo de nuestro proyecto es el dichero *config*. Éste contiene una extensa lista de atributos de configuración, entre los que se pueden encontrar desde parámetros por defecto de configuración de puertos en los servidores virtuales hasta rutas de ficheros que debe conocer el panel. Los atributos se organizan de forma que al principio de cada linea aparece el nombre de la propiedad y a continuación, precedido por un símbolo "=" se encuentra el valor o los diferentes valores separados por comas en el caso de que fuera un atributo multivaluado.

```
ldap=0
ldap_unix=1
ldap_mailstore=$HOME/Maildir/
vpopmail_dir=/home/vpopmail
vpopmail_maildir=mail
vpopmail_user=vpopmail
vpopmail_group=vchkpw
mysql_db=${PREFIX}
other_users=0
other_doms=0
limitnoalias=0
vpopmail_auto=/usr/local/bin/autorespond
bw_ftplog_rotated=1
mysql_mkdb=1
mysql_nopass=0
mysql_nouser=0
show_quotas=0
spam=0
```

Ilustración 8: Extracto del fichero conf de Virtualmin

Cada vez que un ejecutable del panel realiza una llamada a la función *init_config(),* ésta carga en memoria todos los atributos ordenados como una lista de pares *clave/valor* para así poder ser leídos o escritos correctamente por los programas del módulo que lo necesiten.

El contenido del fichero se puede editar a través de diversos formularios que muestra el panel al administrador en las secciones correspondientes a cada familia de valores, pero si se desea también se puede editar a mano, y mientras los valores introducidos sean coherentes el funcionamiento del panel será el correcto.

Una función importante de Virtualmin y que nos interesa conocer es que a la hora de crear un nuevo servidor virtual, ofrece al administrador una plantilla con los parámetros por defecto entre los que se encuentran el puerto e IP de este servidor. Esto, unido al esquema final que queremos conseguir para el sistema hace que nuestra solución deba cambiar esa plantilla con los nuevos valores de comunicaciones para que coincidan con la dirección y puerto al que moveremos Apache. Todos los atributos por defecto de esta plantilla se encuentran en el fichero *config* de configuración mencionado con anterioridad.

4.2. Servidor web Apache

Tras analizar y comprender el panel Virtualmin y su funcionamiento, toca estudiar el servidor web Apache, qué recordemos, es el que se encarga en la situación inicial de recibir, procesar y responder las peticiones que le lleguen a través de la IP en la que esté configurado.

En el sistema de nuestra máquina virtual se encuentra instalado un servidor Apache versión 2.2. El directorio raíz del programa es:

- */etc/httpd/*

Dentro de este directorio hay dos directorios clave para nosotros que son:

- */etc/httpd/conf/*
- */etc/httpd/conf.d/*

Dentro del primer directorio están los ficheros de configuración del servicio web que ofrece Apache. De todos ellos el archivo más importante es *httpd.conf* que contiene todos los atributos de configuración del servicio. En este fichero podemos encontrar, entre otros atributos, el puerto o puertos y dirección o direcciones IP en los que Apache se encuentra escuchando a la espera de peticiones entrantes. También se especifican aquí en qué rutas del sistema debe ir Apache a buscar los objetos necesarios para responder una petición. Además, en el fichero se encuentra definido cualquier servidor virtual que exista en la máquina.

Apache sigue el mismo formato para todos sus ficheros de configuración, incluido del que estamos tratando. Un atributo de configuración ocupa una linea, y los valores que tenga definido este atributo deben escribirse a continuación en la misma línea, separados del atributo por un espacio. Sí el valor de un atributo contiene espacios, es necesario escribirlo entrecomillado con comillas dobles.

Las líneas más importantes en relación a nuestro proyecto son dos:

- Listen 80: Este atributo especifica en qué IP y puerto está escuchando el servidor Apache a la espera de peticiones. En este caso vemos que Apache espera en todas las IP disponibles y en el puerto 80. También podrían haber otros formatos de línea coexistiendo, como Listen IP:puerto, o Listen *:puerto.
- NameVirtualHost *:80: Esta linea indica en qué dirección IP y puerto espera Apache peticiones que se van a enviar a los hosts virtuales. El * indica que se esperan peticiones en todas las direcciones IP disponibles en la máquina.

Los atributos del fichero pueden agruparse en bloques de configuración, especificando así a qué parte concreta del servidor se aplican esos valores contenidos en el bloque. Estos apartados se organizan siguiendo una estructura en formato XML, es decir, se delimitan al inicio y al final por una línea con el nombre del bloque entre los caracteres menor y mayor que (<, >). En el fichero se pueden encontrar bloques de configuración de los siguientes tipos: `<Directory>`, `<DirectoryMatch>`, `<Files>`, `<FilesMatch>`, `<Location>`, `<LocationMatch>`, y `<VirtualHost>`.

Este ultimo tipo de bloques de configuración son los que más nos interesan de cara al proyecto, pues como se puede deducir a partir de la etiqueta que los identifica, cada uno contiene la configuración de cada servidor virtual alojado en el servidor web.

Un servidor virtual es una partición dentro de un servidor web físico (una máquina con Apache en nuestro caso) que puede convivir funcionando simultáneamente con otros servidores virtuales. Gracias a esta abstracción, con una única máquina y un único servidor Apache podemos ofrecer al exterior el servicio de tantos servidores como deseemos, sin que el usuario note que están todos en un único ordenador. Pero entonces los servidores virtuales deben diferenciarse unos de otros y tener una identificación única que permita a Apache decidir a qué servidor concreto va destinada una petición entrante en la máquina.

Esta diferenciación o identificación se puede conseguir mediante el nombre de host. Esto quiere decir que si por ejemplo tenemos un servidor virtual con el atributo de configuración *ServerName* con valor "ejemplo.com", cuando llegue una petición web a Apache, éste mirará la cabecera *host* de la petición, y si coincide con el valor del atributo mencionado, se asignará esa petición a ese servidor virtual.

```
<VirtualHost 127.0.0.1:8080>
    SuexecUserGroup "#513" "#511"
    ServerName prueba.com
    ServerAlias www.prueba.com
    ServerAlias webmail.prueba.com
    ServerAlias admin.prueba.com
    DocumentRoot /home/prueba.com/public_html
</VirtualHost>
```

Ilustración 9: Ejemplo de bloque de configuración de un virtual host.

También es posible hacer esta diferenciación entre servidores virtuales utilizando diferentes direcciones IP y/o puertos para cada host virtual. De esta forma, si una petición entra por la IP y puerto que tenga asignado un servidor virtual, será a este al que Apache mandará la petición.

Volviendo al estudio de los directorios de Apache, pasamos ahora a */etc/httpd/conf.d/* en el que se encuentran los ficheros de configuración que utilizan los módulos de Apache instalados en el equipo. Por ejemplo la extensión *mod_dbd* que añade a Apache la posibilidad de utilizar conexiones con una base de datos SQL. Existen numerosos módulos o extensiones para Apache, convirtiéndolo aún más si cabe en una auténtica navaja suiza de los servidores web[19].

4.3. Varnish Cache

En esta sección estudiamos la herramienta que hace de proxy inverso en nuestro sistema. Vemos como funciona y cómo se configura para poder abordar posteriormente el desarrollo de la solución de la mejor forma posible.

Comenzamos con todo lo referente a la configuración de Varnish y observamos que existen dos grandes categorías de configuración en el programa, la configuración por linea de comandos con los parámetros del servicio, y el fichero VCL, qué configuraremos más adelante.

4.3.1 Configuración del servicio

Los parámetros del servicio y los argumentos de linea de comandos se usan para definir cómo Varnish debería interaccionar con el sistema operativo y el hardware del sistema, mientras que la configuración VCL define cómo debe interactuar el proxy con los servidores web y los clientes. Casi todos los aspectos configurables de Varnish pueden ser editados en tiempo de ejecución sin necesidad de reiniciar el programa, pero otros concretos como la dirección IP, tamaño de la cache y el algoritmo de dispersión requieren un reinicio del servicio.

El formato de los parámetros por consola a la hora de lanzar el servicio es el siguiente:

- -a <[nombre/IP de host]:puerto> → Especifica la dirección en la que Varnish escucha peticiones entrantes.
- -f <fichero VCL> → Indica la ruta y el nombre del fichero VCL que se usará para tratar las peticiones.
- -p <parámetro=valor> → Permite dar valor a los diferentes parámetros configurables.
- -S <archivo *secret*> → Configura en qué fichero se almacenan los credenciales para poder autenticarse a la hora de administrar Varnish de forma remota.
- -T <nombre/IP de host:puerto> → Especifica en qué dirección y puerto abrirá Varnish una conexión para la administración remota del servicio.
- -s <tipo de almacenamiento, opciones> → Indica el dónde y cómo se almacenarán los objetos cacheados.

Dada la flexibilidad del programa, todas las opciones son opcionales, salvo la opción -f para indicar el fichero VCL con la lógica de tratamiento de opciones. Sin embargo, aunque sean opcionales, por norma general siempre se necesita un punto de almacenamiento de la cache con -s, una dirección en la que Varnish escuche con -a. En los dos parámetros en los que se requiere una dirección y un puerto, no es estrictamente necesario especificar una dirección IP concreta, pudiendo utilizar la orden *-a :80* que hará que Varnish espere peticiones en todas las direcciones IP disponibles en el puerto 80.

También se pueden modificar parámetros con -p múltiples veces en la misma linea de comando sin ningún problema.

Todos los parámetros que se especifiquen por linea de comandos a la hora de lanzar el servicio deben ser escritos a mano en el fichero */etc/sysconfig/varnish,* de lo contrario, ante un reinicio del sistema se lanzará Varnish con los parámetros que encuentre en ese fichero.

4.3.2 Arquitectura del servicio

A continuación vamos a describir cómo se puede adaptar de la mejor forma posible el servicio a nuestra máquina y nuestro hardware, pero debemos recordar que buscamos implementar una solución lo más genérica posible, por lo que tampoco podemos concretar ni afinar mucho la configuración en este sentido.

El servicio de Varnish se compone de dos procesos principales. El primero es el proceso de administración, que se encarga de aplicar cambios de configuración, de compilar el fichero VCL, de monitorizar Varnish, de iniciarlo y de recibir e interpretar los comandos por consola que le lleguen.

El segundo proceso principal es el proceso “hijo”. Periódicamente, el proceso de administración se intenta comunicar con el proceso hijo para consultar si sigue lanzado y funcionando correctamente. En caso de que éste no responda en un tiempo razonable, el proceso de administración matará al proceso hijo y lo lanzará de nuevo.

Ahora nos centraremos en conocer más a fondo el proceso hijo y su funcionamiento y utilidad. Este proceso está formado por múltiples hilos y es sobre el que recae la funcionalidad principal del programa. En la siguiente tabla podemos observar los componentes principales de este proceso hijo.

Nombre del hilo	Numero de hilos	Tarea
Cache-worker	Uno por conexión activa	Manejar peticiones
Cache-main	Uno	Inicialización
Acceptor	Uno	Aceptar nuevas conexiones y delegarlas
Epoll	Configurable, 2 por defecto	Controlar y gestionar los *pools* de hilos
Expire	Uno	Eliminar contenido caducado de la cache
Backend poll	Uno por backend	Comprobaciones del estado del servidor web.

Para poder ajustar estos aspectos de Varnish correctamente, debemos pensar en las previsiones de tráfico que va a soportar nuestro servidor. La arquitectura de hilos de la que se compone el programa nos permite tener mas de dos *pools* de hilos, pero según el libro de documentación oficial de Varnish, diferentes experimentos y estadísticas de entornos de altas exigencias en cuanto a rendimiento han demostrado que incrementar el número de *pools* de hilos a más de dos no mejora las prestaciones en general, mientras que reducirlo a uno si empeora el rendimiento drásticamente.

Mientras en el manual aconsejan dejar la mayoría de parámetros con los valores por defecto, hay uno que si es posible que nos convenga ajustar. Ese parámetro es el numero de hilos que contiene cada *pool,* valores definidos por los siguientes pares parámetro/valor por defecto:

- *thread_pool_min: 5*
- *thread_pool_max: 500*

Varnish puede crear y destruir hilos bajo demanda y eliminarlos una vez la carga de trabajo se reduce. Esto permite enfrentarme mejor a los picos de tráfico que se puedan producir. Por ello, es preferible intentar mantener siempre un numero razonable de hilos sin trabajo mientras el tráfico se mantiene en valores normales, que correr siempre el mínimo número de hilos y crear

destruirlos y crear nuevos constantemente conforme varía el trabajo. Esto quiere decir que es una mejor práctica estimar al alza el valor mínimo de hilos.

Con esto, el proxy será capaz de controlar tantas peticiones simultaneas como hilos pueda crear. Así, cuando Varnish acepta una conexión entrante, la petición se delega en un *pool* de hilos que a su vez la delegará en uno de los hilos que contiene siempre que haya alguno disponible. Sí no hay ningún hilo disponible, la conexión de pondrá en una cola de espera, y sí las colas están llenas, la petición se desechará directamente.

5. Desarrollo del módulo

Con todo lo estudiado en los apartados anteriores ya estamos en la situación idónea para desarrollar el módulo de Virtualmin que integre todo el software en el sistema de forma automática.

Por el contrario, en esta sección no vamos a tratar los procedimientos necesarios para poder configurar el funcionamiento que le daremos a la cache por defecto. Ese tema se trata más adelante en la memoria y toda la lógica de funcionamiento en cuanto a tratamiento de peticiones y de respuestas recaen sobre el fichero VCL de Varnish.

Retomando el tema que nos compete en esta sección de la memoria, vamos a recordar que la configuración de Apache se localiza en el fichero */etc/httpd/conf/httpd.conf,* por lo que nuestro módulo deberá trabajar con él, tanto leyéndolo como editándolo. También es necesario recordar que un módulo de Virtualmin se compone internamente de ejecutables en un lenguaje interpretado, que nosotros hemos desarrollado en Python, así como de diversos CGIs que crean toda la interfaz que se le muestra al administrador del sistema.

A continuación procedemos a detallar todos los ficheros ejecutables, de configuración y CGIs que se han desarrollado.

5.1. conf.file

Este fichero de nuestro módulo realiza la función de almacenar todos los parámetros necesarios para la correcta integración y configuración de los diferentes componentes de nuestra solución, así por ejemplo contiene las rutas de los diferentes ficheros de configuración necesarios o el puerto y dirección IP a los que mover Apache entre otros. Su contenido se detalla más adelante. La idea es que nada más el administrador instale el módulo en su sistema, este fichero tenga unos valores por defecto adecuados a la mayoría de entornos imaginables. El fichero está escrito en texto plano y puede ser editado con cualquier editor de texto.

```
IP: 127.0.0.1
nPuertoAp: 9080
nfPlantilla: /etc/webmin/virtual-server/config
rutaVarnish: /etc/varnish/
nfDefaultVcl: /etc/varnish/default.vcl
nfConfVarnish: /etc/sysconfig/varnish
nfConfApache: /etc/httpd/conf/httpd.conf
nfConfRPAF: /etc/httpd/conf.d/mod_rpaf.conf
varnishListenAdd: 192.168.0.103
varnishListenPort: 80
varnishAdmListenAdd: 127.0.0.1
varnishAdmListenPort: 6082
varnishStorageSize: 400M
varnishStorage: malloc,${VARNISH_STORAGE_SIZE}
varnishTTL: 120
varnishMinH: 50
varnishMaxH: 1000
```

Ilustración 10: Contenido por defecto del fichero conf.file de nuestro módulo de integración.

También es posible editar la mayoría de valores contenidos en el fichero mediante un formulario del módulo habilitado para tal uso, sin embargo, tal y como se explicará más adelante, no se ha optado por ofrecer la posibilidad de editar todos y cada uno de los valores de una forma tan sencilla para evitar problemas producidos por manos inexpertas, pues la visión principal de nuestro desarrollo es ofrecer una herramienta funcional y sencilla. Pasamos a detallar el contenido del fichero:

- *IP: 127.0.0.1*

  Dirección IP a la que se va a mover Apache para la integración. El valor por defecto es la dirección *loppback* o *localhost*, que es la 127.0.0.1. Este valor por defecto nos garantiza un extra de seguridad, pues nadie podrá acceder a nuestros servidores virtuales desde fuera sin pasar antes por Varnish.

- *nPuertoAp: 8080*

  Numero de puerto al que se va a mover Apache para la integración. El valor por defecto es el puerto 8080, que atiende a una especie de convencionalismo no escrito, que dice que ése es el puerto alternativo para servicios HTTP.

- *nfPlantilla: /etc/webmin/virtual-server/config*

  Fichero con ruta absoluta donde se almacenan los datos de la plantilla que ofrece Virtualmin al administrador a la hora de crear nuevos servidores virtuales. Necesitamos modificarla para que coincida con la nueva configuración tras la integración. El valor por defecto es la ruta en la que se encuentra el fichero en todos los sistemas basados en RHEL.

- *rutaVarnish: /etc/varnish/*

  Ruta de instalación del programa Varnish Cache. La ruta por defecto coincide con la ruta en la que se instala en todas las distribuciones Linux.

- *nfDefaultVcl: /etc/varnish/default.vcl*

  Fichero con ruta absoluta de configuración de la cache de Varnish. Ésto es el fichero VCL. El valor por defecto es el que se da en los sistemas operativos basados en Linux.

- *nfConfVarnish: /etc/sysconfig/varnish*

  Fichero con ruta absoluta de dónde se configura el servicio de Varnish para el arranque. Aquí se encuentra la llamada al demonio con todos los parámetros necesarios. El valor por defecto coincide con la ruta en la que se encuentra dicho fichero en las distribuciones basadas en RHEL.

- *nfConfApache: /etc/httpd/conf/httpd.conf*

  Fichero con ruta absoluta de configuración de Apache. Tal y como se ha comentado anteriormente, aquí se encuentran configurados los hosts virtuales entre otros. El valor por defecto coincide con el valor en todas las distribuciones basadas en Linux.

- *nfConfRPAF: /etc/httpd/conf.d/mod_rpaf.conf*

  Fichero de configuración del módulo de Apache Rpaf. Este módulo nos permite poder enviar la IP del cliente al servidor web para tener unos logs más coherentes. Se hablará de él más adelante. El valor por defecto es el de la ruta del fichero en los sistemas basados en RHEL.

- *varnishListenAdd: 192.168.0.103*

  Dirección hacia Internet en la que vamos a poner Varnish a escuchar peticiones. El valor por defecto es una IP cualquiera y deberá ser modificado por el administrador a través del formulario para que coincida con una IP válida del sistema.

- *varnishListenPort: 80*

  Puerto en el que vamos a poner Varnish a escuchar peticiones. El puerto recomendado es el puerto HTTP, es decir, el puerto 80 que coincide con el valor por defecto.

- *varnishAdmListenAdd: 127.0.0.1*

  Dirección IP por la cual se podrá acceder a la consola de administración de Varnish. Se recomienda una dirección de la cual tengamos la certeza de que solo el administrador podrá tener acceso. Por defecto viene la dirección *loopback* 127.0.0.1, lo que nos confiere la seguridad de que solo se podrá acceder a este servicio a través de la propia máquina.

- *varnishAdmListenPort: 6082*

  Puerto a través del cual se podrá acceder a la consola de administración de Varnish. El valor por defecto es el que se recomienda en la documentación del programa por convención.

- *varnishStorageSize: 400M*

  Tamaño del espacio de memoria que Varnish va a reservar para la cache. Se pueden utilizar los sufijos k (kilobytes), M (megabytes), G (gigabytes) y T (terabytes). El valor por defecto es 400M, pero deberá ser modificado por el administrador para ajustarse a sus necesidades y limitaciones. Se hablará más adelante de este aspecto.

- *varnishStorage: malloc,${VARNISH_STORAGE_SIZE}*

  Forma de reservar memoria para cache de Varnish. El valor por defecto es con un malloc del tamaño especificado en parámetro anterior. Se deberá modificar este valor por medio de un editor de texto si se desea reservar la memoria para cache con un fichero, pero no es una práctica recomendada pues perdemos parte importante de la mejora de rendimiento.

- *varnishTTL: 120*

  Tiempo de vida (en segundos) durante el cual se mantendrán los objetos en cache. Por defecto se aplican 120 segundos, que es la configuración con la que se instala el software Varnish. El administrador deberá cambiar este atributo mediante el formulario diseñado para ello y adaptarlo a sus necesidades.

- *varnishMinH: 50*

  Número mínimo de hilos que mantendrá Varnish en espera por cada *pool* de hilos. Por defecto se ha decidido dejar 50 hilos siempre activos, pero el valor puede ser cambiado por el administrador en el formulario para adaptar la solución a sus niveles de tráfico. Recordemos que Varnish trabaja con 2 *pools* de hilos por defecto siendo esta la configuración más recomendada.

- *varnishMaxH: 1000*

  Número máximo de hilos que Varnish podrá tener en ejecución por cada *pool* de hilos. Por defecto hemos asignado un valor de 1000 hilos, que combinado con los dos *pool* con los que trabaja Varnish por defecto, hacen un total de 2000 hilos como máximo simultáneos, lo que se traduce en un máximo de 2000 peticiones atendidas simultáneamente. El valor podrá ser editado por el administrador a través del formulario.

5.2. varnishRLib.py

Después de explicar cómo vamos a guardar las configuraciones de nuestro módulo a través del fichero descrito anteriormente, procedemos a desmembrar y analizar la librería que hemos creado en Python *varnishRLib.py*. Podemos ver un extracto del código del fichero en la siguiente imagen.

```
##############################################################################
# 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
        valores=valores[1:] # quita el primer espacio
        valores=valores[1::2] # quita el resto de espacios.
        config[linea[0:fin]]=valores #asigna la de conf
```

Ilustración 11: Extracto de código de la funcion cargaConf de la librería de nuestro módulo.

Dentro de este fichero encontramos todas las funciones que tienen en común los distintos ejecutables del proyecto, es por ésto por lo que podemos denominar al fichero como una librería. Dentro del fichero podemos encontrar funciones para leer nuestro archivo de configuración o también para escribirlo. Las detallamos a continuación.

- *guardaConf( )*

  Función que abre el fichero de configuración *conf.file* y lo recorre línea a línea, comparando las entradas con las de la variable *config*, escribiendo los valores de esta variable en la línea correspondiente.

- *cargaConf( )*

  Función que abre el fichero de configuración y lo recorre línea a línea, cargando los valores encontrados en la variable *config.*

- *imprimeConf( )*

  Función que recorre el fichero de configuración e imprime los distintos valores de configuración encontrados.

- *log (cadenaLog)*

  Función que abre un fichero de log llamado *logFile.log* para escribir en éste el valor de la variable que se le pasa como argumento, precedido de la hora del sistema en el momento de la escritura. Gracias a esta función se mantiene un registro de las acciones realizadas por los ejecutables para facilitar la tarea de debug.

- *escribeFichero (fich, conf)*

  Función que abre el fichero que se le pasa como parámetro a través de la variable *fich,* para posteriormente, añadir al final del mismo el contenido de la variable que se le pasa como parámetro *conf.* También genera una linea de log al ejecutarse indicando qué fichero se ha escrito.

- *leeFichero (fich, conf)*

  Función que abre el fichero que se le pasa como parámetro a través de la variable *fich,* para recorrerlo línea a línea. Devuelve el contenido del fichero leído en la variable *contenido*. Esta función también genera una entrada de log indicando qué fichero se ha leído.

El mecanismo para poder cargar y utilizar estas funciones en cualquier ejecutable es utilizar la instrucción *import varnishRLib.py,* de esta forma, la librería se enlaza con el ejecutable que la importa, permitiendo la llamadas a sus funciones dentro del código mediante la instrucción varnishRLib.*función(parámetros de entrada),* que veremos aparecer numerosas veces en los otros ejecutables.

5.3. apacheConf.py

Sobre este script recaen las funciones iniciales para preparar el servidor Apache para la integración de nuestro proxy inverso. A continuación se detalla el contenido de las funciones que lo componen así como su cometido, adjuntando alguna imagen que ayude al lector a visualizar el código programado.

- *compruebaConfApache( )*

  Esta función se encarga de abrir el fichero de configuración de apache, que se encontrará en la ruta especificada por el valor de *nfConApache* de nuestro fichero de configuración, y de cargar su contenido en una variable. Posteriormente envía ese contenido a la función *ConfApache( ),* la cual realizará las modificaciones necesarias que veremos más adelante en esta misma sección de la memoria. Después mandará ese contenido del fichero de configuración modificado a la función *purgarApache( ),* que realizará otras modificaciones en el contenido. Por último, si se ha modificado algo del fichero, la función lo guardará en su sitio con el nuevo contenido modificado.

- *confApache (conf)*

  La función de la que tratamos ahora, recibe como parámetro una variable cargada con el contenido del fichero de configuración de Apache. La llamada la recibe de la función que acabamos de analizar anteriormente.

  Su funcionamiento consiste en recorrer esa configuración línea a línea para, mediante el uso de expresiones regulares, buscar las entradas que indican en qué dirección o direcciones IP y puerto o puertos se encuentra el servidor web Apache esperando conexiones entrantes.

  En primer lugar se buscan las entradas de tipo "*NameVirtualHost IP:puerto",* de las cuales se extrae tanto el puerto como la IP que contienen para compararlos con los valores de dirección a los que se desea mover el servidor web Apache. En el caso de que no sean los valores deseados se cambia la línea con los nuevos valores de IP y puerto.

  Lo mismo se hace con las entradas tipo *"Listen IP:puerto"* y sus posibles variaciones que pueden ser tanto "*Listen puerto"* como "*Listen *:puerto"*. En todos los casos citados se sustituye tanto el puerto como la dirección IP por los valores configurados en el módulo.

  Por último se localizan las entradas del fichero de tipo "*<VirtualHost IP:puerto>"* para proceder a cambiar el puerto de los servidores virtuales por el puerto configurado para Apache.

  Durante la ejecución de la función, se omitirán todas las entradas que mencionen el puerto SSL, es decir, el puerto 443, ya que Varnish no soporta ese protocolo por lo que los servicios que lo usen deberán continuar ejecutándose sin modificación alguna de sus comunicaciones.

- *purgarApache(conf)*

  A esta función se le llama inmediatamente después de que la función anterior realice las pertinentes modificaciones al fichero de configuración de Apache que recordemos se encuentra cargado en memoria en una variable. La utilidad del código de esta función es la de eliminar cualquier entrada de tipo *"Listen IP:puerto"* que se encuentre repetida a lo largo del fichero. De no hacerlo, al encontrarse Apache con más de una entrada de este tipo con la misma IP y puerto, no sería capaz de arrancar, dando un error en el servicio. Puede verse el código de esta función en la figura 12.

- *comprobarVarnish( )*

  La sencilla tarea de esta función del script es la de comprobar si está actualmente Varnish instalado en la máquina. Para ello intenta abrir el fichero de configuración del servicio, devolviendo un valor *False* en caso de que no pueda encontrarlo o lo que desencadenará la ejecución de la siguiente función.

- *instalarVarnish( )*

  Tal y como se introducía antes, este segmento de código se lanza a ejecución en el caso de que la función anterior determine que Varnish no se encuentra instalado en el equipo. Al ejecutarse, la función instala el repositorio oficial del programa Varnish en su versión 3 para CentOS (o sistemas basados en RHEL). A continuación ejecuta la orden de instalación del programa y una vez completado este proceso, indica al sistema que este programa se lanzará al arrancar la máquina. Puede verse el código de esta función en la imagen 13.

- *comprobarRPAF( )*

  Esta función comprueba sí el módulo de Apache, RPAF, se encuentra instalado en el equipo o no. Para ello se basa en el mismo método que la función *comprobarVarnish( ).*

- *instalarRPAF( )*

  La función de encarga de descargar e instalar el módulo de Apache que le permitirá poder identificar el origen de las peticiones que le lleguen, tal y como se explica más adelante.

- *modPlantillaVmin( )*

  El código dentro de esta función se encarga de abrir el fichero donde se encuentra la información de la plantilla que ofrece Virtualmin al administrador del servidor a la hora de crear nuevos servidores virtuales y cargar el contenido en una variable, para posteriormente mandar el contenido a editar por otra función. Una vez finalizada la modificación, se procede a escribir el fichero con los nuevos valores en su sitio.

- *confPlantilla (conf)*

  Esta función recibe como parámetro de entrada todo el contenido de del fichero indicado por el parámetro de configuración *nfPlantilla.* Recorre dicho contenido línea a línea buscando mediante el uso de expresiones regulares las entradas de tipo "*web_port=”* y *“old_defip=”.* Ambas entradas se sustituyen por el valor del puerto e IP al que se desea mover a Apache. De esta forma, cuando el administrador desee crear un nuevo servidor virtual, los valores por defecto que se le mostrarán para estos parámetros serán los deseados en la integración.

Una vez conocidas todas las funciones que componen el script, procedemos a justificar su utilización. Así tenemos que en primer lugar, para que la integración funcione correctamente, es estrictamente necesario mover al servidor web Apache del puerto 80 a otro puerto, pues en caso contrario se produciría un conflicto con el servicio Varnish, que debe atender peticiones del exterior en el puerto estándar de HTTP, esto es el puerto 80. Para conseguir mover el servicio es necesario cambiar todas las entradas *"Listen"* y *"NameVirtualHost"* encontradas en el fichero de configuración. De esta forma evitaremos los conflictos entre los servicios.

```
#############################################################################
# Funcion que elimina las entradas Listen repetidas para evitar problemas de configuracion
#############################################################################
def purgarApache (conf):
    configurado=False
    for n in range(0,len(conf)):
        linea=conf[n]
        if re.match("Listen \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\:\d{2,4}\n", linea):
                ini=linea.find(" ")
                fin=linea.find(":")
                IPantigua=linea[ini+1:fin]
                puertoAntiguo=linea[fin+1:len(linea)-1]
                if puertoAntiguo==varnishRLib.config["nPuertoAp"][0]
                and IPantigua==varnishRLib.config["IP"][0]:
                        if configurado:
                                conf[n]=""
                        if not configurado:
                                configurado=True
    return conf
```

Ilustración 12: Extracto de código del fichero apacheConf.py que ilustra la función purgarApache.

En cuanto al módulo de Apache, RPAF, resulta necesario para que la integración con el proxy inverso resulte plenamente funcional. En una configuración estándar entre Apache y un proxy inverso, sin el módulo RPAF, las peticiones que le lleguen al servidor web tendrán como IP de origen la IP del proxy inverso, pues es donde se interceptan, se tratan y desde donde se reenvían a Apache. Ésto es lo que sucede a nivel de datagrama IP, y en realidad es la forma correcta de funcionar según la implementación del protocolo, por lo que la modificación de este comportamiento no se contempla sencilla. El principal inconveniente de esta situación es la generación y estudio de logs y ficheros de estadísticas desde Apache, pues de esta manera sería imposible identificar correctamente de qué cliente son las peticiones que recibe, apareciendo en todas nuestro proxy Varnish como origen de éstas.

La solución, en nuestro, caso pasa por recurrir al citado módulo RPAF. Este módulo nos permite poder añadir de forma programática una cabecera HTTP a las peticiones cuando pasan por el proxy inverso. En concreto la cabecera es la "*X-Forwarded-For"* y en ella se indica el valor de la dirección IP donde se originó la petición. Gracias a esta solución, Apache puede interpretar correctamente esa cabecera y consigue generar unos logs y unas estadísticas correctas que nos darán unos resultados verídicos, dando por solventado el problema. Más adelante, cuando veamos la configuración del funcionamiento de la cache de Varnish en el fichero VCL, veremos como incluimos esta cabecera a las peticiones procesadas.

```
#########################################################################
# Funcion que instala Varnish en el sistema.
#########################################################################
def instalarVARNISH ( ):
    print "<br>"+"Instalando y configurando Varnish"
    varnishRLib.log("rpm --nosignature -i https://repo.varnish-
                    cache.org/redhat/varnish-3.0.el6.rpm\n")
    os.system("rpm --nosignature -i https://repo.varnish-cache.org/redhat/varnish-
                    3.0.el6.rpm")
    varnishRLib.log("yum install varnish\n")
    os.system("yum install varnish")
    varnishRLib.log("chkconfig varnish on\n")#Arranque varnish al arrancar la máquina
    os.system("chkconfig varnish on")
```

Ilustración 13: Función del ejecutable apacheConf.py que instala Varnish.

En lo referente al protocolo HTTPS, como ya se ha mencionado en anteriores ocasiones, el proxy Varnish no tiene compatibilidad con éste, por lo que a la hora de hacer la integración no se deberá tocar ninguna entrada de configuración que mencione el puerto 443. De esta forma se consigue que todas las web servidas a través de este puerto permanezcan intactas y ajenas a la instalación del proxy, funcionando con total normalidad.

Por último, la mayoría de acciones que realiza el script generan una entrada en el fichero de log del módulo. Gracias a ésto, en el caso de que se produzca algún error durante la ejecución, será más sencillo localizar el fallo y corregirlo como se considere oportuno.

5.4. VarnishApache.py

Pasamos ahora a comentar y estudiar el ejecutable que sigue a *apacheConf.py* según el orden lógico de ejecución. El script *VarnishApache.py* tiene como misión principal la de integrar el recién instalado Varnish con Apache, haciendo que el proxy envíe las peticiones HTTP al puerto y dirección IP a los cuales hemos movido el servidor web. Al igual que con los scripts anteriores, procederemos a detallar las diferentes funciones que componen el archivo en Python para más tarde justificar su utilización.

- *mueveVirtualServers( )*

  Esta función se encarga de poner todos los servidores virtuales encontrados a lo largo del fichero de configuración de Apache en la dirección IP y puerto que el administrador haya indicado previamente en la configuración del módulo. Para ello, el código abre el fichero de configuración de Apache y lo recorre línea a línea comparando cada entrada mediante expresiones regulares, tal y como se ha hecho en situaciones anteriores.

  Se buscan todas las líneas del fichero que contengan la cadena "*<VirtualHost IP:puerto>,* para posteriormente identificar estos valores. En el caso de que el puerto del host virtual sea el 443, se procede a ignorar la entrada y no modificar nada debido a la incompatibilidad de Varnish con el protocolo HTTPS. Por el contrario, si el host no se encuentra en el puerto 443, se actualiza la entrada del fichero manteniendo el mismo formato pero con los valores para la dirección IP y el puerto correspondientes extraídos del fichero *conf.file* de nuestro módulo.

  La función también localiza y envía cada uno de los bloques de configuración de hosts virtuales a una función que se encargará de modificar las entradas *customlog.* Más adelante se puede observar una porción del código de esta función en la ilustración 14.

- *confCustomLog(bloque)*

  Esta función se encarga de buscar las entradas *customlog dentro de un bloque de configuración de host virtual que se le pasa como argumento.* Las entradas se cambian para enviar los archivos de los de cada servidor virtual al directorio raíz de Varnish Cache, identificando cada fichero con el nombre del host virtual para un mejor control de las estadísticas de cara a las tareas de administración.

- *creaVSconfVarnish( )*

  Este fragmento de código basa su funcionamiento en llamadas al sistema operativo. Lo que hace concretamente es, en primer lugar, copiar el fichero VCL genérico incluido por defecto en el módulo, a su ruta correspondiente. En segundo lugar, realiza la misma acción pero esta vez con el fichero de configuración del servicio Varnish.

- *crearBackend( )*

  Esta función es la responsable de conectar Varnish y Apache y lo consigue escribiendo en el fichero VCL del proxy inverso los valores de la dirección IP y puerto a los que se ha movido el servidor web. Se recorre el fichero de forma secuencial hasta encontrar la entrada "*backend default{"* para añadir a partir de esa línea los valores correspondientes indicados en el fichero de configuración de nuestro módulo. Puede verse el código de esta función en la imagen 15.

- *opcionesVarnish( )*

  El código Python dentro de esta función se encarga de escribir dentro del fichero de configuración del servicio de Varnish los valores que el administrador haya introducido en nuestro fichero de configuración (o los valores por defecto en caso de no modificarlo). Haciendo uso de expresiones regulares, busca las entradas correspondientes a "*VARNISH_LISTEN_ADDRESS", "VARNISH_MIN_THREADS", VARNISH_MAX_THREADS", "VARNISH_TTL", "VARNISH_LISTEN_PORT", "VARNISH_STORAGE_SIZE"* y *"VARNISH_STORAGE"* y actualiza los parámetros dentro del fichero, escribiéndolo en disco con sus valores nuevos de configuración.

- *reiniciaServ( )*

  Para terminar, la función que tratamos en este punto se encarga de reiniciar los servicios de Apache y de Varnish con llamadas al sistema. Gracias a esto, las configuraciones aplicadas tienen efecto al momento.

Una vez que conocemos las diferentes funciones que conforman este script, podemos concretar las tareas que se realizan al ejecutar este código y podemos razonar la necesidad de tales tareas.

```
#############################################################################
# Funcion que recorre el fichero de configuracion de apache buscando entradas de VirtualHost
# para moverlos a la IP y puertos nuevos de apache.
#############################################################################
def mueveVirtualServers():
        inibloque=0
        finbloque=0
        bloque=[]
        lineas=varnishRLib.leeFichero(varnishRLib.config["nfConfApache"][0])
        varnishRLib.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 ):
                    varnishRLib.log(" INFO -> Encontrado VirtualHost "+linea)
                    varnishRLib.log(" INFO -> Moviedo Vhost a IP/puerto de apache\n")
                    inibloque=n
                    print "<br>Virtual Host encontrado: "+linea        #Cogemos IP y puerto
                    ini=linea.find(" ")
                    fin=linea.find(":")
                    IPvieja=linea[ini+1:fin] #IP de VHost Apache
                    puertoviejo=linea[fin+1:len(linea)-2] #Leemos el puerto en el que esta
                    if puertoviejo=="443":
                            print "<br>Host SSL omitido"
                            continue
```

Ilustración 14: Extracto de código de la función mueveVirtualServers del ejecutable VarnishApache.py.

En primer lugar, resulta necesario para una completa integración funcional el mover los servidores virtuales al nuevo puerto y dirección IP donde escucha Apache tras moverlo. De este modo, las peticiones web que entren en el servidor podrán llegar a ser atendidas por el host virtual correcto, ya que en el caso de que no coincidieran estos valores, no se podría comunicar peticiones a estas páginas web.

Continuamos con las entradas *custom log* de los servidores virtuales. Un *custom log* es la forma que tiene Apache de generar información de acceso y de control sobre un servidor virtual, ya que de no utilizarlos, el log que se generase desde Apache sería compartido por todos los diferentes servidores virtuales que estuvieran alojados en nuestro equipo. Así, gracias a este mecanismo, el administrador puede tener unas estadísticas más específicas y podrá ejercer las tareas de administración de una forma más personalizada.

```
#########################################################################
# Funcion que crea el backend de varnish para comunicar con apache
#########################################################################
def crearBackend ():
    lineas=varnishRLib.leeFichero(varnishRLib.config["nfDefaultVcl"][0])
    for n in range(0,len(lineas)):
        linea=lineas[n]
        if re.match("backend default {", linea ):
                lineas[n+1]="    .host =\""+varnishRLib.config["IP"][0]+"\";\n"
                lineas[n+2]="    .port =\""+varnishRLib.config["nPuertoAp"][0]+"\";\n"
    varnishRLib.escribeFichero(varnishRLib.config["nfDefaultVcl"][0],lineas)
    varnishRLib.log(" INFO -> Configurado el BACKEND de Varnish hacia APACHE\n")
```

Ilustración 15: Código de la función que inserta el backend en el fichero VCL de Varnish.

Para configurar el proxy inverso Varnish, la solución por la que se ha optado es copiar al sistema unos ficheros que vienen incluidos en el módulo. Estos archivos vienen con una estructura probada previamente para poder garantizar en la medida de lo posible un correcto funcionamiento de todos los servicios. Así, también nos aseguramos de que los ficheros tienen la estructura necesaria para poder ser configurados a posteriori por nuestras funciones, a través de los formularios habilitados para ello. De esta forma otorgamos al módulo más flexibilidad y genericidad, al mismo tiempo que el administrador gana funcionalidad y sencillez de configuración. El fichero de configuración del servicio ha sido explicado con anterioridad. Y en cuanto al fichero VCL, se trata sobre él en secciones posteriores, por el momento no nos importa más allá de conocer que el *backend* debe definirse al principio de éste. Y recordemos que un *backend* es la forma a la que Varnish denomina al servidor web del que tiene que recoger los objetos de las respuestas para los clientes, en nuestro caso, Apache.

Para finalizar, el reinicio tanto de Varnish como de Apache garantiza que las configuraciones aplicadas tengan efecto inmediato, sin necesidad de que el administrador lo tenga que hacer a mano.

5.5. index.cgi

Este archivo CGI es el que genera la página principal que el administrador verá al acceder a nuestro módulo. Su código es sencillo y su principal función es mostrar los tres iconos de acceso a las tres secciones que componen nuestro módulo, en las que el administrador puede realizar todas las tareas de integración disponibles.

Ilustración 16: Captura de cómo se presenta en el navegador el ejecutable index.cgi de nuestro módulo.

Al ejecutarse, crea una página HTML en la que el usuario puede ver el título del módulo, así como tres imágenes, enlazando cada una de ellas a una sección diferente. Estas imágenes se han extraído de otros módulos del panel Webmin para mantener al máximo posible una cierta homogeneidad con el entorno del panel y hacer más agradable e intuitiva la navegación por nuestra solución.

```
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>Varnish como Proxy inverso de Apache</title></head><body>"
print "<table class='header' width=100%><tr><td id='headln2l' width='15%' valign='top'
                                align='left'><a class='ui_link' href='settings.cgi'>Configuracion del
                                                    Modulo</a></td>"
print "<td id='headln2l' width=15% valign=top align=center>"
print "<font size=+2>Varnish como Proxy inverso Apache</font></td>"
print "<td id='headln2r' width=15% valign=top align=right>"
print "<br></td></tr></table>"
```

Ilustración 17: Extracto de código del fichero index.cgi.

La primera imagen o botón, enlaza con el ejecutable *settings.cgi* que permite al administrador personalizar los valores de la integración del módulo. La imagen o botón central enlaza con el archivo *applySettings.cgi* que se encarga de aplicar esos valores a todos los componentes del sistema. Por último, el botón de la derecha ofrece al administrador acceso a un cuadro de texto para editar directamente el fichero VCL de Varnish, el cual veremos más adelante, así como todos esos CGIs enlazados desde esta página principal.

Debajo de esas imágenes que dan acceso a las secciones principales, el administrador también podrá visualizar dos sencillos botones, uno parará el servicio Varnish en el caso de que esté activo, y otro reiniciará el servicio. De esta forma intentamos evitar en la medida de lo posible que el administrador tenga que acceder a la consola del sistema para realizar esas acciones. Estos botones enlazan con el fichero *startstop.cgi* que veremos más adelante.

Cada uno de los elementos sobre los que se puede hacer clic con el ratón, son un elemento HTML tipo *anchor* que enlazan con los CGIs correspondientes.

Todas las imágenes que se deseen usar a lo largo del módulo deben estar en una carpeta dentro del propio directorio del módulo, así se pueden cargar en la página mediante el atributo *src* de los objetos HTML.

5.6. Startstop.cgi

El código del CGI que tratamos en este apartado realiza las acciones de parada y reinicio del servicio Varnish en el sistema. El funcionamiento se basa en el parámetro de entrada que se le pasa al hacerle la llamada.

Ilustración 18: Captura del CGI startstop.cgi en funcionamiento en el navegador.

Primero carga el valor del formulario desde el que se le llama en la variable *form*. A continuación comprueba que el campo "ac" no esté vacío, en cuyo caso mostrará al administrador un mensaje de error indicando que no existe valor para el campo necesario, por lo que no se realizará ninguna acción.

En el caso de que pasemos esa primera comprobación, se evalúa qué valor contiene ese campo "ac". A partir de aquí existen dos posibilidades. La primera es que el valor coincida con la palabra "reinicia", en cuyo caso el CGI manda al sistema la orden *"service varnish restart",* almacenando en nuestro fichero de log del módulo el registro de la acción realizada.

```
if "ac" not in form:
   print "<H1>Error</H1>"
   print "No existe el campo ac para hacer una accion"
   print "<a href='/varnishReverse/index.cgi'>Indice del Modulo</a>"
else:
      print "<p>Orden recibida: ", form["ac"].value+"</p>"
      if form["ac"].value=="reinicia":
         print "Reiniciando...<br>"
         varnishRLib.log(" INFO -> Reinicio desde
                        WEBMIN"+str(subprocess.call(["service","varnish","restart"])))
         print "Reiniciado<br>"
      if form["ac"].value=="para":
         print "Parando...<br>"
         varnishRLib.log(" INFO -> Servicio parado desde
                        WEBMIN"+str(subprocess.call(["service","varnish","stop"])))
         print "Parado<br>"
```

Ilustración 19: Extracto de código del ejecutable startstop.cgi.

La segunda posibilidad es que el valor del campo coincida con la palabra "para". Sí es así, el ejecutable realizara las mismas acciones que en el caso anterior, pero esta vez la instrucción que le enviará al sistema es la de "*service varnish stop",* que como se puede deducir, para completamente el servicio del proxy.

En cualquier caso, el valor del parámetro de llamada "ac" viene determinado por el botón de la página principal del módulo que pulse el administrador, que es desde dónde se realiza la llamada a esta sección.

5.7. settings.cgi

Este archivo CGI es el que se encarga de mostrar al usuario el formulario (tal y como puede apreciarse en la ilustración número 20) donde podrá introducir los diferentes datos que configurarán la integración en el sistema. Se accede al mismo a través del botón izquierdo de la pantalla principal del módulo, así como desde el enlace superior que reza "Configuración del módulo".

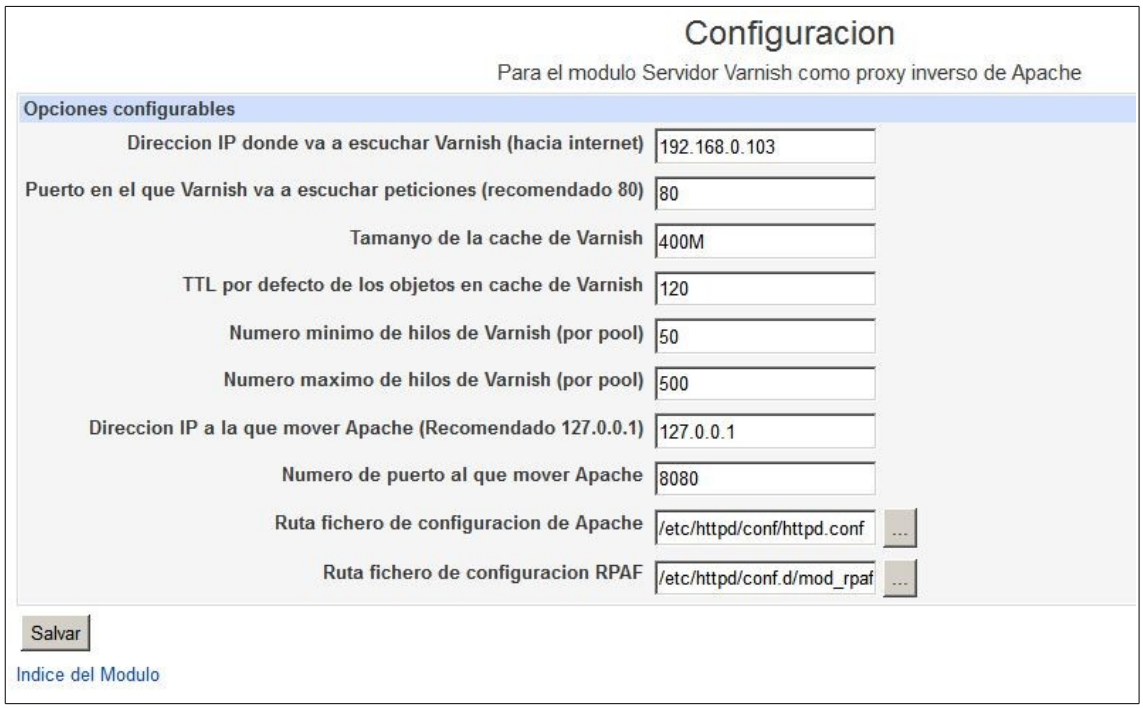

Ilustración 20: Captura del formulario de configuración del módulo.

Al crearse el formulario, los campos de cada parámetro se rellenan con los valores que existan en nuestro fichero de configuración (recordemos que es *conf.file)*. Los distintos campos y sus etiquetas se organizan en una estructura de tabla invisible para el usuario, ofreciendo una sensación de orden.

El formulario HTML está enlazado con el CGI *settingsSave.cgi* a través de su atributo *action,* al cual llamará mediante una petición POST para pasarle todos los valores que haya en los campos en el momento que el administrador pulse el botón de guardar.

```
print "<form class='ui_form' action='settingsSave.cgi' method='post'>"
print "<table class='shrinkwrapper' width=100%><tr><td>"
print "<table class='ui_table' width=100%>"
print "<thead><tr><td><b>Opciones configurables</b></td></tr></thead>"
print "<tbody> <tr class='ui_table_body'> <td colspan=1><table width=100%>"
print "<tr class='ui_form_pair'>"
print "<td class='ui_form_label' width=30% nowrap><b><a name=varnishListenAdd>
                Direccion IP donde va a escuchar Varnish (hacia internet)</a></b></td>"
print "<td class='ui_form_value' colspan=1 ><input class='ui_textbox' type='text'
        name='varnishListenAdd' value='"+varnishRLib.config["varnishListenAdd"][0]+"'
                size=Automatic> </td></tr>"
```

Ilustración 21: Extracto de código del CGI que genera el formulario de configuración del módulo.

Por último también existe un enlace a la página principal del módulo para facilitar la navegación por el mismo, así evitamos el uso del botón atrás del navegador, volviendo a ganar en factor de usabilidad.

5.8. settingsSave.cgi

Este CGI cumple la tarea de actualizar nuestro fichero de configuración (*config.file)* con los valores que le llegan desde el formulario que le realiza la llamada mediante POST. Para ello se basa en las funciones descritas anteriormente de la librería *varnishRLib.py.*

La lógica de funcionamiento del código en Python consiste en cargar en una variable *form* los valores de entrada del ejecutable, que como hemos dicho se corresponderán con los valores que el administrador haya introducido en el formulario visto anteriormente *settings.cgi*. Una vez tenemos los valores cargados en la variable, los vamos analizando uno a uno y actualizando las entradas correspondientes del fichero de configuración, y en el caso de que alguno de ellos esté vacío (por un error de la persona que introdujo la información en el formulario), no se hace nada para evitar errores, pues es preferible un parámetro con un valor por defecto a un parámetro vacío.

```
form = cgi.FieldStorage() #Cargamos campos que recibe como parametro
if not form["nfConfApache"].value=="" : #Si no valor no hacemos nada
    varnishRLib.config["nfConfApache"][0]=form["nfConfApache"].value
if not form["nfConfRPAF"].value=="" :
    varnishRLib.config["nfConfRPAF"][0]=form["nfConfRPAF"].value
if form["nPuertoAp"].value!="" and form["nPuertoAp"].value!="80" :
    varnishRLib.config["nPuertoAp"][0]=form["nPuertoAp"].value
if form["IP"].value!="" :
    varnishRLib.config["IP"][0]=form["IP"].value
if form["varnishListenAdd"].value!="" :
    varnishRLib.config["varnishListenAdd"][0]=form["varnishListenAdd"].value
```

Ilustración 22: Extracto de código del fichero settingsSave.cgi que guarda la configuración en config.file.

Por ultimo, el CGI ejecuta la función de nuestra librería *guardaConf( ),* que como ya dijimos se encarga de escribir el fichero con los nuevos valores en disco para segurar su persistencia.

Una vez realizado todo esto, se muestra al administrador a través del navegador una sencilla página HTML con un mensaje de confirmación indicando que las operaciones se han realizado correctamente. Así como un enlace que devuelve la navegación al índice del módulo.

5.9. applySettings.cgi

A este CGI el administrador accede a través del botón central de la pantalla de índice del módulo, y como su nombre indica, se encarga de aplicar las configuraciones de la integración al sistema en el que está instalado el módulo. Para ello hace uso de los ejecutables vistos anteriormente *apacheConf.py* y *VarnishApache.py.* Lo que hace internamente este CGI es cargar ambos scripts en memoria, para posteriormente lanzar a ejecución las funciones *main( )* de

ambos. De esta forma se consiguen realizar todas las tareas necesarias tanto por una lado para preparar el sistema instalando Varnish y el módulo de Apache RPAF en el caso de que no lo estén previamente, como por otra parte aplicar a ambas herramientas las configuraciones que figuren en el fichero de configuración de nuestro módulo tal y como se ha explicado en las secciones donde tratábamos estos scripts analizando su contenido y funcionamiento.

```
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 Varnish como Proxy inverso de Apache</title></head><body>"
apacheConf.main() #lanzamos script instala Varnish proxy de apache
VarnishApache.main() #lanzamos script configura VHost de Apache
print "<br>Reiniciando VirtualMin(Webmin)"
print "<br><br>"
print "<a href='/varnishReverse/index.cgi'>Indice del Modulo</a>"
print "</body></html>"
sys.stdout.flush()
varnishRLib.log(" INFO - Reiniciando WebMin\n")
subprocess.call(["service","webmin","restart"])
```

Ilustración 23: Extracto de código del fichero applySettings.cgi

A la hora de acceder a la página HTML que genera este CGI, el administrador verá en la ventana una serie de sentencias informativas sobre las diferentes acciones que se estén ejecutando en segundo plano. A la finalización de las cuales, como es habitual en nuestro módulo, se muestra un hiperenlace que nos lleva a la página principal del modulo si lo pulsamos.

Hay que destacar que al final de la ejecución del código de nuestro fichero CGI, se procede a lanzar al sistema la orden de reiniciar el servicio Webmin mediante la instrucción "*service webmin restart*". Debido a esto, la mayoría de las veces la ejecución es tan rápida que el navegador no es capaz de mostrar todas y cada una de las sentencias informativas que mencionábamos anteriormente, aunque sí se registran todas las acciones en el conocido fichero de log de nuestro módulo de integración.

```
Parando httpd: [ OK ] Iniciando httpd: [ OK ] Stopping Varnish Cache: [ OK ]
Comprobando configuracion de Apache...
Listen 127.0.0.1:8080
Apache escuchando en puerto: 8080
Listen 192.168.0.103:443
Apache escuchando en puerto: 443
Listen SSL omitido
NameVirtualHost 127.0.0.1:8080
IP127.0.0.1
NameVirtualHost 127.0.0.1:8080
```

Ilustración 24: Captura de parte de los mensajes informativos que proporciona el módulo al aplicar la integración.

5.10. editFile.cgi

Este es el CGI cuya ejecución muestra al administrador un cuadro de texto en el que poder editar directamente el fichero que controla el comportamiento de la cache de Varnish, es decir, el fichero VCL del que se habla más adelante. Para acceder a la pantalla que genera el script, el administrador deberá pulsar en el icono de la derecha del todo de la ventana principal de nuestro módulo de Virtualmin.

```
fichero=varnishRLib.config["nfDefaultVcl"][0]
print "<form class='ui_form' action='editFileSave.cgi' method='post'
                                                enctype='multipart/form-data'>"
print "<input class='ui_hidden' type='hidden' name='file' value='"+fichero+"'>"
print "<textarea class='ui_textarea' name='data' rows='20' cols='80' style='width:100%'>"
cont=varnishRLib.leeFichero(fichero)
for linea in cont:
    linea=linea.replace("\n","")
    print linea
#Cargamos las lineas del fichero en el cuadro de texto
print "</textarea><br>"
print "<table class='ui_form_end_buttons' ><tr>"
print "<td><input class='ui_submit' type='submit' name='save' value='Salvar'>"
print "</td>"
print "</tr></table>"
```

Ilustración 25: Extracto de código del fichero editFile.cgi que carga un fichero en pantalla.

Al ejecutar este fichero, en el navegador de la persona que esté administrando el servidor aparecerá un cuadro de texto, más concretamente un objeto HTML *textarea,* tal y como puede verse en la imagen 26. El script se encarga de rellenar ese cuadro de texto con el fichero VCL que esté especificado en el atributo *nfDefaultVcl* de nuestro archivo de configuración del módulo.

El código simplemente abre el fichero y lo recorre de forma secuencial, copiando cada línea del mismo dentro del elemento *textarea.*

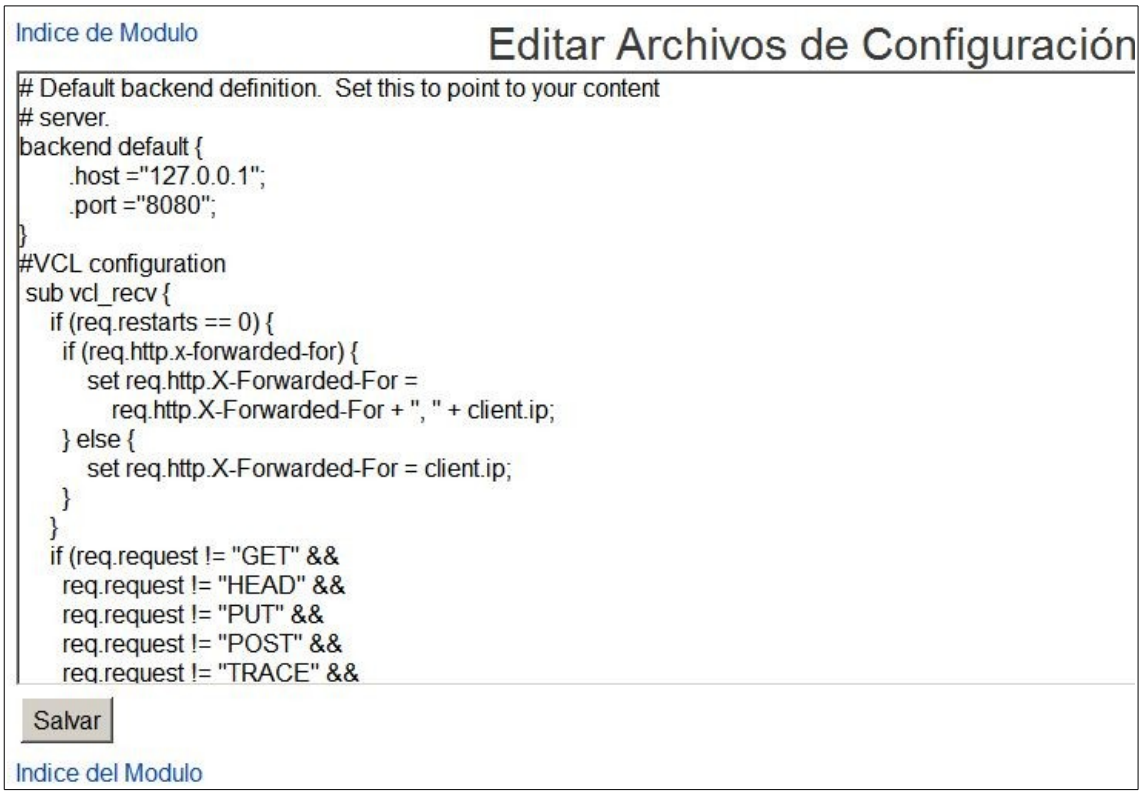

Ilustración 26: Captura de la ventana de nuestro módulo que permite editar el fichero VCL de Varnish.

Al mismo tiempo, se le muestra al administrador un botón que lanza a ejecución el CGI *editFIleSave,* del cual hablaremos a continuación, pero cuyo trabajo a grandes rasgos consiste en guardar los cambios que se hayan hecho a través del cuadro de texto.

Por último se muestra un enlace al índice del módulo, para permitir a la persona que esté navegando poder retornar a la pantalla principal sin utilizar los botones de su navegador web.

5.11. editFileSave.cgi

Éste es el CGI que se lanza a ejecución cuando el administrador pulsa el botón para guardar los cambios efectuados en el fichero de Varnish VCL desde la ventana generada por el script *editFile.cgi.*

En el código del fichero observamos que en primer lugar se comprueba que a través de la llamada que recibe el script se envíe una variable llamada *file*. En caso contrario se detiene la ejecución y no se hace nada.

```
form = cgi.FieldStorage() #Se cargan los parametros
if "file" not in form and "data" not in form :
    exit()#Si no hay fichero ni datos salimos
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 "<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' />"
```

Ilustración 27: Extracto de código del fichero editFileSave.cgi.

En caso de que, efectivamente, tengamos esa variable, cargamos su contenido en la variable *form*. Así, posteriormente, mediante el uso de las funciones de nuestra librería *varnishRLib* guardamos todo el contenido recibido en el fichero VCL que indique la entrada *nfDefaultVCL* de nuestro archivo de configuración.

Como paso final, el código manda al sistema la orden de reiniciar el servicio Varnish para que los cambios tengan efectos inmediatos.

```
Stopping Varnish Cache: [ OK ] Starting Varnish Cache: [ OK ] Guardando configuracion en
/etc/varnish/default.vcl

Guardado

Reiniciando Varnish...
Reiniciado
Indice del Modulo
```

Ilustración 28: Captura de los mensajes de confirmación que genera el módulo al guardar los cambios en el fichero VCL.

Durante todo el proceso se muestran en el navegador sentencias informativas de las acciones que se llevan a cabo en segundo plano, guardando también registro de estas acciones en el fichero de log de nuestro módulo.

Al final de la ventana que ve el administrador, como es habitual en el entorno de nuestro módulo, se muestra un enlace a la pantalla principal, es decir, a *index.cgi.*

5.12. module.info / install_check.pl

Estos dos ficheros son de obligada inclusión a la hora de desarrollar cualquier módulo para el panel Webmin y ayudan al propio panel a integrar el módulo y gestionarlo.

En primer lugar, el fichero *module.info* contiene los metadatos referentes a nuestro módulo. Se compone de cuatro atributos seguidos del símbolo "=" y del valor que necesitemos darles para nuestro módulo.

En nuestro caso los atributos y sus valores son:

- *desc=Vanrish proxy inverso*

  Este es el nombre identificativo de nuestro módulo, es decir, con qué nombre va a aparecer en la lista dinámica de módulos de Webmin.

- *os_support=*-linux*

  Esta entrada indica los sistemas operativos para los que nuestro módulo es compatible, por ello soporta múltiples valores separados por comas. En nuestro caso el sistema operativo para el que ofrecemos compatibilidad es cualquier distribución Linux.

- *version=0.9*

  Como se puede deducir por en nombre, en esta entrada del fichero indicamos el número de versión de nuestro módulo. En nuestro caso hemos optado por ofrecer una versión previa a la 1.0, es decir, una versión beta, pues aún con todo el desarrollo correcto, sería recomendable darle varios meses de rodaje para pulir posibles errores que se hayan escapado y añadir funcionalidades en las que no hayamos caído.

- *category=servers*

  Esta línea del archivo sirve para indicar la sección de los menús desde la cual se podrá acceder a nuestro módulo. En nuestro caso, como el objetivo es gestionar servidores virtuales hemos decidido incluirla en la sección de servidores, donde puede encontrarse también el módulo de Apache.

Por último, según la documentación de Webmin, el fichero *install_check.pl* es necesario para que el propio panel pueda verificar que el módulo se ha instalado correctamente, para lo cuál deberá devolver el valor 0 al ejecutarse. Así que simplemente hemos incluido la instrucción *return 0.*

5.13. Fichero varnish

Este archivo, que durante la instalación del módulo se copia al directorio de Varnish en */etc/sysconfig/varnish/,* tiene definidas las diferentes variables de entorno que utiliza el servicio del proxy para arrancar de la manera deseada. Puede verse un estracto del fichero en la imagen 29. Los valores de sus entradas vienen definidos por defecto para una configuración estándar. Pero recordemos que la mayoría de esos parámetros son configurables a través del formulario de configuración de nuestro módulo. Ya se han estudiado la inmensa mayoría de entradas que contiene previamente cuando explicamos nuestro fichero de configuración *config.file.*

```
VARNISH_STORAGE_SIZE=300M
#Backend storage specification
VARNISH_STORAGE="file,${VARNISH_STORAGE_FILE},${VARNISH_STORAGE_SIZE}"
#Default TTL used when the backend does not specify one
VARNISH_TTL=120

DAEMON_OPTS="-a ${VARNISH_LISTEN_ADDRESS}:${VARNISH_LISTEN_PORT} \
        -f ${VARNISH_VCL_CONF} \
        -T ${VARNISH_ADMIN_LISTEN_ADDRESS}:${VARNISH_ADMIN_LISTEN_PORT} \
        -t ${VARNISH_TTL} \
        -w ${VARNISH_MIN_THREADS},${VARNISH_MAX_THREADS},$
                                                {VARNISH_THREAD_TIMEOUT} \
        -u varnish -g varnish \
        -S ${VARNISH_SECRET_FILE} \
        -s ${VARNISH_STORAGE}"
```

Ilustración 29: Extracto del código del fichero de configuración del servicio Varnish

5.14. Generación e instalación del módulo

Un fichero de instalación de cualquier módulo de Webmin es un directorio comprimido en formato *.tar* al que le cambiamos la extensión por *.wbm.* Con esto en mente, en nuestro caso hemos utilizado el programa *7zip* [20] para comprimir el esquema de directorios y ficheros que hemos mencionado en todos los puntos anteriores y del cual se puede ver un esquema en la siguiente ilustración. De esta forma conseguimos generar con éxito el archivo que necesita el panel de control para instalar correctamente nuestro módulo en el sistema. En nuestro caso generamos el fichero *varnishReverse.wbm.*

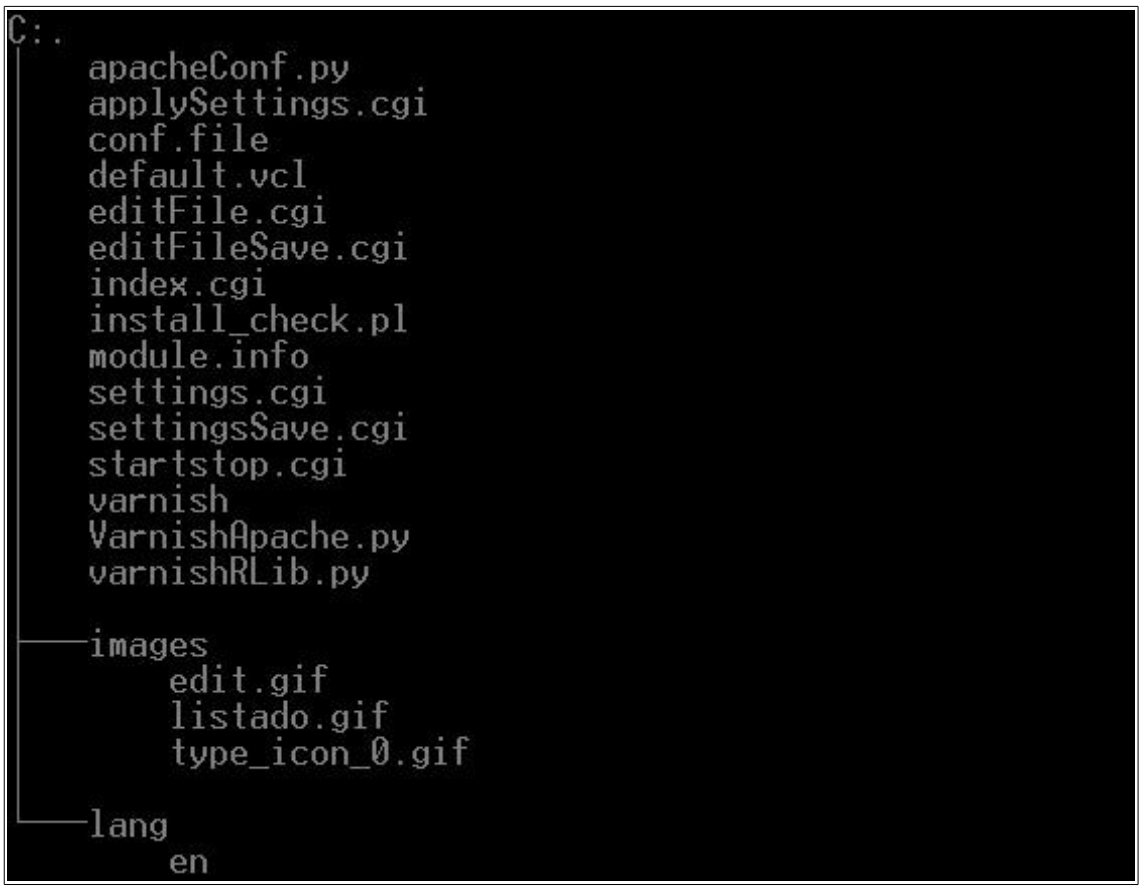

Ilustración 30: Esquema en árbol del directorio de ficheros y carpetas que se comprime para generar el fichero .wbm.

Posteriormente hemos procedido a instalar nuestro desarrollo en la máquina virtual CentOS, que recordemos, ya tiene instalado un panel Virtualmin y un servidor web Apache. Los pasos para realizar la instalación del módulo se ilustran a continuación.

En primer lugar debemos acceder al panel de control Virtualmin desde el navegador de la máquina donde tenemos el archivo *.wbm* y autenticarnos como un usuario válido para realizar las tareas de administración. Para ello ingresaremos en un navegador web la página *https://192.168.0.103:10000* (en el caso de que el sistema CentOS tenga una dirección IP diferente será esa la que debamos ingresar). Entonces nos aparecerá la ventana donde podremos loguearnos con nuestras credenciales, tal y como puede verse en la imagen 7.

El paso siguiente será navegar hasta la sección de Webmin desde la que se gestiona la configuración del propio panel. Para ello haremos clic en el botón de acceso a Webmin que se encontrará en la esquina superior izquierda de la pantalla, y a continuación desplegaremos el submenú lateral llamado “Webmin”, llegando a la pantalla que puede verse en la imagen 31.

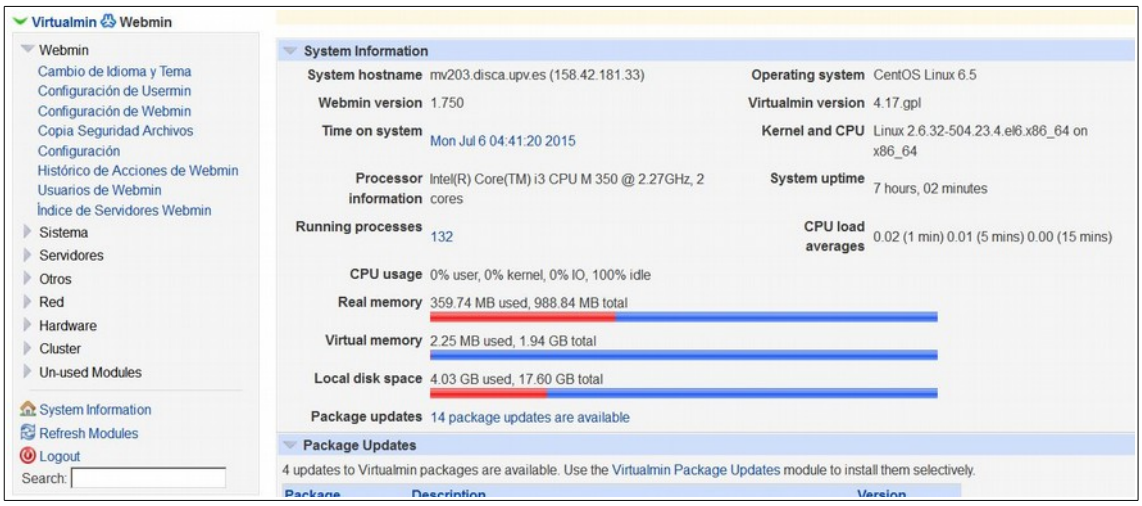

Ilustración 31: Captura de la página principal de Webmin con el primer menú lateral desplegado.

Una vez ahí, como administradores deberemos acceder a la sección "Configuración de Webmin" y nos aparecerá una pantalla con todos los iconos que dan acceso a las diferentes opciones de configuración del panel, tal y como muestra la siguiente imagen.

Ilustración 32: Captura de el menú de configuración principal del panel Webmin.

El botón que nos interesa para instalar nuestro módulo es el icono llamado "Módulos de Webmin", pulsaremos en él y accederemos a la pestaña desde la que se añaden nuevas extensiones a nuestro panel. En esta pestaña aparecen numerosas opciones a través de las cuales podríamos instalar nuestro módulo, pero en nuestro caso concreto, la más cómoda es la alternativa "Desde archivo a cargar". Así pues, activamos la opción y hacemos clic en el botón "Examinar...". Se nos desplegará un navegador de archivos y seleccionaremos nuestro módulo que, recordemos, empaquetamos anteriormente como "varnishReverse.wbm". A continuación deberemos pulsar sobre el botón "Instalar módulo".

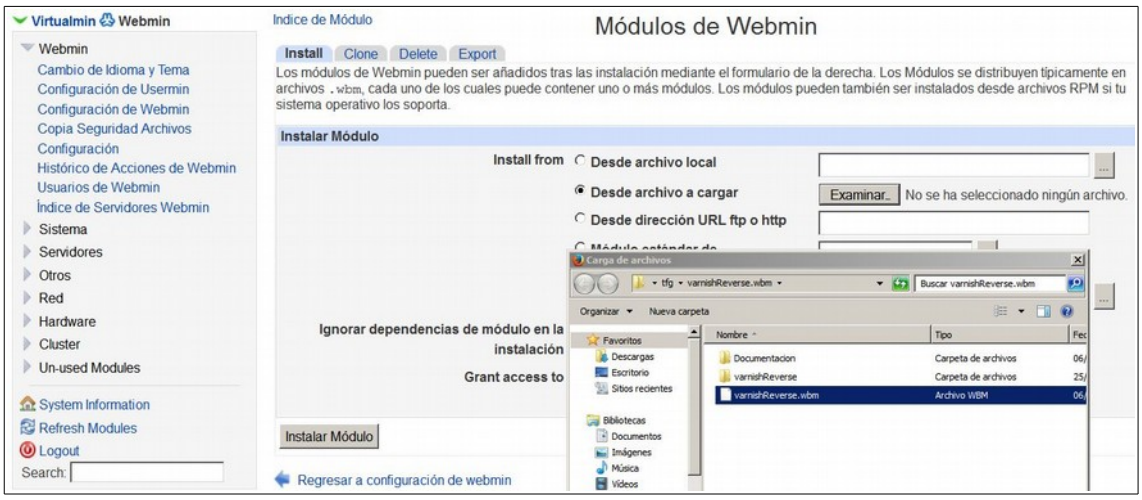

Ilustración 33: Captura del desplegable desde el cual subimos nuestro módulo en formato .wbm

Cuando la instalación se complete, el panel nos mostrará un mensaje confirmándonos que el proceso ha concluido con éxito. En ese mismo mensaje tendremos un enlace que podremos pinchar para ir a la página principal de nuestro módulo.

Ilustración 34: Mensaje de confiramción de Webmin tras instalar con éxito nuestro módulo.

A partir de entonces podremos encontrar nuestra extensión para Varnish en el panel de Webmin, dentro del menú desplegable de "Servidores", pues esa es la sección que indicamos dentro del fichero *module.info* tal y como veíamos unas páginas atrás (y como puede verse en la ilustración 35). Y así es como instalamos nuestro módulo en el panel de control Virtualmin.

Ilustración 35: Localización del acceso a nuestro módulo dentro del panel de control webmin.

6. VCL

La forma de configurar las políticas de cacheo de Varnish es a través del lenguaje de configuración de Varnish (VCL). La forma de cargar y aplicar estas políticas a nivel interno empieza con el proceso de administración, que interpreta el fichero a lenguaje C y posteriormente le pasa el nuevo código a un compilador, GCC en el caso de nuestra máquina. Por último el código resultante se enlaza a la instancia Varnish.

Como resultado de ésto, cambiar la forma en la que el programa trata las peticiones consume muy pocos recursos y además, al efectuarse la compilación fuera del proceso hijo, no se corre el riesgo de que el cambio afecte negativamente a cualquier tratamiento de una petición que esté en curso.

Antes de entrar de lleno con el código VCL, hay que remarcar que es un lenguaje de domino, lo cual quiere decir que a lo largo del tratamiento algunos datos solo son accesibles dentro de ciertas partes de la secuencia de ejecución, la información la detallaremos más adelante. A continuación exponemos cuatro premisas que es importante tener presentes a la hora de programar en VCL.

- Cada petición se procesa por separado.
- Cada petición es completamente independiente de cualquiera que se ejecute al mismo tiempo, antes o después de ésta.
- *return(x);* hace que la petición salga del estado actual y pase al siguiente.
- Los diferentes estados por los que pasa una petición están relacionados pero aislados unos de otros.

Ahora que hemos hablado de estados tenemos que intentar aclarar qué son exactamente. Cuando Varnish procesa una petición, comienza analizándola gramaticalmente, verificando que efectivamente sea una petición HTTP válida. Una vez se completa este proceso, la petición es enviada al estado *vcl_recv,* donde se tratará y se decidirá si se pasa a un estado siguiente o a otro. Todo esto se entiende mejor con el diagrama de flujo del procesamiento de peticiones en la siguiente ilustración.

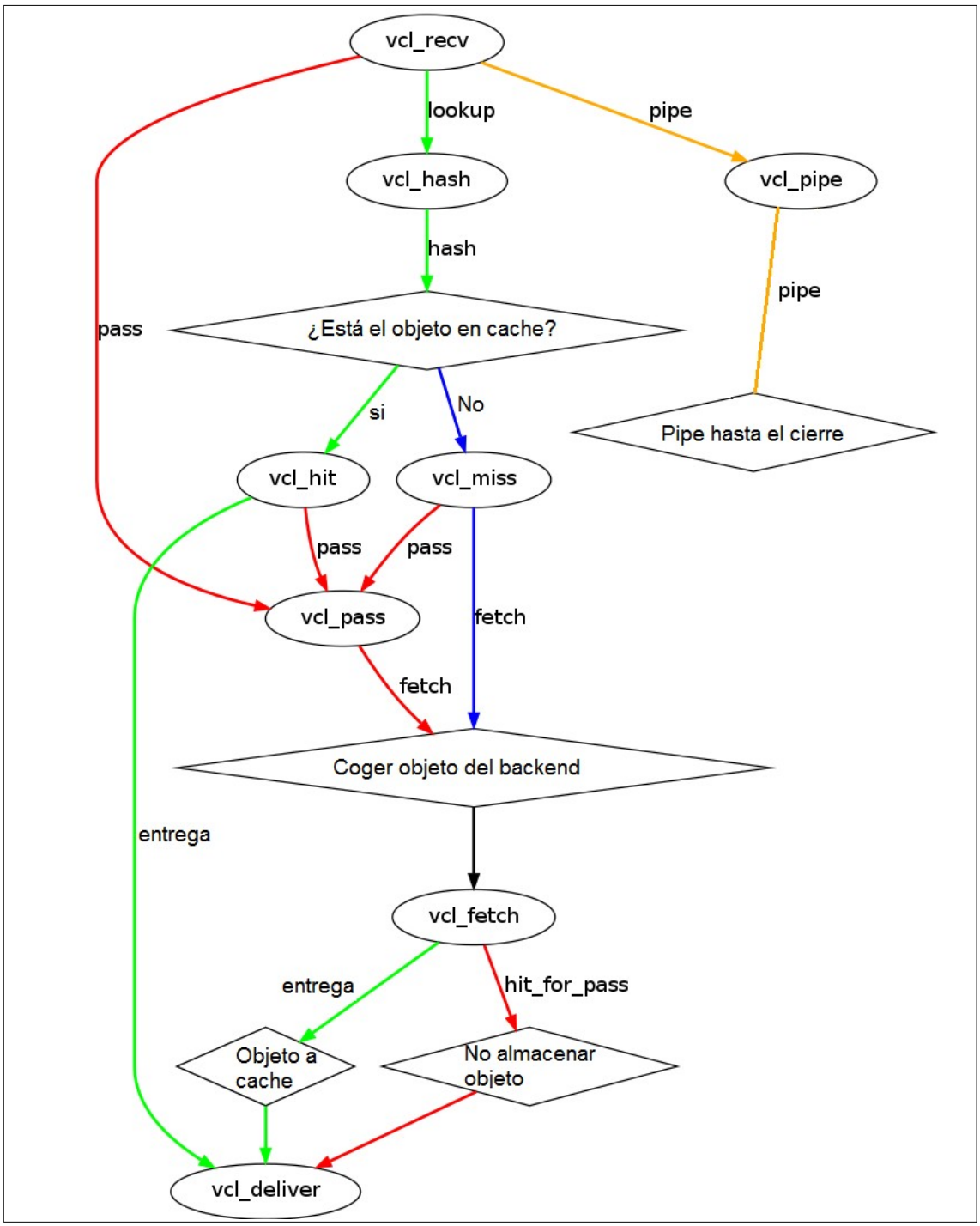

Ilustración 36: Diagrama de flujo de ejecución del fichero VCL de Varnish.

6.1. Disponibilidad de las variables en VCL

Tal y como hemos dicho anteriormente, Varnish crea variables tanto de las peticiones del cliente como de las respuestas que recibe de nuestro servidor, así como de la respuesta que acaba enviando al cliente. Lógicamente, estas variables no se encuentran disponibles por igual a lo

largo de todas las funciones del diagrama de flujo de Varnish. Por eso, y como medida aclaratoria que nos servirá para poder comprender mejor nuestro fichero VCL, hemos añadido una tabla donde explicamos la disponibilidad de las diferentes variables, así como la posibilidad de leer (R) o de escribir sobre ellas (W).

Variable	recv	fetch	pass	miss	hit	error	deliver	pipe	hash
req.*	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
bereq.*		R/W	R/W	R/W				R/W	
obj.hits					R		R		
obj.ttl					R/W	R/W			
obj.grace					R/W				
obj.*					R	R/W			
beresp.*		R/W							
resp.*						R/W	R/W		

Vamos a intentar explicar a grandes rasgos las variables principales, lo cual nos permitirá terminar de utilizarlas mejor durante el desarrollo de nuestro fichero VCL.

- *req*

  Este es el objeto “petición” (*request* en inglés) original. Es decir, lo que le llega a Varnish directamente del cliente.

- *bereq*

  Esta variable es la petición tal y como se la envía Varnish a nuestro servidor web Apache, es decir, es la petición que le llega al *backend (backend request* en inglés). Adelantamos que podría ser ligeramente diferente a la petición original. Por ejemplo podríamos convertir las peticiones HEAD en peticiones GET.

- *beresp*

  Este objeto, para Varnish, es la respuesta tal cual nos la manda el servidor web Apache (*backend response* en inglés).

- *obj*

  Esta variable es el siguiente según la lógica de procesamiento de VCL. Se genera a partir de *beresp* y las características de éste que nos convenga, y por definición, es el objeto tal y como lo almacenamos en la cache.

- *resp*

  Es el objeto respuesta que definitivamente vamos a enviar al cliente. Se genera a partir de *obj*.

Tal y como hemos visto, los objetos se generan a partir de otros, siempre según nuestras necesidades y según lo especifiquemos en el fichero VCL, por lo que hay que tener en cuenta que, por ejemplo, cualquier modificación de *beresp* podrá tener efecto en el *obj* correspondiente, y posteriormente en el *resp* generado.

6.2. Backend y heath-checks

Un *backend* en Varnish es un servidor web al que se le pasarán las peticiones y del que se recogerán las respuestas. En nuestro caso concreto nuestro *backend* es el servidor web Apache, pero en otras situaciones típicas del mundo de los servidores web, una máquina que ejecuta Varnish, podría tener dos o más *backends* definidos, siendo estos maquinas completamente distintas a las que les manda las peticiones, realizando una labor de balanceador de carga.

El proxy inverso Varnish también soporta una herramienta con una gran utilidad. Si el usuario lo define, el programa es capaz de comprobar el estado del servidor web con el que trabaja y actuar en consecuencia. De esta forma, podemos por ejemplo, aumentar el tiempo de los objetos en cache si detectamos que Apache está caído, para que los usuarios puedan seguir navegando y nosotros atender a la incidencia técnica cuanto antes. Esta técnica se denomina *heath check* y la utilizamos en nuestro fichero VCL por defecto.

6.2.1 Código por defecto

A continuación se detalla el código que incluimos por defecto en el fichero VCL referente a la definición del *backend* y a los *health-checks* para después explicar su funcionamiento.

```
#Declaracion del backend. El servidor web
backend default {
 .host = "192.168.0.103";
 .port = "8080";
      #Instrucciones para comprobar que apache esta arriba
 .probe = {
      .url = "/"; /*URL de consulta*/
      .interval = 10s;/*Tiempo entre consultas*/
      .window = 8;/*Consultas a tener en cuenta*/
      .threshold = 4;/*Consultas positivas para OK*/
      }
}
```

Ilustración 37: Código que se aplica por defecto al realizar la integración con el módulo refrente al backend y los health-checks.

Como podemos apreciar en la figura 37, nuestro código por defecto define un *backend* en una dirección IP que el administrador deberá cambiar casi seguro desde el formulario de configuración. Lo mismo sucede con el puerto.

Por último, se añade un *health-check* genérico, que hará que Varnish consulte a apache en la URL base de todos los servidores a los que lleguen peticiones. Hará las consultas en intervalos de diez segundos y para determinar el correcto funcionamiento o no del servidor web, tendrá en cuenta las ultimas 8 comprobaciones realizadas, declarando al servidor Apache caído si más de cuatro resultan fallidas.

Como puede apreciarse, los valores corresponden a situaciones genéricas para ofrecer una funcionalidad eficaz en la mayoría de entornos, pero siempre será una buena práctica que el administrador del servidor ajuste esos parámetros a sus necesidades.

6.3. vcl_recv

Tal y como se puede ver en el anterior diagrama de flujo, cuando una petición entra a Varnish, éste ejecuta la función *vcl_recv*. En esta sección del tratamiento de peticiones es el lugar indicado para realizar ciertas acciones siempre que sea de interés para nuestro servidor y las páginas que tenemos alojadas en él. Entre ellas destacamos:

- Modificar y normalizar los datos procedentes del cliente para reducir la redundancia en la cache, por ejemplo eliminando “www” de las URLs.

- Decidir las políticas de cacheo de objetos, por ejemplo no almacenar en cache peticiones POST o solo almacenar ciertas URLs.
- Ejecutar reglas de reescritura de algunos datos de cliente utilizados por algunas aplicaciones web específicas.
- Añadir barreras de seguridad para evitar ataques.

Todas estas situaciones han sido tratadas e implementadas más adelante, recordando que nuestro desarrolla busca ser una solución general intentando abarcar la máxima funcionalidad posible para el máximo numero de servidores posibles sin intentar resolver situaciones muy concretas que puedan responder a necesidades de ciertos servidores y sus páginas.

Volviendo a *vcl_recv,* una vez realizadas las acciones anteriores, es posible tomar tres caminos diferentes.

En primer lugar podemos hacer un *pass,* que supone ejecutar el resto de la lógica de procesamiento pero sin buscar coincidencias en la cache y sin tratar de almacenar la respuesta generada. Es decir, se pedirá al *backend* la respuesta y se le enviará al cliente, sin más.

Otra opción, llegados al final de *vcl_recv,* es hacer un *pipe.* Ésto no es ni más ni menos que crear una conexión directa entre nuestro servidor *backend* y el cliente. Toda la transmisión a través de la *pipe* será interpretada por Varnish como simples bytes que retransmitirá a un lado o a otro sin intentar comprenderlos. Por este motivo, cualquier petición que se genere a consecuencia del *pipe* mientras el tiempo *keep-alive connection* siga activo pasará también por el proxy con la misma técnica.

La opción que más nos interesa es buscar (o hacer un *lookup*) de la petición en la cache, respondiendo con la respuesta correspondiente en caso de que se encontrara o almacenando la respuesta generada por el *backend* en el caso de que no se encontrara en la cache.

Por último, en *vcl_recv* podemos generar una respuesta de error desde Varnish, indicando una redirección o cualquier cosa que nos interese. Ésto se hace con *error.*

6.3.1 Código por defecto

Procedemos a ilustrar el código de la función *vcl_recv( )* que incluimos por defecto al aplicar la integración con nuestro módulo.

```
#VCL configuracion
sub vcl_recv {

    #Anñadimos a la peticion la IP del cliente para estadisticas
    if (req.restarts == 0) {
       if (req.http.x-forwarded-for) {
                set req.http.X-Forwarded-For =
                req.http.X-Forwarded-For + ", " + client.ip;
       }
    else {
                set req.http.X-Forwarded-For = client.ip;
       }
    }
    if (req.request != "GET" &&
    req.request != "HEAD" &&
    req.request != "PUT" &&
    req.request != "POST" &&
    req.request != "TRACE" &&
    req.request != "OPTIONS" &&
    req.request != "DELETE") {
    /* Peticion no exitente en RFC2616*/
        return (pipe);
    }

    if (req.request != "GET" && req.request != "HEAD") {
    /*Solo cacheamos GET y HEAD*/
       return (pass);
    }
    if (req.http.Authorization || req.http.Cookie) {
    /*Si hay cookie o se negocia una conexion no cacheamos*/
       return (pass);
    }

    #Controlamos que el servidor esta arriba
    if (req.backend.healthy) {
       set req.grace = 30s;
    }
    else { /*Si Apache down, grace y aguantamos los obj 1h*/
       set req.grace = 1h;
    }
  return (lookup);
}
```

Ilustración 38: Código que incluimos por defecto en la función vcl:recv( ).

La primera parte del código es la que hace uso del mencionado módulo RPAF de Apache. Lo que hace es añadir en una cabecera de la petición que va a enviar al servidor web, la dirección IP del cliente que originó la petición en primera instancia. Gracias a esto, como hemos comentado anteriormente, las estadísticas de Apache son más completas y se registran los accesos con la IP que deben registrarse, y no con la del servicio Varnish, que es lo que sucedería en caso de que no aplicáramos ésta técnica.

En segundo lugar, nuestro código comprueba que la petición entrante se una que cumpla con el estándar HTTP según el RFC2616. En caso contrario, se entiende que Varnish no va a ser capaz de tratar la petición, por lo que crea una conexión directa entre cliente y servidor, es decir, se hace un *pipe.*

El código que sigue se encarga de intentar mandar a cache tan solo las peticiones HEAD y GET, pues no resulta de interés ni utilidad por lo general intentar guardar en cache otros tipos de peticiones como pueda ser POST. También discrimina las peticiones que tienen una *cookie* o que llevan la cabecera de negocio de conexión *authorization*. Si la petición cumple cualquiera de estas condiciones, se derivará como un *pass,* es decir, no intentará recuperar la respuesta de la cache.

Por último vemos dónde se realizan los *health-checks* que hemos definido en la sección anterior. Tal y como puede verse, en caso de que Varnish declare el servidor Apache caído, pondrá los objetos petición en modo *grace* durante una hora*,* es decir, se obviará el tiempo TTL y se mantendrán los objetos relacionados con sus peticiones durante sesenta minutos, dando tiempo al administrador a solucionar las incidencias técnicas. Como resulta evidente, este mecanismo deberá eliminarse en caso de que un sitio web no pueda servir contenido antiguo a los clientes.

6.4. vcl_fetch

La función *vcl_fetch* es la equivalente a *vcl_recv* pero por parte del servidor web o *backend*. Mientras que en una podemos utilizar la diferente información proporcionada por el cliente para actuar en consecuencia con unas políticas de cache u otras, en *vcl_fetch* tenemos la diferencia de que el que nos proporciona información útil es el servidor web.

En esta función se aconseja hacer las siguientes tareas:

- Sobrescribir el tiempo que se mantiene en cache un objeto de una URL determinada.
- Eliminar las cabeceras *set-cookie* que sean dispensables.
- Crear un objeto *hit_for_pass*.

Dentro de la función tenemos disponibles diferentes datos de la respuesta, de los cuales el más importante y usado es *beresp.ttl*. Esta variable asociada a un objeto respuesta del servidor indica a Varnish durante cuánto tiempo se mantendrá el objeto generado en la cache. Por defecto éste valor coincidirá con el valor que le hayamos asignado al servicio Varnish en el campo *default_TTL.*

Para terminar la ejecución de esta función tenemos dos caminos diferentes. Por un lado podemos hacer un *return(deliver),* con lo cual Varnish enviará al cliente su respuesta y la guardará en cache sí es posible. Por otro lado, tenemos la opción de terminar con un *return(hit_for_pass).* Esta instrucción hará que Varnish almacene en cache un objeto de tipo homónimo *hit_for_pass* enlazado a la petición que lo generó. Es muy importante tener en cuenta

que el TTL del objeto *hit_for_pass* se aplica como a un objeto normal. De este modo, cualquier petición posterior que sea igual que la que generó éste objeto, sí entra a Varnish dentro del tiempo TTL del objeto, entrará en la función *pass( )* de forma automática. Es decir, creando un *hit_for_pass* estaremos almacenando en cache la decisión de que esas peticiones hagan un *pass* durante un periodo de tiempo determinado.

6.4.1 Código por defecto

A continuación ilustramos la función *vcl_fetch( )* que incluimos por defecto en nuestro fichero VCL.

```
sub vcl_fetch {
   if (beresp.ttl <= 0s ||
      beresp.http.Set-Cookie ||
      beresp.http.Vary == "*") {
         /* Marcamos como HitForPass durante 2 min*/
         set beresp.ttl = 120 s;
         return (hit_for_pass);
   }
      /*Activamos modo grace*/
      set beresp.grace = 1h;
   return (deliver);
}
```

Ilustración 39: Código de la función vcl_fetch( ) que utilizamos por defecto.

La primera parte de código de nuestra función hace que en caso de que el servidor web Apache conteste una petición con una cabecera *set-cookie* marcaremos esa respuesta como *hit_for_pass* y la acción se recordará para todas las peticiones que generen la misma respuesta durante los próximos dos minutos. Es extremadamente extraño que se desee guardar en cache una respuesta con una cabecera de ese tipo.

Por último, al final del código podemos ver como asignamos un valor de *grace* de una hora a todas las respuestas que se guarden en cache. Recordemos que este tiempo solo se tendrá en cuenta en cuanto se caiga el servidor Apache. Mientras eso no suceda, el valor del atributo TTL es el que manda.

6.5. vcl_hash

En esta función se define la clave de dispersión que será usada para asociar un objeto en cache con la petición que le corresponde. En otras palabras, decide qué es único en una petición. Se ejecuta inmediatamente después de *vcl_recv.*

Un posible uso de esta sección podría ser añadir un nombre de usuario concreto para un objeto de la cache que sea de uso específico para ese usuario, sin embargo, es una práctica arriesgada pues de esta forma dispersaríamos mucho el contenido de la cache, perdiendo eficacia y eficiencia, ya que el objetivo es almacenar en cache los objetos que reciban más peticiones por parte de los clientes en general, no de uno unicamente.

6.5.1 Código por defecto

Detallamos el código que utilizamos para identificar un objeto respuesta con su petición en la cache.

```
sub vcl_hash {

      #Utilizamos la ULR de la peticion
      hash_data(req.url);

      #Y la cabecera HOST si existe
      if (req.http.host) {
          hash_data(req.http.host);
      } else {
          #O la IP del servidor
        hash_data(server.ip);
      }

   return (hash);
}
```

Ilustración 40: Código de la función vcl_hash( ) que incluimos por defecto en el módulo.

El sencillo código de nuestra función de dispersión de Varnish nos proporciona una solución eficaz para la mayoría de sistemas. Las peticiones se diferencian unas de otras según la URL y la cabecera *host* (o la dirección IP de destino en caso de que la petición no tenga dicha cabecera). Gracias a esto obtenemos un correcto funcionamiento de la cache, evitando la redundancia de objetos.

6.6. vcl_hit

El código de esta sección se ejecuta en cuanto un objeto se localiza con éxito en la cache de Varnish. En esta función se puede cambiar el atributo TTL de un objeto, por ejemplo ampliar el TTL en el caso de que nuestro servidor Apache esté caído. También es la función desde la que se pueden realizar purgas de los objetos, ésto es eliminarlos de la cache.

6.6.1 Código por defecto

En el caso de que el objeto se encuentre en la cache, nuestro código simplemente activará la función que se encarga de entregarlo al cliente.

```
sub vcl_hit {
   return (deliver);
}
```

Ilustración 41: Código por defecto que incluimos en la función vcl_hit( )

6.7. vcl_miss

Se ejecuta después de buscar un objeto en la cache y no encontrarlo. La principal función que se le da a esta sección es realizar la purga de los objetos de la cache. En versiones de Varnish anteriores, también se utilizaba esta función para modificar las cabeceras de las peticiones que vamos a enviar al servidor web, pero actualmente esas acciones se realizan en *vcl_recv.*

6.7.1 Código por defecto

Al igual que ocurre con la función *hit( ),* el código de *vcl_miss( )* es muy sencillo pero funcional. Cuando un objeto no se encuentra en cache, se activa la función que se encarga de pedirlo al servidor web Apache.

```
sub vcl_miss {
   return (fetch);
}
```

Ilustración 42: Código por defecto que incluimos en la función vcl_miss( ).

6.8. vcl_pass

Esta función podemos englobarla en el mismo grupo que *vcl_hit* y *vcl_miss.* Su código se ejecuta después de que una búsqueda en cache o la función *vcl_recv* determinan que el objeto no se encuentra en cache, pero que tampoco es posible cachearlo (por ejemplo si tiene *cookies* de usuario). La utilidad de esta función es limitada, pero típicamente se puede utilizar para controlar aspectos que hayas implementado tanto en *vcl_hit* como en *vcl_miss.* El uso más extendido para esta función es evitar que una petición de purga de la cache llegue a nuestro servidor Apache.

6.8.1 Código por defecto

Tal y como sucede con las ultimas dos funciones anteriores, el código de esta función es básico y funcional. Lanza la petición al servidor como *pass* en el caso de que en *vcl_recv( )* se devuelva un *pass* o al buscar el objeto correspondiente en la cache, éste esté marcado como *hit_for_pass.*

```
sub vcl_pass {
   return (pass);
}
```

Ilustración 43: Código por defecto de la función vcl_pass( ).

6.9. vcl_deliver

La función de la que tratamos ahora es el punto de salida de la lógica de ejecución del fichero VCL para todas las funciones menos para *vcl_pipe.* Su utilidad es principalmente testear y debuguear, pues permite modificar directamente la respuesta que envía Varnish al cliente, independiente de sí ha salido de cache o del servidor web Apache. Aquí podemos eliminar cabeceras HTTP que no nos interesen, o añadir las que deseemos (que no habremos añadido antes porque no queremos que se almacenen en cache). Las principales variables que podemos modificar en esta función son las siguientes.

- *resp.http.**

  Conjunto de cabeceras que se van a enviar al cliente. Pueden añadirse o quitarse.

- *resp.status*

  Código de estado de la respuesta HTTP (200, 404, 503, etc).

- *resp.response*

  Mensaje de respuesta, normalmente asociado al código de la respuesta ("*Ok*", "*File not found*", "*Service enavailable*", etc).

- *obj.hits*

  Número de veces que se ha obtenido un objeto con éxito de la cache. Puede ser tratado como una cadena, por lo que no habrá que convertir ningún tipo de datos a otro para usarlo.

- *req.restarts*

  Numero de reinicios que ha provocado una respuesta en la lógica VCL.

6.9.1 Código por defecto

A continuación de ilustra y explica el código por defecto que hemos incluido en la función *vcl_deliver( )*.

```
sub vcl_deliver {
      if (obj.hits > 0) {
         set resp.http.X-Cache = "HIT "+obj.hits;
      } else {
           set resp.http.X-Cache = "MISS";
      }
   return (deliver);
}
```

Ilustración 44: Código por defecto de la función vcl_deliver( ).

Nuestra función *vcl_deliver( )* ayuda al administrador a probar la efectividad de la cache. Lo que hace es añadir en el objeto que se devuelve al cliente, una cabecera HTTP *X-Cache* que valdrá *MISS* en el caso de que el objeto no se haya recuperado de la cache, o *HIT* en el caso de que sí se haya respondido a la petición con el objeto de cache. Además, añadimos el número de veces que se ha conseguido un acierto en el objeto, antes de que sea retirado de memoria cuando acabe su TTL.

6.10. vcl_error

Tal y como el nombre de la función indica, esta función se ejecuta cuando sucede un error a la hora de procesar una petición. Es posible generar una página HTML sintética desde el código de esta sección, por lo que Varnish no tiene por qué recuperar esa página de error que se le va a mostrar al usuario de ningún servidor web concreto.

6.10.1 Código por defecto

El código que incluimos en la función de error de Varnish es el mismo que ofrecen en su documentación como solución estándar. La decisión responde a que al ser un mensaje de error, es muy probable que un administrador desee personalizarlo y adaptarlo a su sitio web para mejorar lo máximo posible la experiencia de los usuarios de su web en el caso de que se produzca una situación de fallo.

```
sub vcl_error {
        set obj.http.Content-Type = "text/html; charset=utf-8";
        set obj.http.Retry-After = "5";
        synthetic {"
           <?xml version="1.0" encoding="utf-8"?>
           <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
           "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
           <html>
           <head>
                   <title>"} + obj.status + " " + obj.response + {"</title>
           </head>
           <body>
                   <h1>Error "} + obj.status + " " + obj.response + {"</h1>
                   <p>"} + obj.response + {"</p>
                   <h3>Guru Meditation:</h3>
                   <p>XID: "} + req.xid + {"</p>
                   <hr>
                   <p>Varnish cache server</p>
           </body>
           </html>
        "};
        return (deliver);
}
```

Ilustración 45: Código por defecto de la función vcl_error( ).

Tal y como puede apreciarse, se trata de una página HTML sintética, es decir, que se genera al vuelo e incluye cierta información referente al objeto que ha causado el error.

7. Pruebas de funcionamiento

7.1. Pruebas de funcionamiento del módulo

Para poder comenzar a probar nuestro módulo, debemos preparar el sistema para crear un entorno donde podamos observar con certeza si el módulo realiza correctamente sus tareas o no. Por este motivo procedemos a arrancar nuestra máquina virtual y nos conectaremos al panel Webmin que ya trae preinstalado.

El primer paso que hemos hecho ha sido poner el servidor web Apache a esperar peticiones en la dirección IP de nuestra máquina virtual y en dos puertos, en el 80 y en el 443. Para conseguirlo podemos proceder de dos formas diferentes. Una de ellas es editar a mano el fichero de Apache donde de se especifica en qué direcciones y puerto va a escuchar peticiones. Y la otra forma es a través de la interfaz gráfica que nos ofrece Webmin para configurar Apache. Independientemente del método que decidamos utilizar, los resultados serán los mismos, pues lo que hace la interfaz gráfica a nivel interno, es precisamente modificar el fichero localizado en */etc/httpd/conf/,* por eso vamos a ilustrar cómo hacerlo desde el entorno gráfico para posteriormente ver los cambios en el fichero.

Sabiendo ésto, vamos al menú desplegable de Webmin llamado "Servidores" y hacemos clic en "Servidor web Apache". Una vez ahí, pinchamos sobre la pestaña "*Global configuration",* en la que nos aparecerán todas las opciones de Apache. Aquí pulsaremos sobre el icono de "Redes y direcciones" para acceder al menú desde el cual editar las direcciones en las que se encuentra Apache, tal y como se ve en la figura 46. Una vez aquí, en la sección del formulario donde dice "Escuchar en direcciones y puertos", introduciremos la dirección IP de nuestra máquina y en el cuadro de puerto pondremos el puerto 80. Haremos lo mismo en el recuadro inferior, solo que esta vez pondremos el puerto 443. Por último haremos clic en el botón inferior para salvar los datos de configuración.

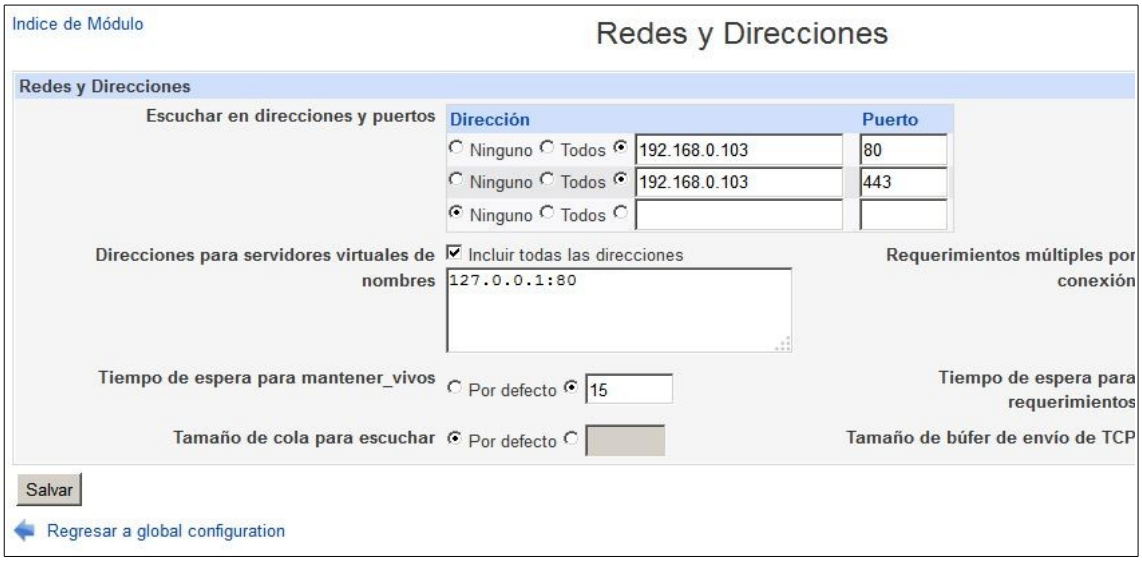

Ilustración 46: Captura del formulario de configuración de "Redes y direcciones" de Apache.

Podemos ver como se traducen estas acciones a entradas del fichero de configuración de Apache, para lo cual tendremos que volver a la pestaña de "*Global configuration"* del servidor web y esta vez pulsaremos sobre el icono de "Editar archivos de configuración". Si buscamos dentro del fichero podremos ver las lineas "*Listen IP:puerto"* que hemos creado a través de la interfaz gráfica.

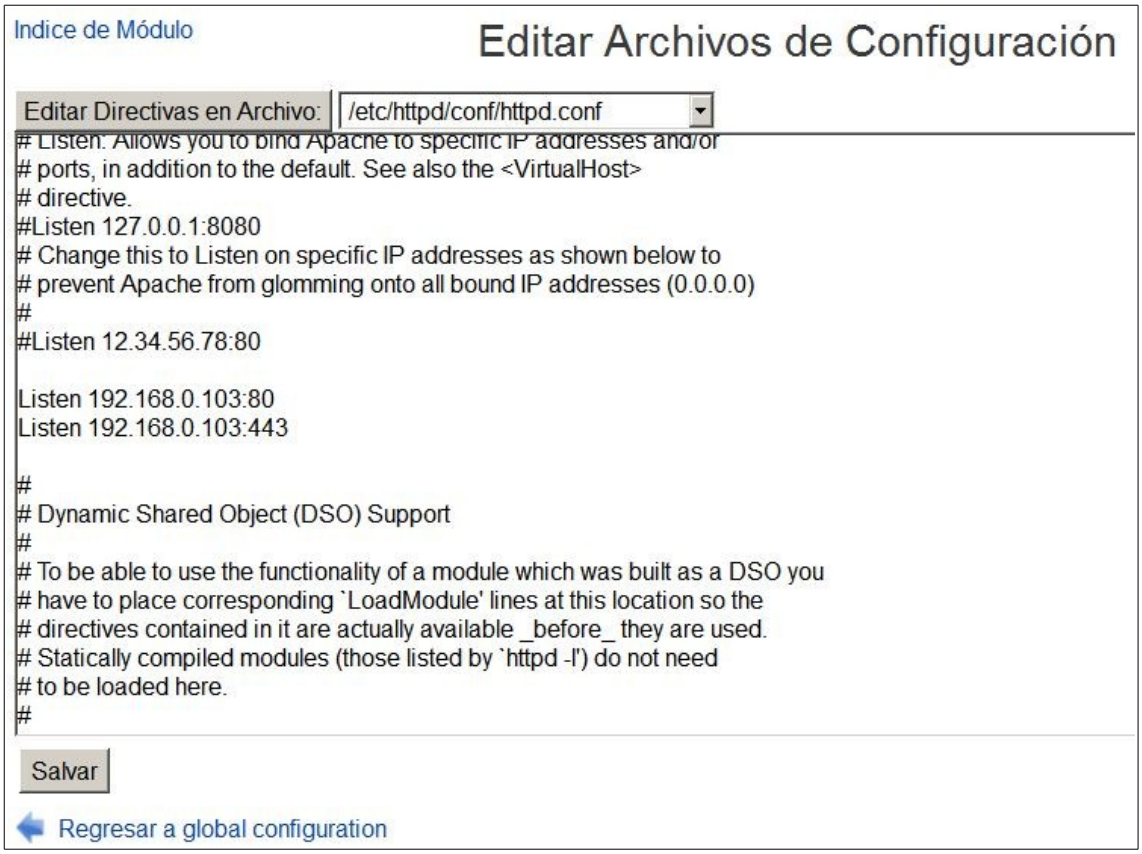

Ilustración 47: Comprobación de que los datos introducidos en el formulario anterior han modificado el fichero de configuración de Apache.

El siguiente paso es crear dos servidores virtuales en Apache. Uno de ellos estará en la dirección IP de la máquina y en el puerto HTTP (puerto 80) y el último lo pondremos en la misma dirección IP de la máquina pero en puerto HTTPS (puerto 443). Para ello accedemos de nuevo al panel de gestión del servidor Apache, y abrimos la pestaña que está situada más a la derecha, que se llama "*Create virtual host"*. Aquí rellenaremos el formulario de creación de servidor virtual, especificando una dirección IP (la de nuestra máquina), un puerto, un directorio raíz de documentos y un nombre para el servidor. Una vez rellenemos todo pulsaremos en el botón "Crear servidor". Repetimos el paso para el segundo servidor y ya tenemos listos nuestros dos servidores virtuales.

Ilustración 48: Listado con los hosts virtuales que hemos creado en Apache para realizar las pruebas del módulo.

Ahora es momento de probar el funcionamiento de nuestro módulo, para corroborar que la integración se realiza correctamente. Accedemos al mismo desde el menú lateral desplegable de Webmin, tal y como vimos en la figura 35. Pinchamos en el primer elemento de la página principal del módulo, es decir, el que reza "Editar opciones de la integración". Una vez ahí, introducimos los datos que consideremos oportunos como administradores. Podemos ver la configuración que hemos aplicado en nuestro caso en la imagen 49.

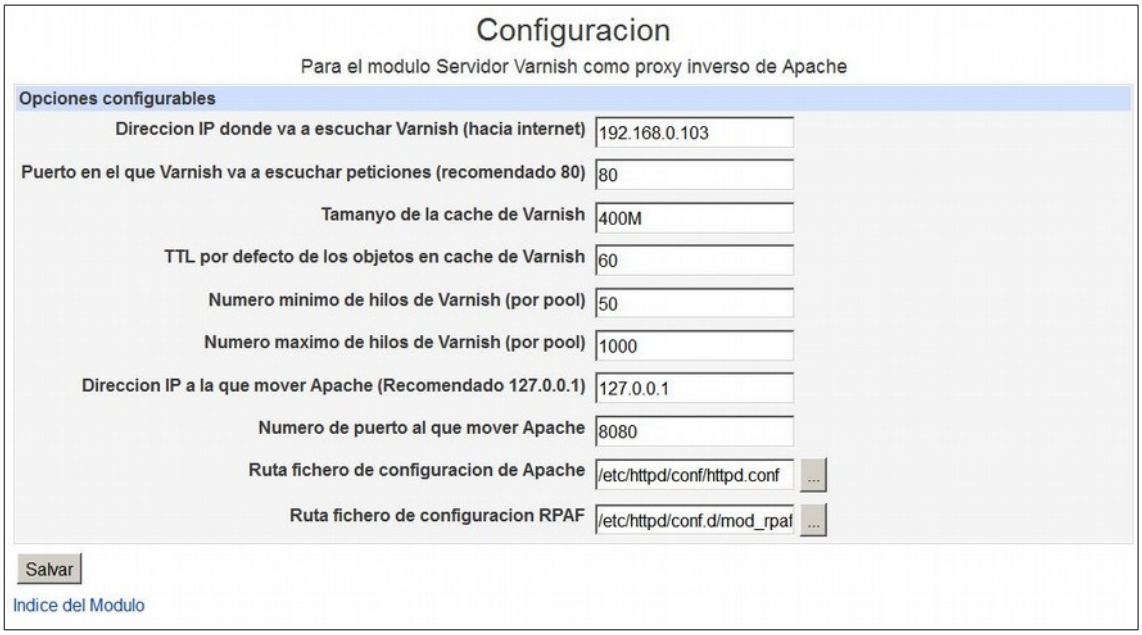

Ilustración 49: Datos introducidos en el formulario de configuración del panel de integración para las pruebas.

Hacemos clic en el botón inferior para salvar los datos de la configuración y regresamos a la pantalla principal del módulo. Una vez ahí pulsamos sobre el icono central, el que dice "Integrar Varnish / Aplicar configuración al sistema" para hacer la integración. Una vez hecho, nos debe de aparecer una pantalla en la que se detallan las diversas acciones que nuestro módulo ha hecho. Es importante comentar que por la forma de funcionamiento de Webmin, es posible que

no se complete la lista que se nos genera en el navegador, pero las acciones se llevan a cabo igualmente.

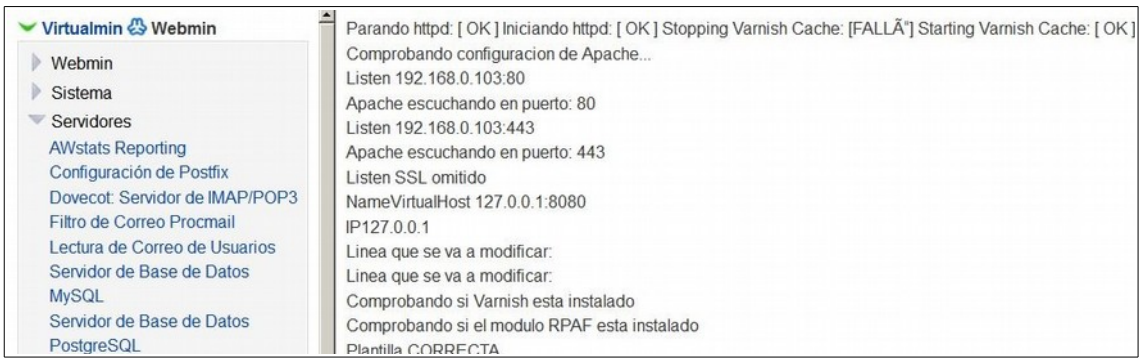

Ilustración 50: Captura de la lista de acciones realizadas por nuestro módulo para llevar a cabo la integración.

Para comprobar que todo ha salido correctamente, podemos ir a los formularios de configuración de Apache y comprobar los nuevos datos que vemos. Para considerar que todo ha salido correctamente, el servidor virtual "prueba.es" debería seguir en la misma dirección IP y puerto, ya que estaba en el 443 que Varnish no toca por incompatibilidad. Por el contrario el servidor virtual "prueba.com" deberá haberse movido a la dirección IP 127.0.0.1 y al puerto 8080. Además, Apache deberá haber dejado de escuchar en el puerto 80 para pasar a esperar peticiones a la IP 127.0.0.1 y al puerto 8080, igual que el servidor virtual.

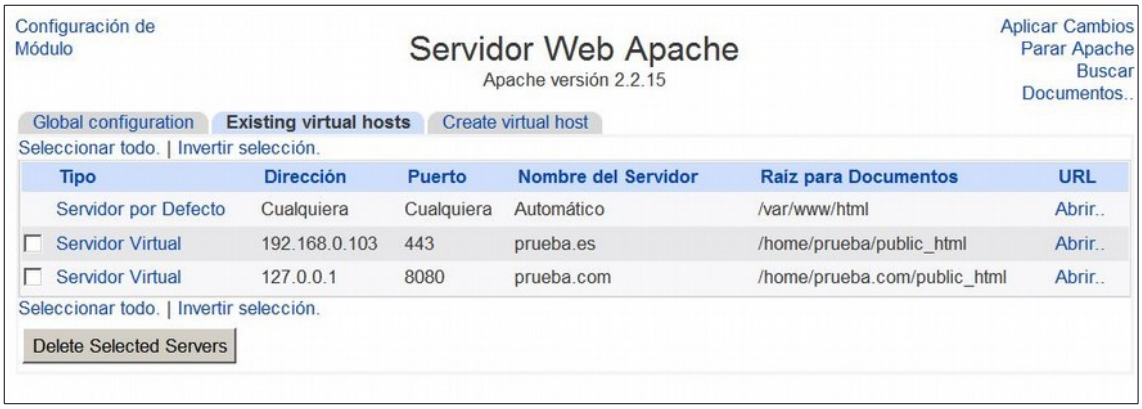

Ilustración 51: Captura de la configuración de los hosts virtuales de Apache tras la integración.

Como podemos apreciar en la imagen superior, efectivamente los cambios en los hosts virtuales se han aplicado correctamente. Pasamos a comprobar las direcciones de escucha de Apache en la imagen 52. Comprobamos que los cambios han surtido efecto de la manera deseada, habiendo aparecido la conexión del servidor web en la dirección *localhost* y desapareciendo cualquier conexión que hubiera en el puerto 80.

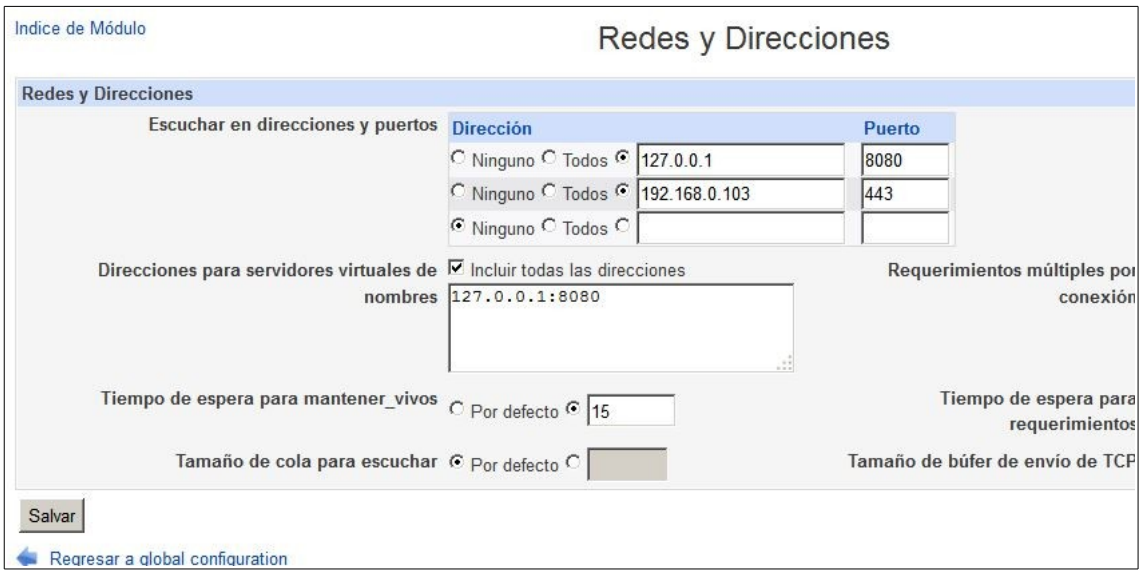

Ilustración 52: Captura de la configuración de direcciones de Apache tras la integración.

Podemos concluir por tanto, que el módulo que hemos desarrollado realiza las tareas de integración de forma correcta. En primer lugar, mueve el servidor web Apache y sus servidores virtuales a la dirección IP y puerto que el administrador especifica, respetando los servicios SSL que hubiera con anterioridad. Y por último, coloca el servicio Varnish en la dirección establecida a través del formulario de configuración del módulo.

7.2. Pruebas de funcionamiento de la cache

En esta sección procedemos a probar que la cache de Varnish esté haciendo su trabajo de forma correcta. Para realizar las pruebas hemos configurado en Apache dos servidores virtuales, “prueba.es” y “prueba.com”. Ambos servidos en la dirección IP 127.0.0.1 y en el puerto 8080. Varnish, por su parte, se encuentra esperando peticiones en la dirección IP de la máquina hacia el exterior, es decir, en la dirección 192.168.0.103 en el puerto HTTP 80.

Para realizar las pruebas utilizamos la herramienta cURL, que se encuentra disponible de forma nativa en las distribuciones Linux. Esta herramienta nos permite lanzar peticiones HTTP a donde nosotros deseemos y especificando las cabeceras que deseemos. También nos mostrará la respuesta que recibamos al completo.

7.2.1 Pruebas de acierto / fallo de la cache

En primer lugar vamos a comprobar el funcionamiento básico de la cache. Vamos a pedir las páginas de prueba que tenemos en el servidor y comprobaremos si nuestras peticiones producen aciertos en la cache o por el contrario producen fallos y por ende, crean tráfico en el servidor web.

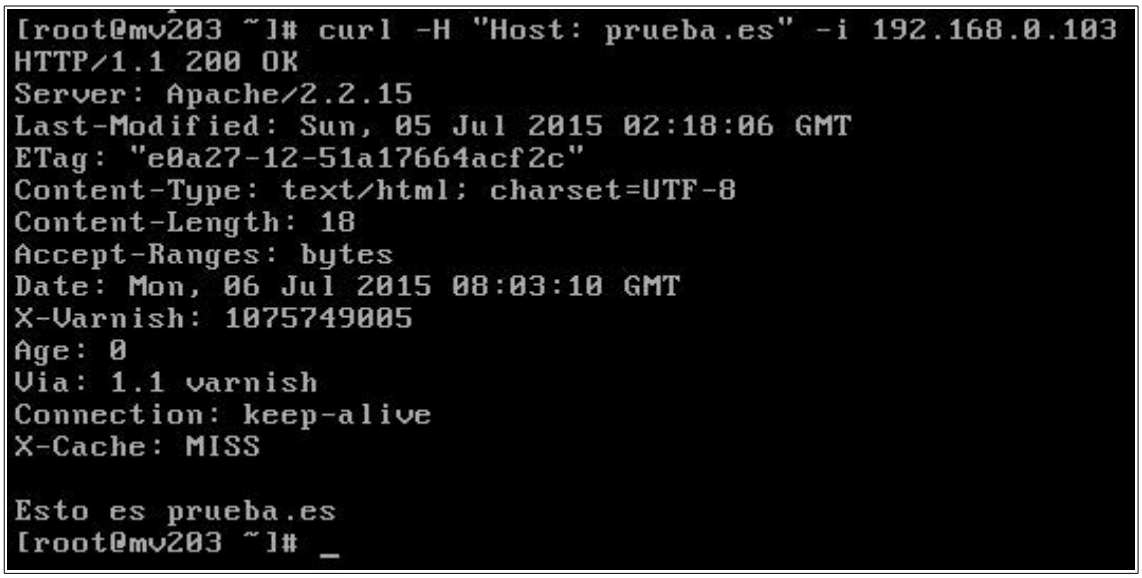

Ilustración 53: Lanzamiento de la primera petición HTTP con cURL.

Como podemos observar en la imagen 53, dentro de la respuesta tenemos la cabecera "*X-Varnish: xxxx"* que nos indica que esta respuesta ha sido procesada por un hilo del servicio de proxy inverso. Sin embargo, al fijarnos en la cabecera "*X-Cache: MISS"* podemos observar que la petición no ha producido un acierto en la cache. Lancemos ahora una segunda petición.

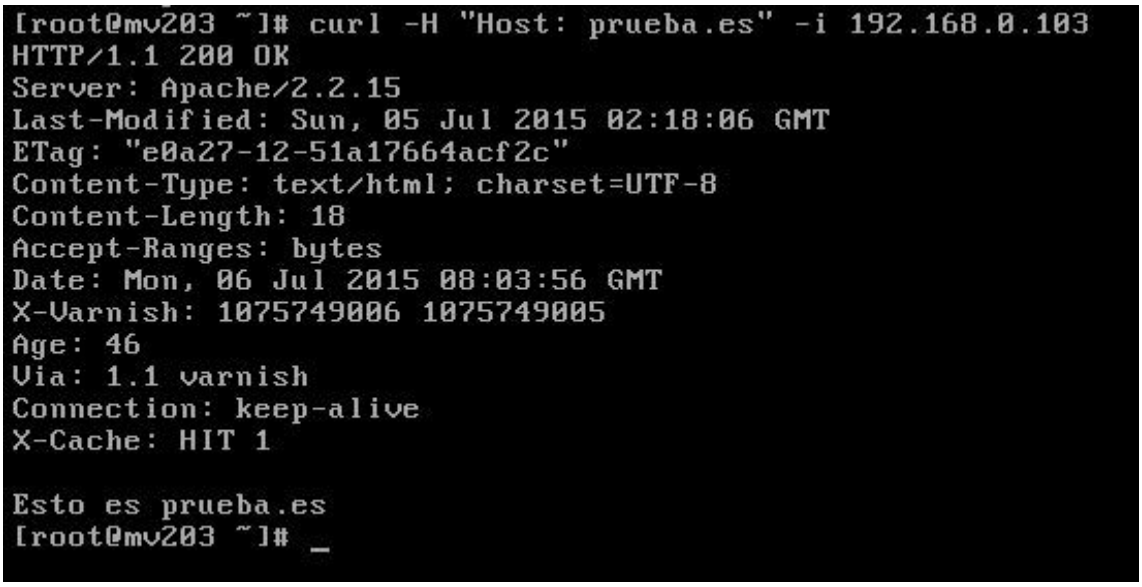

Ilustración 54: Lanzamiento de la segunda petición HTTP con cURL.

En la ilustración 54 podemos ver el resultado de lanzar una segunda petición HTTP a nuestra máquina pidiendo el mismo recurso. En esta ocasión podemos leer la cabecera "*X-Cache: HIT 1",* lo que nos indica que la respuesta recibida viene directamente de un acierto en la cache. Esto se debe a que nuestra primera petición generó el objeto respuesta a cachear, por lo que al

lanzar una segunda petición que genere una clave *hash* equivalente, obtenemos un acierto en cache. Lo mismo sucedería si lanzásemos más peticiones equivalentes en un periodo no más largo que el TTL que haya establecido el administrador. Una vez un objeto en la cache sobrepase ese tiempo, será eliminado, repitiendo el ciclo desde la ilustración 53 en el caso de que realizáramos otra petición.

7.2.2 Pruebas de heath-check

En esta sección pasamos a probar como se comporta nuestro proxy inverso en caso de que el servidor web Apache se venga abajo.

El procedimiento a seguir va a ser el siguiente:

- En primer lugar, lanzaremos una petición de la web "prueba.com", generando un objeto en cache.
- Después, pararemos el servidor Apache con la instrucción "*service httpd stop",* con lo que nuestro proxy inverso se quedará solo.
- Volveremos a lanzar una petición al host "prueba.com", después de esperar el tiempo TTL, para ver si nos sirve el objeto de la cache que debería haber entrado en modo *grace* al detectar Varnish la caída del servidor web.
- Lanzaremos una petición al host "prueba.es" para comprobar que al no estar el objeto en cache y estar Apache caído, Varnish no es capaz de darnos una respuesta correcta.

Procedemos con el primer paso, crear el objeto de la petición al host "prueba.com":

```
[root@mv203 ~]# curl -H "Host: prueba.com" -i 192.168.0.103
HTTP/1.1 200 OK
Server: Apache/2.2.15
Last-Modified: Sun, 05 Jul 2015 02:55:57 GMT
ETag: "e0a94-13-51a17eda3bc2e"
Content-Type: text/html; charset=UTF-8
Content-Length: 19
Accept-Ranges: bytes
Date: Mon, 06 Jul 2015 08:23:29 GMT
X-Varnish: 1075749028 1075749022
Age: 6
Via: 1.1 varnish
Connection: keep-alive
X-Cache: HIT 6

Esto es prueba.com
[root@mv203 ~]# _
```

Ilustración 55: Lanzamiento de petición HTTP al host "prueba.com" para asegurarnos de que el objeto se encuentra en cache.

Como podemos observar, el objeto se encuentra en cache. Es momento de desconectar Apache:

```
[root@mv203 ~]# service httpd stop
Parando httpd:                                             [  OK  ]
[root@mv203 ~]# _
```

Ilustración 56: Orden de parada al servicio de Apache.

Ahora, con Apache desconectado, esperamos más de 120 segundos, que es el tiempo TTL de los objetos en cache que hemos puesto por defecto. Una vez pase ese tiempo, lanzamos otra petición equivalente al host "prueba.com".

```
[root@mv203 ~]# curl -H "Host: prueba.com" -i 192.168.0.103
HTTP/1.1 200 OK
Server: Apache/2.2.15
Last-Modified: Sun, 05 Jul 2015 02:55:57 GMT
ETag: "e0a94-13-51a17eda3bc2e"
Content-Type: text/html; charset=UTF-8
Content-Length: 19
Accept-Ranges: bytes
Date: Mon, 06 Jul 2015 08:25:39 GMT
X-Varnish: 1075749044 1075749022
Age: 135
Via: 1.1 varnish
Connection: keep-alive
X-Cache: HIT 22

Esto es prueba.com
[root@mv203 ~]# _
```

Ilustración 57: Lanzamiento de la segunda petición al host "prueba.com" fuera del tiempo TTL establecido.

Fijándonos en la imagen 57, podemos ver como se ha comportado el proxy inverso ante la caída del servidor web. Nos ha contestado a la petición con el objeto que tenía en cache, pero si nos fijamos en la cabecera "*Age: 135"* vemos que ese valor es superior al máximo de 120 establecido por defecto. Esto sucede porque al desconectar Apache, Varnish pone los objetos de la cache en estado *grace,* el cual hemos especificado que aguante durante una hora sirviendo el contenido cacheado. Cuando un objeto está en estado *grace,* su atributo TTL es omitido, y se mantiene en cache tanto tiempo como hayamos indicado al atributo *obj.grace.*

Sí por el contrario, realizamos una petición HTTP al host "prueba.es", cuyo objeto de respuesta no se encontraba en la cache antes de tumbar Apache, vemos como efectivamente Varnish nos responde con un error, pues no es capaz de servirnos el objeto solicitado ni desde la cache ni (evidentemente) desde el servidor web desconectado .

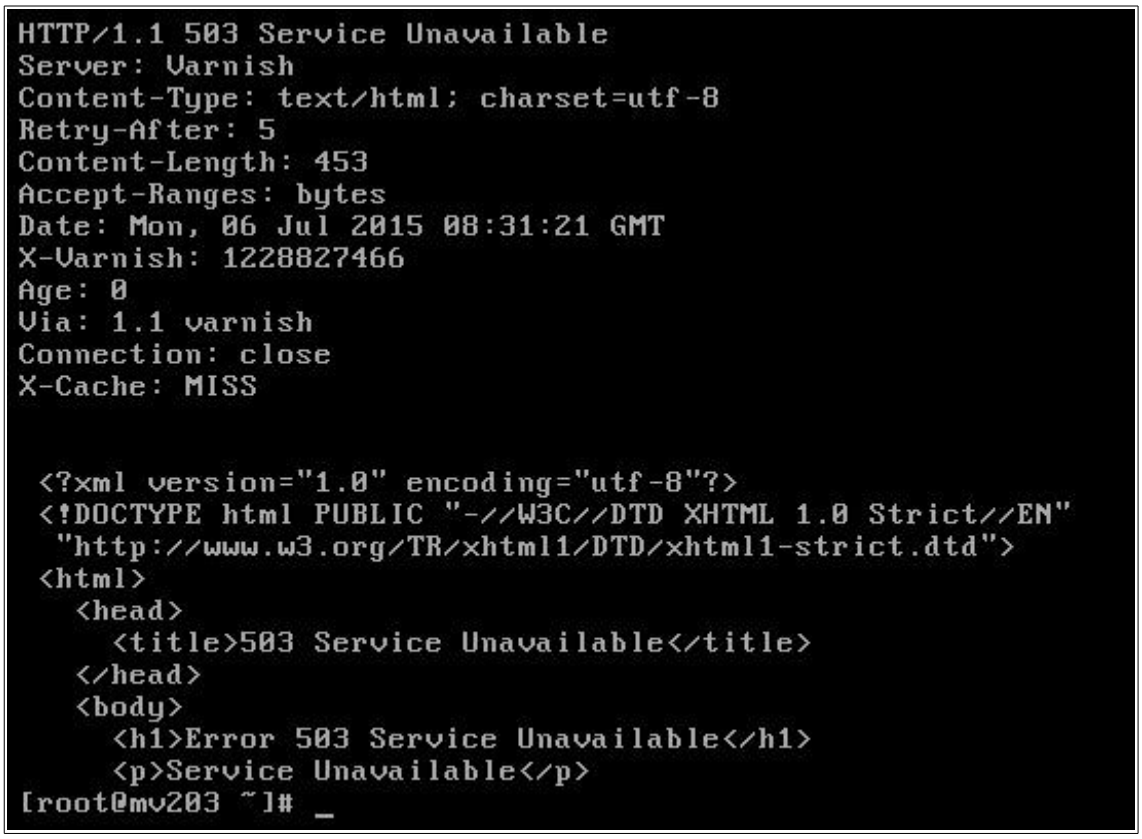

Ilustración 58: Respuesta de error que nos devuelve Varnish al no poder satisfacer nuestra petición por ningún medio.

Como hemos podido comprobar con las pruebas realizadas, tanto nuestro módulo de integración, como la configuración VCL que incluimos por defecto funciona correctamente.

8. Conclusión

El desarrollo de este proyecto nos ha enseñado cómo a la vez que la tecnología evoluciona, van apareciendo nuevas necesidades por cubrir. Es parte de nuestra formación como ingenieros el descubrir, abordar y aprovechar esas necesidades de la forma más eficiente y eficaz posible.

He aprendido cómo desarrollar un módulo para el panel de control Virtualmin. Que además, es capaz de añadir nuevas herramientas al sistema y configurarlas para que trabajen en armonía con todos los demás componentes. Mejorando las prestaciones y servicios de sistemas ampliamente utilizados en el sector.

Al mismo tiempo, hemos tenido que aprender a desarrollar una solución lo más genérica posible, teniendo que decidir donde poner el límite en ciertas configuraciones, para no reducir el número de potenciales beneficiados de nuestro desarrollo.

A nivel puramente formativo, he tenido que aprender un nuevo lenguaje de programación, Python, que nunca había utilizado antes en todos los años de formación académica. También he conseguido sentirme cómodo utilizando la herramienta Varnish Cache, de la que nunca antes había oído hablar. Y en general, he aprendido las bases del uso de cualquier proxy inverso y he mejorado mucho mis habilidades de trabajo con servidores web.

Al final hemos conseguido cumplir el objetivo, adquiriendo nuevos conocimientos y mejorando los que ya teníamos por el camino. La satisfacción recibida por el trabajo bien realizado nos anima a seguir afrontando retos, sin importar de qué magnitud sean, porque gracias a ello creceremos como ingenieros y como personas.

8.1. Trabajo futuro

Pese a haber cumplido el objetivo del proyecto, la naturaleza de nuestro desarrollo hace que éste tenga innumerables posibilidades de ampliación y mejora. Desde añadir más funcionalidades, hasta ampliar la compatibilidad con más entornos.

Una posible vía de mejora sería que el módulo leyese la configuración y arquitectura del sistema para ajustar el servicio Varnish automáticamente lo máximo posible, sin que el administrador tenga que tocar nada. Esto nos permitiría omitir muchos valores por defecto iniciales.

Otra sugerencia para mejorar nuestro desarrollo puede ser añadir a la interfaz de nuestro módulo una nueva ventana en la que el administrador pueda ver el estado del servicio Varnish Cache en tiempo real, así como estadísticas o avisos importantes de errores.

Una última posibilidad de ampliación que se estuvo barajando es la de incluir un amplio abanico de ficheros VCL por defecto, cada uno respondiendo a las necesidades de diferentes sitios web tipo. De esta forma, el administrador tan solo tendría que elegir qué tipo de sitio web está alojando en su servidor para conseguir una configuración por defecto algo más adaptada a sus necesidades.

9. Referencias

[1] Internet Live Stats. Total number of websites. [última consulta 20/05/2015]

http://www.internetlivestats.com/total-number-of-websites/

[2] Netcraft Ltd. Web Server Survey. [última consulta 20/05/2015]

http://news.netcraft.com/archives/category/web-server-survey/

[3] CentOS.org. Wiki CentOS. [última consulta 15/06/2015]

http://wiki.centos.org/

[4] Python Software Foundation. [última consulta 16/06/2015]

https://www.python.org/

[5] Python Software Foundation. Wiki Python. [última consulta 16/06/2015]

https://wiki.python.org/

[6] Apache Software Foundation. Wiki Apache. [última consulta 16/06/2015]

http://wiki.apache.org/general/

[7] Apache Software Foundation. [última consulta 16/06/2015]

http://httpd.apache.org

[8] Kurose, James F. y Ross, Keith w (2010). Redes de computadoras. Un enfoque descendente (5a ed). Madrid: Pearson. [última consulta 1/07/2015]

[9] Webmin. Webmin documentation. [última consulta 16/06/2015]

http://doxfer.webmin.com/Webmin/Introduction/

[10] Webmin. Webmin modules. [última consulta 16/06/2015]

http://doxfer.webmin.com/Webmin/Webmin_Modules/

[11] Virtualmin Inc. Specs Comparation. [última consulta 20/06/2015]

https://www.virtualmin.com/compare.html

[12] Varnish Software. Wiki Varnish. [última consulta 5/07/2015]

https://www.varnish-cache.org/trac

[13] Varnish Software. Main. [última consulta 20/06/2015]

https://www.varnish-cache.org

[14] Varnish Software. Working with virtual hosts. [última consulta 26/06/2015]

https://www.varnish-software.com/blog/getting-virtual-hosts-right-varnish-cache

[15] StackExchange. Programers section. [última consulta 20/06/2015]

http://programmers.stackexchange.com/questions/83818/what-does-proxy-to-mean

[16] CCM.net. Diferencias entre porxy y proxy inverso. [última consulta 16/06/2015]

http://es.ccm.net/contents/297-servidores-proxy-y-servidores-de-proxy-inversos

[17] Apache Software Foundation. Name-based Virtual hosting. [última consulta 20/06/2015]

http://httpd.apache.org/docs/current/vhosts/name-based.html

[18] Apache Software Foundation. VirtualHost Examples. [última consulta 1/07/2015]

http://httpd.apache.org/docs/2.2/vhosts/examples.html

[19] Apache Sotware Foundation. Module Index. [última consulta 1/07/2015]

http://httpd.apache.org/docs/2.2/mod/

[20] Igor Pavlov. 7-zip main site. [última consulta 21/06/2015]

www.7-zip.org/

[21] Nate Haug. Configuring Varnish for high avaliability. [última consulta 30/06/2015]

https://www.lullabot.com/articles/configuring-varnish-for-highavailability-with-multiple-web-servers

[22] Varnish Software. Oficial Varnish 3.0 Docs. [última consulta 4/07/2015]

https://www.varnish-cache.org/docs/3.0/index.html

[23] Kwalsh. Online Python Interpreter. [última consulta 25/06/2015]

http://mathcs.holycross.edu/~kwalsh/python/

[24] Blackhold. Configurando Apache para alto rendimiento. [última consulta 30/06/2015]

http://blackhold.nusepas.com/2010/05/configurando-apache-para-la-guerra-alto-rendimiento/

[25] Varnish Software. VCL Examples. [última consulta 3/07/2015]

https://www.varnish-cache.org/trac/wiki/VCLExamples

[26] Varnish Software. 1O common Vanrish mistakes. [última consulta 3/07/2015]

https://www.varnish-software.com/blog/10-varnish-cache-mistakes-and-how-avoid-them

[27] Stackoverflow. Varnish Hit for Pass meaning. [última consulta 26/06/2015]

http://stackoverflow.com/questions/12691489/varnish-hit-for-pass-means

[28] Ned Productions Ltd. A perfect configured Varnish VCL. [última consulta 4/07/2015]

http://www.nedproductions.biz/wiki/a-perfected-varnish-reverse-caching-proxy-vcl-script

[29] Varnish Software. Why no SSL? [última consulta 21/06/2015]

https://www.varnish-cache.org/docs/3.0/phk/ssl.html

[30] BuiltWith Pty Ltd. Web server usage statistics. [última consulta 2/07/2015]

http://trends.builtwith.com/web-server

[31] Virtualmin Inc. Open Source Web Hosting and Cloud Control Panels [última consulta 4/07/2015]

http://www.virtualmin.com/

[32] Wessels Duane (2001). Web caching. O'Reilly [última consulta 5/07/2015]

[33] Varnish Software. The Varnish Book. [última consulta 5/07/2015]