Administración avanzada del sistema operativo GNU/Linux

Josep Jorba Esteve
Remo Suppi Boldrito

PID_00212464

Material docente de la UOC

UOC
Universitat Oberta de Catalunya

Josep Jorba Esteve

Ingeniero Superior en Informática. Doctor ingeniero en Informática por la UAB. Profesor de los Estudios de Informática, Multimedia y Telecomunicaciones de la UOC, Barcelona.

Remo Suppi Boldrito

Ingeniero de Telecomunicaciones. Doctor en Informática por la UAB. Profesor del Departamento de Arquitectura de Computadores y Sistemas Operativos en la Universidad Autónoma de Barcelona.

Cuarta edición: septiembre 2014
© Josep Jorba Esteve, Remo Suppi Boldrito
© Fundació per a la Universitat Oberta de Catalunya
Av. Tibidabo, 39-43, 08035 Barcelona
Diseño: Manel Andreu
Realización editorial: Oberta UOC Publishing, SL
Depósito legal: B-19.882-2014

© 2014, FUOC. Se garantiza permiso para copiar, distribuir y modificar este documento según los términos de la GNU Free Documentation License, Version 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera. Se dispone de una copia de la licencia en el apartado "GNU Free Documentation License" de este documento.

Introducción

Los sistemas GNU/Linux han llegado a un grado de madurez importante que los hace válidos para integrarlos en cualquier ambiente de trabajo, desde el escritorio del PC personal hasta el servidor de una gran empresa. El objetivo principal de este curso es introducirnos en el mundo de la administración de los sistemas GNU/Linux. Aprenderemos cómo proporcionar desde GNU/Linux los servicios necesarios a diferentes ambientes de usuarios y máquinas. El campo de la administración de sistemas es enorme, hay muchas tareas, muchos problemas por tratar, hay que tener grandes conocimientos de hardware y software, y no está de más aplicar un poco de psicología para tratar con los usuarios finales de los sistemas.

El curso no pretende abordar una distribución GNU/Linux particular, pero se han escogido un par de ellas para tratar los ejemplos: Debian y Fedora. Respecto al campo de la administración, esta se intentará gestionar desde el nivel más bajo posible, normalmente la línea de comandos y los ficheros de configuración. Se comentarán, en su caso, herramientas de nivel más alto, pero hay que tener cuidado con estas últimas, ya que suelen ser fuertemente dependientes de la distribución utilizada e incluso de la versión de esta; además, estas herramientas suelen variar mucho entre versiones. La administración de bajo nivel suele ser mucho más dura, pero sabemos dónde estamos actuando, dónde podemos ver los resultados y, además, nos aporta muchos conocimientos extra sobre las diferentes tecnologías utilizadas.

La primera edición de estos materiales fue realizada a finales de 2003, y por ello se han ido desarrollando cuatro ediciones, que se han trabajado sobre las diferentes versiones de las distribuciones Debian y Fedora disponibles hasta la fecha actual. De estas distribuciones cabe señalar que, esencialmente, no han cambiado en los aspectos básicos de administración pero sí han incluido nuevas opciones, módulos, versiones del núcleo, formas de configuración, etc. La distribución Debian es un paradigma dentro del movimiento Open Source, por no pertenecer a ninguna empresa y estar confeccionada solo por las aportaciones de los voluntarios distribuidos por todo el mundo. Debian, además, integra exclusivamente software libre (pueden añadirse otros aparte). Por otra parte, Fedora es la distribución que cuenta, además de su comunidad, con el soporte de una de las empresas más solventes del panorama comercial (Red Hat) que, quizás por ello, sea la que brinde más soporte a nivel empresarial (mediante servicios de pago). En Debian el soporte depende de los voluntarios y del conocimiento compartido de los usuarios.

Como la administración de sistemas es un campo muy amplio, este manual solo pretende introducirnos en este apasionante (y cómo no, también a veces

frustrante) mundo. Veremos algunas de las tareas típicas y cómo tratar los diferentes problemas; si bien la administración es un campo que se aprende día a día, con el esfuerzo continuo. Y desde aquí advertimos de que este manual es un trabajo abierto, que con sus aciertos y los más que probables errores, se puede ver complementado con los comentarios de sus (sufridores) usuarios, de modo que son bienvenidos los comentarios y las sugerencias de cualquier tipo para la mejora de los materiales. Esta última adaptación se realiza sobre una nueva estructura del máster adaptado al Espacio Europeo de Educación Superior (EEES), que lo adecúa a las nuevas metodologías de aprendizaje, de conformidad con la normativa vigente de ámbito estatal.

Agradecimientos

Los autores agradecen a la Fundación para la Universitat Oberta de Catalunya (http://www.uoc.edu) la financiación de la edición de esta obra, enmarcada en el máster internacional en software libre ofrecido por la citada institución.

Contenidos

Módulo didáctico 1
El núcleo Linux
Josep Jorba Esteve
1. El núcleo del sistema GNU/Linux
2. Personalización o actualización del núcleo
3. Proceso de configuración y compilación
4. Aplicación de parches al núcleo
5. Módulos del núcleo
6. Virtualización en el núcleo
7. Presente del núcleo y alternativas
8. Taller de configuración del núcleo a las necesidades del usuario

Módulo didáctico 2
Administración de servidores
Remo Suppi Boldrito
1. Administración de servidores
1.1. Sistema de nombres de dominio (*Domain Name System, DNS*)
1.2. NIS (YP)
1.3. Servicios de conexión remota: telnet y ssh
1.4. Servicios de transferencia de ficheros: FTP
1.5. Active Directory Domain Controller con Samba4
1.6. Servicios de intercambio de información a nivel de usuario
1.7. *Internet Message Access Protocol* (IMAP)
1.8. Grupos de discusión
1.9. *World Wide Web* (httpd)
1.10. Servidor de WebDav
1.11. Servicio de *proxy*: Squid
1.12. OpenLdap (Ldap)
1.13. Servicios de archivos (NFS, *Network File System*)
1.14. Servidor de wiki
1.15. Gestión de copias de respaldo (*backups*)
1.16. *Public Key Infrastructure* (PKI)

Módulo didáctico 3
Administración de datos
Remo Suppi Boldrito
1. Administración de datos
1.1. PostgreSQL
1.2. El lenguaje SQL
1.3. MySQL

1.4. MariaDB
1.5. SQLite
1.6. Source Code Control System (CVS y Subversion)
1.7. Mantis Bug Tracker

Módulo didáctico 4

Administración de seguridad

Josep Jorba Esteve

1. Tipos y métodos de los ataques
2. Seguridad del sistema
3. Seguridad local
4. SELinux
5. Seguridad en red
6. Herramientas de seguridad: Detección de vulnerabilidades e intrusiones
7. Protección mediante filtrado (*TCP wrappers* y cortafuegos)
8. Análisis de registros
9. Taller: análisis de la seguridad mediante herramientas

Módulo didáctico 5

Sintonización, optimización y alta disponibilidad

Remo Suppi Boldrito

1. Sintonización, optimización y alta disponibilidad
1.1. Aspectos básicos
1.2. Monitorización
1.3. Alta disponibilidad en Linux (High-Availability Linux)

Módulo didáctico 6

Clúster, Cloud y DevOps

Remo Suppi Boldrito

1. Clúster, Cloud y DevOps
1.1. Virtualización
1.2. Beowulf
1.3. Beneficios del cómputo distribuido
1.4. Memoria compartida. Modelos de hilos (*threading*)
1.5. OpenMP
1.6. MPI, *Message Passing Interface*
1.7. Rocks Cluster
1.8. FAI
1.9. Logs

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent

and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

- D. Preserve all the copyright notices of the Document.

- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

- H. Include an unaltered copy of this License.

- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties —for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides promi-

nent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

El núcleo Linux

Josep Jorba Esteve

PID_00212476

UOC
Universitat Oberta de Catalunya

Índice

Introducción 5

Objetivos 6

1. El núcleo del sistema GNU/Linux 7

2. Personalización o actualización del núcleo 17

3. Proceso de configuración y compilación 20
3.1. Compilación de las ramas 2.6.x y 3.x del kernel Linux 22
3.2. Compilación del núcleo en Debian (*Debian Way*) 31

4. Aplicación de parches al núcleo 36

5. Módulos del núcleo 39
5.1. DKMS: módulos recompilados dinámicamente 42

6. Virtualización en el núcleo 45
6.1. KVM 47
6.2. VirtualBox 54
6.3. Xen 57

7. Presente del núcleo y alternativas 64

8. Taller de configuración del núcleo a las necesidades del usuario 69
8.1. Configuración del núcleo en Debian 69
8.2. Configuración del núcleo en Fedora/Red Hat 71
8.3. Configuración de un núcleo genérico 74

Resumen 77

Actividades 78

Bibliografía 79

Introducción

El núcleo (en inglés *kernel*) del sistema operativo GNU/Linux (al que habitualmente denominamos Linux) [Vasb], es la parte central del sistema: se encarga de ponerlo en funcionamiento y, una vez este es ya utilizable por las aplicaciones y los usuarios, se encarga de gestionar los recursos de la máquina, controlando la gestión de la memoria, los sistemas de ficheros, las operaciones de entrada/salida, los procesos y su intercomunicación.

Origen de Linux

El núcleo Linux se remonta al año 1991, cuando Linus Torvalds lo ofreció para el uso de la comunidad. Es de los pocos sistemas operativos que, siendo ampliamente usados, se dispone de su código fuente.

Su origen se remonta al año 1991, cuando en agosto, un estudiante finlandés llamado **Linus Torvalds** anunció en una lista de noticias, que había creado su propio núcleo de sistema operativo, funcionando conjuntamente con software GNU, y lo ofrecía a la comunidad de desarrolladores para que lo probara y sugiriera mejoras para hacerlo más utilizable. Este es el origen del núcleo del sistema operativo que más tarde se llamaría **GNU/Linux**.

Una de las particularidades de Linux es que, siguiendo la filosofía de software libre, se nos ofrece el código fuente del núcleo del propio sistema operativo (del *kernel*), de manera que es una herramienta perfecta para la educación, en temas de análisis y diseño de sistemas operativos.

La otra ventaja principal es que, disponiendo de los archivos de código fuente, podemos recompilarlos, para adaptarlos mejor a nuestro sistema, y como veremos en el módulo "Sintonización, optimización y alta disponibilidad", configurarlos posteriormente para dar un mejor rendimiento al sistema.

En este módulo veremos cómo manejar este proceso de preparación de un núcleo para nuestro sistema GNU/Linux: cómo, partiendo de los archivos fuente, podemos obtener una nueva versión del núcleo adaptada a nuestro sistema. Veremos cómo se desarrollan las fases de configuración, la posterior compilación y la realización de pruebas con el nuevo núcleo obtenido.

Además, examinaremos cómo el núcleo ha ido añadiendo toda una serie de características a lo largo de su evolución, que lo han convertido en competitivo frente a otros sistemas. En especial, observaremos algunas características de la virtualización que se nos ofrecen con soporte desde el propio núcleo.

Objetivos

En este módulo se muestran los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Conocer el funcionamiento del núcleo (en inglés, *kernel*) y de los procesos de configuración asociados.
2. Poder configurar y crear el núcleo del sistema en las distribuciones más habituales.
3. Entender el uso de los módulos del núcleo y decidir su integración (o no) dentro de la parte estática del núcleo.
4. Conocer las técnicas de virtualización y, en particular, de las incluidas en el núcleo.
5. Saber adaptar el núcleo a las necesidades particulares del usuario, para sintonizar sus sistemas.

1. El núcleo del sistema GNU/Linux

El núcleo o *kernel* es la parte básica de cualquier sistema operativo [Tan87, Tan06], y sobre él descansa el código de los servicios fundamentales para controlar el sistema completo. Básicamente, su estructura puede separarse en una serie de componentes, o módulos de gestión orientados a:

- **Gestión de procesos**: qué tareas se van a ejecutar y en qué orden y con qué prioridad. Un aspecto importante es la planificación de la CPU: ¿cómo se optimiza el tiempo de la CPU para ejecutar las tareas con el mayor rendimiento o interactividad posible con los usuarios?

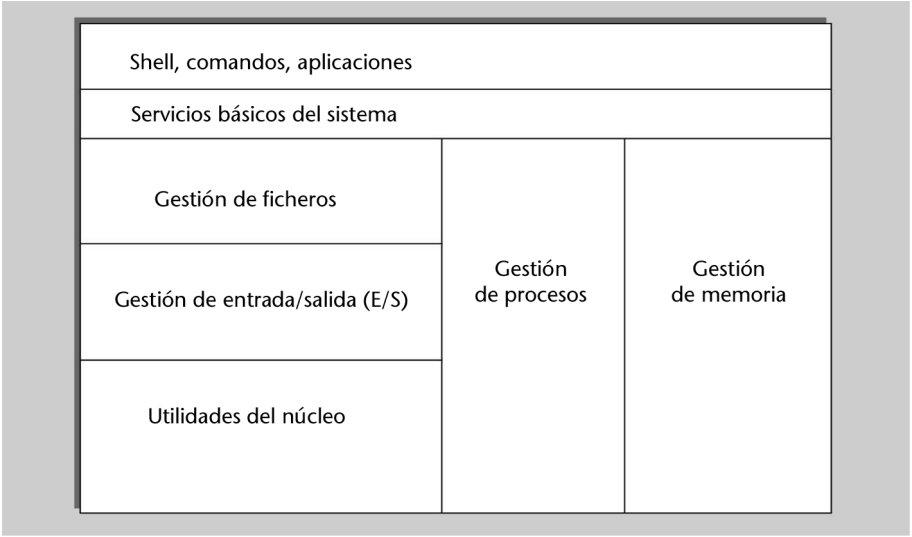
- **Intercomunicación de procesos y sincronización**: ¿cómo se comunican tareas entre sí, con qué diferentes mecanismos y cómo pueden sincronizarse grupos de tareas?

- **Gestión de entrada/salida (E/S)**: control de periféricos y gestión de recursos asociados.

- **Gestión de memoria**: optimización del uso de la memoria, sistema de paginación y memoria virtual.

- **Gestión de ficheros**: cómo el sistema controla y organiza los ficheros presentes en el sistema, y accede a los mismos.

Figura 1. Funciones básicas de un núcleo respecto a las aplicaciones y comandos ejecutados



En los sistemas privativos, el núcleo, o *kernel*, está perfectamente "oculto" bajo las capas del software del sistema operativo, y el usuario final no tiene una perspectiva clara de qué es ese núcleo ni tiene tampoco ninguna posibilidad de cambiarlo u optimizarlo, si no es por el uso de esotéricos editores de "registros" internos, o programas especializados de terceros (normalmente de alto coste). Además, el núcleo suele ser único, es el que proporciona el fabricante, el cual se reserva el derecho de introducir las modificaciones que quiera y cuando quiera, así como tratar los errores que aparezcan en plazos no estipulados, mediante actualizaciones que nos ofrece como "parches" de errores (o grupos de ellos denominados comúnmente *service packs* o *updates*).

Uno de los principales problemas de esta aproximación es precisamente la disponibilidad de estos parches: disponer de las actualizaciones de los errores a su debido tiempo y, si se trata de problemas de seguridad, todavía con más razón, ya que hasta que no estén corregidos no podemos garantizar la seguridad del sistema para problemas ya conocidos. Muchas organizaciones, grandes empresas, gobiernos, instituciones científicas y militares no pueden depender de los caprichos de un fabricante para solucionar los problemas de sus aplicaciones críticas.

En este caso, el núcleo Linux ofrece una solución de código abierto, con los consecuentes permisos de modificación, corrección, posibilidad de generación de nuevas versiones y actualizaciones de forma rápida, por parte de cualquiera que quiera y tenga los conocimientos adecuados para realizarlo. Esto permite a los usuarios con necesidades críticas controlar mejor sus aplicaciones y el propio sistema, y poder elaborar sistemas con el propio sistema operativo "a la carta", personalizado al gusto de cada usuario final. También permite disponer, a su vez, de un sistema operativo con código abierto, desarrollado por una comunidad de programadores coordinados mediante Internet y accesible ya sea para educación, por disponer del código fuente y abundante documentación, o para la producción final de los sistemas GNU/Linux adaptados a necesidades individuales o de un determinado colectivo.

Al disponer del código fuente, se pueden aplicar mejoras y soluciones de forma inmediata, a diferencia del software privativo, donde debemos esperar a las actualizaciones del fabricante. Podemos, además, personalizar el núcleo tanto como necesitemos, requisito esencial, por ejemplo, en aplicaciones de alto rendimiento, críticas en el tiempo o en soluciones con sistemas empotrados (para dispositivos móviles u otra electrónica de consumo).

A continuación repasamos un poco la historia del núcleo [Kera, Kerb]. El núcleo Linux lo comenzó a desarrollar un estudiante finlandés llamado Linus Torvalds, en 1991, con la intención de realizar una versión parecida a MINIX [Tan87, Tan06] (versión para PC de UNIX [Bac86]) para el procesador 386 de Intel. La primera versión publicada oficialmente fue la de Linux 1.0 en marzo de 1994, en la cual se incluía sólo la ejecución para la arquitectura i386 y

MINIX

El núcleo tiene sus orígenes en el sistema MINIX, desarrollado por Andrew Tanenbaum, como un clon de UNIX para PC.

soportaba máquinas de un solo procesador. Linux 1.2 fue publicado en marzo de 1995 y fue la primera versión en dar cobertura a diferentes arquitecturas, como Alpha, Sparc y Mips. Linux 2.0, en junio de 1996, añadió más arquitecturas y fue la primera versión en incorporar soporte multiprocesador (SMP) [Tum]. En Linux 2.2, de enero de 1999, se incrementaron las prestaciones de soporte SMP de manera significativa, y se añadieron controladores para gran cantidad de hardware. En la 2.4, en enero del 2001, se mejoró el soporte SMP, se incorporaron nuevas arquitecturas y se integraron controladores para dispositivos USB, PC Card (PCMCIA de los portátiles), parte de PnP (*plug and play*), soporte de RAID y volúmenes, etc. En la rama 2.6 del núcleo (diciembre de 2003), se mejoró sensiblemente el soporte SMP, se introdujo una mejor respuesta del sistema de planificación de CPU, el uso de hilos (*threads*) en el núcleo, mejor soporte de arquitecturas de 64 bits, soporte de virtualización y una mejor adaptación a dispositivos móviles.

En julio de 2011, Linus Torvalds anunció la versión Linux 3.0, no por sus grandes cambios tecnológicos, sino por iniciar una serie de cambios posteriores y para conmemorar el 20 aniversario de Linux. Algunos hitos más en la rama 3.x:

- 3.3 (marzo de 2012): se realiza una mezcla de los fuentes de Linux y se incorpora Android, que básicamente es un kernel Linux con ciertas modificaciones; tras varios intentos en esta versión, se realiza la unión de ambos. Se introduce también la capacidad de poder arrancar directamente a traves de las nuevas EFI, sustitución de las BIOS.

- 3.6 (septiembre de 2012): mejoras en el soporte del sistema de ficheros Btrfs (cuotas, *snapshots*), pensado para sustituir en el futuro a ext4.

- 3.7 (diciembre de 2012): se incorpora el soporte para diversas arquitecturas ARM.

- 3.8 (febrero de 2013): como curiosidad se elimina el soporte a procesadores 386, y se realizan importantes mejoras en varios sistemas de ficheros.

- 3.13 (enero de 2014): soporte de NFtables, la siguiente generación de reglas de firewall (sustituyendo a iptables).

- 3.15 (junio de 2014): se mejora el soporte de EFI, la escritura en sistemas de ficheros FUSE, y el soporte para repertorios vectoriales de nuevas CPUs Intel.

Respecto al proceso de desarrollo, desde su creación por Linus Torvalds en 1991 (versión 0.01), el núcleo lo ha seguido manteniendo él mismo, pero a medida que su trabajo se lo permitía y a medida que el núcleo maduraba (y crecía), se ha visto obligado a mantener las diferentes versiones estables

del núcleo gracias a diferentes colaboradores, mientras que Linus continua (en la medida de lo posible) desarrollando y recopilando aportaciones para la última versión de desarrollo del núcleo. Los colaboradores principales en estas versiones han sido [Lkm]:

- 2.0 David Weinehall.
- 2.2 Alan Cox (también desarrolla y publica parches para la mayoría de versiones).
- 2.4 Marcelo Tosatti.
- 2.5 Linus Torvalds (rama de desarrollo).
- 2.6 Greg Kroah-Hartman (versiones estables, entre otros) / Linus Torvalds, Andrew Morton (*releases* de desarrollo).
- 3.x Greg Kroah-Hartman / Linus Torvalds (*releases* de desarrollo).

Complejidad del núcleo

El núcleo hoy en día ha alcanzado unos grados de madurez y complejidad significativos

Para ver un poco la complejidad del núcleo de Linux, veamos una tabla con un poco de su historia resumida en las diferentes versiones iniciales y en el tamaño respectivo del código fuente (no es un indicador absoluto de la complejidad, pero da indicios de su evolución), confrontando con una versión de las ultimas ramas del kernel. En la tabla solo se indican las versiones de producción; el tamaño esta especificado en miles de líneas del código de los paquetes fuentes del núcleo:

Versión	Fecha de publicación	Líneas de código (en miles)
0.01	09-1991	10
1.0	03-1994	176
1.2	03-1995	311
2.0	06-1996	649
2.2	01-1999	1.800
2.4	01-2001	3.378
2.6	12-2003	5.930
3.10	06-2013	15.803

Como podemos comprobar, se ha evolucionado de unas diez mil líneas a seis millones en las primeras versiones de la rama 2.6; las últimas versiones de la rama 3.x tienen ya más de quince millones de líneas.

En estos momentos el desarrollo continúa en la rama 3.x del núcleo, la última versión estable, que incluye la mayoría de distribuciones como versión principal (algunas todavía incluyen algunas 2.6.x, pero 3.x suele ser la opción por defecto en la instalación).

Aunque ahora la rama principal sea la 3.x, cierto conocimiento de las anteriores versiones es imprescindible, ya que con facilidad podemos encontrar máquinas con distribuciones antiguas que no se hayan actualizado, que es posible que debamos mantener o realizar un proceso de migración a versiones más actuales.

En la rama del 2.6, durante su desarrollo se aceleraron de forma significativa los trabajos del núcleo, ya que tanto Linus Torvalds como Andrew Morton (que mantuvieron varias de las ramas de Linux 2.6 en desarrollo) se incorporaron (durante 2003) al Open Source Developement Laboratory (OSDL), un consorcio de empresas cuyo fin es promocionar el uso de Open Source y GNU/Linux en la empresa (en el consorcio se encuentran, entre otras muchas empresas con intereses en GNU/Linux: HP, IBM, Sun, Intel, Fujitsu, Hitachi, Toshiba, Red Hat, Suse, Transmeta, etc.). En aquel momento se dio una situación interesante, ya que el consorcio OSDL hizo de patrocinador de los trabajos, tanto para el mantenedor de la versión estable del núcleo (Andrew) como para el de la de desarrollo (Linus), trabajando a tiempo completo en las versiones y en los temas relacionados. Linus se mantiene independiente, trabajando en el núcleo, mientras Andrew se fue a trabajar a Google, donde continuaba a tiempo completo sus desarrollos, realizando parches con diferentes y nuevas aportaciones al núcleo, en la que se conoce como rama de desarrollo `-mm`. Después de cierto tiempo, OSDL se reconvirtió en la fundación The Linux Foundation.

Hay que tener en cuenta que con las versiones actuales del núcleo se ha alcanzado ya un alto grado de desarrollo y madurez, lo que hará que cada vez se amplíe más el tiempo entre la publicación de las versiones estables, no así de las revisiones parciales o de desarrollo, aspecto en el que los mantenedores esperan una nueva versión cada 2 o 3 meses.

Enlace de interés

Linux Foundation: http://www.linuxfoundation.org

Además, otro factor a considerar es el tamaño y el número de personas que están trabajando en el desarrollo actual. En un principio había unas pocas personas que tenían un conocimiento global del núcleo entero, mientras que hoy en día tenemos un importante número de personas que lo desarrollan (se cree que cerca de varios miles) con diferentes contribuciones, aunque el grupo duro se estima en unas pocas docenas de desarrolladores.

También cabe tener en cuenta que la mayoría de desarrolladores (de los miles) solo tienen unos conocimientos parciales del núcleo y, ni todos trabajan simultáneamente, ni su aportación es igual de relevante (algunas aportaciones solo corrigen errores sencillos). En el otro extremo, son unas pocas personas (como los mantenedores) las que disponen de un conocimiento total del núcleo. Esto supone que se puedan alargar los desarrollos y que se tengan que depurar las aportaciones, para comprobar que no entren en conflicto entre ellas, o que se deba escoger entre posibles alternativas de prestaciones para un mismo componente.

Respecto a la numeración de las versiones del núcleo de Linux [Ker, Ces06, Lov10], cabe tener en cuenta los aspectos siguientes:

1) Hasta la rama del núcleo 2.6.x, las versiones del núcleo Linux se regían por una división en dos series: una era la denominada “experimental” (con nume-

ración impar en la segunda cifra, como 1.3.xx, 2.1.x o 2.5.x) y la otra era la de producción (serie par, como 1.2.xx, 2.0.xx, 2.2.x, 2.4.x y más). La serie experimental eran versiones que se movían rápidamente y se utilizaban para probar nuevas prestaciones, algoritmos, controladores de dispositivo, etc. Por la propia naturaleza de los núcleos experimentales, podían tener comportamientos impredecibles, como pérdidas de datos, bloqueos aleatorios de la máquina, etc. Por lo tanto, no estaban destinadas a utilizarse en máquinas para la producción, a no ser que se quisiese probar una característica determinada (con los consecuentes peligros).

Los núcleos de producción (serie par) o estables eran los núcleos con un conjunto de prestaciones bien definido, con un número bajo de errores conocidos y controladores de dispositivos probados. Se publicaban con menos frecuencia que los experimentales y existían variedad de versiones, unas de más o menos calidad que otras. Las distribuciones GNU/Linux se suelen basar en una determinada versión del núcleo estable, no necesariamente el último núcleo de producción publicado.

2) En la numeración del núcleo Linux (utilizada en la rama 2.6.x), se siguen conservando algunos aspectos básicos: la versión viene indicada por unos números X.Y.Z, donde normalmente X es la versión principal, que representa los cambios importantes del núcleo; Y es la versión secundaria, y habitualmente implica mejoras en las prestaciones del núcleo: Y es par en los núcleos estables e impar en los desarrollos o pruebas; Z es la versión de construcción, que indica el número de la revisión de X.Y, en cuanto a parches o correcciones hechas.

En los últimos esquemas se llega a introducir cuartos números, para especificar Z cambios menores, o diferentes posibilidades de la revisión (con diferentes parches añadidos que corrigen fallos). La versión así definida con cuatro números es la que se considera estable (*stable*). También se usan otros esquemas para las diversas versiones de prueba (normalmente no recomendables para entornos de producción), como sufijos `-rc` (*release candidate*), `-mm` que son núcleos experimentales con gran introducción de parches que suponen nuevas prestaciones adicionales como pruebas de diferentes técnicas novedosas, o los `-git` que son una especie de "foto" diaria del desarrollo del núcleo. Estos esquemas de numeración están en constante cambio para adaptarse a la forma de trabajar de la comunidad del núcleo y a sus necesidades para acelerar el desarrollo.

Los distribuidores no suelen incluir la última versión del núcleo, sino la que ellos han probado con más frecuencia y pueden verificar que es estable para el software y componentes que incluyen. Partiendo de este esquema de numeración clásico (que se siguió durante las ramas 2.4.x hasta los inicios de la 2.6), hubo algunas modificaciones para adaptarse al hecho de que el núcleo (rama 2.6.x) se vuelve más estable (fijando X.Y a 2.6) y cada vez las revisiones son menores (por significar un salto de versión de los primeros números), pero el desarrollo continuo y frenético sigue.

3) En la rama 3.x, cuando después del 2.6.39 Linus decidió renombrar a 3.x, el segundo número se ha usado como número de revisión por secuencia temporal a medida que salían los nuevos kernels, y puede añadirse un tercer número para designar correcciones de fallos respecto a *bugs* o seguridad. Se espera que, de manera semejante, en un determinado momento se progrese a las ramas 4.x, 5.x. etc. En cuanto a las versiones diferentes del kernel, ahora se suele denominar *mainline* a la versión de desarrollo que suele mantener Linus, y que aparece periódicamente cada 2 o 3 meses. Cuando aparece liberada la versión pasa al estado de *Stable*, en la cual se realizan varias iteraciones (indicadas por el tercer número) por *bugs*, normalmente de 2 a 3 por mes, aunque hay solo unas pocas revisiones del estable, porque mientras tanto el *mainline* siguiente se vuelve estable. Y como caso excepcional, hay algunos kernels denominados *longterm*, para propósitos de mantenimiento de larga duración, en los que se incluye el soporte de parches que se hagan en otras versiones, para portarlos a estas. Pueden ser, por ejemplo, kernels que hayan sido escogidos por distribuciones de tipo LTS (*Long Term Support*) o interesantes por contener alguna prestación especial. Hay que diferenciar que los kernels de distribución suelen ser mantenidos por las propias distribuciones, normalmente como paquetes en sus repositorios, y pueden incluir niveles de parche adicionales propios de la distribución. Estos kernels especiales de las distribuciones pueden detectarse con el comando `uname -r`, que nos da la versión del kernel actual, por los números adicionales a los predefinidos en la numeración del kernel, incluidos en la salida del comando.

4) Para obtener el último núcleo publicado (que normalmente se denomina *stable* o *vanilla*), hay que acudir al archivo de núcleos Linux (disponible en http://www.kernel.org). También podrán encontrarse aquí algunos parches al núcleo original, que corrigen errores detectados *a posteriori* de la publicación del núcleo.

Enlace de interés

Repositorio de núcleos Linux:
http://www.kernel.org

Algunas de las características técnicas [Ces06, Kan][Lov10] del núcleo Linux que podríamos destacar son:

- Núcleo de tipo monolítico: básicamente es un gran programa creado como una unidad, pero conceptualmente dividido en varios componentes lógicos.

- Tiene soporte para carga y descarga de porciones del núcleo bajo demanda; estas porciones se llaman *módulos* y suelen ser características del núcleo o controladores de dispositivo.

- Hilos de núcleo: Para el funcionamiento interno se utilizan varios hilos (*threads* en inglés) de ejecución internos al núcleo, que pueden estar asociados a un programa de usuario o bien a una funcionalidad interna del núcleo. En Linux no se hacía un uso intensivo de este concepto en origen, pero ha pasado a ser un concepto fundamental para el rendimiento, en especial debido a la aparición de las CPU *multicore*. En las diferentes revi-

siones de la rama 2.6.x se ofreció un mejor soporte, y gran parte del núcleo se ejecuta usando diversos hilos de ejecución.

- Soporte de aplicaciones multihilo: soporte de aplicaciones de usuario de tipo multihilo (*multithread*), ya que muchos paradigmas de computación de tipo cliente/servidor necesitan servidores capaces de atender múltiples peticiones simultáneas dedicando un hilo de ejecución a cada petición o grupo de ellas. Linux tiene una biblioteca propia de hilos que puede usarse para las aplicaciones multihilo, con las mejoras que se introdujeron en el núcleo, que también han permitido un mejor uso para implementar bibliotecas de hilos para el desarrollo de aplicaciones.

- El núcleo es de tipo no apropiativo (*nonpreemptive*): esto implica que dentro del núcleo no pueden pararse llamadas a sistema (en modo supervisor) mientras se está resolviendo la tarea de sistema, y cuando ésta acaba, se prosigue la ejecución de la tarea anterior. Por lo tanto, el núcleo dentro de una llamada no puede ser interrumpido para atender a otra tarea. Normalmente, los núcleos apropiativos están asociados a sistemas que trabajan en tiempo real, donde debe permitirse lo anterior para tratar eventos críticos. Hay algunas versiones especiales del núcleo de Linux para tiempo real (ramas `-rt`, de *realtime*), que permiten esto por medio de la introducción de unos puntos fijos donde las tareas del núcleo pueden interrumpirse entre si. También se ha mejorado especialmente este concepto en la rama 2.6.x del núcleo, que en algunos casos permite interrumpir algunas tareas del núcleo, reasumibles, para tratar otras, prosiguiendo posteriormente su ejecución. Este concepto de núcleo apropiativo también puede ser útil para mejorar tareas interactivas, ya que si se producen llamadas costosas al sistema, pueden provocar retardos en las aplicaciones interactivas.

- Soporte para multiprocesador, tanto lo que se denomina *multiprocesamiento simétrico* (SMP) como *multicore*. Este concepto suele englobar máquinas que van desde el caso simple de 2 hasta 64 CPU colocadas en diferentes zócalos físicos de la máquina. Este tema se ha puesto de especial actualidad con las arquitecturas de tipo *multicore*, que permiten de 2 a 8 o más núcleos de CPU en un mismo zócalo físico, en máquinas accesibles a los usuarios domésticos. Linux puede usar múltiples procesadores, donde cada procesador puede manejar una o más tareas. Pero originalmente había algunas partes del núcleo que disminuían el rendimiento, ya que están pensadas para una única CPU y obligan a parar el sistema entero en determinados bloqueos. SMP es una de las técnicas más estudiadas en la comunidad del núcleo de Linux, y se han obtenido mejoras importantes en en las ramas 2.6 y 3. Del rendimiento SMP y multicore depende en gran medida la adopción de Linux en los sistemas empresariales, en la faceta de sistema operativo para servidores.

- Sistemas de ficheros: el núcleo tiene una buena arquitectura de los sistemas de ficheros, ya que el trabajo interno se basa en una abstracción de

un sistema virtual (VFS, *virtual file system*), que puede ser adaptada fácilmente a cualquier sistema real. Como resultado, Linux es quizás el sistema operativo que más sistemas de ficheros soporta, desde su propio ext2 inicial, hasta msdos, vfat, ntfs, sistemas con *journal* como ext3, ext4, ReiserFS, JFS(IBM), XFS(Silicon), ZFS (Oracle), Btrfs, NTFS, iso9660 (CD), udf, etc. y se van añadiendo más en las diferentes revisiones del núcleo.

Otras características menos técnicas (un poco de *marketing*) que podríamos destacar:

1) Linux es gratuito: junto con el software GNU, y el incluido en cualquier distribución, podemos tener un sistema tipo UNIX completo prácticamente por el coste del hardware; y por la parte de los costes de la distribución GNU/Linux, podemos obtenerla prácticamente gratis. Pero no está de más pagar por una distribución completa, con los manuales y apoyo técnico, a un coste menor comparado con lo que se paga por algunos sistemas privativos, o contribuir con la compra al desarrollo de las distribuciones que más nos gusten o nos sean prácticas.

2) Linux es personalizable: la licencia GPL nos permite leer y modificar el código fuente del núcleo (siempre que tengamos los conocimientos adecuados).

3) Linux se ejecuta en hardware antiguo bastante limitado; es posible, por ejemplo, crear un servidor de red con un 386 con 4 MB de RAM (hay distribuciones especializadas en bajos recursos y en procesadores o arquitecturas obsoletas).

4) Linux es un sistema de altas prestaciones: el objetivo principal en Linux es la eficiencia y se intenta aprovechar al máximo el hardware disponible.

5) Alta calidad: los sistemas GNU/Linux son muy estables, con una baja proporción de fallos, y reducen el tiempo dedicado a mantener los sistemas.

6) El núcleo es bastante reducido y compacto: es posible colocarlo, junto con algunos programas fundamentales en un solo (formato antiguo de) disco de 1,44 MB (existen varias distribuciones de un solo disquete con programas básicos).

7) Linux es compatible con una gran parte de los sistemas operativos, puede leer ficheros de prácticamente cualquier sistema de ficheros y puede comunicarse por red para ofrecer y recibir servicios de cualquiera de estos sistemas. Además, también con ciertas bibliotecas puede ejecutar programas de otros sistemas (como MS-DOS, Windows, BSD, Xenix, etc.) en la arquitectura x86 32 o 64 bits o bien virtualizar máquinas completas, pudiendo disponer de imágenes virtuales de distribuciones GNU/Linux ya preparadas para actuar de sistemas invitados en gestores de virtualización.

8) Linux dispone de un completísimo soporte: no hay ningún otro sistema que tenga la rapidez y cantidad de parches y actualizaciones que Linux, ni en

los sistemas privativos. Para un problema determinado, hay infinidad de listas de correo y foros que en pocas horas pueden permitir solucionar cualquier problema. El único problema está en los controladores de hardware reciente, que muchos fabricantes todavía se resisten a proporcionar, si no es para sistemas privativos. Pero esto está cambiando poco a poco, y varios de los fabricantes más importantes de sectores como tarjetas de vídeo (NVIDIA, AMD ATI) e impresoras (Epson, HP) comienzan ya a proporcionar los controladores para sus dispositivos, bien sean de código abierto, o binarios utilizables por el núcleo.

2. Personalización o actualización del núcleo

Como usuarios o administradores de sistemas GNU/Linux, debemos tener en cuenta las posibilidades que nos ofrece el núcleo para adaptarlo a nuestras necesidades y equipos.

Normalmente, construimos nuestros sistemas GNU/Linux a partir de la instalación en nuestros equipos de alguna de las distribuciones de GNU/Linux, ya sean comerciales, como Red Hat o Suse, o comunitarias, como Debian y Fedora.

Estas distribuciones aportan, en el momento de la instalación, una serie de núcleos Linux binarios ya preconfigurados y compilados, y normalmente tenemos que elegir qué núcleo del conjunto de los disponibles se adapta mejor a nuestro hardware. Hay núcleos genéricos para una arquitectura, para un modelo de procesador o bien orientados disponer de una serie de recursos de memoria, otros que ofrecen una mezcla de controladores de dispositivos [Cor05], posibilidades de virtualización, etc.

Otra opción de instalación suele ser la versión del núcleo. Normalmente las distribuciones usan una versión para instalación que consideran lo suficientemente estable y probada como para que no cause problemas a los usuarios. Por ejemplo, a día de hoy muchas distribuciones vienen con una versión de la rama 3.x del núcleo por defecto, que se consideraba la versión más estable del momento en que salió la distribución. En algunos casos en el momento de la instalación puede ofrecerse la posibilidad de usar como alternativa versiones más modernas, con mejor soporte para dispositivos más modernos (de última generación), pero quizás no tan probadas.

Los distribuidores suelen, además, modificar el núcleo para mejorar el comportamiento de su distribución o corregir errores que han detectado en el núcleo en el momento de las pruebas. Otra técnica bastante común en las distribuciones comerciales es deshabilitar prestaciones problemáticas, que pueden causar fallos o que necesitan una configuración especifica de la máquina, o bien una determinada prestación no se considera lo suficientemente estable para incluirla activada.

Personalización del núcleo

La posibilidad de actualizar y personalizar el núcleo a medida ofrece una buena adaptación a cualquier sistema, lo que permite así una optimización y sintonización del núcleo al sistema destino.

Esto nos lleva a considerar que, por muy bien que un distribuidor haga el trabajo de adaptar el núcleo a su distribución, siempre nos podemos encontrar con una serie de problemas u objetivos que no podemos realizar con la situación actual:

- El núcleo no está actualizado a la última versión estable disponible; no se dispone de soporte para algunos dispositivos modernos.

- El núcleo estándar no dispone de soporte para los dispositivos que tenemos, porque no han sido habilitados.

- Los controladores que nos ofrece un fabricante necesitan una nueva versión del núcleo o modificaciones.

- A la inversa, el núcleo es demasiado moderno, tenemos hardware antiguo que ya no tiene soporte en los últimos núcleos.

- El núcleo, tal como está, no obtiene las máximas prestaciones de nuestros dispositivos.

- Algunas aplicaciones que queremos usar requieren soporte de un núcleo nuevo o de algunas de sus prestaciones.

- Queremos estar a la última, nos arriesgamos, instalando últimas versiones del núcleo Linux.

- Nos gusta investigar o probar los nuevos avances del núcleo o bien queremos tocar o modificar el núcleo.

- Queremos programar un controlador para un dispositivo no soportado.

- Etc.

Por estos y otros motivos podemos no estar contentos con el núcleo que tenemos. Se nos plantean entonces dos posibilidades: actualizar el núcleo binario de la distribución o bien personalizarlo a partir de los paquetes fuente.

Vamos a ver algunas cuestiones relacionadas con las diferentes opciones y qué suponen:

1) Actualización del núcleo de la distribución. El distribuidor normalmente publica también las actualizaciones que van surgiendo del núcleo. Cuando la comunidad Linux crea una nueva versión del núcleo, cada distribuidor la une a su distribución y hace las pruebas pertinentes. Después del periodo de prueba, se identifican posibles errores, los corrige y produce la actualización del núcleo pertinente respecto a la que ofrecía en los CD de la distribución. Los usuarios pueden descargar la nueva revisión de la distribución del sitio web, o bien actualizarla mediante algún sistema automático de paquetes vía repositorio. Normalmente, se verifica qué versión tiene el sistema, se descarga el núcleo nuevo y se hacen los cambios necesarios para que la siguiente vez el sistema funcione con el nuevo núcleo, y se mantiene la versión antigua por si hay problemas.

Este tipo de actualización nos simplifica mucho el proceso, pero no tiene porqué solucionar nuestros problemas, ya que puede ser que nuestro hardware no esté todavía soportado o la característica a probar del núcleo no esté todavía en la versión que tenemos de la distribución; cabe recordar que no tiene porqué usar la última versión disponible (por ejemplo en kernel.org), sino aquella que el distribuidor considere estable para su distribución.

Si nuestro hardware tampoco viene habilitado por defecto en la nueva versión, estamos en la misma situación. O sencillamente, si queremos la última versión, este proceso no nos sirve.

2) Personalización del núcleo. En este caso, iremos a los paquetes fuente del núcleo y adaptaremos "a mano" el hardware o las características deseadas. Pasaremos por un proceso de configuración y compilación de los paquetes fuente del núcleo para, finalmente, crear un núcleo binario que instalaremos en el sistema, y tenerlo, así, disponible en el siguiente arranque del sistema.

Ved también

La personalización del núcleo es un proceso que se describe con detalle en los apartados siguientes.

También aquí podemos encontrarnos con dos opciones más: o bien por defecto obtenemos la versión "oficial" del núcleo (kernel.org) o bien podemos acudir a los paquetes fuente proporcionados por la propia distribución. Hay que tener en cuenta que distribuciones como Debian y Fedora hacen un trabajo importante de adecuación del núcleo y de corrección de errores del que afectan a su distribución, con lo cual podemos, en algunos casos, disponer de correcciones adicionales al código original del núcleo. Otra vez más los paquetes fuente ofrecidos por la distribución no tienen porqué corresponder a la última versión estable publicada.

Este sistema nos permite la máxima fiabilidad y control, pero a un coste de administración alto, ya que debemos disponer de conocimientos amplios de los dispositivos y de las características que estamos escogiendo (qué significan y qué implicaciones pueden tener), así como de las consecuencias que puedan tener las decisiones que tomemos.

3. Proceso de configuración y compilación

La personalización del núcleo [Vasb] es un proceso costoso, necesita amplios conocimientos del proceso a realizar y, además, es una de las tareas críticas, de la cual depende la estabilidad del sistema, por la propia naturaleza del núcleo, puesto que es, para el sistema operativo, su elemento central.

Obtención de un núcleo personalizado

El proceso de obtención de un nuevo núcleo personalizado pasa por obtener los paquetes fuente, adaptar la configuración, compilar e instalar el núcleo obtenido en el sistema.

Cualquier error de procedimiento puede comportar la inestabilidad o la pérdida del sistema. Por lo tanto, no está de más realizar cualquier tarea de copia de seguridad de los datos de usuarios, datos de configuraciones que hayamos personalizado o, si disponemos de dispositivos adecuados, una copia de seguridad completa del sistema. También es recomendable disponer de algún disquete de arranque (o distribución LiveCD con herramientas de rescate) que nos sirva de ayuda por si surgen problemas, o bien un disquete/CD/archivo de rescate (*rescue disk*) que la mayoría de distribuciones permiten crear desde los CD de la distribución (o directamente proporcionan alguno como CD de rescate para la distribución). Actualmente muchos de los LiveCD de las distribuciones ya proporcionan herramientas de rescate suficientes para estas tareas, aunque también existen algunas distribuciones especializadas para ello.

Sin embargo, ante sistemas en producción, siempre es importante tomar las medidas de precaución y hacer las copias de seguridad necesarias. O bien trabajar con un sistema equivalente, en estado de test o preproducción, donde podamos previamente probar los efectos de un nuevo kernel.

Examinemos el proceso necesario para instalar y configurar un núcleo Linux. En los subapartados siguientes, trataremos:

1) El proceso de compilación para las ramas 2.6.x y 3.x

2) Detalles específicos de las versiones 3.x.

3) Un caso particular para la distribución Debian, que dispone de un sistema propio (*Debian way*) de compilación más flexible.

Enlace de interés

Para Fedora recomendamos consultar el siguiente enlace: http://fedoraproject.org/wiki/Building_a_custom_kernel.

Respecto las versiones 2.6.x, en este módulo mantenemos la explicación por razones históricas, ya que las distribuciones actuales prácticamente ya no las ofrecen, pero debemos considerar que en más de una ocasión nos veremos obligados a migrar un determinado sistema a nuevas versiones, o bien a mantenerlo en las antiguas debido a incompatibilidades o a la existencia de hardware antiguo no soportado por las distribuciones actuales.

Los conceptos generales del proceso de compilación y configuración se explicarán en el primer subapartado (2.6.x), ya que la mayoría de ellos son genéricos, y observaremos posteriormente las diferencias respecto de las nuevas versiones.

También hay que añadir que, con las últimas distribuciones, cada vez es más casual la necesidad de reconstruir o recompilar el propio núcleo, debido, entre otras consideraciones, a que:

- Antiguamente la mayoría de los controladores estaban integrados en el núcleo y había que recompilarlo por completo si queríamos incluir o excluir un controlador determinado. Hoy en día, como veremos en el apartado 5, pueden recompilarse los controladores o módulos concretos, no el núcleo en si mismo.

- Para sintonizar el núcleo, antiguamente había que recompilarlo. En muchos casos (no todos) puede realizarse la sintonización de algunos elementos del núcleo mediante el acceso al sistema de ficheros `/proc` o `/sys`.

- En algunos casos de distribuciones comerciales (versiones empresariales para las que se paga soporte), los núcleos y el sistema completo están soportados por el equipo de la distribución, y a veces pueden perderse las licencias de soporte o las garantías por realizar cambios de este estilo.

- Por otro lado, las distribuciones tienen una velocidad bastante rápida en cuanto a integrar parches y nuevos núcleos a medida que se generan.

Por contra, una personalización del núcleo, mediante compilación, nos puede permitir:

- Escoger qué partes incluir o excluir del núcleo, dando un soporte concreto a una plataforma o un hardware muy concreto. Esto es imprescindible si estamos, por ejemplo, en situaciones de hardware empotrado (*embedded*).

- Sintonizar, de esta última manera, el consumo de memoria u otros recursos para adaptarse mejor a recursos limitados (CPU, memoria, disco, etc.).

- Versiones concretas de las distribuciones implican usar ciertas versiones específicas del núcleo. En muchos casos no podemos obtener nuevas actualizaciones del núcleo si no actualizamos la distribución concreta completamente a una nueva *release* de esta.

- Probar versiones de núcleo todavía no disponibles en las distribuciones. Aunque hay cierta rapidez de integración de los nuevos núcleos, se puede tardar semanas o meses en disponer de los nuevos núcleos vía distribución. Por otra parte, algunas distribuciones son muy conservadoras en cuanto a

la versión de núcleo utilizada, y hay que esperar varias versiones completas de la distribución (un periodo largo de tiempo, por ejemplo 6 meses) para llegar a una versión concreta de núcleo.

- Probar versiones beta, o parches de núcleo, con el objetivo de integrarlo rápidamente en algún sistema con problemas concretos, o bien sencillamente por cuestiones de evaluación de las nuevas posibilidades o de un mejor rendimiento.

- Participar directamente en el desarrollo del núcleo, estudiando posibles mejoras del código actual, proponiendo y probando propuestas concretas. Es típico de algunos componentes, así como estudiar diferentes estrategias de planificación de CPU, de gestión de memoria, mejorar parámetros del núcleo o colocar alguna prestación nueva a algún sistema de ficheros.

En los siguientes subapartados veremos las diferentes posibilidades en cuanto a la configuración y compilación de las diferentes ramas de desarrollo del núcleo Linux.

3.1. Compilación de las ramas 2.6.x y 3.x del kernel Linux

Las instrucciones son específicas para la arquitectura x86/x86_64(o amd64) de Intel, mediante usuario no-root la mayor parte del proceso (puede hacerse como usuario normal y, de hecho, es altamente aconsejable por seguridad), y únicamente el proceso final de instalación es necesario root para integrar el kernel final en el sistema. La compilación del kernel es un proceso costoso en tiempo y disco para los kernels actuales, se recomienda disponer de espacio de disco abundante (100 GB, o más, para kernels actuales), y realizar el proceso con el máximo de CPU y memoria disponible para acelerar el proceso. Las diferentes fases que están involucradas son:

1) Obtener el núcleo. Podemos acudir a http://www.kernel.org (o a su servidor ftp) y descargar la versión *stable* o *longterm* que queramos probar. Por otro lado, en la mayoría de las distribuciones de GNU/Linux, como Fedora/Red Hat o Debian, también ofrecen como paquete el código fuente del núcleo (normalmente con algunas modificaciones incluidas); si se trata de la versión del núcleo que necesitamos, quizá sea preferible usar estas (mediante los paquetes `kernel-source`, `kernel-version`, `linux-source`, o similares). Si queremos los últimos núcleos, quizá no estén disponibles en la distribución y tendremos que acudir a *kernel.org*.

2) Desempaquetar el núcleo. Los paquetes fuente del núcleo solían colocarse y desempaquetarse sobre el directorio `/usr/src`, aunque se recomienda utilizar algún directorio aparte para no mezclarlos con ficheros fuente que pueda traer la distribución. Por ejemplo, si los paquetes fuente venían en un fichero comprimido de tipo bzip2:

```
bzip2 -dc linux-2.6.32.63.tar.bz2 | tar xvf -
```

Si los paquetes fuente venían en un fichero `gz` (o `tar.gz`), reemplazamos bzip2 por gzip, o en los recientes (usados por kernel.org) `.tar.xz` o `.xz` reemplazamos por `unxz` (paquete xz-utils). Al descomprimir los paquetes fuente se habrá generado un directorio llamado `linux-version_kernel`, donde entraremos para establecer la configuración del núcleo.

En este punto, podría ser interesante parchear el núcleo, lo cual podríamos realizar debido a que tenemos un código fuente adicional (en forma de parche) que mejora algún problema conocido de la versión o bien porque queremos proponer o probar un cambio de código en el núcleo. También podría darse el caso de que un fabricante de hardware ofreciese algún soporte o corrección de fallos para un controlador de dispositivo, como un parche para una versión de núcleo concreta.

Ved también

El proceso de parchear el núcleo se trata en el apartado 4.

Una vez dispongamos de los paquetes necesarios, procedemos al proceso de compilación en el directorio utilizado para los paquetes fuente. Cabe recordar que todo el proceso puede realizarse como usuario normal, solo partes muy concretas, como la instalación final del núcleo o de módulos dinámicos, es necesario hacerlas usando el usuario root.

También pueden surgir problemas de seguridad por realizar la compilación en modo root de fuentes de núcleo desconocidas o no fiables. Al respecto, tanto los paquetes fuente de *kernel.org* como las proporcionadas por las distribuciones suelen contener firmas (o repositorios firmados) que pueden usarse para verificar la integridad de los ficheros de fuentes. Hay que evitar, a toda costa, usar fuentes de núcleo o de módulos proporcionados por emisores no fiables.

Herramientas de configuración y compilación

Antes de comenzar los pasos previos a la compilación, debemos asegurarnos de disponer de las herramientas correctas, en especial del compilador gcc, make y otras utilidades gnu complementarias en el proceso. Un ejemplo son las *module-init-tools*, que ofrecen las diferentes utilidades para el uso y gestión de los módulos de núcleo dinámicos. Asimismo, para las diferentes opciones de configuración hay que tener en cuenta una serie de prerrequisitos en forma de bibliotecas asociadas a la interfaz de configuración usada (por ejemplo, las ncurses para la interfaz menuconfig).

Por ejemplo, en una distribución Debian estable (en otras distribuciones u versiones, consultar los paquetes similares), entre otros son necesarios los paquetes: *build-essential libncurses5-dev libqt4-dev libqt4-qt3support qt4-qmake libgtk2.0-dev libglib2.0-dev libglade2-dev modules-init-tools gcc g++*, en el caso de Debian, necesitaremos una instalación previa de estos paquetes, por ejemplo, realizada con la lista anterior, y un *apt-get install* de los paquetes mencionados.

Se recomienda, en general, consultar la documentación del núcleo (ya sea vía paquete o en el directorio raíz de las fuentes del núcleo) para conocer qué prerrequisitos, así como versiones de estos, son necesarios para el proceso. Se recomienda examinar los ficheros `README` en el directorio "raíz", y el `Documentation/Changes`, que cita los requisitos mínimos de versiones de los programas prerrequisitos de la compilación, o el índice de documentación del núcleo en `Documentation/00-INDEX`.

Si hemos realizado anteriores compilaciones en el mismo directorio, deberemos garantizar que el directorio utilizado esté limpio de compilaciones anteriores; podemos limpiarlo con `make mrproper` (realizado desde el directorio raíz de las fuentes).

3) Configuración del núcleo. Para el proceso de configuración del núcleo [Vasb], tenemos varios métodos alternativos, que nos presentan interfaces diferentes para ajustar los múltiples parámetros del núcleo (que suelen almacenarse en un fichero de configuración, normalmente `.config` en el directorio "raíz" de las fuentes). Las diferentes alternativas son:

- `make config`: desde la línea de comandos se nos pregunta por cada opción y se nos pide confirmación (y/n), si deseamos o no la opción, o se nos piden los valores necesarios, o seleccionar una de las posibilidades de la opción. Es la configuración larga, en la que se nos piden muchas respuestas; podemos tener que responder a varios centenares de preguntas (o más dependiendo de la versión).

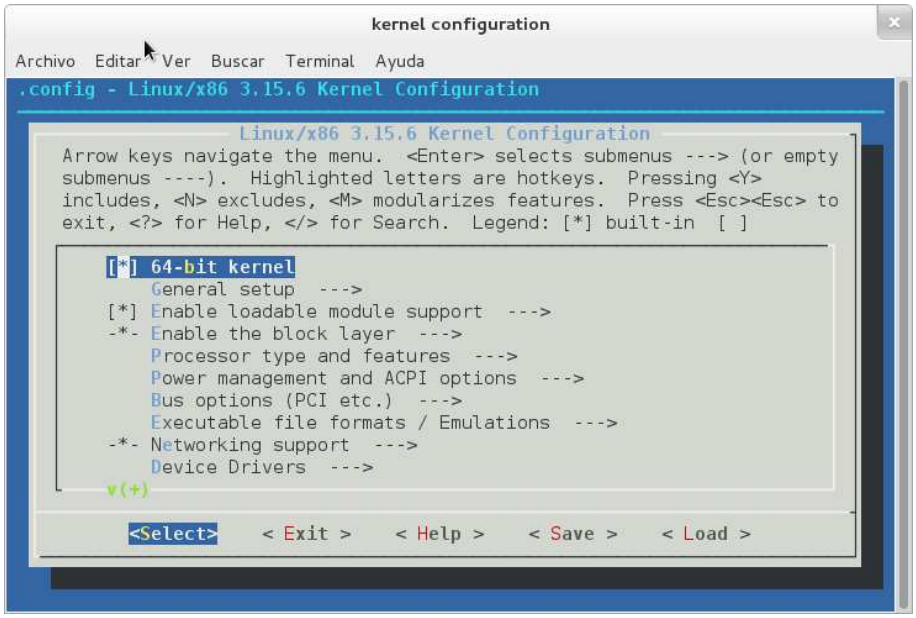
- `make oldconfig`: sirve por si queremos reutilizar una configuración ya usada (normalmente almacenada en un fichero `.config`, en el directorio "raíz" de las fuentes previas que hayamos compilado con anterioridad); hay que tener en cuenta que solo es válida si estamos compilando la misma versión del núcleo, ya que diferentes versiones del núcleo pueden variar en sus opciones. Pero en las últimas versiones del núcleo, esta opción detecta la configuración anterior, y solamente nos pregunta por las opciones nuevas disponibles en la nueva versión, ahorrando un tiempo considerable en las elecciones. Por otra parte, en general la configuración de un kernel disponible en la distribución está también guardada en el fichero `/boot/config-version` correspondiente (recordar que con `uname -r` tenemos la versión del kernel actual), el cual podemos copiar directamente como fichero `.config` en la raíz de los fuentes del kernel, y con `make oldconfig` configurar únicamente las nuevas opciones.

- `make menuconfig`: configuración basada en menús textuales, bastante cómoda (figura 2); podemos habilitar o inhabilitar lo que queramos, y es más rápida y versátil para la navegación entre opciones que el `make config`.

- `make xconfig`: la más cómoda, basada en diálogos gráficos en X Window. Es necesario tener instaladas las bibliotecas de desarrollo de qt3 o qt4, ya que esta configuración tiene estos requerimientos al estar basada en la librería Qt3/Qt4 (KDE). La configuración se basa en cuadros de diálogo, botones y casillas de activación. Es bastante rápida y dispone de ayuda con comentarios de muchas de las opciones.

- `make gconfig`: similar al caso de xconfig, pero con la interfaz basada en gtk (Gnome). Como prerrequisito necesita los paquetes de desarrollo de las librerías Gnome (entre otros *libgtk2.0-dev*, *libglib2.0-dev* y *libglade2-dev*, los nombres son dependientes de la distribución y versión).

Figura 2. Configuración del núcleo 3.x desde la interfaz textual de menuconfig



Una vez se haya hecho el proceso de configuración, hay que guardar el fichero `.config`, ya que la configuración consume un tiempo importante. Además, puede ser de utilidad disponer de la configuración realizada (`.config`) si se está planeando hacerla posteriormente en varias máquinas parecidas o idénticas.

Otro tema importante en las opciones de configuración es que en muchos casos se nos va a preguntar si una determinada característica la queremos integrada en el núcleo o como módulo. Esta es una decisión más o menos importante, ya que el tamaño y el rendimiento del núcleo (y, por tanto, del sistema entero) en algunos casos puede depender de nuestra elección.

Ved también

La gestión de módulos se trata en el apartado 5.

El núcleo de Linux, como imagen binaria una vez compilado, ha comenzado a ser de gran tamaño, tanto por complejidad como por los controladores de dispositivo [Cor05] que incluye. Si integrásemos todas sus posibilidades, se podría crear un fichero del núcleo bastante grande y ocupar mucha memoria, lo que ralentizaría algunos aspectos de funcionamiento. Los módulos del núcleo [Hen] son un método que permite separar parte del núcleo en pequeñas porciones, que se cargarán dinámicamente bajo demanda cuando, por carga explícita o por uso de sus características, sean necesarias.

La elección más normal es integrar dentro del núcleo lo que se considere básico para el funcionamiento, o crítico en rendimiento, y dejar como módulos aquellas partes o controladores de los que se vaya a hacer un uso esporádico o que se necesite conservar por si se producen futuras ampliaciones del equipo.

- Un caso claro son los controladores de dispositivo: si estamos actualizando la máquina, puede ser que a la hora de crear el núcleo no conozcamos con

seguridad qué hardware va a tener, por ejemplo, qué tarjeta de red, pero sí que sabemos que estará conectada a red; en este caso, el soporte de red estará integrado en el núcleo, pero por lo que respecta a los controladores de las tarjetas, podremos seleccionar unos cuantos (o todos) y ponerlos como módulos. Así, cuando tengamos la tarjeta podremos cargar el módulo necesario, o si después tenemos que cambiar una tarjeta por otra, solo tendremos que cambiar el módulo que se va a cargar. Si hubiese solo un controlador integrado en el núcleo y cambiamos la tarjeta, esto obligaría a reconfigurar y recompilar el núcleo con el controlador de la tarjeta nueva.

- Otro caso que suele aparecer (aunque no es muy común) es cuando necesitamos dos dispositivos que son incompatibles entre sí, o está funcionando uno o el otro (esto pasaba antiguamente, por ejemplo, con impresoras con cable paralelo y algunos dispositivos hardware que se conectaban al puerto paralelo). Por lo tanto, en este caso tenemos que colocar como módulos los controladores y cargar o descargar el que sea necesario en cada momento. Aunque también se han solucionado actualmente estos problemas, con soportes de daemons que controlan casos de *(*hotplug) (conexión en caliente) de dispositivos, como por ejemplo *udev*, que aporta soluciones en este sentido, gestionando dinámicamente las entradas del directorio `/dev` a medida que los dispositivos se conectan o desconectan en el sistema.

- Otro ejemplo podrían formarlo los sistemas de ficheros (*filesystems*). Normalmente esperaremos que nuestro sistema tenga acceso a algunos de ellos, por ejemplo ext2, ext3 o ext4 (propios de Linux), vfat (de los Windows 95/98/ME) y los daremos de alta en la configuración del núcleo. Si en otro momento tuviéramos que leer otro tipo no esperado, por ejemplo datos guardados en un disco o partición de sistema NTFS de Windows NT/XP, no podríamos: el núcleo no sabría o no tendría soporte para hacerlo. Si tenemos previsto que en algún momento (pero no habitualmente) se tenga que acceder a estos sistemas, podemos dejar los demás sistemas de ficheros como módulos.

4) Compilación del núcleo. Una vez disponible una configuración de la versión del núcleo, mediante `make` comenzaremos el proceso de compilación:

```
$ make
```

Este proceso puede acelerarse si disponemos de una máquina multiprocesador o multicore; `make` dispone de la opción -jn, con la que podemos especificar, con n, el número de procesadores o cores que emplearemos para la compilación.

Cuando este proceso acabe, tendremos la parte integral del núcleo y nos faltarán las partes que se pidieron colocar como módulos:

```
$ make modules
```

Atajos Debian

Algunas distribuciones permiten modos más simplificados, por ejemplo Debian, permite mediante *make deb-pkg* realizar todo el proceso de compilación y preparar la instalación mediante la creación de los paquetes `.deb` para la instalación del kernel. Finalmente solo quedará instalar con *dpkg -i* los paquetes resultantes.

Hasta este momento hemos hecho la configuración y compilación del núcleo. Esta parte podía hacerse desde un usuario normal (altamente recomendable) o bien el root, pero ahora necesitaremos forzosamente usuario root, porque pasaremos a la parte de la instalación del nuevo kernel compilado en el sistema.

5) Instalación. Comenzamos instalando los módulos:

```
# make modules_install
```

esto nos instalará los módulos construidos en el sistema de fichero para que el nuevo kernel los pueda encontrar en el lugar adecuado. Normalmente los módulos están disponibles en el directorio `/lib/modules/version_kernel`, donde `version_kernel` será la numeración del kernel que acabamos de construir.

Y para la instalación del nuevo núcleo (desde el directorio de compilación `dir/linux-version`, siendo `version` la versión usada), simplemente con:

```
# make install
```

Para este último paso la mayoría de distribuciones incluyen un soporte externo con un script denominado *installkernel* (normalmente proporcionado por el paquete *mkinitrd*), que es usado por el sistema de compilación del kernel para colocarlo en el lugar adecuado, modificar el *bootloader* (lilo o Grub), de manera que no se deban realizar pasos extras. Hay que señalar que algunas de tipo *from scratch*, como *Gentoo*, no incluyen el script, por tanto consultar la documentación específica de kernel para estas distribuciones.

En el proceso de esta última fase *install*, se desarrollan las siguientes fases:

- Se verifica que el kernel haya estado bien construido.

- Se instala la porción estática del kernel en el directorio `/boot`, y se nombra el fichero con la versión del kernel que se ha construido (típicamente `/boot/vmlinuz-version`).

- Las imágenes de ramdisk que puedan ser necesarias son creadas usando aquellos módulos que se hayan creado previamente y sean necesarios para tales imágenes en arranque de la màquina. Típicamente esta imagen aparecerá como `/boot/initrd.img-version`. Respecto a la creación de *initrd*, hace falta disponer de las herramientas adecuadas; en Debian, por ejemplo, en el proceso de compilación es necesario como prerrequisito el paquete *initramfs-tools* (para kernels superiores a 2.6.12, antiguamente cumplían esta función las *mkinitrd-tools*, ya obsoletas). También durante estas primeras fases se suelen generar los ficheros `/boot/System.map-version` y `/boot/config-version`, que contienen, respectivamente, información

sobre los símbolos disponibles en el kernel y la configuración empleada en el kernel en su compilación.

- El *bootloader* (lilo/Grub) correspondiente del sistema es notificado de la existencia del nuevo kernel, creándose la entrada necesaria del menú, de manera que el usuario pueda seleccionarlo en el siguiente arranque de la máquina.

Finalmente, después de estas fases, el kernel está instalado correctamente y podemos proceder al reinicio, y probar nuestra imagen nueva del kernel. Si el proceso se ha realizado correctamente, dispondremos de una entrada de *bootloader* correspondiente, y en caso de que surgieran problemas con la nueva imagen, dispondríamos de las entradas de los kernels antiguos para volver a la situación estable anterior.

En la instalación dependiendo del kernel, y de su uso posterior, puede también ser necesaria la instalación de los *headers* de desarrollo del kernel, y de los *firmwares* asociados (los últimos kernels han incluido algunos *firmwares* de dispositivos directamente en el kernel), estos últimos procesos los desarrollaremos con:

```
# make headers_install
# make firmware_install
```

finalizando de esta manera el proceso de instalación del kernel completo.

Este último proceso, de instalación automática a través de *make install* (y añadidos), también puede realizarse de forma manual, si no se dispone del script antes comentado (installkernel), con el siguiente proceso (algunos detalles dependen de la arquitectura):

```
# make modules install
# make kernelversion
```

(el último comando obtiene un número KERNEL_VERSION de la versión construida, en lo siguiente colocar el número obtenido donde aparezca la expresión KERNEL_VERSION)

```
# cp arch/i386/boot/bzImage /boot/bzImage-KERNEL_VERSION
# cp System.map /boot/System.map-KERNEL_VERSION
```

El archivo `bzImage` es el núcleo recién compilado, que se coloca en el directorio `/boot`. Normalmente el núcleo antiguo se encontrará en el mismo directorio boot con el nombre `vmlinuz` o bien `vmlinuz-versión-anterior` y `vmlinuz` como un enlace simbólico al núcleo antiguo. Una vez tengamos nuestro núcleo, es mejor conservar el antiguo, por si se producen fallos o un mal funcionamiento del nuevo y así poder recuperar el viejo. El fichero

`System.map`, un fichero que contiene los símbolos disponibles en el núcleo, necesario para el proceso de arranque del núcleo, también se coloca en el mismo directorio `/boot`.

También puede ser necesaria la creación de la imagen de disco ramdisk `initrd` con las utilidades necesarias según la distribución (o su versión).

Respecto a la creación del `initrd`, en Fedora/Red Hat este se creará automáticamente con la opción `make install`. En Debian deberemos usar las técnicas conocidas como *Debian way* o bien crearlo explícitamente con `mkinitrd` (versiones de núcleo $\leq$ 2.6.12) o, posteriormente con kernels actuales usar, bien `mkinitramfs`, o una utilidad denominada `update-initramfs`, especificando la versión del núcleo (se asume que este se llama `vmlinuz-version` dentro del directorio `/boot`):

```
# update-initramfs -c -k 'version'
```

aunque cabe señalar que en las distribuciones actuales, y versiones del kernel, es habitual que el initramfs se cree solo en el proceso de compilación y posterior instalación (en el *make install* igualmente). Aún así se recomienda la remodelación de este (vía previa configuración en */etc/initramfs-tools/initramfs.conf*), debido a que suele presentar un tamaño respetable, que puede ser ineficiente en algunas máquinas por incluir más módulos de los imprescindibles, para lo que puede jugarse (en el susodicho initramfs.conf) con la configuración *MODULES=most* o *dep* (que seguramente minimizará el tamaño de forma significativa). Después podemos recrear el initrd mediante el comando anterior, o por ejemplo substituyendo los parámetros por:

```
#update-initramfs -t -u -k version
```

que en este caso nos actualizará el initrd si el proceso de instalación ya nos creó uno previo.

6) Configuración del arranque. El siguiente paso es decirle al sistema con qué núcleo tiene que arrancar. Este paso depende del sistema de arranque de Linux, y del *bootloader* usado:

- Desde arranque con LiLo [Skoa], ya sea en el Master Boot Record (MBR) o desde partición propia, hay que añadir al fichero de configuración (en `/etc/lilo.conf`), por ejemplo, las líneas:

  ```
  image = /boot/bzImage-KERNEL_VERSION
      label = KERNEL_VERSION
  ```

  donde `image` es el núcleo que se va arrancar y `label` será el nombre con el que aparecerá la opción en el arranque. Podemos añadir estas líneas

o modificar las que hubiera del núcleo antiguo. Se recomienda añadirlas y dejar el núcleo antiguo para poder recuperarlo si aparecen problemas. En el fichero `/etc/lilo.conf` puede haber una o más configuraciones de arranque, tanto de Linux como de otros sistemas (como Windows); cada arranque se identifica por su línea `image` y el `label` que aparece en el menú de arranque. Hay una línea "`default=label`" donde se indica el label por defecto que se arrancará. También podemos añadirle a las líneas anteriores un "`root=/dev/...`" para indicar la partición de disco donde estará el sistema de archivos principal (el /). Recordar que los discos tienen dispositivos como `/dev/sda` (primer disco) `/dev/sdb` (segundo), y la partición se indicaría como "`root=/dev/sda2`" si el / de nuestro Linux estuviese en la segunda partición del primer disco. Además, con `append=` podemos añadir parámetros al arranque del núcleo. Después de cambiar la configuración del LiLo hay que escribirla físicamente en el disco (se modifica el sector de arranque) para que esté disponible en el siguiente arranque:

```
/sbin/lilo -v
```

- Arranque con Grub: La gestión en este caso es bastante simple; cabe añadir una nueva configuración formada por el núcleo nuevo y añadirla como una opción más al fichero de configuración del Grub, y rearrancar procediendo de forma parecida a la del LiLo, pero recordando que en Grub basta con editar el fichero y rearrancar. También es mejor dejar la antigua configuración para poder recuperarse de posibles errores o problemas con el núcleo recién compilado. Hay que señalar que la configuración de Grub (normalmente disponible en `/boot/grub/menu.lst` o `grub.cfg`) depende mucho de la versión, sea Grub-Legacy (Grub 1.x) o Grub 2, se recomienda examinar la configuración de kernels ya presentes, para adaptar el recién creado, y consultar la documentación GNU, y el manual disponible vía *info grub*.

Lectura recomendada

Sobre Grub podéis consultar *Grub bootloader* y *Grub Manual* accesibles desde la web del proyecto GNU: http://www.gnu.org

Una vez disponemos de los ficheros correctos en `/boot` y del *bootloader* actualizado, ya podemos proceder al rearranque con `shutdown -r now`, escoger nuestro núcleo y, si todo ha ido bien, podremos comprobar con `uname -r` que disponemos de la nueva versión del núcleo. También es particularmente interesante examinar algunos registros, como `/var/log/messages` (u logs equivalentes dependiendo de la distribución) y el comando `dmesg`, para examinar el registro de salida de mensajes producidos por el nuevo núcleo en el arranque y detectar si ha aparecido algún problema de funcionalidad o con algún dispositivo concreto.

Si tuviésemos problemas, podemos recuperar el antiguo núcleo, escogiendo la opción del viejo núcleo, y luego retocar el `bootloader` para volver a la antigua configuración o estudiar el problema y reconfigurar y recompilar el núcleo de nuevo.

3.2. Compilación del núcleo en Debian (*Debian Way*)

En Debian, además de los métodos comentados en los subapartados previos, hay que añadir la configuración por el método denominado *Debian Way*. Es un método que nos permite construir el núcleo de una forma flexible y rápida, adaptada a la distribución.

Enlace de interés

Para Fedora recomendamos consultar el siguiente enlace: http://fedoraproject.org/wiki/Building_a_custom_kernel.

Para el proceso necesitaremos una serie de utilidades (hay que instalar los paquetes o similares): `kernel-package`, `ncurses-dev`, `fakeroot`, `wget` y `bzip2`.

Podemos observar el método [Debk] desde dos perspectivas: reconstruir un núcleo equivalente al proporcionado por la distribución como núcleo base (cambiando opciones) o bien crear un núcleo con una numeración de versión-revisión personalizada.

Enlace de interés

Puede verse el proceso *Debian Way* de forma detallada en: http://kernel-handbook.alioth.debian.org/

Respecto a los paquetes de fuentes del núcleo, Debian proporciona los paquetes fuente usados en su distribución, que pueden llegar a ser bastante diferentes de los de la versión *vanilla* o *pristine* obtenida como estable de *kernel.org*. Ello es debido a que en Debian producen múltiples revisiones con diversos parches que van añadiendo, muchos de ellos a partir de fallos que se detectan *a posteriori* en las siguientes versiones *vanilla* del núcleo.

En las versiones estables de Debian, la distribución suele escoger una revisión xx de la rama 2.6.xx o 3.xx, de manera que el núcleo suele quedarse (generalmente) en esta numeración para toda la vida de la versión concreta de Debian estable, y así, cuando se actualiza con revisiones menores la distribución, solo se actualizan parches del núcleo (sin cambiar el número principal). Cuando Debian produce la siguiente versión estable se salta a una nueva versión del núcleo. Durante la duración de una versión estable de la distribución, Debian suele producir diferentes modificaciones (*patchlevels*) o revisiones del núcleo que se escogió.

Debian ha cambiado varias veces la gestión de sus paquetes asociados a los paquetes fuente del núcleo. A partir de la versión 2.6.12 es habitual encontrar en el repositorio Debian una versión `linux-source-versión` que contiene la versión de los fuentes del núcleo con los últimos parches aplicados (véase el apartado 4). Esta versión del paquete de los fuentes del núcleo es la que usaremos para crear un núcleo personalizado, en la mencionada *Debian way*. Este paquete fuente es usado para crear los paquetes binarios de la distribución asociados al núcleo y también es el indicado para usar en caso de querer aplicar parches al núcleo actual de la distribución, o por si queremos probar modificaciones del núcleo a nivel de código.

Examinemos primero esta opción y después, al final del subapartado, comentaremos la personalización.

Para la reconstrucción del kernel actual Debian, podemos proceder de dos maneras: bien obtenemos el kernel directamente de los fuentes (con todos los parches incluidos) o bien reconstruimos los paquetes Debian oficiales del kernel.

En el primer caso, obtenemos las fuentes directas, y para la compilación del núcleo actual procedemos usando el paquete disponible como *linux-source-version*; en este caso de ejemplo version=3.2 (versión principal del kernel para una Debian estable concreta) como versión del kernel Debian de la distribución, pero dependiendo en cada momento del kernel actual, tendremos que escoger los fuentes que se correspondan con la versión principal de nuestro kernel actual (ver *uname -r*):

```
# apt-get install linux-source-3.2
$ tar jxf /usr/src/linux-source-3.2.tar.bz2
```

En este caso las descargará normalmente en `/usr/src`, aunque podemos copiarlas a otro espacio o disco de trabajo, ya que el proceso necesita un espacio importante (cerca de 10 GB o más), si no disponemos de él en `/usr/src`. Una vez tengamos el fuente, con tar, descomprimirá los paquetes fuente dejándolos en un árbol de directorios a partir del directorio `linux-version`, donde versión ahora será la revisión del kernel disponible (3.2.xx). Para la configuración y compilación posterior a partir de este directorio, podemos realizarla por el método clásico o por la *Debian Way* que examinamos en este apartado.

Para el segundo caso (la reconstrucción de los paquetes Debian oficiales), el objetivo es obtener unos paquetes *deb* finales equivalentes a los que ofrece la distribución, pero con la personalización que queramos realizar.

Para este proceso, comenzaremos por obtener algunas herramientas que necesitaremos:

```
# apt-get install build-essential fakeroot
# apt-get build-dep linux
```

Estas líneas básicamente nos instalan el entorno de compilación para el núcleo (de hecho los paquetes del compilador gcc necesarios y herramientas propias de la *Debian Way*) y por último se comprueban las dependencias de los fuentes por si son necesarios paquetes fuente extra de desarrollo. Después de esta configuración de herramientas iniciales, podemos pasar a la descarga de los fuentes (puede hacerse desde cualquier usuario, no es necesario root para el proceso):

```
$ apt-get source linux
```

que nos descargará los fuentes y los descomprimirá en el directorio actual, además de otros paquetes que incluyen parches respecto a la versión original del kernel. El directorio necesario donde comenzar el proceso lo encontraremos como `linux-version`. En este momento es cuando podremos personalizar la configuración del kernel por los métodos examinados en la sección previa, ya que la configuración es la que trae el kernel Debian por defecto, con lo que podremos cambiar aquellos componentes u opciones del kernel que nos interesen antes de volver a reconstruir los paquetes.

Para realizar la construcción de los paquetes kernel para la arquitectura (se recomiendan como mínimo 20 GB disponibles de disco), según la configuración preestablecida del paquete (semejante a la incluida en los paquetes oficiales del núcleo `linux-image` en Debian):

```
$ cd linux-version
$ fakeroot debian/rules binary
```

Con esto dispondremos de los paquetes binarios (y otros de soporte independientes de la arquitectura) del núcleo (ficheros `*.deb`) disponibles para la instalación (vía `dpkg -i`).

Existen algunos procedimientos extra para la creación del núcleo en base a diferentes niveles de parche (*patch*) proporcionados por la distribución, y posibilidades de generar diferentes configuraciones finales (puede verse la referencia de la nota para complementar estos aspectos).

Pasamos ahora a la otra opción que habíamos comentado al inicio, a los aspectos de personalización, cuando queremos cambiar ciertas opciones del núcleo y crear una versión personal del mismo (a la que daremos una numeración de revisión propia).

En este caso, más habitual, cuando deseamos un núcleo personalizado, deberemos realizar un proceso semejante a través de un paso de personalización típico (por ejemplo, mediante `make menuconfig` o `oldconfig`). Los pasos son, en primer lugar, la obtención y preparación del directorio (aquí obtenemos los paquetes de la distribución, pero es equivalente obteniendo los paquetes fuente desde *kernel.org*). En este caso sea 3.x la versión de kernel disponible de fuentes en el repositorio, o también comenzando en el segundo paso, el kernel estable obtenido de *kernel.org* (este último quizás esté en formato *xz* en lugar de *bzip2*):

```
# apt-get install linux-source-3.2
$ tar -xvjf /usr/src/linux-source-3.2.tar.bz2
$ cd linux-source-3.2
```

de donde obtenemos los paquetes fuente y los descomprimimos (la instalación del paquete deja el archivo de fuentes en `/usr/src`; en el caso de que proceda de kernel.org ya depende de donde hayamos realizado la descarga inicial). Es recomendable que antes de proceder al paso de descompresión con tar, lo realicemos en un espacio de disco con capacidad suficiente (recomendados 10-20 GB).

A continuación realizamos la configuración de parámetros. Como siempre, podemos basarnos en ficheros `.config` que hayamos utilizado anteriormente para partir de una configuración conocida (para la personalización, con *make oldconfig*, también puede usarse cualquiera de los otros métodos, `xconfig`, `gconfig`, etc.):

```
$ make menuconfig
```

El kernel preparado de esta manera, a no ser que indiquemos lo contrario, tiene la depuración activada, lo que resulta muy útil para la depuración en caso de errores (utilizando herramientas como *kdump, crash* o *SystemTap*), aunque, por otra parte, nos aumenta el espacio necesario del kernel para su construcción; cuando esto no sea necesario, antes de pasar a la construcción podemos deshabilitar la opción de incluir información de depuración:

```
$ scripts/config --disable DEBUG_INFO
```

A continuación, la construcción final del núcleo:

```
$ make clean
$ make KDEB_PKGVERSION=custom.1.0 deb-pkg
```

donde creamos un identificador para el núcleo construido (`custom.1.0`) que se añadirá al nombre del paquete binario del núcleo, posteriormente visible en el arranque con el comando `uname`.

Así, el proceso finalizará con la obtención de los paquetes asociados a la imagen del núcleo, que podremos finalmente instalar (examinar de los pasos anteriores el nombre del paquete obtenido, que sera el que incluiremos aquí):

```
# dpkg -i ../linux-image-3.xx.yy_custom.1.0_i386.deb
```

Esto nos descomprimirá e instalará el núcleo y generará una imagen `initrd` adicional si fuera necesario. Además, nos configura el *bootloader* con el nuevo núcleo por defecto (hay que vigilar con este paso: vale la pena haber obtenido

antes una copia de seguridad del *bootloader* para no perder ninguna configuración estable).

Ahora directamente con un `shutdown -r now` podemos probar el arranque con el nuevo núcleo.

Añadimos también otra peculiaridad a tener en cuenta en Debian, que puede ser útil con algún hardware: la existencia de utilidades para añadir módulos dinámicos de núcleo proporcionados por terceros; en particular, la utilidad `module-assistant`, que permite automatizar todo este proceso a partir de los paquetes fuente del módulo.

Enlace de interés

Habitualmente no será necesario, pero si se necesita alguna reconfiguración del `initrd` generado, se recomienda leer el siguiente enlace, donde se comentan algunas herramientas Debian disponibles: http://kernel-handbook.alioth.debian.org/ch-initramfs.html

Necesitamos disponer de los *headers* del núcleo instalado (disponible en el paquete `linux-headers-version`) o bien de los paquetes fuente que utilizamos en su compilación (también pueden obtenerse durante la instalación del núcleo mediante un `make headers_install`, o mediante el paquete deb obtenido con el *Debian way*). A partir de aquí `module-assistant` puede utilizarse interactivamente y seleccionar entre una amplia lista de módulos registrados previamente en la aplicación, y puede encargarse de descargar el módulo, compilarlo e instalarlo en el núcleo existente.

En la utilización desde la línea de comandos, también podemos simplemente especificar (`m-a`, equivalente a `module-assistant`):

```
# m-a prepare
# m-a auto-install nombre_modulo
```

Así se prepara el sistema para posibles dependencias, descarga fuentes del módulo, compila y, si no hay problemas, instala para el presente núcleo. En la lista interactiva de `module-assistant` podemos observar el nombre de los módulos disponibles.

4. Aplicación de parches al núcleo

En algunos casos también puede ser habitual la aplicación de parches (*patches*) al núcleo [Lkm].

Un fichero de parche (*patch file*) respecto al núcleo de Linux es un fichero de texto ASCII que contiene las diferencias entre el código fuente original y el nuevo código, con información adicional de nombres de fichero y líneas de código. El programa *patch* (ver `man patch`) sirve para aplicarlo al árbol del código fuente del núcleo (normalmente, dependiendo de la distribución, en `/usr/src/linux`).

Los parches suelen necesitarse cuando un hardware especial necesita alguna modificación en el núcleo, o se han detectado algunos errores (*bugs*) de funcionalidad posteriores a alguna distribución concreta de una versión del núcleo, o bien quiere añadirse una nueva prestación sin generar una versión de núcleo nueva. Para corregir el problema (o añadir la nueva prestación), se suele distribuir un parche en lugar de un nuevo núcleo completo. Cuando ya existen varios de estos parches, se unen con diversas mejoras del núcleo anterior para formar una nueva versión del mismo. En todo caso, si tenemos hardware problemático o el error afecta a la funcionalidad o a la estabilidad del sistema y no podemos esperar a la siguiente versión del núcleo, será necesario aplicar el/los parche(s).

El parche se suele distribuir en un fichero comprimido tipo bz2 (bunzip2, aunque también puede encontrarse en gzip con extensión .gz, o xz con extensión .xz), como por ejemplo podría ser:

```
patchxxxx-version-pversion.xz
```

donde xxxx suele ser algún mensaje sobre el tipo o finalidad del parche, versión sería la versión del núcleo al cual se le va a aplicar el parche, y `pversion` haría referencia a la versión del parche, del que también pueden existir varias. Hay que tener en cuenta que estamos hablando de aplicar parches a los paquetes fuente del núcleo (normalmente instalados, como vimos, en `/usr/src/linux` o directorio similar del usuario usado en la compilación del núcleo).

Una vez dispongamos del parche, tendremos que aplicarlo. Veremos el proceso a seguir en algún fichero `Readme` que acompaña al parche, pero generalmente el proceso sigue los pasos (una vez comprobados los requisitos previos) de descomprimir el parche en el directorio de los ficheros fuente y aplicarlo sobre las fuentes del núcleo, como por ejemplo:

```
cd /usr/src/linux (o /usr/src/linux-version, o directorio que usemos).
unxz patch-xxxxx-version-pversion.xz
patch -p1 < patch-xxxxx-version-pversion
```

También puede aplicarse previamente con la opción `patch -p1 -dry-run` que solo procede a realizar un primer test, para asegurarnos previamente que no haya alguna condición de error cuando se sustituya el código. Si no hay error, volvemos a aplicar entonces sin la opción de test.

Posteriormente, una vez aplicado el parche, tendremos que recompilar el núcleo para volverlo a generar.

Los parches pueden obtenerse de diferentes lugares. Lo más normal es encontrarlos en el sitio de almacén de los núcleos *vanilla* (http://www.kernel.org), que tiene un archivo completo de los mismos. Determinadas comunidades Linux (o usuarios individuales) también suelen ofrecer algunas correcciones, pero es mejor buscar en los sitios estándar para asegurar un mínimo de confianza en estos parches y evitar problemas de seguridad con posibles parches "piratas". Otra vía es el fabricante de hardware que puede ofrecer ciertas modificaciones del núcleo (o de controladores en forma de módulos dinámicos de núcleo) para que funcionen mejor sus dispositivos (un ejemplo conocido es NVIDIA y sus controladores Linux propietarios para sus tarjetas gráficas).

Por último, señalaremos que muchas distribuciones de GNU/Linux (Fedora/Red Hat, Debian) ya ofrecen núcleos parcheados por ellos mismos y sistemas para actualizarlos (algunos incluso de forma automática, como en el caso de Fedora/Red Hat y Debian). Normalmente, en sistemas de producción es más recomendable seguir las actualizaciones del fabricante, aunque éste no ofrecerá necesariamente el último núcleo publicado, sino el que crea más estable para su distribución, con el inconveniente de perder prestaciones de última generación o alguna novedad en las técnicas incluidas en el núcleo.

Núcleo actualizado

En sistemas que se quieran tener actualizados, por razones de test o de necesidad de las últimas prestaciones, siempre se puede acudir a http://www.kernel.org y obtener el núcleo más moderno publicado, siempre que la compatibilidad de la distribución lo permita.

Por último, cabe comentar la incorporación de una tecnología comercial al uso de parches en Linux, Ksplice (mantenida por Oracle), que permite a un sistema Linux añadir parches al núcleo sin necesidad de parar y rearrancar el sistema. Básicamente, Ksplice determina a partir de los paquetes fuente cuáles son los cambios introducidos por un parche o una serie de ellos, y comprueba en memoria cómo afectan a la imagen del núcleo en memoria que se encuen-

Ksplice

Ksplice es una tecnología muy útil para servidores empresariales en producción. Podéis consultar su web: http://www.ksplice.com

tra ejecutándose. Se intenta entonces parar la ejecución en el momento en que no existan dependencias de tareas que necesiten las partes del núcleo a parchear. Entonces se procede a cambiar, en el código objeto del núcleo, las funciones afectadas, apuntando a las nuevas funciones con el parche aplicado y modificando datos y estructuras de memoria que tengan que reflejar los cambios. Actualmente es un producto comercial (de Oracle), pero algunas distribuciones de comunidad ya lo están incluyendo debido al soporte gratuito que se ofrece para algunas (entre ellas Ubuntu Desktop y Fedora). En los casos de producción en empresas, con servidores en los que es importante no disminuir el tiempo de servicio, puede ser una tecnología crítica para usar, altamente recomendable tanto para disminuir el tiempo de pérdida de servicio como para minimizar incidentes de seguridad que afecten al núcleo.

Básicamente ofrecen un servicio denominado *Ksplice Uptrack* que es una especie de actualizador de parches para el núcleo en ejecución. La gente de Ksplice sigue el desarrollo de los parches fuente del núcleo, los prepara en forma de paquetes que puedan incorporarse a un núcleo en ejecución y los hace disponibles en este servicio *uptrack*. Una herramienta gráfica gestiona estos paquetes y los hace disponibles para la actualización durante la ejecución.

SUSE también anuncio su idea de integrar una tecnología semejante, denominada *kGraft*, como mecanismo para aplicar parches al kernel sin reiniciar el sistema, de forma semejante a *Ksplice*. Pero en este caso la idea de los desarrolladores es intentar incluirla en la línea principal del kernel, con lo que estaría disponible como parte del código del kernel. En esta tecnología, de la que ya existen algunas versiones que se estan integrando en las versiones de desarrollo del kernel, básicamente el parche se integra como un módulo del kernel (.ko) el cual es cargado con *insmod*, y reemplaza a las funciones kernel afectadas por el parche, incluso durante el tiempo de ejecución de estas funciones. La idea de kGraft es permitir arreglar *bugs* críticos de kernel sin afectar a los sistemas que necesitan gran estabilidad y disponibilidad.

kGraft

Nueva tecnología de SUSE para parche en ejecución del kernel: https://www.suse.com/promo/kgraft.html

5. Módulos del núcleo

Flexibilidad del sistema

Los módulos aportan una flexibilidad importante al sistema, permitiendo que se adapte a situaciones dinámicas.

El núcleo es capaz de cargar dinámicamente porciones de código (módulos) bajo demanda [Hen], para complementar su funcionalidad (se dispone de esta posibilidad desde la versión 1.2 del núcleo). Por ejemplo, los módulos pueden añadir soporte para un sistema de ficheros o para dispositivos de hardware específicos. Cuando la funcionalidad proporcionada por el módulo no es necesaria, el módulo puede ser descargado y así liberar memoria.

Normalmente bajo demanda, el núcleo identifica una característica no presente en el núcleo en ese momento, contacta con un hilo (*thread*) del núcleo denominado `kmod` (en las versiones del núcleo 2.0.x el *daemon* era llamado `kerneld`) y este ejecuta un comando `modprobe` para intentar cargar el módulo asociado a partir de una cadena con el nombre de módulo o bien de un identificador genérico. Esta información en forma de alias entre el nombre y el identificador se consulta en el fichero `/etc/modules.conf`, aunque en las recientes distribuciones se ha migrado a una estructura de subdirectorios donde se incluye un fichero de opciones por módulo, que puede verse en el subdirectorio `/etc/modprobe.d`

A continuación se busca en `/lib/modules/version-kernel/modules.dep` para saber si hay dependencias con otros módulos. Finalmente, con el comando `insmod` se carga el módulo desde `/lib/modules/version_kernel/` (el directorio estándar para los módulos), la `version-kernel` es la versión del núcleo actual y se utiliza el comando `uname -r` para determinarla. Por tanto, los módulos en forma binaria están relacionados con una versión concreta del núcleo, y suelen colocarse en `/lib/modules/version-kernel`. Los módulos se reconocen como archivos dentro de la estructura de este directorio, con `.ko` como extensión de archivo.

En general, el administrador debe conocer cómo se cargan los módulos en el sistema. La mayor parte de veces por el proceso anterior, los módulos de la mayoría del hardware y necesidades concretas son detectados automáticamente en arranque o por demanda de uso y cargados en el momento correspondiente. En muchos casos no deberemos realizar ningún proceso como administradores. Pero en algunos casos, habrá que preparar alguna sintonización del proceso o de los parámetros de los módulos, o en algunos casos añadir nuevos módulos ya en forma binaria o por compilación a partir de los fuentes.

Si hay que realizar alguna compilación de módulos a partir de sus fuentes, se tiene que disponer de los paquetes fuente y/o *headers* de la versión del núcleo al cual está destinado.

Existen unas cuantas utilidades que nos permiten trabajar con módulos (solían aparecer en un paquete de software llamado `modutils`, que se reemplazó por `module-init-tools`):

- `lsmod`: Podemos ver los módulos cargados en el núcleo (la información se obtiene del pseudofichero `/proc/modules`). Se listan los nombres, las dependencias con otros (entre corchetes, [ ]), el tamaño del módulo en bytes y el contador de uso del módulo; esto permite descargarlo si la cuenta es cero.

Ejemplo

Algunos módulos en un Debian:

```
Module                  Size  Used by    Tainted: P
agpgart                37344   3   (autoclean)
apm                    10024   1   (autoclean)
parport_pc             23304   1   (autoclean)
lp                      6816   0   (autoclean)
parport                25992   1   (autoclean) [parport_pc lp]
snd                    30884   0
af_packet              13448   1   (autoclean)
nvidia               1539872  10
es1371                 27116   1
soundcore               3972   4   [snd es1371]
ac97_codec             10964   0   [es1371]
gameport                1676   0   [es1371]
3c59x                  26960   1
```

- `modprobe`: Intenta la carga a mano de un módulo y de sus dependencias.
- `insmod`: Carga un módulo determinado.
- `depmod`: Analiza dependencias entre módulos y crea un fichero de dependencias.
- `rmmod`: Saca un módulo del núcleo.
- `depmod`: Usado para generar el fichero de dependencias de los módulos, que se encuentra en `/lib/modules/version-kernel/modules.dep` y que incluye las dependencias de todos los módulos del sistema. Si se instalan nuevos módulos de núcleo, es interesante ejecutar manualmente este comando para actualizar las dependencias. También se suelen generar automáticamente al arrancar el sistema.
- Se pueden usar otros comandos para depuración o análisis de los módulos, como `modinfo`, que lista informaciones asociadas al módulo (como licencias, descripción, uso y dependencias), o ficheros `/proc/kallsyms`, que nos permiten examinar los símbolos exportados por los módulos.

Ya sea por el mismo núcleo o por el usuario manualmente con `insmod`, normalmente para la carga se especificará el nombre del módulo y, opcionalmente, determinados parámetros; por ejemplo, en el caso de dispositivos suele ser habitual especificar las direcciones de los puertos de E/S o bien los recursos de IRQ o DMA. Por ejemplo:

```
insmod soundx io=0x320 irq=5
```

La carga general de módulos, en función del momento y la forma, puede hacerse manualmente, como hemos comentado, mediante `initrd/initramfs` (que supone una precarga en tiempo de arranque del sistema) o por medio de `udev` (en situaciones dinámicas de uso de dispositivos removibles, o de conexión en caliente).

En el caso de `initrd/initramfs`, cuando el sistema arranca, se necesitan inmediatamente algunos módulos, para acceder al dispositivo y al sistema de ficheros raíz del sistema, por ejemplo controladores específicos de disco o tipos de sistemas de ficheros. Estos módulos, necesarios se cargan mediante un sistema de ficheros especial en RAM denominado `initrd/initramfs`. Dependiendo de la distribución GNU/Linux se utilizan estos términos de forma indiferente, aunque en algunos casos se han producido cambios a lo largo de la vida de la distribución. Por convención, suele denominarse a este elemento como *filesystem* RAM inicial, y se le refiere comúnmente como `initramfs`.

El sistema inicial de ficheros en RAM, `initramfs`, es cargado por el *bootloader* en la especificación de la entrada correspondiente a la carga del núcleo correspondiente (por ejemplo en la línea/opción `initrd` de la entrada correspondiente de Grub).

En la mayoría de distribuciones, ya sea con el núcleo distribuido originalmente o bien con nuestra configuración del núcleo, suele crearse un `initramfs` inicial. En todo caso, si dependiendo de la distribución no se produce (puede no ser necesario), entonces podemos crearlo y sintonizarlo manualmente. El comando `mkinitramfs` (o un posterior *update-initramfs*) permite crearlo a partir de sus opciones genéricas, que se pueden configurar en el archivo `/etc/initramfs-tools/initramfs.conf` y, específicamente, los módulos que se cargarán en inicio automáticamente, que podemos encontrarlos en */etc/initramfs-tools/modules*.

Un `mkinitramfs -o new_initrd_file` nos permitirá crearlo, y normalmente podemos proceder a copiarlo en el directorio `/boot` para hacerlo accesible al *bootloader* usado. Por ejemplo, mediante un cambio en Grub de su fichero de configuración `/boot/grub/menu.lst o grub.cfg`, modificando la línea de `initrd` oportuna. En cualquier caso, siempre es interesante establecer en el *bootloader* una configuración alternativa durante estas pruebas, para poder reiniciar y probar la nueva configuración, pero de la misma manera mantener la configuración estable antigua.

Durante el proceso de arranque, además del `initramfs` necesario para el arranque inicial, se producirá también la carga del resto de módulos por detección automática. Si no se carga algún módulo deseado siempre puede forzarse su carga al incluir su nombre implícitamente en el fichero de configuración `/etc/modules`.

También puede darse o desearse el caso contrario: evitar la carga de un módulo que puede detectarse erróneamente o para el que existe más de una alterna-

tiva posible. En este caso se utilizan técnicas de listas negras de módulos (típicamente la lista negra se guarda en `/etc/modprobe.d/blacklist.conf`), aunque puede crearse en un fichero arbitrario en ese directorio, simplemente añadiendo una línea *blacklist nombremodulo*.

5.1. DKMS: módulos recompilados dinámicamente

Respecto a los módulos dinámicos, un problema clásico ha sido la recompilación de estos frente a nuevas versiones del núcleo. Los módulos dinámicos de terceros, no incluidos *a priori* en el núcleo, necesitan de código fuente para compilarse, proporcionado por la comunidad o por el fabricante del hardware del dispositivo.

Durante la compilación, normalmente es necesario disponer de los paquetes de desarrollo del núcleo, de los paquetes del código fuente del núcleo y de sus *headers* para desarrollo, con la misma numeración que el núcleo usado actualmente, para el que se quiere compilar el módulo. Este hecho obliga a una recompilación constante con el cambio de versiones del núcleo del sistema, en especial ahora que las distribuciones distribuyen revisiones del núcleo con un menor tiempo, ya sea para solventar potenciales problemas de seguridad o para corregir errores detectados.

Algunas distribuciones, para minimizar esta problemática, distribuyen un entorno denominado DKMS, que permite facilitar la recompilación automática de un módulo con los cambios de versión del núcleo. Esto normalmente se produce en arranque al detectar un numero de núcleo: todos los módulos de terceros registrados por el sistema DKMS se recompilan a partir de los archivos fuente de los módulos y de los archivos fuente o *headers* del núcleo nuevo. De esta manera el proceso total es transparente al usuario. Una vez realizado este proceso, el sistema o el usuario, mediante comandos de manipulación de módulos (como `modprobe`), pueden utilizar directamente el nuevo módulo, o si estaba configurado en arranque, una vez producido éste el nuevo módulo se utilizará.

Algunas distribuciones ofrecen solo el paquete base (dkms) del sistema, en algunas se proporcionan paquetes dkms preparados para módulos concretos, o incluso el fabricante puede ofrecer su módulo con soporte dkms.

El proceso habitualmente pasa por los siguientes pasos:

1) Obtener los archivos fuente del módulo (un paquete o un archivo TGZ suele ser lo más normal).

2) Instalar los paquetes necesarios para el proceso: `dkms`, `kernel-source`, `kernel-headers` (los nombres dependen de la distribución, y en el caso de

los paquetes fuente del núcleo, hay que tener en cuenta que sean las versiones asociadas a la versión del núcleo actual para la que se quiere instalar el módulo; normalmente es suficiente con los headers, pero dependiendo del módulo pueden ser necesarios más paquetes de fuentes).

3) Activar el servicio DKMS en arranque. Habitualmente el servicio es denominado `dkms_autoinstaller`

4) Crear un directorio en `/usr/src` para los paquetes fuente del módulo y colocarlos allí.

5) Crear un fichero `dkms.conf` en el directorio anterior, que le especifica como construir (compilar) e instalar el módulo.

6) Añadir el servicio a la base de datos DKMS, compilarlo e instalar, normalmente con unos comandos:

```
dkms add -m nombre-modulo -v numero-version-modulo
dkms build   -m nombre-modulo -v numero-version-modulo
dkms install -m nombre-modulo -v numero-version-modulo
```

Respecto al fichero `dkms.conf` mencionado, podría ser como sigue (donde `nombre-modulo` es el nombre del módulo y `version`, el código numérico de su versión):

```
#
# /usr/src/nombre-modulo/dkms.conf
#

PACKAGE_NAME="nombre-modulo"
PACKAGE_VERSION="version"
CLEAN="make clean"
MAKE[0]="make module"
BUILD_MODULE_NAME[0]="nombre-modulo"
DEST_MODULE_LOCATION[0]="/kernel/drivers/video"
AUTOINSTALL="yes"

# End Of File
```

En este caso tenemos ubicados los paquetes fuente del módulo en un directorio `/usr/src/nombre-modulo` donde ponemos este fichero `dkms.conf`. La opción `MAKE` da los pasos para compilar y construir el módulo, previamente limpiados con `CLEAN`. `BUILD_MODULE_NAME` establece el nombre del módulo construido (cuidado en este punto porque depende del sistema de compilación y puede coincidir con el nombre del módulo general o no, algunos paquetes fuente permiten construir varios controladores/módulos diferentes, con diferente denominación).

`DEST_MODULE_LOCATION` define dónde se instalará el módulo final en el árbol asociado al núcleo; en este caso suponemos que es un controlador de vídeo (recordad que la raíz está en `/lib/modules/version-kernel`, lo que se coloca aquí es a partir de esta raíz). `AUTOINSTALL` permite que se reconstruya automáticamente el módulo durante cambios del núcleo actual.

En los casos de

`CLEAN`, `MAKE`, `BUILD_MODULE_NAME` y `DEST_MODULE_LOCATION`

se recomienda consultar el fichero de explicación (normalmente un `README` o `INSTALL`) que acompaña a los paquetes fuentes de los módulos, ya que pueden ser necesarios comandos adicionales, o tener que modificarlos para que se compile y se instale correctamente el módulo.

Por último mediante:

```
# dkms status
```

podemos conocer que módulos están activos en este momento y en qué kernels de los disponibles en el sistema.

6. Virtualización en el núcleo

Una de las áreas en expansión, en la administración de IT, es la virtualización de sistemas y su integración con entornos Cloud de tipo público/privado, como veremos más adelante. Con el tiempo, GNU/Linux ha ido incorporando diferentes posibilidades, provenientes tanto de soluciones comerciales como de diferentes proyectos de código abierto.

La virtualización de sistemas es un recurso básico actual en las empresas y organizaciones para mejorar su administración de sistemas, disminuir costes y aprovechar los recursos de hardware de manera más eficiente.

En general, en el pasado si necesitábamos varias instancias de uno (o más) sistemas operativos, teníamos que adquirir un servidor para cada instancia a implantar. La corriente actual es comprar servidores mucho más potentes y utilizar virtualización en estos servidores para implantar los diferentes sistemas en desarrollo o producción.

Normalmente, en virtualización disponemos de un sistema operativo instalado (que habitualmente se denomina sistema *host*) y una serie de máquinas virtuales sobre este sistema (denominados sistemas *guest*). Aunque también hay soluciones que sustituyen al sistema operativo *host* por una capa denominada *hypervisor*.

La virtualización como solución nos permite optimizar el uso de nuestros servidores o bien, por ejemplo en el caso de escritorio, disponer de máquinas de test de otros sistemas operativos conviviendo en la misma máquina simultáneamente. En el caso de GNU/Linux disponemos de múltiples soluciones que permiten tanto un caso como el otro. También podemos disponer tanto de GNU/Linux como sistema *host* que aloja máquinas virtuales, como utilizarlo como máquina virtual sobre otro sistema diferente o bien sobre otro sistema *host* también GNU/Linux. Un esquema, este último, particularmente útil en el caso de administración porque nos permitirá, por ejemplo, examinar y ejecutar diferentes distribuciones GNU/Linux sobre un mismo sistema *host* base, para observar diferencias entre ellas o personalizar nuestras tareas o scripts para cada distribución.

Existen muchas soluciones de virtualización, pero por mencionar algunas de las más populares en sistemas GNU/Linux, disponemos (ordenamos de más a menos en relación directa con el núcleo):

- KVM
- Xen

- LXC
- OpenVZ
- VirtualBox
- VMware

VMware es uno de los líderes comerciales en soluciones de virtualización, y dispone de productos de virtualización para escritorio (VMware Workstation/Fusion), mientras que para servidor dispone de un producto VMware vSphere que está disponible para Linux con descarga gratuita. El caso de servidor permite gestionar varias máquinas virtuales con un interfaz de gestión simple. Otras líneas de productos VMware implementan necesidades mayores para centros de datos (*data centers*) y entornos de Cloud Computing, con gestión elaborada de máquinas, tolerancia a fallos, migraciones y otras necesidades explícitas para centros de datos.

Enlace de interés

Podéis visitar la web de VMware en: http://www.vmware.com

Oracle VirtualBox ofrece virtualización orientada a escritorio, que nos permite una opción bastante sencilla para probar máquinas con diferentes sistemas operativos. Dispone de versión de código libre utilizable en gran número de distribuciones GNU/Linux.

OpenVZ es una solución de virtualización que utiliza el concepto de contenedor de máquina virtual. Así el *host* arranca con un núcleo común a las máquinas virtuales (existen paquetes de imágenes de núcleo con soporte OpenVZ integrado en las distribuciones, como por ejemplo en Debian), que permite arrancar máquinas con el núcleo en común pero cada una dentro de un entorno aislado del resto. OpenVZ solo permite máquinas virtuales *guest* Linux (debido al núcleo compartido).

LXC (LinuX Containers) es un sistema de virtualización a nivel de operativo que permite correr múltiples sistemas Linux de forma aislada. En lugar de crear máquinas virtuales completas, se basa en el uso de una funcionalidad del kernel de Linux, denominada cgroups, juntamente con *chroot*, que permite encapsular un espacio de ejecución de procesos y gestión de red de forma aislada. La idea es similar a OpenVZ, pero en este caso no necesita actuaciones de módulos o parches de kernel, ya que funciona bajo el kernel *vanilla*, sin modificaciones. Algunos sistemas para desplegamiento de contenedores, como *Docker*, lo utilizan como base para implementar los contenedores.

Xen usa el concepto de *hypervisor*, utilizando un tipo de virtualización denominada *paravirtualización*, en la que se elimina el concepto de *host-guest* y se delega a la capa de *hypervisor* la gestión de los recursos físicos de la máquina, de manera que permita el máximo acceso a los recursos de hardware por parte de las máquinas virtuales. En estos casos se necesitan núcleos sintonizados que puedan beneficiarse de las posibilidades de la paravirtualización. En el caso de GNU/Linux en la mayoría de distribuciones se ofrecen núcleos optimizados para Xen (véase el caso de Debian, para el que existen imágenes binarias para xen de los núcleos). En general, la paravirtualización y la capa

de *hypervisor* para acceder al hardware aprovechan las facilidades de las CPU actuales, con recursos de hardware dedicados a facilitar la virtualización. Si se dispone de las características se pueden usar sistemas como Xen, sino, puede utilizarse un sistema más clásico de *host-guest* como por ejemplo VirtualBox.

En general, para ver si la CPU dispone de soporte de virtualización, hay que examinar sus datos en `/proc/cpuinfo`, en concreto el *flag* `vmx`, para procesadores Intel, o svm en procesadores AMD, que puede verse en la sección `flags`:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 23
model name      : Intel(R) Xeon(R) CPU E5405  @ 2.00GHz
stepping        : 10
cpu MHz         : 1994.999
cache size      : 6144 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
apicid          : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2
ss ht tm syscall   nx lm constant_tsc pni monitor ds_cpl
vmx tm2 ssse3 cx16 xtpr sse4_1 lahf_lm
bogomips        : 3989.99
clflush size    : 64
cache_alignment : 64
address sizes   : 38 bits physical, 48 bits virtual
power management:
```

6.1. KVM

Finalmente, la solución en que nos centraremos en este subapartado, **KVM**, está presente desde el núcleo 2.6.20, como solución incluida para la virtualización de sistemas. Es una solución parecida a Xen, pero con diferencias en cuanto a la implementación. Xen es un producto complejo, con diferentes capas y, en especial, su diseño de capa hipervisora. Por contra, KVM se implementa como un módulo del núcleo existente, que se complementa con otras

Enlace de interés

Enlace web de KVM en:
http://www.linux-kvm.org

soluciones. Es la opción por defecto para virtualización en distribuciones como Debian y Fedora.

Normalmente una solución de virtualización basada en KVM se compone de una mezcla de:

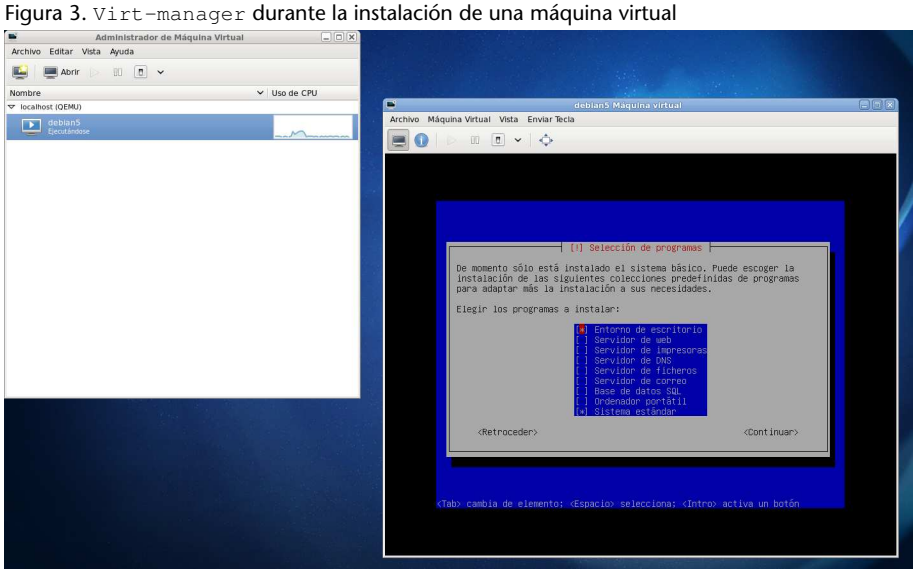
- El módulo de núcleo `kvm.ko`, que proporciona la infraestructura base de virtualización, y además un módulo adicional según el modelo de procesador `kvm-intel.ko` o `kvm-amd.ko`. Estos módulos deberán cargarse manualmente (`modprobe`) o habilitarlos durante el arranque para disponer de soporte de virtualización KVM.

- Qemu, un simulador cruzado de arquitecturas de CPU, que nos permite simular una CPU virtual sobre otra de igual o diferente arquitectura real. KVM utiliza una versión modificada de Qemu para la gestión y creación de las máquinas *guest* (este paquete suele aparecer como `qemu-kvm` en las distribuciones).

- La biblioteca `libvirt`, que es una API que permite una gestión de la virtualización independiente del sistema de virtualización usado (sea Xen, KVM u otros).

- Utilidades basadas en `libvirt` para la creación de las VM *guest* y su mantenimiento, como `virt-manager` una interfaz gráfica de gestión de máquinas virtuales (figura 3), `virt-install`, una utilidad de línea para la gestión de máquinas virtuales, o `virtsh`, un *shell* basado en comandos de gestión de las máquinas virtuales. Normalmente estas utilidades suelen necesitar un servicio de sistema, denominado `libvirtd` (en algunas distribuciones `libvirt-bin`). Tenemos que asegurarnos de poner en marcha el servicio (`/etc/init.d/libvirtd start`, `/etc/init.d/libvirt-bin start` o equivalentes con Systemd) o bien de cargarlo al inicio.

Figura 3. `Virt-manager` durante la instalación de una máquina virtual



A continuación veremos los procesos básicos asociados a la utilización de KVM y describiremos algunas de las fases de la instalación de KVM y la puesta en marcha de algunas máquinas virtuales.

Soporte de virtualización

Hay que tener cuidado con el hecho de que muchas de las máquinas actuales permiten desactivar/activar en BIOS el soporte de virtualización; comprobemos primero que no esté desactivado.

Para comenzar debemos examinar si disponemos de soporte de harwdare en la CPU: como hemos comentado buscamos el *flag* `vmx` en `/proc/cpuinfo` (para procesadores Intel) o el *flag* `svm` para procesadores AMD. Por ejemplo, con (sustituir `vmx` por `svm` en AMD):

```
grep vmx /proc/cpuinfo
```

obtendremos como resultado la línea de *flags* (si existe el *flag* `vmx`, si no, ningún resultado). También podemos obtener varias líneas en CPU *multicore* y/o Intel con *hyperthreading*, donde obtenemos una línea de *flags* por cada elemento de cómputo (CPU con HT o múltiples núcleos con/sin HT).

Una vez determinado el correcto soporte de virtualización, procedemos a instalar los paquetes asociados a KVM; estos ya dependerán de la distribución, pero en general, el mismo "kvm" ya obtendrá la mayoría de paquetes requeridos como dependencias. En general, los paquetes recomendados a instalar (vía apt o yum) son `kvm`, `qemu-kvm`, `libvirt-bin`, `virt-viewer` entre otros. Dependiendo de la distribución pueden cambiar algunos nombres de paquetes, y en especial con el nombre `virt` existen varios paquetes de utilidades de generación y monitorización de máquinas virtuales tanto de KVM como de Xen, ya que la librería libvirt nos permite abstraer varios sistemas diferentes de virtualización.

Si se permite a los usuarios del sistema el uso de máquinas virtuales, hay que añadir los nombres de usuario a grupos específicos (vía comandos `adduser` o `usermod`), normalmente a los grupos `kvm` o `libvirt` (dependiendo de la distribución, Fedora o Debian, y versión de KVM):

```
# adduser <usuario> libvirt
```

En este momento podemos ejecutar el comando de shell para la gestión de máquinas KVM, `virsh`), que nos permitiría ver las disponibles:

```
$ virsh --connect qemu:///system list
```

esto permite conectarse al gestor local, y preguntarle por la lista de máquinas disponibles.

El siguiente paso (opcional) es facilitar el uso de red a las máquinas virtuales. Por defecto KVM utiliza NAT, dando direcciones IP privadas de tipo 10.0.2.x, y

accediendo mediante la red de la máquina *host*. En otro caso, si queremos una configuración diferente (por ejemplo que permita acceso externo a las máquinas virtuales) tendremos que permitir hacer de *bridge* a la red actual; en este caso es necesario instalar el paquete `bridge-utils` y configurar un dispositivo especial de red denominado `br0`, en Debian en la configuración de red presente en `/etc/network/interface` y en Fedora puede crearse un fichero asociado al dispositivo como `/etc/sysconfig/network-scripts/ifcfg-br0`. Por ejemplo, en Debian podría colocarse un ejemplo de configuración como:

```
auto lo
iface lo inet loopback

auto br0
iface br0 inet static
       address 192.168.0.100
       network 192.168.0.0
       netmask 255.255.255.0
       broadcast 192.168.0.255
       gateway 192.168.0.1
       bridge_ports eth0
       bridge_fd 9
       bridge_hello 2
       bridge_maxage 12
       bridge_stp off
```

Enlace de interés

Por otra parte, un tema importante es la gestión de red, que es uno de los temas complejos en virtualización; en el caso de KVM se recomienda examinar: http://www.linux-kvm.org/page/Networking.

Esta configuración permite que se cree un dispositivo `br0` para reemplazar a `eth0`. Así las tarjetas de red virtuales redirigirán el tráfico asignadas a este dispositivo. `bridge_ports` especifica cuál será el dispositivo físico real que se utilizará.

Como comentamos, esta parte de configuración de red es opcional, y solo tiene sentido si queremos acceder desde el exterior a nuestras máquinas virtuales. Por el contrario, en un entorno de virtualización de escritorio puede ser suficiente con el modo NAT por defecto, ya que las máquinas dispondrán de salida de red a través de la red del *host*.

A partir de estas configuraciones, ya estaremos capacitados para crear las imágenes de las máquinas virtuales. Hay diferentes conjuntos de comandos para realizarlo, bien usando kvm directamente (comando `kvm`), utilidades asociadas a qemu para creación de estas imágenes (`qemu-img`) o utilidades asociadas a libvirt (`virt-install`). El proceso pasa por crear la imagen de disco (o espacio de disco) asociado a la máquina *guest* como un fichero y después realizar la instalación del sistema operativo en ese fichero (imagen de disco), bien desde el CD/DVD de instalación del sistema operativo o bien también desde una imagen `*.iso` del CD/DVD.

Por ejemplo, supongamos una instalación de un *guest* determinado (por ejemplo, disponemos de unos CD/DVD o de archivos de imágenes ISO de la distribución Debian).

Creamos el espacio de disco para la máquina virtual (en este caso 8 GB):

```
# dd if=/dev/zero of=~/debianVM.img bs=1M count=8192
```

Mediante el comando `dd` creamos así un fichero de salida de 8192 bloques de 1 MB, es decir, 8 GB, que nos formará la unidad de disco para nuestra máquina virtual (también existe un comando alternativo para crear imágenes, `qemu-img`, véase página man).

Lo siguiente es obtener el medio de instalación del sistema operativo a instalar, bien proporcionado como CD/DVD y, por tanto, accesible en el dispositivo `/dev/cdrom` (o equivalente si tenemos más unidades de CD/DVD) o, por contra, a partir de una imagen `iso` del soporte. En cualquier caso, este último siempre lo podemos obtener a partir de los discos con:

```
dd if=/dev/cdrom of=debian-install.iso
```

que nos genera así la imagen del CD/DVD o directamente descargar la imagen ISO de la distribución.

Ahora que disponemos de la imagen binaria y el medio de instalación del sistema operativo, podemos proceder a instalar, con diferentes utilidades. En general suele utilizarse `virt-install` como comando para crear la VM, pero también existe la posibilidad de usar el mencionado `qemu-kvm` directamente como opción más simple, que suele aparecer (dependiendo de la distribución) como comando denominado `qemu-kvm`, `qemu-system-x86_64` o simplemente como `kvm` (en algunas distribuciones, kvm ya no se utiliza o solo está por compatibilidad):

```
$ qemu-system-x86_64 -enable-kvm -m 512 -cdrom debian-install.iso \
-boot d -hda debianVM.img
```

En este caso crearía una máquina virtual (arquitectura x86_64) básica de 512 MB de memoria principal usando nuestro disco de 8 GB creado previamente y arrancando la máquina a partir de nuestra imagen `iso` del medio de instalación del sistema operativo. Se puede sustituir el fichero `iso` por `/dev/cdrom` o `/dev/sr0` si usamos los discos de instalación.

La otra posible alternativa de creación se realiza mediante la utilidad de línea de comandos `virt-install`, mucho más completa, pero con la dificultad de

tener que especificar muchos más parámetros. Por ejemplo, podríamos indicar la creación de la máquina anterior mediante:

```
virt-install --connect qemu:///system -n debian -r 512 \
--vcpus=2 -f debianVM.img -s 8 -c debianinstall.iso --vnc \
--noautoconsole --os-type linux --os-variant debianLenny
```

que entre otros parámetros, coloca el método de conexión a la máquina virtual, el nombre de la VM `debian`, define 512 MB de memoria, hace disponibles 2 núcleos de la máquina física, utiliza el disco virtual `debianVM`, de 8 GB (`-s` permite que si no existe, lo cree previamente con ese tamaño de GB), utiliza la `iso` como medio de instalación y permitirá conexiones gráficas con vnc con la máquina virtual. Además definimos que el sistema que va a instalarse es Linux, en especial una versión de Debian. Los parámetros de tipo y variante del sistema operativo son altamente dependientes de la versión de `virt-install`, de manera que vale la pena consultar su página man (`man virt-install`) y la lista de sistemas operativos compatibles con la versión KVM del núcleo y el conjunto de utilidades `qemu` y `libvirt`.

Enlace de interés

La lista de compatibilidad de KVM puede encontrarse en: http://www.linux-kvm.org/page/Guest_Support_Status

En este punto solo hemos creado la máquina, pero no estamos conectados a ella, y hemos realizado una configuración muy básica de sus parámetros. Con otras utilidades, por ejemplo las basadas en `libvirt`, como la interfaz gráfica `virt-manager`, podemos personalizar más la VM creada, por ejemplo añadiendo o quitando hardware virtual al sistema. Con `virt-manager` podemos añadir la conexión a la máquina de la que hemos creado la imagen, por ejemplo al *localhost*, lo que nos permitirá después tenerla accesible mediante conexión a *qemu:///system*, y también arrancarla al momento.

Después podemos visualizar la consola (de texto o gráfica) de la máquina recién creada mediante:

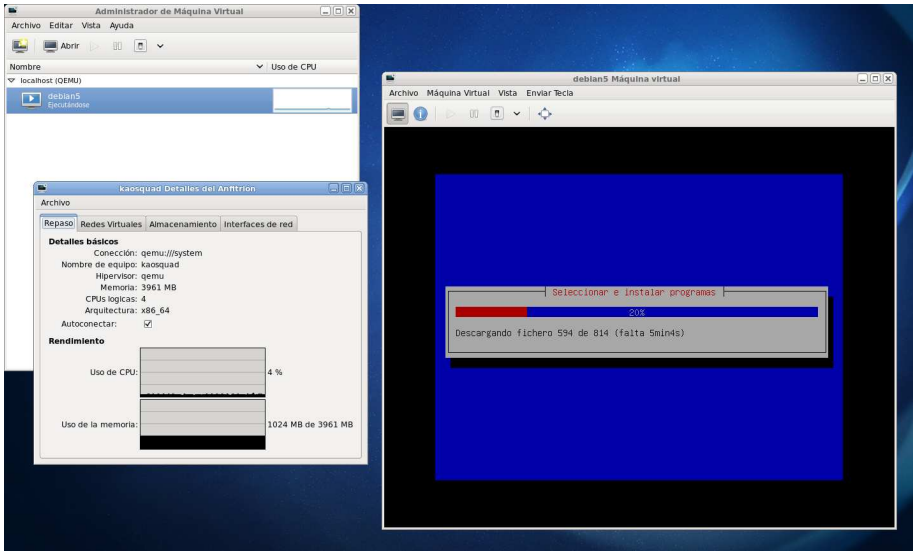
```
virt-viewer -c qemu:///system nombreVM
```

si está en el mismo sistema *localhost*, o si estamos en una máquina remota con:

```
virt-viewer -c qemu+ssh://ip/system nombreVM
```

o usando directamente la interfaz `virt-manager` (figura 4).

Figura 4. `Virt-manager` conectado a la máquina virtual durante el proceso de instalación, observando los recursos usados por la máquina virtual



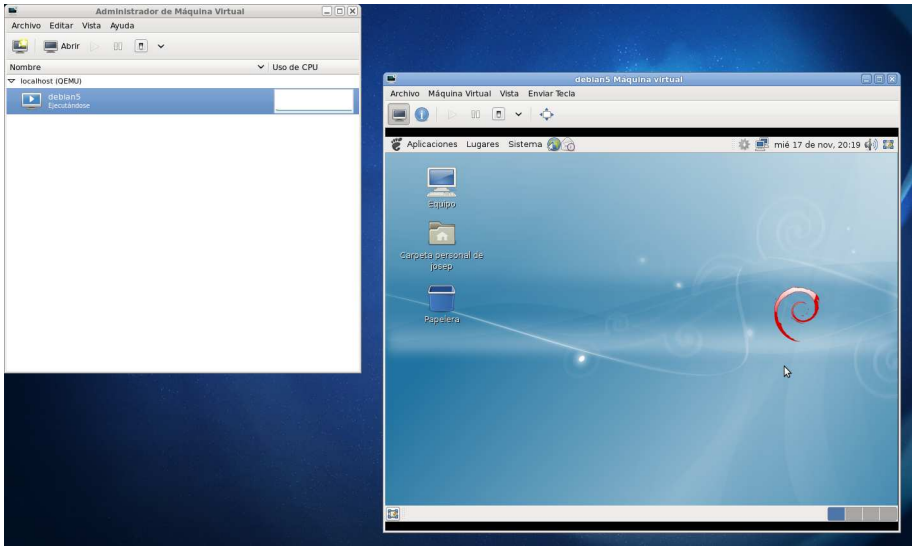
Esto nos permite conectar gráficamente con la máquina virtual creada y, por ejemplo, continuar con la instalación del sistema operativo (se habrá iniciado con el arranque anterior con `virt-install` o `qemu-system-x86_64`). Una vez instalado el sistema, ya podemos usar cualquier sistema, ya sea gráfico (`virt-manager`) o de línea de comandos (`virtsh`) para gestionar las máquinas *guest* virtuales y poder arrancar y pararlas.

Por ejemplo, con `virsh`:

```
virsh --connect qemu:///system
```

conectamos con el *shell* de comandos; comandos como `list` nos dan las máquinas activas, `list -all`, todas las máquinas disponibles, y otros como `start`, `shutdown`, `destroy`, `suspend` o `resume` nos dan diferentes posibilidades de gestión de cada máquina virtual.

Figura 5. `virt-manager` conectado a la máquina virtual recién instalada



6.2. VirtualBox

VirtualBox es una solución de virtualización de escritorio que está obteniendo resultados importantes, un proyecto en estos momentos mantenido por Oracle.

En el caso de Debian, que usaremos, está disponible en el repositorio oficial. Pero para otras distribuciones, pueden encontrarse en https://www.virtualbox.org/wiki/Linux_Downloads paquetes preparados para gran numero de distribuciones, así como repositorios propios que añadir a las distribuciones.

Esta solución de virtualización incluye tanto herramientas de tipo gráfico como utilidades a nivel de línea de comandos para virtualizar máquinas de 32bits y 64bits de arquitectura Intel (x86 y x86_64). VirtualBox se ofrece mayoritariamente con licencia GPL; aún así existe una parte adicional propietaria, denominada *Extension Pack*, que ofrece libre de coste Oracle para uso personal. Este paquete adicional ofrece algunas facilidades extra relacionadas con emulación de Hardware extra (USB2 por ejemplo, por defecto solo se ofrece USB1 sin pack), y ofrece acceso gráfico a las máquinas huésped a través de RDP (*Remote Desktop Protocol*).

Para la instalación en Debian se debe instalar el paquete *virtualbox* y es recomendado disponer de los headers correspondientes al kernel actual (paquete *linux-headers-version* siendo la versión la correspondiente al kernel actual). Los headers se van a utilizar para la compilación de una serie de módulos que virtualbox instala dinámicamente en el kernel. Normalmente, la instalación proporciona también una configuración de *dkms* para estos módulos que permite que se recompilen después de cambios de kernel en la distribución. En algunas distribuciones a veces pasa un tiempo hasta que están disponibles los módulos actualizados para una versión concreta de kernel; por tanto, antes de actualizar el kernel en un sistema con virtualización virtualbox, es recomendable comprobar que la versión de kernel ya tiene el soporte adecuado.

```
# apt-get install linux-headers-version virtualbox
```

VirtualBox puede arrancarse ya a través del comando `virtualbox`. Los módulos, si teníamos los headers adecuados, se habrán compilado y cargado en el sistema, y se cargarán en cada arranque. Si no deseamos que se arranquen automáticamente, podemos cambiar en `/etc/default/virtualbox` el parámetro LOAD_VBOXDRV_MODULE y ponerlo a 0.

Una vez arrancado el entorno gráfico con `virtualbox`, podemos crear una nueva VM mediante la opción del icono "Nueva", que nos arrancará el asistente de creación de máquinas virtuales. El proceso pasa por determinar:

- El operativo de la máquina huésped y su nombre.

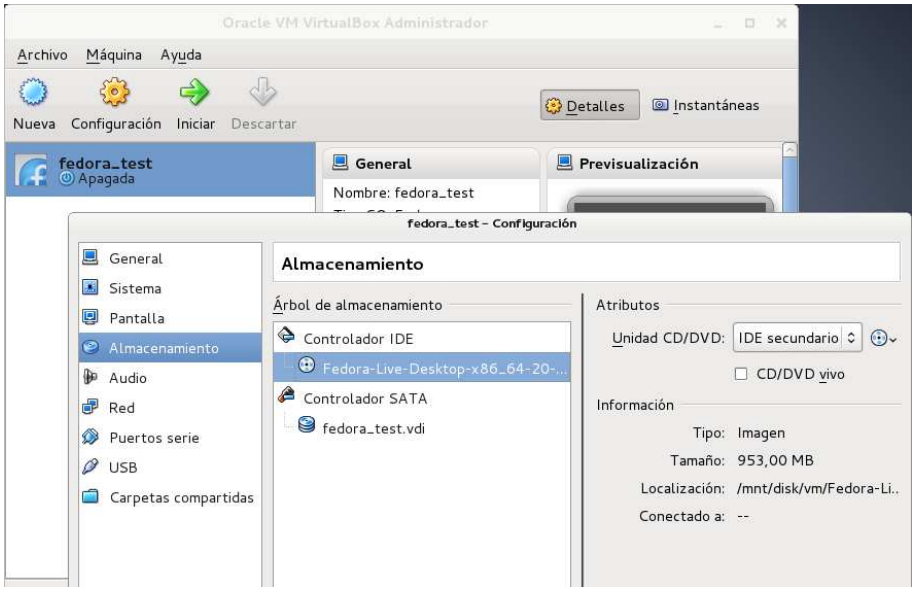
- La memoria de nuestro sistema, que proporcionaremos a la máquina huésped; generalmente se recomienda que el total de máquinas VM no consuman más del 50 % de la memoria. Dependiendo del uso, es frecuente entre 512 MB a 2 GB por máquina virtual.

- Creación del disco duro virtual de la máquina. Podemos crear un disco nuevo o usar una imagen de disco previamente creada para la nueva máquina.

- Formato de la imagen del disco. En virtualbox, por defecto, es VDI. Pero podemos crear otros, que quizá podamos compartir con otros sistemas de virtualización.

- Asignamos el tamaño del disco virtual de forma dinámica o reservamos el total del espacio. Esto nos permitirá reservar espacio de disco a medida que se vaya usando o, por contra, tener un disco más rápido, lo que nos permitirá mejores prestaciones del disco virtual.

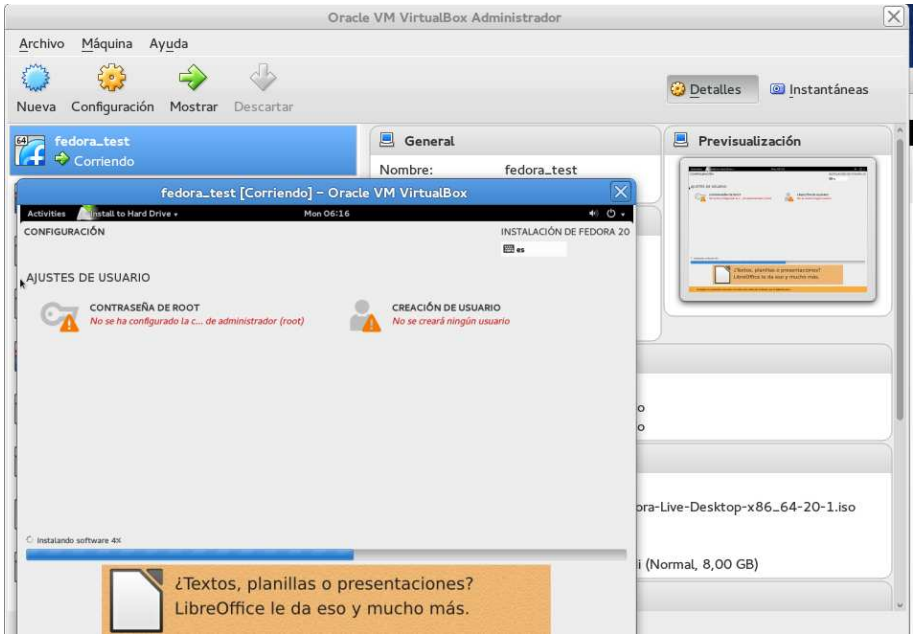
Figura 6. Máquina virtual creada en el gestor de VirtualBox añadiendo ISO para instalar.



Con este proceso ya disponemos de una máquina creada y podemos retocar su configuración seleccionándola; en especial, podemos conectar en almacenamiento una imagen ISO de una distribución a su CDROM virtual. Esto nos permitirá arrancar la máquina desde CD, arrancando así la imagen de instalación. En el proceso inicial también podemos cambiar la configuración de red; hay diversos modos disponibles: desde el NAT, por defecto, a modos que permiten redes privadas para las máquinas VM, o incluir una IP pública para nuestra máquina VM (ver documentación VirtualBox para la administración de red http://www.virtualbox.org/manual/UserManual.html). Véase en la figura 6 una máquina creada, con su sección de configuración de almacena-

miento a la que se le acaba de conectar una imagen ISO (de una distribución Fedora), si procedemos al arranque de la máquina con *Iniciar*. En la figura 7 siguiente podemos ver la máquina huésped durante el proceso de instalación.

Figura 7. Máquina virtual en el proceso de instalación de una distribución GNU/Linux.



Una vez tengamos la VM con sistema operativo instalado, ya la tendremos funcional. Para mejorar el rendimiento de las máquinas huésped, VirtualBox también proporciona una serie de controladores para el sistema huésped, conocidos como *VirtualBox Guest Additions*, que permiten mejoras de la gestión de la red virtual, de la aceleración de los gráficos y la posibilidad de compartir carpetas con la máquina host (anfitrión). Una vez arrancada e instalada la VM, desde el menú Dispositivos ->Insertar imagen CD Guest Additions, pueden instalarse.

Cabe señalar que además de todo este proceso, determinados proveedores proporcionan imágenes VDI de máquinas VirtualBox ya preparadas para su funcionamiento, para que no tengamos que pasar por todo el proceso de creación. Normalmente son imágenes denominadas Cloud, y deberemos buscar imágenes VDI ya preparadas u otros formatos compatibles con VirtualBox. Sin embargo, se puede incurrir en ciertos riesgos de seguridad. Conviene evitar el uso de VM ya customizadas si el proveedor no es de confianza o no es directamente la misma distribuidora; además, algunas máquinas VM arbitrarias podrían contener problemas de seguridad o introducir malware o herramientas destinadas al ataque de nuestros sistemas. Por eso se recomienda usar solo imágenes de proveedores adecuados.

Por ultimo, cabe señalar el tema del control mediante interfaz de usuario. VirtualBox proporciona los comandos `vboxmanage` y `VBoxHeadless` para permitirnos la mayor parte de la gestión de las máquinas desde línea de coman-

dos, e incluso si ya disponemos de imágenes VDI, podremos realizar toda la gestión sin ningún uso de interfaz gráfica.

Un resumen de algunos comandos interesantes:

- Arrancar la máquina en modo servidor sin interfaz VirtualBox; al arrancar nos dará el puerto de conexión al servidor gráfico (VRDE). También con vboxmanage, con interfaz GUI y sin interfaz:

```
$ VBoxHeadless -startvm "nombre_maquina"
$ vboxmanage startvm "nombre_maquina"
$ vboxmanage startvm "nombre_maquina" --type headless
```

- Información de la máquina virtual, los detalles de su configuración, discos, red, memoria, cpu y otros:

```
$ vboxmanage showvminfo "nombre_maquina"
```

- Apagar la máquina:

```
$ vboxmanage controlvm "nombre_maquina" poweroff
```

- Modificar el servicio de pantalla:

```
$ vboxmanage modifyvm "nombre_maquina" --vrde on --vrdeport
puerto_VRDE --vrdeaddress ip_maquina
```

- Conexión gráfica a la máquina ejecutándose. Si la máquina está en NAT sin ip asignada, la ip=0.0.0.0 para la conexión, si no la ip correspondiente. Para pleno funcionamiento se necesitan instaladas las Guest Additions, las cuales tienen como requisito el paquete de headers del kernel correspondiente en la máquina huésped:

```
$ rdesktop -a 16 -N ip_maquina:puerto_VRDE
```

- En particular, con el siguiente veremos múltiples opciones de control de las VMs que tenemos disponibles:

```
$ vboxmanage controlvm
```

6.3. Xen

En el caso de Xen, tratamos con un entorno de hypervisor a nivel de máquina física que permite correr múltiples instancias del sistema operativo o de diferentes operativos en paralelo en una sola máquina.

Algunas de las características de Xen que cabe destacar son las siguientes:

Enlace de interés

Se recomienda leer esta introducción: http://wiki.xen.org/wiki/Xen_Overview, para comprender algunos conceptos de virtualización asociados a Xen.

- El hypervisor, o componente monitor de máquinas virtuales, es relativamente pequeño y consume pocos recursos de la máquina.

- Xen es agnóstico del sistema operativo. La mayoría de instalaciones usan Linux como sistema base; Xen aprovecha los servicios básicos, el denominado dominio 0 de Xen. Dom0 es una VM especializada con privilegios para el acceso directo al hardware. Como base para Xen, pueden usarse otros operativos, como algunos de la familia BSD.

- Xen es capaz de aislar el funcionamiento de un driver de sistema en una máquina virtual. Si este falla o es comprometido, la VM que contiene el driver puede rearrancarse sin afectar al resto del sistema.

- Pueden distinguirse dos modos de funcionamiento, por un lado a) el denominado Xen Full Virtualization (HVM), que usando las extensiones hardware de virtualización (las HVM), utiliza emulación de PC (Xen se basa en Qemu para ello). En este caso no se requiere soporte de kernel, pero por contra suele ser una solución de menor rendimiento debido a la emulación necesaria. Por otra parte, el modo b) Xen Paravirtualization (PV), es una virtualización eficiente y ligera en uso de recursos, que no requiere extensiones HVM, pero si disponer de kernels y drivers con soporte de PV. De manera que el hypervisor puede ejecutarse eficientemente sin emulación de máquina, o emulación de componentes virtuales de hardware.

- El soporte de Paravirtualización. Los huéspedes de este tipo son optimizados para poder ejecutarse como máquina virtual (algunos estudios han proporcionado solo sobrecargas en torno al 2-8 %, mientras que en emulación las penalizaciones se suelen acercar a un 20 %), dando mejor rendimiento que otros sistemas que se basan en componentes adicionales. Además, Xen puede incluso ejecutarse en hardware que no soporta extensiones de virtualización.

  Linux dispone del soporte de paravirtualización desde la versión 2.6.23, llamado *paravirt_ops* (o simplemente *pvops*). Se trata de un kernel Linux de Dominio 0 (disponible para arquitecturas x86, x86_64 y ia64). Este dom0 es el sistema que tienen los controladores de dispositivo para comunicarse con el hardware subyacente, ejecuta las herramientas de administración de Xen y proporciona los discos virtuales y el subsistema de red virtual al resto de sistemas huésped (denominados domU's).

  Enlace de interés

  Una referencia para explicar el concepto de PVHVM: http://www.slideshare.net/fullscreen/xen_com_mgr/linux-pv-on-hvm/

  También hay que mencionar que en las últimas generaciones de procesadores ha mejorado sensiblemente el soporte y las posibilidades de las extensiones HVM, lo que ha hecho viable nuevas aproximaciones híbridas para pasar de virtualización PV a PVHVM; básicamente consiste en huésped HVM pero con drivers especiales para los discos y red.

- Como necesitamos de base el mencionado Dominio 0, Xen necesitará portar los sistemas operativos para adaptarse a la API de Xen, a diferencia de los VM tradicionales, que proporcionan entornos basados en software para simular el hardware. En estos momentos hay ports (de los kernels pvops) para NetBSD, FreeBSD y Linux entre otros.

- Gracias al código aportado por Intel y AMD a Xen, se han adaptado diferentes extensiones de virtualización hardware que permiten que los sistemas operativos sin modificaciones se ejecuten en máquinas virtuales Xen, lo que ha aportado mejoras de rendimiento y ofrecido la posibilidad de ejecutar sistemas GNU/Linux y Windows sin modificación alguna.

- Hay soporte para la migración de máquinas VM en caliente entre equipos físicos. En este sentido, una máquina virtual en ejecución puede ser copiada y trasladada de equipo físico sin detener la ejecución, tan solo para pequeñas sincronizaciones.

A continuación veremos algunos usos simples de Xen sobre una distribución Debian, que dispone de la particularidad de que es fácil construir VM huéspedes basadas en Debian misma de forma bastante flexible a partir de los repositorios de Debian.

Comenzaremos por la instalación del dominio 0 (`dom0`). Básicamente se realiza con una instalación Debian típica, donde únicamente se tendrán en cuenta las particiones a emplear para reservar espacio posterior a los huéspedes. En las versiones actuales de Xen se suele reservar 4 GB de disco para el `dom0` (su /), y 1 GB de memoria para la RAM que necesitará. El espacio extra puede dedicarse a almacenar las VM huéspedes, o en el caso de *storage* podemos por ejemplo usar volúmenes LVM que nos permitirán crecer los *filesystems* a medida que los necesitemos por parte del `dom0` o las domU's (huéspedes).

Por ejemplo, podríamos realizar 3 particiones (sda1,2,3), y colocar `/ (root)` y `swap` en las dos primeras y un LVM en la tercera, con un volumen físico y un grupo (*vg0*, por ejemplo). Instalamos el sistema Debian.

Pasamos a instalar el hypervisor mediante un metapaquete disponible en Debian que nos hace todo el trabajo de dependencias y utilidades necesarias:

```
# apt-get install xen-linux-system
```

Podemos comprobar también si nuestra máquina soporta extensiones HVM, ya sean de Intel o AMD (en algunos casos pueden estar desactivadas en la BIOS de la máquina, y entonces deberemos habilitarlas previamente), con:

```
egrep '(vmx|svm)' /proc/cpuinfo
```

Si están disponibles, Xen podrá utilizar más eficientemente la paravirtualización, basándose en el soporte de virtualización de los procesadores modernos. En varios estudios (de *benchmarking*) se ha comprobado que PV+HVM (sobre PV pura) en diferentes *bencharmaks* obtiene beneficios que van del 15 % hasta el 200 o 300 %.

***Benchmarking* PVHVM**

Algunas referencias: https://developer.rackspace.com/blog/welcome-to-performance-cloud-servers-have-some-benchmarks/ y https://xen-orchestra.com/debian-pvhvm-vs-pv/

Una vez realizada la instalación previa del sistema Xen, observaremos que la distribución Debian, en su arranque `Grub2` (en este caso), tiene una nueva entrada denominada `Xen4`. Es la que nos permitirá arrancar la máquina, bajo el hypervisor Xen gestionando el `dom0`. La entrada Xen no está puesta por defecto, con lo cual se seguirá cargando por defecto el kernel del sistema. Con estas instrucciones la pondremos por defecto:

```
dpkg-divert --divert /etc/grub.d/08_linux_xen --rename /etc/grub.d/20_linux_xen
update-grub
```

El siguiente paso será configurar la red para el dominio 0. La más común es usar un *bridge* por software (necesitamos disponer del paquete bridge-utils); un ejemplo sencillo para `/etc/network/interfaces`:

```
#The loopback network interface
auto lo
iface lo inet loopback

iface eth0 inet manual

auto xenbr0
iface xenbr0 inet dhcp
   bridge_ports eth0

#other possibly useful options in a virtualized environment
  #bridge_stp off       # disable Spanning Tree Protocol
  #bridge_waitport 0    # no delay before a port becomes available
  #bridge_fd 0          # no forwarding delay

## configure a (separate) bridge for the DomUs without
## giving Dom0 an IP on it
#auto xenbr1
#iface xenbr1 inet manual
#   bridge_ports eth1
```

Otras técnicas en la configuración son opcionales, como por ejemplo: ajustar el uso de memoria para `dom0`, añadir a `/etc/default/grub` la siguiente línea para reservar memoria para el `dom0` (1 GB en este caso), y posteriormente un *update-grub*:

```
GRUB_CMDLINE_XEN="dom0_mem=1024M"
```

En Xen se utiliza una técnica denominada *ballooned* que lo que hace es asignar inicialmente la mayoría de la memoria al `dom0`, y a medida de que van apareciendo domU's se la va reduciendo. Con esto la fijamos de manera estática. Hay que comprobar que sea suficiente; si sufrimos algún crash del kernel Xen hay que aumentarla. También podemos obviar esta personalización de memoria y dejar la técnica de *balloning* por defecto. También hay que hacer el proceso equivalente en `/etc/xen/xend-config.sxp` con:

```
(dom0-min-mem 1024)
(enable-dom0-ballooning no)
```

Con estos cambios ya podremos rearrancar la máquina de nuevo con el `dom0` de Xen disponible con opciones básicas. La configuración de Xen en producción escapa a los términos de espacio de esta sección, y recomendamos consultar los manuales Xen para configurar, en especial: a) las CPUs disponibles a las máquinas Guest y `dom0`; b) comportamiento de Guests en rearranque; c) también es posible, para depuración, activar la consola por puerto serie para mensajes de depuración.

Con la configuración actual, pasaremos a detallar una instalación básica de domU (huésped) de tipo Debian mismo, ya que se nos ofrecen ciertas facilidades para el proceso automático:

```
apt-get install xen-tools
```

Así instalamos scripts de utilidades que nos provee Xen y después modificaremos `/etc/xen-tools/xen-tools.conf` para definir `dir`=/mnt/disk y `passwd`=1, con la dirección del parche donde podremos nuestras máquinas creadas (en /mnt/disk, en nuestro caso un volumen especial para guardar las guest).

Con el comando siguiente, y la configuración previa, podremos construir una imagen de Xen, una VM ya preparada con la imagen de Debian correspondiente (en este caso una wheezy):

```
# xen-create-image --hostname nombre_maquina --ip ip_maquina
--vcpus 1 --pygrub --dist wheezy
```

Coloquemos en el comando anterior un nombre de VM y una ip disponible (ya sea pública o privada), y comenzará la construcción de la imagen. En este caso se utilizan los repositorios de Debian para descargar los paquetes adecuados para construir la imagen.

Observaremos que nos da las características por defecto de la máquina, y que nos comienza a generar las imágenes de las particiones de *root* y *swap* en el directorio (a partir de nuestro caso /mnt/disk), *domains/nombre_maquina* donde residirán las imágenes de la VM guest que se está creando. Finalmente nos pedirá el *password* de *root* de la máquina creada y finalizará:

```
General Information
--------------------
Hostname       :  deb
Distribution   :  wheezy
Mirror         :  http://mirror.switch.ch/ftp/mirror/debian/
Partitions     :  swap            128Mb (swap)
                  /               4Gb   (ext3)
Image type     :  sparse
Memory size    :  128Mb
Kernel path    :  /boot/vmlinuz-3.2.0-4-amd64
Initrd path    :  /boot/initrd.img-3.2.0-4-amd64

Networking Information
----------------------
IP Address 1   : 10.0.0.2 [MAC: 00:16:3E:2D:D7:A2]

Creating partition image: /mnt/disk/domains/deb/swap.img
Creating swap on /mnt/disk/domains/deb/swap.img
Creating partition image: /mnt/disk/domains/deb/disk.img
Creating ext3 filesystem on /mnt/disk/domains/deb/disk.img
Installation method: debootstrap
Running hooks
Creating Xen configuration file

Setting up root password
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
All done

Logfile produced at:
 /var/log/xen-tools/deb.log

Installation Summary
---------------------
Hostname        :  deb
Distribution    :  wheezy
IP-Address(es)  :  10.0.0.2
RSA Fingerprint :  4e:f3:0a:47:c1:15:14:71:15:28:da:e2:9f:df:42:2b
Root Password   :  N/A
```

Para ejecutar la máquina creada (*nombre_maquina=deb* en este ejemplo):

```
$ xm create /etc/xen/deb.cfg
```

Y para borrar una imagen de VM:

```
$ xen-delete-image VMs_name
```

Para listar los dominios activos:

```
$xm list
Name              ID   Mem VCPUs      State   Time(s)
Domain-0           0  1023     1     r-----    247.9
deb                2   128     1     -b----      5.4
```

Y para conectar una consola a la máquina disponible (al dominio con ID=2 en este caso):

```
$xm console 2
```

Lo que nos permite realizar la conexión a la máquina virtual y realizar el proceso de login.

7. Presente del núcleo y alternativas

Los avances en el núcleo de Linux en determinados momentos fueron muy rápidos, pero actualmente, ya con una situación bastante estable con los núcleos de la rama 2.6.x y la rama posteriormente derivada 3.x, cada vez pasa más tiempo entre las versiones que van apareciendo. En cierta manera esto es bastante positivo: permite tener tiempo para corregir errores cometidos, ver aquellas ideas que no funcionaron bien y probar nuevas ideas, que, si resultan, se incluyen.

El núcleo, en evolución

El núcleo continúa evolucionando, incorporando las últimas novedades en soporte de hardware y mejoras en las prestaciones.

Comentaremos en este apartado algunas de las ideas de los últimos núcleos y algunas que están previstas, para dar indicaciones de lo que será el futuro próximo en el desarrollo del núcleo.

En la antigua rama 2.4.x del núcleo [Ces06] se realizaron algunas aportaciones en:

- Cumplimiento de los estándares IEEE POSIX, lo que permite que muchos de los programas existentes de UNIX pueden recompilarse y ejecutarse en Linux.
- Mejor soporte de dispositivos: PnP, USB, puerto paralelo, SCSI, etc.
- Soporte para nuevos sistemas de ficheros, como UDF (CD-ROM reescribibles como un disco). Otros sistemas con *journal*, como los Reiser de IBM o el ext3, que permiten tener un registro (*journal*) de las modificaciones de los sistemas de ficheros, y así poder recuperarse de errores o tratamientos incorrectos de los ficheros.
- Soporte de memoria hasta 4 GB. En su día surgieron algunos problemas (con núcleos 1.2.x) que no soportaban más de 128 MB de memoria (una cantidad que en aquel tiempo era mucha memoria).
- Se mejoró la interfaz `/proc`. Se trata de un pseudosistema de ficheros (el directorio `/proc`) que no existe realmente en disco, sino que es simplemente una forma de acceder a datos del núcleo y del hardware de una manera organizada.
- Soporte del sonido en el núcleo: se añadieron parcialmente los controladores Alsa que antes se configuraban por separado.
- Se incluyó soporte preliminar para el RAID software y el gestor de volúmenes dinámicos LVM1.

En la antigua rama del núcleo 2.6.x [Ces06, Pra03][Lov10], se dispuso de importantes avances respecto a la anterior (con las diferentes revisiones .x de la rama 2.6):

- Mejores prestaciones en SMP, importante para sistemas multiprocesadores muy utilizados en entornos empresariales y científicos.

- Mejoras en el planificador de CPU (*scheduler*). En particular, se introducen avances para mejorar el uso de tareas interactivas de usuario, imprescindibles para mejorar el uso de Linux en un ambiente de escritorio.

- Mejoras en el soporte *multithread* para las aplicaciones de usuario. Se incorporan nuevos modelos de hilos NGPT (IBM) y NPTL (Red Hat) (con el tiempo se consolidó finalmente la NPTL).

- Soporte para USB 2.0 y, posteriormente, para USB 3.0.

- Controladores Alsa de sonido incorporados en el núcleo.

- Nuevas arquitecturas de CPU de 64 bits, se soportan AMD x86_64 (también conocida como amd64) y PowerPC 64 y IA64 (arquitectura de los Intel Itanium).

- Sistemas de ficheros con *journal*: JFS, JFS2 (IBM) y XFS (Silicon Graphics).

- Mejoras con *journal* en los sistemas de ficheros propios, `ext3` y `ext4`, con mejoras del tamaño máximo de los archivos y de rendimiento general.

- Mejoras de prestaciones en E/S y nuevos modelos de controladores unificados.

- Mejoras en implementación de TCP/IP y el sistema NFSv4 (compartición de sistema de ficheros por red con otros sistemas).

- Mejoras significativas para núcleo apropiativo: permite que internamente el núcleo gestione varias tareas que se pueden interrumpir entre ellas, imprescindible para implementar eficazmente sistemas de tiempo real y también para aumentar el rendimiento de tareas interactivas.

- Suspensión del sistema y restauración después de reiniciar (por núcleo).

- UML, User Mode Linux, una especie de máquina virtual de Linux sobre Linux que permite ver un Linux (en modo usuario) ejecutándose sobre una máquina virtual. Esto es ideal para la propia depuración, ya que se puede desarrollar y probar una versión de Linux sobre otro sistema, y es útil tanto para el propio desarrollo del núcleo como para un análisis de seguridad del mismo. En versiones posteriores este concepto evolucionó hacia el módulo KVM.

- Técnicas de virtualización incluidas en el núcleo: en las distribuciones se han ido incorporando diferentes técnicas de virtualización, que necesitan extensiones en el núcleo. Cabe destacar, por ejemplo, núcleos modificados para Xen, Virtual Server (Vserver), OpenVZ o el propio módulo KVM.

- Nueva versión del soporte de volúmenes LVM2.

- Nuevo pseudosistema de ficheros `/sys`, destinado a incluir la información del sistema, y dispositivos que se irán migrando desde el sistema `/proc`, dejando este último con información relacionada con los procesos y su desarrollo en ejecución, así como la información dinámica del propio núcleo.

- Módulo FUSE para implementar sistemas de ficheros en espacio de usuario (en especial usado para el caso de NTFS).

Para conocer los cambios de las versiones más recientes de Linux, pueden examinarse los ficheros `ChangeLog` que acompañan a cada versión del núcleo en su código fuente, o consultar un registro histórico que se mantiene en *kernelnewbies.org*, en especial http://kernelnewbies.org/LinuxChanges, que mantiene los cambios de la última versión y pueden consultarse los del resto de versiones (http://kernelnewbies.org/LinuxVersions).

En las versiones 3.x del kernel, básicamente se han mejorado diferentes aspectos de la rama previa 2.6.x, aportando nuevas prestaciones, que también son el futuro inmediato de futuras versiones en el Kernel, que se centrarán en:

- Incremento de la tecnología de virtualización en el núcleo, para soportar diferentes configuraciones de sistemas operativos y diferentes tecnologías de virtualización, así como un mejor soporte del hardware para virtualización incluido en los procesadores que surjan en las nuevas arquitecturas. Están bastante soportadas x86 y x86_64, con KVM, por ejemplo, pero otras no lo están o solamente parcialmente.

- El soporte de SMP (máquinas multiprocesador), de CPU de 64 bits (Xeon, nuevos multicore de Intel y Opteron de AMD), el soporte de CPU *multicore* y la escalabilidad de aplicaciones multihilo en estas CPU.

- La mejora de sistemas de ficheros para clusterización y grandes sistemas distribuidos.

- Mejoras en sistemas de ficheros estándar Linux para adaptarlos a nuevas necesidades, como el caso de Btrfs y diversas mejoras introducidas en ext4, así como las mejoras introducidas y mejor soporte en el kernel para XFS y ZFS.

- Por el contrario, la mejora en núcleos más optimizados para dispositivos móviles (*smartphones*, *tablets*, etc.). Por ejemplo, se ha incorporado el nú-

cleo de Android, como plataforma, al código fuente del kernel. Y existen diversas iniciativas como Ubuntu Phone o Tizen, para proporcionar nuevas plataformas basándose en kernel Linux para entornos móviles. En especial, el soporte del kernel para arquitecturas ARM (de las más utilizadas en entornos móviles), ha traído la posibilidad de usar Linux en multitud de nuevos dispositivos.

- Mejora en el cumplimiento de los estándar POSIX.

Enlace de interés

Para saber más sobre POSIX podéis visitar la siguiente web:
http://www.unix.org/

- Mejora de la planificación de la CPU. Aunque se hicieron muchos avances en este aspecto, todavía hay un bajo rendimiento en algunas situaciones. En particular, en el uso de aplicaciones interactivas de escritorio se están estudiando diferentes alternativas, para mejorar este y otros aspectos relacionados con el escritorio y el uso del rendimiento gráfico.

- Soporte para las nuevas EFI/UEFI como sustitutos de las antiguas BIOS en entornos de PC x86/x86_64.

- Soporte para un nuevo sistema de filtrado IP NFtables, que substituirá progresivamente a los firewalls mediante iptables.

También, aunque se aparta de los sistemas Linux, la Free Software Foundation (FSF), y su proyecto GNU, siguen trabajando para acabar un sistema operativo completo. Cabe recordar que el proyecto GNU tenía como principal objetivo conseguir un clon UNIX de software libre, y las utilidades GNU solo son el software de sistema necesario. A partir de 1991, cuando Linus consigue conjuntar su núcleo con algunas utilidades GNU, se dio un primer paso que ha acabado en los sistemas GNU/Linux actuales. Pero el proyecto GNU sigue trabajando en su idea de terminar el sistema completo. En este momento disponen ya de un núcleo en el que pueden correr sus utilidades GNU. A este núcleo se le denomina Hurd; y a un sistema construido con él se le conoce como GNU/Hurd. Ya existen algunas distribuciones de prueba, en concreto, una Debian GNU/Hurd.

Enlace de interés

Para saber más sobre el proyecto GNU podéis visitar la siguiente página web:
http://www.gnu.org/gnu/thegnuproject.html

Hurd fue pensado como el núcleo para el sistema GNU hacia 1990, cuando comenzó su desarrollo, ya que entonces la mayor parte del software GNU estaba desarrollado y solo faltaba el núcleo. Fue en 1991 cuando Linus combinó GNU con su núcleo Linux y creó así el inicio de los sistemas GNU/Linux. Pero Hurd sigue desarrollándose. Las ideas de desarrollo en Hurd son más complejas, ya que Linux podría considerarse un diseño "conservador", que partía de ideas ya conocidas e implantadas.

Enlace de interés

Podéis leer las opiniones de Richard Stallman sobre GNU y Linux en:
http://www.gnu.org/gnu/linux-and-gnu.html

En concreto, Hurd estaba pensada como una colección de servidores implementados sobre un micronúcleo Mach [Vah96], que es un diseño de núcleo tipo micronúcleo (a diferencia de Linux, que es de tipo monolítico) desarrollado por la Universidad Carnegie Mellon y posteriormente por la Universidad

de Utah. La idea básica era modelar las funcionalidades del núcleo de UNIX como servidores que se implementarían sobre un núcleo básico Mach. El desarrollo de Hurd se retrasó mientras se estaba acabando el diseño de Mach, y este se publicó finalmente como software libre, que permitiría usarlo para desarrollar Hurd. En este punto debemos comentar la importancia de Mach, ya que muchos sistemas operativos se han basado en ideas extraídas de él, el más destacado de los cuales es el MacOS X de Apple.

El desarrollo de Hurd se retrasó más por la complejidad interna, ya que existían varios servidores con diferentes tareas de tipo *multithread* (de ejecución de múltiples hilos) y la depuración era extremadamente difícil. Pero hoy en día se dispone de algunas versiones de test, así como de versiones de prueba de distribución GNU/Hurd producidas por Debian. Con todo, el proyecto en si no es especialmente optimista de cara a obtener sistemas en producción, debido tanto a la complejidad como a la falta de soporte para dispositivos.

Puede que en un futuro no tan lejano pueda haber avances y coexistencia de sistemas GNU/Linux con GNU/Hurd, o incluso que sea sustituido el núcleo Linux por el Hurd, si se hicieran avances importantes en su desarrollo. Esto sería una solución si en algún momento Linux se estanca (ya que su diseño monolítico puede causar problemas si se hace mucho más grande). En cualquier caso, tanto unos sistemas como otros tienen un prometedor futuro por delante. El tiempo dirá hacia dónde se inclina la balanza.

8. Taller de configuración del núcleo a las necesidades del usuario

En este apartado vamos a ver un pequeño taller interactivo para el proceso de actualización y configuración del núcleo en el par de distribuciones utilizadas: Debian y Fedora.

Una primera cosa imprescindible, antes de comenzar, es conocer la versión actual que tenemos del núcleo, mediante `uname -r`, para poder determinar cuál es la versión siguiente que queremos actualizar o personalizar. Y otra es la de disponer de medios para arrancar nuestro sistema en caso de fallos: el conjunto de CD/DVD de la instalación, el disquete (o CD) de rescate (actualmente suele utilizarse el primer CD/DVD de la distribución) o alguna distribución en LiveCD que nos permita acceder al sistema de ficheros de la máquina, para rehacer configuraciones que hayan causado problemas. Además, deberíamos hacer copia de seguridad de nuestros datos o configuraciones importantes.

Veremos las siguientes posibilidades:

1) Actualización del núcleo de la distribución. Caso automático de Debian.

2) Actualización automática en Fedora.

3) Personalización de un núcleo genérico (tanto Debian como Fedora). En este último caso los pasos son básicamente los mismos que los que se presentan en el apartado de configuración, pero haremos algunos comentarios adicionales.

8.1. Configuración del núcleo en Debian

En el caso de la distribución Debian, la instalación puede hacerse también de forma automática mediante el sistema de paquetes de APT. Puede hacerse tanto desde línea de comandos como con gestores APT gráficos (synaptic, por ejemplo).

Vamos a realizar la instalación por línea de comandos con `apt-get`, suponiendo que el acceso a los paquetes fuente apt (sobre todo a los Debian originales) está bien configurado en el fichero de `/etc/apt/sources.list`. Veamos los pasos:

1) Actualizar la lista de paquetes:

```
# apt-get update
```

2) Listar paquetes asociados a imágenes del núcleo:

```
# apt-cache search linux-image
```

3) Elegir una versión adecuada a nuestra arquitectura (genérica, x86 o i386 para Intel o amd o, en particular para 64 bits, versiones amd64 para intel y amd). El código de la versión indica la versión del núcleo, la revisión de Debian del núcleo y la arquitectura. Por ejemplo, 3.14-1-amd64 es un núcleo para x86_64 AMD/Intel, revisión Debian 1 del núcleo 3.14.

4) Comprobar, para la versión elegida, que existan los módulos accesorios extras. Con `apt-cache` buscamos si existen otros módulos dinámicos que puedan ser interesantes para nuestro hardware, según la versión del núcleo a instalar. Recordad que, como vimos en la *Debian Way*, también existe la utilidad `module-assistant`, que nos permite automatizar este proceso si los módulos están soportados. En el caso en que los módulos necesarios no estuvieran soportados, esto nos podría impedir actualizar el núcleo si consideramos que el funcionamiento del hardware problemático es vital para el sistema.

5) Buscar, si queremos disponer también del código fuente del núcleo, los `linux-source-version` (en este caso 3.14, es decir, el número principal) y los `linux-headers` correspondientes, por si más tarde queremos hacer un núcleo personalizado (en este caso, el núcleo genérico correspondiente parcheado por Debian).

6) Instalar lo que hayamos decidido. Si queremos compilar desde los paquetes fuente o simplemente disponer del código:

```
# apt-get install linux-image-version
```

(si fueran necesarios algunos módulos) y

```
# apt-get install linux-source-version-generica
# apt-get install linux-headers-version
```

7) Instalar el nuevo núcleo, por ejemplo en el *bootloader* LiLo o Grub (en las últimas versiones encontraremos este por defecto). Normalmente este paso se hace automáticamente, pero no estaría de más realizar alguna copia de seguridad previa de la configuración del *bootloader*
(ya sea `/etc/lilo.conf` o `/boot/grub/menu.lst o grub.cfg`).

Si se nos pregunta si tenemos el `initrd` activado, habrá que verificar el fichero de LiLo (`/etc/lilo.conf`) e incluir la nueva línea en la configuración LiLo de la imagen nueva:

```
initrd = /initrd.img-version (o /boot/initrd.img-version)
```

Una vez hecha esta configuración, tendríamos que tener un LiLo parecido al siguiente listado (fragmento del fichero), suponiendo que `initrd.img` y `vmlinuz` sean enlaces a la posición de los ficheros del nuevo núcleo:

```
default = Linux
image = /vmlinuz
  label = Linux
  initrd = /initrd.img
# restricted
# alias = 1
image = /vmlinuz.old
  label = LinuxOLD
  initrd = /initrd.img.old
# restricted
# alias = 2
```

Tenemos la primera imagen por defecto, la otra es el núcleo antiguo. Así, desde el menú LiLo podremos pedir una u otra o, simplemente cambiando el `default`, recuperar la antigua. Siempre que realicemos cambios en el archivo `/etc/lilo.conf` no debemos olvidarnos de reescribirlos en el sector correspondiente con el comando `/sbin/lilo` o `/sbin/lilo -v`.

En el caso de Grub, que suele ser la opción normal en las distribuciones (ahora Grub2 en particular), se habría creado una nueva entrada (hay que tener presente realizar la copia de seguridad previa, porque podemos haber perdido alguna entrada anterior dependiendo del funcionamiento o configuración de Grub; por ejemplo, puede limitarse el número máximo de entradas):

```
title           Debian GNU/Linux, kernel 2.6.32-5-amd64
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.32-5-amd64 \
                root=UUID=4df6e0cd-1156-444e-bdfd-9a9392fc3f7e ro
initrd          /boot/initrd.img-2.6.32-5-amd64
```

donde aparecerían los ficheros binarios del núcleo y `initrd`. Con la etiqueta `root` en el núcleo aparece el identificador de la partición raíz del sistema donde está instalado el núcleo, identificador que es común a todas las entradas de núcleos del mismo sistema. Antes se utilizaba un esquema con `root=/dev/hda0` o `/dev/sda0`, pero este esquema ya no es útil, porque en la detección de los discos puede cambiar el orden de estos en el sistema; así, se prefiere etiquetar las particiones. Estas etiquetas pueden obtenerse mediante el comando del subsistema udisks `udisksctl info -b /dev/particion`, ver symlinks, o también con el comando `dumpe2fs /dev/particion | grep UUID` o, si la partición se encuentra montada en el sistema, consultando el archivo `/etc/fstab`.

8.2. Configuración del núcleo en Fedora/Red Hat

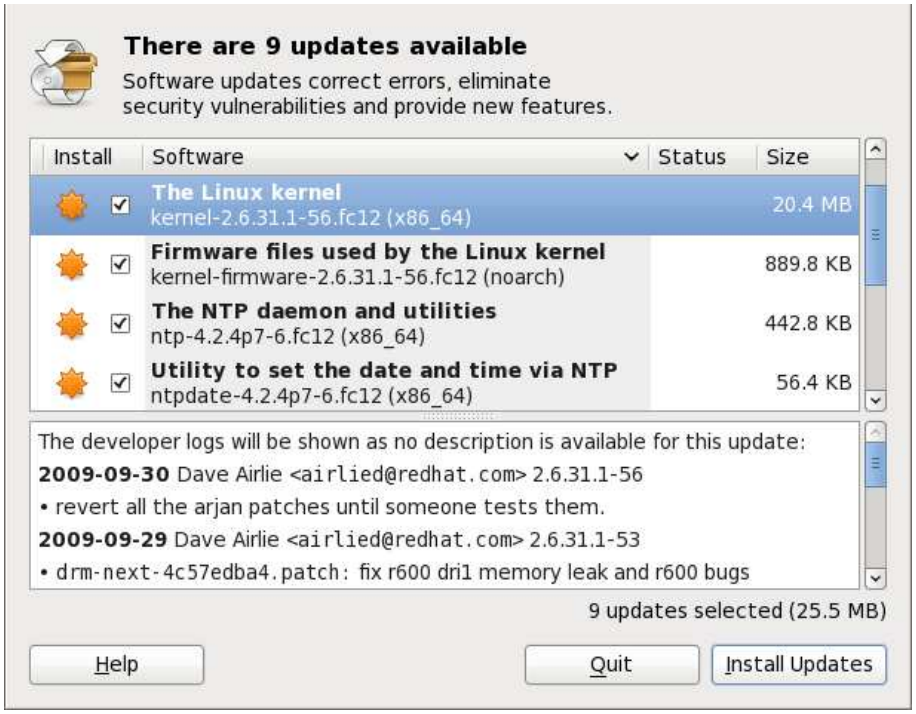
La actualización del núcleo en la distribución Fedora/Red Hat es totalmente automática por medio de su servicio de gestión de paquetes (`yum` en línea de comandos, por ejemplo), o bien mediante los programas gráficos que incluye

la distribución, dependiendo de su versión, para la actualización (PackageKit en Fedora o pup en Red Hat empresarial o equivalentes como CentOS). Normalmente, las encontraremos en la barra de tareas o en el menú de herramientas de sistema de Fedora/Red Hat.

Este programa de actualización básicamente verifica los paquetes de la distribución actual frente a una base de datos de Fedora/Red Hat, y ofrece la posibilidad de descargar los paquetes actualizados, entre ellos los del núcleo. Este servicio, en el caso de Red Hat empresarial, funciona por una cuenta de servicio y Red Hat lo ofrece por pago. Con este tipo de utilidades la actualización del núcleo es automática. Hay que comprobar las utilidades disponibles en los menús Herramientas/Administración, ya que las herramientas gráficas disponibles en una distribución son altamente dependientes de su versión, puesto que son actualizadas con frecuencia.

Por ejemplo, en la figura 8, observamos que una vez puesto en ejecución nos ha detectado una nueva versión del núcleo disponible y podemos seleccionarla para que nos la descargue.

Figura 8. Herramienta de Fedora que muestra la actualización del núcleo disponible



En Fedora podemos utilizar las herramientas gráficas equivalentes o usar directamente yum, si conocemos la disponibilidad de nuevos núcleos:

```
# yum install kernel kernel-source
```

Una vez descargada, se procederá a su instalación, normalmente también de forma automática, ya dispongamos de Grub o LiLo como gestores de arran-

que. En el caso de Grub, suele ser automático y deja un par de entradas en el menú, una para la versión más nueva y otra para la antigua. Por ejemplo, en esta configuración de Grub (el fichero está en `/boot/grub/grub.cfg` o bien `/boot/grub/menu.lst`), tenemos dos núcleos diferentes, con sus respectivos números de versión:

```
#fichero grub.conf
default = 1
timeout = 10
splashimage = (hd0,1)/boot/grub/splash.xpm.gz

title Linux (2.6.30-2945)
root (hd0,1)
kernel /boot/vmlinuz-2.6.30-2945 ro
       root = UUID=4df6e0cd-1156-444e-bdfd-9a9392fc345f
initrd /boot/initrd-2.6.30-18.9.img

title LinuxOLD (2.6.30-2933)
root (hd0,1)
kernel /boot/vmlinuz-2.4.30-2933 ro
       root = UUID=4df6e0cd-1156-444e-bdfd-9a9392fc345f
initrd /boot/initrd-2.4.30-2933.img
```

Notad que la configuración actual es Grub-Legacy; para Grub2 los campos son parecidos pero hace falta consultar las *menuentry* correspondientes a cada opción.

Cada configuración incluye un título, que aparecerá en el arranque; el root, o partición del disco desde donde arrancar; el directorio donde se encuentra el fichero correspondiente al núcleo y el fichero `initrd` correspondiente.

En el caso de que dispongamos de LiLo* como gestor en la Fedora/Red Hat, el sistema también lo actualiza (fichero `/etc/lilo.conf`), pero luego habrá que reescribir el arranque con el comando `/sbin/lilo -v` manualmente.

***Por defecto se usa Grub.**

Cabe señalar, asimismo, que con la instalación anterior teníamos posibilidades de descargar los paquetes fuente del núcleo; éstos, una vez instalados, están en `/usr/src/linux-version`, y pueden configurarse y compilarse por el procedimiento habitual, como si fuese un núcleo genérico. Hay que mencionar que la empresa Red Hat lleva a cabo un gran trabajo de parches y correcciones para el núcleo (usado después en Fedora), y que sus núcleos son modificaciones al estándar genérico con bastantes añadidos, por lo cual puede ser mejor utilizar los fuentes propios de Red Hat, a no ser que queramos un núcleo más nuevo o experimental que el que nos proporcionan.

8.3. Configuración de un núcleo genérico

Vamos a ver el caso general de instalación de un núcleo a partir de sus fuentes. Supongamos que tenemos unas fuentes ya instaladas en `/usr/src` (o el prefijo correspondiente).

Normalmente, tendremos un directorio `linux, linux-version` o sencillamente el número versión; este será el árbol de los paquetes fuente del núcleo. Estos pueden provenir de la misma distribución (o puede ser que los hayamos *descargado de* una actualización previa) y en primer lugar será interesante comprobar si son los últimas disponibles, como ya hemos hecho antes con Fedora o Debian. Por otro lado, si queremos tener las últimas y genéricas versiones, podemos ir a *kernel.org* y *descargar* la última versión disponible (mejor la estable que las experimentales, a no ser que estemos interesados en el desarrollo del núcleo). Descargamos el archivo y descomprimimos en `/usr/src` (u otro elegido, quizá mejor) los paquetes fuente del núcleo. También podríamos buscar si existen parches para el núcleo y aplicarlos (según hemos *comentado* en el apartado 4).

Ved también

Sería conveniente releer el apartado 3.

A continuación comentaremos los pasos que habrá que realizar. El procedimiento que se indica en esta parte del taller es genérico, pero puede dar algún problema dependiendo de la distribución usada. Se recomienda seguir en lo posible el subapartado 3.1, donde se comenta el caso de configuración de un núcleo *vanilla*, o bien los comentarios en el caso Debian, en el subapartado 3.2, o la referencia [Fedk] para el caso Fedora.

Con las consideraciones comentadas podemos seguir también el proceso siguiente:

1) Limpiar el directorio de pruebas anteriores (si es el caso):

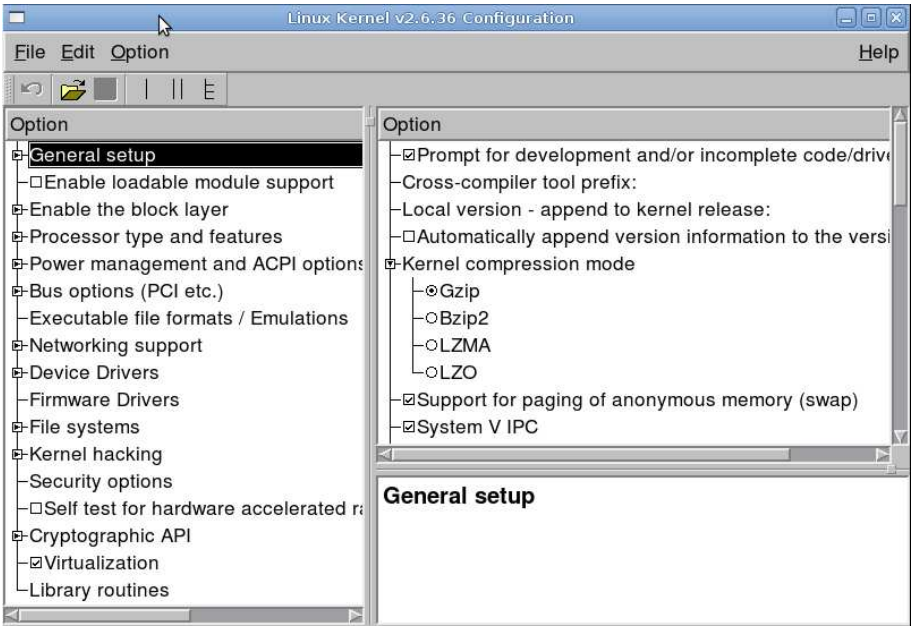
```
make mrproper
```

2) Configurar el núcleo con, por ejemplo, `make menuconfig` (o `xconfig`, figura 9, `gconfig` o `oldconfig`). Lo vimos en el subapartado 3.1.

3) Compilación y creación de la imagen del núcleo: `make`. El proceso puede durar desde algunas decenas de minutos a una hora en hardware moderno o varias horas en hardware muy antiguo. Cuando finaliza, la imagen se halla en: `/usr/src/directorio-fuentes/arch/i386/boot` (en el caso de arquitectura Intel de 32/64 bits).

4) Ahora compilamos los módulos con `make modules`. Hasta este momento no hemos modificado nada en nuestro sistema. Ahora tendremos que proceder a la instalación.

Figura 9. Configuración (xconfig) del núcleo por menús gráficos



Pero hay también que tener cuidado si estamos compilando una versión que es la misma (exacta numeración) que la que tenemos (los módulos se sobrescribirán); en este caso es mejor realizar una copia de seguridad de los módulos:

```
cd /lib/modules
tar -cvzf old_modules.tgz versionkernel-antigua/
```

Así, tenemos una versión en `.tgz` que podríamos recuperar después en caso de problemas. Y, finalmente, instalamos los módulos con:

```
# make modules install
```

5) Ahora podemos pasar a la instalación del núcleo, por ejemplo (de manera manual) con:

```
# cd /usr/src/directorio-Fuentes/arch/i386/boot
# cp bzImage /boot/vmlinuz-versionkernel
# cp System.map /boot/System.map-versionkernel
# ln -s /boot/vmlinuz-versionkernel /boot/vmlinuz
# ln -s /boot/System.map-versionkernel /boot/System.map
```

Así colocamos el fichero de símbolos del núcleo (`System.map`) y la imagen del núcleo. Cabe recordar que puede ser necesaria también una imagen de `initrd`.

6) Ya solo nos queda poner la configuración necesaria en el fichero de configuración del gestor de arranque, ya sea LiLo (`/etc/lilo.conf`), Grub-Legacy o Grub2 (`/boot/grub/menu.lst o grub.cfg`) según las configuraciones que ya vimos con Fedora o Debian. Y recordad que en el caso de LiLo, habrá que volver a actualizar la configuración con `/sbin/lilo` o `/sbin/lilo -v`.

Cabe recordar que como vimos, todo este proceso en los kernels modernos puede hacerse simplemente con un:

```
# make install
```

7) Reiniciar la máquina y observar los resultados (si todo el proceso ha sido correcto).

Resumen

En este módulo se han examinado diferentes características del núcleo (en inglés, *kernel*) de Linux en sus ramas 2.6.x y 3.x. Asimismo, se han comentado algunos de sus procesos de actualización, configuración y sintonización para el sistema, útiles tanto para la optimización del uso de memoria como para maximizar las prestaciones en una arquitectura de CPU concreta, así como la adaptación al hardware (dispositivos) presente en el sistema.

También hemos examinado las posibilidades que ofrecen los módulos dinámicos como mecanismo de extensión del núcleo y ampliación del hardware soportado.

La inclusión en el núcleo de técnicas de virtualización nos permite, a partir del núcleo y ciertas utilidades, construir un entorno de virtualización potente y flexible para generar diferentes combinaciones de máquinas virtuales que residen en un sistema *host* GNU/Linux.

Actividades

1. Determinad la versión actual del núcleo Linux incorporada en vuestra distribución. Comprobad las actualizaciones disponibles de forma automática, ya sea en Debian (apt) o en Fedora/Red Hat (vía yum).

2. Efectuad una actualización automática de vuestra distribución. Comprobad posibles dependencias con otros módulos utilizados y con el *bootloader* (LiLo o Grub-Legacy o Grub2) utilizado. Dependiendo del sistema puede ser recomendable una copia de seguridad de los datos importantes del sistema (cuentas de usuarios y ficheros de configuración modificados), o bien realizar el proceso en otro sistema del que se disponga para pruebas.

3. Para vuestra rama del núcleo, determinad la última versión disponible (consultad la página web http://www.kernel.org) y realizad una instalación manual con los pasos examinados en el módulo. La instalación final puede dejarse como opcional, o bien poner una entrada dentro del *bootloader* para las pruebas del nuevo núcleo. Observad también, dependiendo del sistema, la posibilidad de realizar una copia de seguridad previa, en especial de las configuraciones estables de los *bootloaders*.

4. En el caso de la distribución Debian, además de los pasos manuales, existe, como vimos, una forma especial (recomendada) de instalar el núcleo a partir de sus fuentes mediante el paquete `kernel-package`. Proceded con los pasos necesarios para crear una versión personalizada de un núcleo *vanilla*.

5. Elaborad una máquina virtual basada en KVM, que tenga como *guest* otra distribución GNU/Linux diferente a la del sistema *host*, a partir de su instalación vía CDROM o vía imagen ISO de la distribución.

Bibliografía

[Arc] **Arcomano, R.**. *Kernel Analysis-HOWTO*. The Linux Documentation Project.

[Bac86] **Bach, M. J.** (1986). *The Design of the UNIX Operating System*. Prentice Hall.

[Ces06] **Cesati, M.; Bovet, D.** (2006). *Understanding the Linux Kernel* (3.ª ed.). O'Reilly.

[Cor05] **Corbet, J.; Rubini, A.; Kroah-Hartman, G.** (2005). *Linux Device Drivers* (3.ª ed.). O'Reilly.

[Debk] **Debian Kernel Handbook Project**. *Debian Linux Kernel Handbook*. <http://kernel-handbook.alioth.debian.org>

[Fedk] **Fedora Project**. *Building a custom kernel*. <http://fedoraproject.org/wiki/Building_a_custom_kernel>

[Gor] **Gortmaker, P.** (2003). *The Linux BootPrompt HOWTO*. The Linux Documentation Project.

[Grub1] **GNU**. *Grub bootloader*. <http://www.gnu.org/software/grub/grub-legacy.html>

[Grub2] **GNU**. *Grub Manual*. <http://www.gnu.org/software/grub/manual/>

[Hen] **Henderson, B.** *Linux Loadable Kernel Module HOWTO*. The Linux Documentation Project.

[Kan] **Kanis, I.**. *Multiboot with GRUB Mini-HOWTO*. The Linux Documentation Project.

[Ker] **Rusty Russell**. *Unreliable Guide To Hacking The Linux Kernel*. <https://www.kernel.org/doc/htmldocs/kernel-hacking/index.html>

[Kera] **Kernelnewbies.org**. *Kernel Newbies*. <http://www.kernelnewbies.org>

[Kerb] **Kernel.org**. *Linux Kernel Archives*. <http://www.kernel.org>

[Lkm] **Lkm**. *Linux Kernel Mailing List*. <http://www.tux.org/lkml>

[Lov10] **Love, R.** *Linux Kernel Development*. 3rd Edition. 2010. Addison-Wesley.

[Mur] **Murphy, G. L.** *Kernel Book Project*. <http://kernelbook.sourceforge.net>

[OSDa] **OSDL**. *Open Source Developement Laboratories*. (Ahora *The Linux Foundation*) <http://www.linuxfoundation.org>

[Pra03] **Pranevich, J.** (2003). *The Wonderful World of Linux 2.6*. <http://www.kniggit.net/wwol26.html>

[Pra11] **Pranevich, J.** (2011). *The Wonderful World of Linux 3.0*. <http://www.kniggit.net/wwol30/>

[Skoa] **Skoric, M.** *LILO mini-HOWTO*. The Linux Documentation Project.

[Tan87] **Tanenbaum, A.** (1987). *Sistemas operativos: Diseño e Implementación*. Prentice Hall.

[Tan06] **Tanenbaum, A.; Woodhull, A. S.** (2006). *Operating Systems Design and Implementation*. 3rd Edition. Prentice Hall.

[Tum] **Tumenbayar, E.** (2002). *Linux SMP HOWTO*. The Linux Documentation Project.

[Vah96] **Vahalia, U.** (1996). *UNIX Internals: The New Frontiers*. Prentice Hall.

[Vasb] **Vasudevan, A.** *The Linux Kernel HOWTO*. The Linux Documentation Project.

[Zan] **Zanelli, R.**. *Win95 + WinNT + Linux multiboot using LILOmini-HOWTO*. The Linux Documentation Project.

Sobre estas fuentes de referencia e información:

[Kerb] Sitio que proporciona un repositorio de las diversas versiones del núcleo Linux y sus parches.

[Kera] [lkm] Sitios web que recogen una parte de la comunidad del núcleo de Linux. Dispone de varios recursos de documentación y listas de correo de la evolución del núcleo, su estabilidad y las nuevas prestaciones que se desarrollan.

[Debk] Es un manual imprescindible sobre los procesos de compilación del núcleo en la distribución Debian. Suele actualizarse con los cambios producidos en la distribución. [Fedk] aporta una referencia similar para el caso Fedora.

[Ces06] Libro sobre el núcleo de Linux 2.4, que detalla los diferentes componentes y su implementación y diseño. Existe una primera edición sobre el núcleo 2.2 y una nueva actualización al núcleo 2.6. [Lov10] es un texto alternativo más actualizado. Para la rama 3.x estos libros siguen siendo útiles, ya que la mayoría de conceptos del kernel se mantienen inalterados.

[Pra03] Artículo que describe algunas de las principales novedades de la rama 2.6 del núcleo Linux. [Pra11] es el equivalente para 3.0+.

[Ker] [Mur] Proyectos de documentación del núcleo, incompletos pero con material útil.

[Bac86] [Vah96] [Tan87] [Tan86] Algunos textos sobre los conceptos, diseño e implementación de los núcleos de diferentes versiones UNIX y Linux.

[Skoa][Zan01][Kan][Grub1][Grub2] Recursos para tener más información sobre los cargadores LiLo, Grub y Grub2.

[Gru2][Grub1] Sitios oficiales de Grub2 y el anterior original Grub (ahora conocido como Grub Legacy.)

Administración de servidores

Remo Suppi Boldrito

PID_00212473

Universitat Oberta de Catalunya

Índice

Introducción 5

Objetivos 6

1. Administración de servidores 7
1.1. Sistema de nombres de dominio (*Domain Name System, DNS*) 7
1.1.1. Servidor de nombres caché 8
1.1.2. Configuración de un dominio propio 10
1.1.3. Otros DNS 14
1.2. NIS (YP) 17
1.2.1. ¿Cómo iniciar un cliente local de NIS en Debian? 17
1.2.2. ¿Qué recursos se deben especificar para utilizar el NIS? 18
1.2.3. ¿Cómo se debe configurar un servidor? 19
1.3. Servicios de conexión remota: telnet y ssh 21
1.3.1. Telnet y telnetd 21
1.3.2. SSH, *Secure shell* 22
1.3.3. VPN SSL (via tun driver) 25
1.3.4. Túneles encadenados 26
1.4. Servicios de transferencia de ficheros: FTP 27
1.4.1. Cliente ftp (convencional) 28
1.4.2. Servidores FTP 29
1.5. *Active Directory Domain Controller* con Samba4 30
1.5.1. Pasos preliminares 31
1.5.2. Compilar Samba4 33
1.6. Servicios de intercambio de información a nivel de usuario ... 36
1.6.1. El *Mail Transport Agent* (MTA) 36
1.6.2. External SMTP 37
1.7. *Internet Message Access Protocol* (IMAP) 38
1.7.1. Aspectos complementarios 39
1.8. Grupos de discusión 41
1.9. *World Wide Web* (httpd) 42
1.9.1. Servidores virtuales 44
1.9.2. Apache + PHP + Mysql + PhpMyAdmin 47
1.9.3. Otros servidores httpd 48
1.10. Servidor de WebDav 49
1.11. Servicio de *proxy*: Squid 51
1.11.1. Proxy SOCKS 54
1.12. OpenLdap (Ldap) 56
1.13. Servicios de archivos (NFS, *Network File System*) 60
1.14. Servidor de wiki 61
1.14.1. Instalación rápida 62

1.14.2. Instalación de servidor 62
1.15. Gestión de copias de respaldo (*backups*) 64
1.15.1. Programas habituales de copias de respaldo 64
1.15.2. rdiff-backup y rdiff-backups-fs 66
1.16. *Public Key Infrastructure* (PKI) 67

Actividades 72

Bibliografía 73

Introducción

La interconexión entre máquinas y las comunicaciones de alta velocidad han permitido que los recursos que se utilicen no estén en el mismo sitio geográfico del usuario. UNIX (y por supuesto GNU/Linux) es probablemente el máximo exponente de esta filosofía, ya que desde su inicio ha fomentado el intercambio de recursos y la independencia de dispositivos. Esta filosofía se ha plasmado en algo común hoy en día como son los servicios. Un servicio es un recurso (que puede ser universal o no) y que permite, bajo ciertas condiciones, obtener información, compartir datos o simplemente procesar la información a distancia. Nuestro objetivo es analizar los servicios que permiten el funcionamiento de una red. Generalmente, dentro de esta red existirá una máquina (o varias, según las configuraciones) que hará posible el intercambio de información entre las demás. Estas máquinas se denominan servidores y contienen un conjunto de programas que permiten que la información esté centralizada y sea fácilmente accesible. Estos servicios permiten la reducción de costes y amplían la disponibilidad de la información, pero se debe tener en cuenta que un servicio centralizado presenta inconvenientes, ya que puede quedar fuera de servicio y dejar sin atención a todos los usuarios. En este módulo se verán los principales servicios que permiten que una máquina GNU/Linux juegue un papel muy importante en una infraestructura tecnológica, tanto en centralizar y distribuir datos como en ser punto de información, acceso o comunicación. Por otro lado y con el avance de las arquitecturas (software) orientada a servicios (SOA - *Service Oriented Architecture*), y las tecnologías de desarrollo de aplicaciones que se han estandarizado en este paradigma de diseño de sistemas distribuidos, GNU/Linux se ha transformado en la infraestructura por excelencia que da soporte a la creación de sistemas de información altamente escalables. Este tipo de arquitectura (SOA) se ha transformado en una parte esencial del desarrollo de software distribuido, ya que permite la creación de sistemas distribuidos eficientes, que aprovechan toda la infraestructura subyacente, y establece una interfaz bien definida a la exposición e invocación de servicios web (de forma común pero no exclusivamente) facilitando la interacción entre los sistemas propios y externos.

Servicios replicados

Una arquitectura de servidores debe tener los servicios replicados *(mirrors)* para solventar los inconvenientes que supone.

Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Presentar los aspectos más relevantes de los conceptos involucrados, tanto a nivel teórico como práctico, en la estructura de servidores/servicios en un sistema GNU/Linux.
2. Analizar los conceptos relativos a servicios y servidores específicos de un sistema GNU/Linux.
3. Experimentar con la configuración y adaptar la instalación de servicios a un entorno determinado.
4. Analizar y participar en discusiones sobre las posibilidades actuales y futuras de nuevos servicios y los obstáculos que existen básicamente en aspectos de seguridad en los diferentes entornos de trabajo GNU/Linux (servidor, escritorio multimedia, escritorio ofimàtica, enrutador o *router*,...).

1. Administración de servidores

Los servicios se pueden clasificar en dos tipos: de vinculación ordenador-ordenador o de relación hombre-ordenador. En el primer caso, se trata de servicios requeridos por otros ordenadores, mientras que en el segundo, son servicios requeridos por los usuarios (aunque hay servicios que pueden actuar en ambas categorías). Dentro del primer tipo se encuentran servicios de nombres, como el *Domain Name System* (DNS), el servicio de información de usuarios (NIS-YP), el directorio de información LDAP o los servicios de almacenamiento intermedio (*proxies*). Dentro de la segunda categoría se contemplan servicios de conexión interactiva y ejecución remota (ssh, telnet), transferencia de ficheros (ftp), intercambio de información a nivel de usuario, como el correo electrónico (MTA, IMAP, POP), *news*, *World Wide Web*, *Wiki* y archivos (NFS). Para mostrar las posibilidades de GNU/Linux Debian-FC, se describirá cada uno de estos servicios con una configuración mínima y operativa, pero sin descuidar aspectos de seguridad y estabilidad.

1.1. Sistema de nombres de dominio (*Domain Name System, DNS*)

La funcionalidad del servicio de DNS es convertir nombres de máquinas (legibles y fáciles de recordar por los usuarios) en direcciones IP o viceversa.

A la consulta de cuál es la IP de *nteum.remix.cat*, el servidor responderá 192.168.0.1 (esta acción es conocida como *mapping*); del mismo modo, cuando se le proporcione la dirección IP, responderá con el nombre de la máquina (conocido como *reverse mapping*).

El *Domain Name System* (DNS) es una arquitectura arborescente que evita la duplicación de la información y facilita la búsqueda. Por ello, un único DNS no tiene sentido sino como parte del árbol. Una de las aplicaciones más utilizadas que presta este servicio se llama *named*, se incluye en la mayoría de distribuciones de GNU/Linux (`/usr/sbin/named`) y forma parte de un paquete llamado `bind` (actualmente versión 9.x), coordinado por el ISC (Internet Software Consortium). El DNS es simplemente una base de datos, por lo cual, es necesario que las personas que la modifiquen conozcan su estructura, ya que, de lo contrario, el servicio quedará afectado. Como precaución debe tenerse especial cuidado en guardar las copias de los archivos para evitar cualquier in-

terrupción en el servicio. Los servidores DNS, que pueden convertir la mayoría de los nodos de DNS en su IP correspondiente y se les llama servidores DNS recursivos. Este tipo de servidor no puede cambiar los nombres de los nodos de DNS que hay, simplemente se preguntan otros servidores DNS la IP de un nodo DNS dado. Los servidores DNS autorizados pueden gestionar/cambiar las direcciones IP de los nodos DNS que gestionan y normalmente son con los cuales contactará un servidor DNS recursivo con el fin de conocer la IP de un nodo DNS dado. Una tercera variante de los servidores DNS son los DNS caché, que simplemente almacenan la información obtenida de otros servidores DNS recursivos.

1.1.1. Servidor de nombres caché

En primer lugar, se configurará un servidor de DNS para resolver consultas, que actúe como caché para las consultas de nombres *resolver, caching only server*). Es decir, la primera vez consultará al servidor adecuado porque se parte de una base de datos sin información, pero las veces siguientes responderá el servidor de nombres caché, con la correspondiente disminución del tiempo de respuesta. Para instalar un servidor de esta características, instalamos los paquetes `bind9 y dnsutils` (p. ej., en Debian `apt-get bind9 dnsutils`).

Como se puede observar en el directorio */etc/bind* encontraremos una serie de archivos de configuración entre ellos el principal es named.conf que incluye a otros *named.conf.options, named.conf.local, named.conf.default-zones* (podemos encontrar que esta configuración puede variar entre distribuciones pero es similar). El archivo */etc/bind/named.conf.options* contiene las opciones generales para el servidor y los otros dos archivos nos servirán para configurar nuestras zonas en el servidor de nombres que veremos en el apartado siguiente. Para configurar nuestro *caching only server* debemos considerar que las pregunta en primer lugar las resolveremos nosotros pero como es evidente que no tenemos la respuesta se deberá redirigir la pregunta a servidores de DNS en la red. Para ello, es interesante considerar los de servicios como OpenDNS (http://www.opendns.com/opendns-ip-addresses/) que son las IP: 208.67.222.222 y 208.67.220.220 o bien servicios como los de Google (https://developers.google.com/speed/public-dns/?csw=1) que responden a las direcciones IP: 8.8.8.8 y 8.8.4.4. A continuación modificamos el archivo */etc/bind/named.conf.options* para quitar los comentarios de la sección *forwarders* poniendo lo siguiente:

```
forwarders {
    // OpenDNS servers
     208.67.222.222;
     208.67.220.220;
    // Podríamos incluir nuestro ISP/router -verificar la IP-
     192.168.1.1;
    };
```

Se podrían agregar en el mismo archivo al final y reemplazando las que hay (se puede dejar esta configuración para un segundo paso) opciones de seguridad para que solo resuelva las peticiones de nuestra red:

```
// Security options
 listen-on port 53 { 127.0.0.1; 192.168.1.100; };
 allow-query { 127.0.0.1; 192.168.1.0/24; };
 allow-recursion { 127.0.0.1; 192.168.1.0/24; };
 allow-transfer { none; };
// Las siguientes dos opciones ya se encuentran por defecto en el
// archivo pero si no tenemos IPv6 podemos comentar la última.
 auth-nxdomain no; # conform to RFC1035
 // listen-on-v6 { any; };
```

Se puede verificar la configuración con `named-checkconf`. A continuación modificamos el archivo */etc/resolv.conf* para añadir `nameserver 127.0.0.1` para que las preguntas a DNS la librería *gethostbyname(3)* la realice al propio *host* y el archivo */etc/nsswitch.conf* verificando que la línea *hosts* quede como: `hosts: files dns` indicará el orden que se resolverán las peticiones (primero localmente al */etc/hosts* y luego al servidor DNS). Por último solo resta reiniciar el servidor con `service bind9 restart` y hacer las pruebas de funcionamiento. En nuestro caso hemos ejecutado `dig www.debian.org` -se han omitido líneas para resumir la información-:

```
; «» DiG 9.8.4-rpz2+rl005.12-P1 «» www.debian.org
...
;; QUESTION SECTION:
;www.debian.org. IN A

;; ANSWER SECTION:
www.debian.org. 300 IN A 130.89.148.14
www.debian.org. 300 IN A 5.153.231.4
...
;; Query time: 213 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
...
```

Donde podemos observar que la *query* ha tardado 213 milisegundos. Si la realizamos por segunda vez (ya ha quedado la consulta almacenada en nuestro servidor caché):

```
; «» DiG 9.8.4-rpz2+rl005.12-P1 «» www.debian.org
...
;; QUESTION SECTION:
;www.debian.org. IN A

;; ANSWER SECTION:
www.debian.org. 293 IN A 5.153.231.4
www.debian.org. 293 IN A 130.89.148.14
...
;; Query time: 1 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
...
```

En este caso la consulta ha tardado 1 milisegundo verificando que el servidor caché ha funcionado. También podemos hacer la consulta inversa con `nslookup 130.89.148.14` lo cual nos dará el nombre de la máquina que

tiene asignada esta IP (en este caso es el servidor al cual está asignado el dominio www.debian.org)

```
Server: 127.0.0.1
Address: 127.0.0.1#53
Non-authoritative answer:
14.148.89.130.in-addr.arpa name = klecker4.snt.utwente.nl.
Authoritative answers can be found from:
. nameserver = l.root-servers.net.
. nameserver = b.root-servers.net.
. nameserver = j.root-servers.net.
...
```

Es importante considerar si se desea instalar el servidor de DNS en una máquina virtual, al ser un servicio que se debe acceder desde fuera, el servidor DNS (y la mayoría de los servidores) no puede estar en NAT sobre un *host* sino que debe tener IP propia del segmento de red en la cual presta el servicio y por lo tanto el adaptador de red de la máquina virtual debe estar en modo puente (*bridged*). Para configurar los clientes dentro de nuestra red bastará con modificar el archivo */etc/resolv.conf* si son GNU/Linux (o modificar */etc/network/interfaces* para agregarlos como `dns-nameservers IP` si se tiene instalado el paquete `resolvconf`) y en Windows modificar las propiedades IPv4 del adaptador para introducir la IP de nuestro servidor DNS caché.

1.1.2. Configuración de un dominio propio

DNS posee una estructura en árbol y el origen es conocido como "." (ver `/etc/bind/db.root`). Bajo el "." existen los TLD (*Top Level Domains*) como **org**, **com**, **edu**, **net**, etc. Cuando se busca en un servidor, si este no conoce la respuesta, se buscará recursivamente en el árbol hasta encontrarla. Cada "." en una dirección (por ejemplo, nteum.remix.cat) indica una rama del árbol de DNS diferente y un ámbito de consulta (o de responsabilidad) diferente que se irá recorriendo en forma recursiva de izquierda a derecha.

Otro aspecto importante, además del dominio, es el `in-addr.arpa` (*inverse mapping*), el cual también está anidado como los dominios y sirve para obtener nombres cuando se consulta por la dirección IP. En este caso, las direcciones se escriben al revés, en concordancia con el dominio. Si nteum.remix.world es la 192.168.0.30, entonces se escribirá como 30.0.168.192, en concordancia con nteum.remix.world para la resolución inversa de IP ->nombre. En este ejemplo utilizaremos un servidor de DNS atiende a una red interna (192.168.0.0/24) y luego tiene una IP pública pero a a fines del ejemplo y poder hacer las pruebas internas la configuraremos en un IP privada de otro segmento (172.16.0.80) y un dominio ficticio (remix.world) pero en el caso de llevar este servidor a producción se debería cambiar por la IP externa que tenga y el dominio correspondiente que se haga obtenido y tengamos a nuestra disposición.

Para configurar un servidor de nombres propio en primer lugar cambiaremos el archivo */etc/bind/named.conf* para definir dos zonas (una interna y otra ex-

terna). El archivo quedará como (observad que hemos comentado la línea que incluye named.conf.default-zones ya que la incluiremos después):

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
//include "/etc/bind/named.conf.default-zones";
include "/etc/bind/named.conf.internal-zones";
include "/etc/bind/named.conf.external-zones";
```

Modificamos el archivo *named.conf.internal-zones* para indicarle los archivos donde se encontrarán realmente los nombres de las máquinas que en este caso será dos: uno para para resolución nombre ->IP llamado remix.world.lan y otro para la resolución inversa IP ->nombre llamado 0.168.192.db.

```
view "internal" {
    match-clients {
        localhost;
        192.168.0.0/24;
    };
// set zone for internal
    zone "remix.world" {
        type master;
        file "/etc/bind/remix.world.lan";
        allow-update { none; };
    };
// set zone for internal reverse
    zone "0.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/0.168.192.db";
        allow-update { none; };
    };
include "/etc/bind/named.conf.default-zones";
};
```

De la misma forma modificamos el archivo *named.conf.external-zones* para indicarle los archivos que resolverán esta zona y que también serán dos: remix.world.wan y 80.0.16.172.db para la resolución directa e inversa respectivamente.

```
view "external" {
// define for external section
    match-clients { any; };
// allow any query
    allow-query { any; };
// prohibit recursion
    recursion no;
// set zone for external
    zone "remix.world" {
        type master;
        file "/etc/bind/remix.world.wan";
        allow-update { none; };
    };
// set zone for external reverse
    zone "80.0.16.172.in-addr.arpa" {
        type master;
        file "/etc/bind/80.0.16.172.db";
        allow-update { none; };
    };
};
```

También se debe modificar el *named.conf.options*:

```
options {
    directory "/var/cache/bind";
// ...
// forwarders {
// 158.109.0.9;
// 158.109.0.1;
// };
// query range you permit
    allow-query { localhost; 192.168.0.0/24; };
// the range to transfer zone files
    allow-transfer { localhost; 192.168.0.0/24; };
// recursion range you allow
// allow-recursion { localhost; 192.168.0.0/24; };


    dnssec-validation auto;
    auth-nxdomain no; # conform to RFC1035
//listen-on-v6 { any; };
};
```

Ahora es necesario definir los archivos que realmente resolverán las IP/nombres de las zonas. Comenzamos por *remix.server.lan*:

```
$TTL 86400
@ IN SOA nteum.remix.world. root.remix.world. (
    2013050601 ;Serial
    3600 ;Refresh
    1800 ;Retry
    604800 ;Expire
    86400 ;Minimum TTL
)

    IN NS nteum.remix.world.
    IN A 192.168.0.30
    IN MX 10 nteum.remix.world.

nteum IN A 192.168.0.30
```

Se debe tener en cuenta el "." al final de los nombres de dominio. El significado de los registros indican: **SOA** (*Start of Authority*) debe estar en todos los archivos de zona al inicio, después de `TTL` (*Time to Live*) que indica el tiempo en segundos que los recursos de una zona serán válidos, el símbolo `@` significa el origen del dominio; **NS**, el servidor de nombres para el dominio, **MX** (*Mail eXchange*), indica a donde se debe dirigir el correo para una determinada zona, y **A** es la Ip que se debe asignar a un nombre. Para el archivo de resolución inversa *0.168.192.db*:

```
$TTL 86400
@ IN SOA nteum.remix.world. nteum.remix.world. (
    2013050601 ;Serial
    3600 ;Refresh
    1800 ;Retry
    604800 ;Expire
    86400 ;Minimum TTL
)

    IN NS nteum.remix.world.
    IN PTR remix.world.
    IN A 255.255.255.0
30 IN PTR nteum.remix.world.
```

Donde podemos observar que los registros tienen un formato como el siguiente: `<último-dígito-IP>IN PTR <FQDN-of-system>` donde PTR (*Registro PoinTeR*) crea un apuntador hacia el nombre de la IP que coincide con el registro. De la misma forma procedemos para la zona externa con el archivo *remix.world.wan*:

```
$TTL 86400
@ IN SOA nteum.remix.world. root.remix.world. (
    2013050601 ;Serial
    3600 ;Refresh
    1800 ;Retry
    604800 ;Expire
    86400 ;Minimum TTL
)

    IN NS nteum.remix.world.
    IN A 172.16.0.82
    IN MX 10 nteum.remix.world.
nteum IN A 172.16.0.82
```

Y su resolución inversa *80.0.16.172.db*:

```
$TTL 86400
@ IN SOA nteum.remix.world. nteum.remix.world. (
    2013050601 ;Serial
    3600 ;Refresh
    1800 ;Retry
    604800 ;Expire
    86400 ;Minimum TTL
)

    IN NS nteum.remix.world.
    IN PTR remix.world.
    IN A 255.255.255.248
82 IN PTR nteum.remix.world.
```

Ahora solo restaría cambiar el archivo */etc/resolv.conf* para incluir `nameserver 192.168.0.30` y reiniciar el servidor `service bind9 restart`. Se debe mirar */var/log/syslog* para observar si hay errores en el arranque del servidor y corregirlos hasta que su reinicio esté libre de estos (los más comunes son los símbolos utilizados como comentarios, los puntos al final de dominio o espacios antes de los registros A-PTR). Posteriormente podemos probar con `dig nteum.remix.world` y tendremos una salida como:

```
; «» DiG 9.8.4-rpz2+rl005.12-P1 «» nteum.remix.world
...
;; QUESTION SECTION:
;nteum.remix.world. IN A
;; ANSWER SECTION:
nteum.remix.world. 86400 IN A 192.168.0.30
;; AUTHORITY SECTION:
remix.world. 86400 IN NS nteum.remix.world.
;; Query time: 0 msec
...
```

Si hacemos la petición inversa con `dig -x 192.168.0.30` tendremos:

```
...
;; QUESTION SECTION:
;30.0.168.192.in-addr.arpa. IN PTR
;; ANSWER SECTION:
30.0.168.192.in-addr.arpa. 86400 IN PTR nteum.remix.world.
;; AUTHORITY SECTION:
0.168.192.in-addr.arpa. 86400 IN NS nteum.remix.world.
;; ADDITIONAL SECTION:
nteum.remix.world. 86400 IN A 192.168.0.30
...
```

Un aspecto interesante es poner alias lo cual significa incluir un registro CNAME en *remix.world.lan* como `ftp IN CNAME nteum.remix.world.` Dado que es un servicio esencial es necesario muchas veces disponer de un servicio secundario (*slave*) y bind9 permite crear muy fácilmente servicios de este tipo a partir del servidor primario por transferencia de las tablas (http://www.server-world.info/en/note?os=Debian_7.0&p=dns&f=5).

1.1.3. Otros DNS

MaraDNS es un servidor DNS que puede funcionar como DNS recursivo (caché) o como *authoritative name server* y que se caracteríza por su extremadamente fácil configuración. MaraDNS ha sido optimizado para un pequeño número de dominios en forma simple y eficiente y que permite crear un dominio propio y servirlo sin grandes esfuerzos aunque contiene todos los elementos para configurarlo en forma similar a bind9. En su diseño se ha puesto especial cuidado a la seguridad y utiliza una librería de gestión de cadenas de caracteres que resisten a los principales ataques a los DNS por desbordamiento (*buffer overflows*)[6]. EL paquete MaraDNS incluye un DNS autorizado (`maradns`), un servidor DNS recursivo con posibilidades de caché y que se llama Deadwood. La decisión de poner un servidor autorizado o recursivo dependerá de la función que se busque. Si simplemente es para obtener otros sitios en Internet se deberá configurar un servidor recursivo, en cambio se se desea registrar dominios y se tienen una red de ordenadores dentro de este dominio hay que configurar un servidor DNS con autoridad.

Como pruebas de este servicio configuraremos diferentes opciones de MaraDNS para Debian. Si bien el paquete está incluido en el repositorio* es una versión que tiene problemas de seguridad y su autor recomienda bajar el código fuente y compilarlo/instalarlo. Para ello debemos obtener el archivo desde http://maradns.samiam.org/download.html, descomprimirlo (`tar xjvf maradns-x.x.xx.tar.bz2` donde x.x.xx es la versión del software descargado) y dentro de directorio ejecutar `./configure; make` y si no hay errores ejecutar `make install`. Los servidores se instalarán en */usr/local/sbin* y otros archivos complementarios en */usr/local/bin*. Los archivos de configuración en */etc/mararc y /etc/dwood3rc* y el directorio de configuración */etc/maradns*.

***http://maradns.samiam.org/debian.html**

En primer lugar para configurar un DNS recursivo y caché modificar el archivo */etc/dwood3rc* para incluir:

```
bind_address="127.0.0.1"
chroot_dir = "/etc/maradns"
#upstream_servers = {}
#upstream_servers["."]="8.8.8.8, 8.8.4.4"
recursive_acl = "127.0.0.1/16" # Quien puede utilizar la caché

# Parámetros de configuración del servidor
maxprocs = 8 # Maximum number of pending requests
handle_overload = 1 # Send SERVER FAIL when overloaded
maradns_uid = 99 # UID Deadwood runs as
maradns_gid = 99 # GID Deadwood runs as
maximum_cache_elements = 60000

# File cache (readable and writable by the maradns_uid/gid)
cache_file = "dw_cache"
```

También es posible que Deadwood pueda contactar con otros DNS recursivos antes que con los root DNS servers. Para ello quitar el comentario de las líneas upstream_servers (en este caso se han puestos los dns públicos de Google). Luego podremos ejecutar `/usr/local/sbin/Deadwood` o si se quiere ejecutar como *daemon* se deberá utilizar el programa `duende` incluido en el paquete: `/usr/local/bin/duende /usr/local/sbin/Deadwood`.

Para configurar un DNS autorizado debemos modificar el archivo */etc/mararc* con los siguiente valores: cvs2 el dominio que gestionará este servidor y el archivo que tendrá la información en este caso db.local, el directorio donde se encontrará los archivos de configuración (/etc/maradns) y la IP de donde responderá.

```
csv2 = {}
csv2["remix.world."] = "db.local"
ipv4_bind_addresses = "127.0.0.1"
chroot_dir = "/etc/maradns"
```

El archivo /etc/maradns/db.local simplemente tendrá una línea por cada registro que deseamos que transforme el dns por ejemplo, `nteum.remix.world. 192.168.0.30 ~`. Luego se debe poner en marcha el servidor* o si se desea ejecutar como *daemon* habrá que usar el programa `/usr/local/bin/duende /usr/local/sbin/maradns`. Una de las cuestiones interesantes es cómo puedo dar servicio de forma *authoritative* para mis dominios en una Intranet pero además ser recursivo cuando la petición sea a dominios externos? Es decir tengo unos pocos nombres (FQDN) que pertenecen a mi dominio (por ejemplo remix.world) sobre mi Intranet y deseo al mismo tiempo asegurar la resolución de dominio FQDN externos (por ejemplo debian.org). La solución planteada por el autor (S. Trenholme**) pasa por una configuración combinada de MaraDNS 2.x y Deadwood 3.x poniendo MaraDNS que escuche sobre localhost (127.0.0.1) mientras Deadwood sobre la IP del servidor (192.168.1.37 en nuestro caso). Los archivos de configuración */etc/mararc* y */etc/dwood3rc* serán:

***/usr/local/sbin/maradns**

****http://woodlane.webconquest.com/pipermail/list/2011-August/000928.html**

```
# Archivo /etc/mararc
# Simplemente indicar que escuche sobre localhost y el dominio que será responsable
ipv4_bind_addresses = "127.0.0.1"
timestamp_type = 2
verbose_level = 3
hide_disclaimer = "YES"
csv2 = {}
csv2["remix.world."] = "db.local"
chroot_dir = "/etc/maradns"

#Archivo /etc/dwood3rc
# Indicarle que además de nuestro dominio donde queremos consultar el resto
root_servers = {}
root_servers["remix.world."] = "127.0.0.1"
root_servers["."]="198.41.0.4, 192.228.79.201, 192.33.4.12, 199.7.91.13,"
root_servers["."]+="192.203.230.10, 192.5.5.241, 192.112.36.4, 128.63.2.53, "
root_servers["."]+="192.36.148.17, 192.58.128.30, 193.0.14.129, 199.7.83.42, "
root_servers["."]+="202.12.27.33"
# Donde escuchará la peticiones = IP del servidor Deadwood
bind_address="192.168.1.37"
# Habilitar la respuesta de IP privadas
filter_rfc1918=0
# IPs que podrán hacer peticiones
recursive_acl = "192.168.1.0/24"
# Caché de disco para Deadwood
cache_file = "dw_cache"
# Directorio raíz
chroot_dir = "/etc/maradns"
```

Finalmente en nuestras máquinas poner como */etc/resolv.conf* la IP de nuestro servidor `nameserver 192.168.1.37` y poner en marcha ambos servidores (luego de verificar que funciona se pueden poner como *daemons* y bajar el nivel de log con `verbose_level = 1`.

Otro servidor interesante que viene en muchas distribuciones es **Dnsmasq** que permite en forma simple configurar un servidor DNS (*forwarder*) y un servidor DHCP en el mismo paquete para redes pequeñas/medianas. este puede atender peticiones de nombres de máquinas que no están en los DNS globales e integra el servidor de DHCP para permitir que máquinas gestionadas por el DHCP aparezcan en el DNS con nombres ya configurados y además soporta gestión de IP dinámicas y estáticas para el DHCP y permite el acceso BOOTP/TFTP para máquinas sin disco y que hacen su inicialización por red.

La forma de configurar rápidamente un DNS para máquinas de dominio propio y que haga de *forwarder* para los dominios externos es `apt-get update; apt-get install dnsmasq`. Si lo que se desea es un simple DNS ya está configurado teniendo en cuenta que en */etc/resolv.conf* tendremos alguna cosa como `nameserver 8.8.8.8`. Con esto podemos probar los externos con `dig debian.org @localhost` (o simplemente `dig debian.org`) pero también responderá a todas las máquinas que tengamos definidas en */etc/hosts* como por ejemplo si tenemos una línea como `192.168.1.37 nteum.remix.world nteum` podremos ejecutar `dig nteum @localhost` y nos responderá con la IP correspondiente. Es por ello que si la red es simple podemos agregar las máquinas en este archivo pero si es más compleja podemos utilizar el archivo de configuración */etc/dnsmasq.conf* que incluye una gran cantidad de opciones para organizar los dominios internos y otros parámetros no solo para la configuración del DNS sino también para el servidor de DHCP.[7]

1.2. NIS (YP)

Con el fin de facilitar la administración y dar comodidad al usuario en redes de diferentes tamaños que ejecutan GNU/Linux (o algún otro sistema operativo con soporte para este servicio), se ejecutan servicios de *Network Information Service*, NIS (o *Yellow Pages*, YP, en la definición original de Sun). GNU/Linux puede dar apoyo como cliente/servidor de NIS. La información que se puede distribuir en NIS es: usuarios (*login names*), contraseñas (*passwords*, `/etc/passwd`), directorios de usuario (*home directories*) e información de grupos (*group information*, `/etc/group`), redes, hosts, etc., lo cual presenta la ventaja de que, desde cualquier máquina cliente o desde el mismo servidor, el usuario se podrá conectar con la misma cuenta y contraseña y al mismo directorio (aunque el directorio deberá ser montado anteriormente sobre todas las máquinas cliente por NFS o mediante el servicio de `automount`). [10, 11]

La arquitectura NIS es del tipo cliente-servidor, es decir, existe un servidor que dispondrá de todas las bases de datos y unos clientes que consultarán estos datos en forma transparente para el usuario. Por ello, se debe pensar en la posibilidad de configurar servidores "de refuerzo" (llamados secundarios) para que los usuarios no queden bloqueados ante la caída del servidor principal. Es por ello que la arquitectura se denomina realmente de múltiples servidores (*master+mirrors-clients*).

1.2.1. ¿Cómo iniciar un cliente local de NIS en Debian?

Un cliente local es el que anexa el ordenador a un dominio NIS ya existente: primero se debe verificar que se tienen instalados los paquetes `netbase` (red básica TCP/IP) y `rpcbind` (servidor que convierte números RPC –*Remote Procedure Call*– en puertos DARPA) y `nis` (específico). Es importante destacar que el paquete `rpcbind`, que ya está en la mayoría de las distribuciones, es un servicio que sustituye al tradicional `portmap`. Este ha quedado obsoleto por diferentes causas vinculadas a los cambios en el kernel de NFSV3/V4, pero especialmente, al no tener soporte para IPv6. Es necesario verificar la instalación de los dos primeros paquetes para todos los programas que ejecutan RPC, incluyendo NFS y NIS. Se recomienda usar el comando `apt-get` (o también `dpkg, aptitude, synaptic-pkexec`) y se puede verificar si está instalado con `apt-cache pkgnames` en modo texto.

El procedimiento de instalación del paquete NIS solicitará un dominio (`NIS domainname`). Este es un nombre que describirá el conjunto de máquinas que utilizarán el NIS (no es un nombre de *host*). Hay que tener en cuenta que `NISNteum` es diferente de `Nisnteum` como nombre de dominio. Para configurarlo, se puede utilizar el comando `nisdomainname`, dominio que se almacenará en `/proc/sys/kernel/domainname`. También se puede reconfigugar el paquete con `dpkg-reconfigure nis` y volveremos a comenzar, po-

demos ver que el rpcbind está en marcha con `ps -edaf | grep rpcbind` o también con `rpcinfo -p` y nos mostrará diferentes líneas -para diferentes programas, puertos/protocolos, versiones- como portmapper. Para reiniciar el rpcbind, podemos hacer `service rpcbind restart`. Es importante tener en cuenta que la primera vez que instalamos en NIS, que será el cliente en nuestro caso, el sistema de instalación intentará poner los daemons en marcha pero fallará, ya que no tenemos ningún servidor configurado (que será el siguiente paso) y dará un mensaje de *[....] Starting NIS services: ypbind[....] binding to YP server...failed (backgrounded)*.

El cliente NIS usa el comando `ypbind` para encontrar un servidor para el dominio especificado, ya sea mediante `broadcast` (no aconsejado por inseguro) o buscando el servidor indicado en el archivo de configuración `/etc/yp.conf` (recomendable) que puede tener básicamente dos tipos de configuración como las mostradas abajo -solo se debe indicar una de ellas aunque es recomendable la primera-.

Sintaxis del archivo `/etc/yp.conf`

`domain nisdomain server hostname`: indica que se utiliza el *hostname* para el dominio `nisdomain`. En nuestro caso podría ser `domain NISnteum server 192.168.1.30` Se podría tener más de una entrada de este tipo para un único dominio.

`ypserver hostname`: indica que se utiliza *hostname* como servidor introduciendo la dirección IP del servidor NIS. Si se indica el nombre, asegúrese de que se puede encontrar la IP por DNS o que la misma figura en el archivo `/etc/hosts`, ya que, de otro modo, el cliente se bloqueará.

Para iniciar el servicio se debe ejecutar:

`/etc/init.d/nis stop` *para detenerlo*
`/etc/init.d/nis start` *para iniciarlo*
o también con el comando `service nis start|stop`

A partir de este momento, el cliente NIS estará funcionando. Ello se puede confirmar con `rpcinfo -u localhost ypbind`, que mostrará las dos versiones del protocolo activo o, cuando el servidor esté funcionando, se puede utilizar el comando `ypcat mapname` (por ejemplo, `ypcat passwd`, que mostrará los usuarios NIS definidos en el servidor) donde las relaciones entre *mapnames* y las tablas de la base de datos NIS están definidas en `/var/yp/nicknames`.

1.2.2. ¿Qué recursos se deben especificar para utilizar el NIS?

A partir de la inclusión de lib6 en las distribuciones de GNULinux (esto es Debian5 o FC4) es necesario orientar la consulta del *login* a las bases de datos adecuadas con los pasos siguientes:

1) Verificar el fichero `/etc/nsswitch.conf` y asegurarse de que las entradas `passwd`, `group`, `shadow` y `netgroup` son similares a:

```
passwd: compat nis
group: compat nis
shadow: compat nis
netgroup: nis
hosts: files dns nis
```

2) Consultad la sintaxis de este archivo en `man nsswitch.conf`.

3) En algunas configuraciones puede ser necesario agregar al final del fichero *./etc/pam.d/common-session* (si no existe ya la línea) `session optional pam_mkhomedir.so skel=/etc/skel umask=077` para crear de forma automática el directorio home si no existe pero no es lo habitual ya que generalmente se desea que este se monte del servidor NFS.

4) Con esta configuración se podrá realizar una conexión local (sobre el cliente NIS) a un usuario que no esté definido en el fichero `/etc/passwd`, es decir, un usuario definido en otra máquina (*ypserver*). Por ejemplo, se podría hacer `ssh -l user localhost`, donde `user` es un usuario definido en *ypserver*.

1.2.3. ¿Cómo se debe configurar un servidor?

Antes de instalar el servidor, hay que tener instalado el paquete nis y rpcbind sobre la máquina que hará de servidor (`apt-get install nis rpcbind`) y configurar el dominio -el mismo que hemos introducido cuando configuramos el cliente (NISnteum en nuestro caso). Luego se debe modificar el archivo *./etc/default/nis* para modificar la línea `NISSERVER=master` para indicarle el rol de servidor. También (si bien se puede hacer en un segundo paso) se debe ajustar el archivo *./etc/ypserv.securenets* para modificar la línea que indica `0.0.0.0 0.0.0.0` que da acceso a todo el mundo y restringirlo a nuestra red, por ejemplo `255.255.255.0 192.168.1.0`. Por último, se debe modificar el *./etc/hosts* para que tengamos la máquina con el nombre definido en *./etc/hostname* (se puede cambiar/visualizar con el comando `hostname` o editando el fichero) con una línea FQND como por ejemplo `192.168.1.30 nteum.remix.world nteum`. La configuración del servidor se realiza con el comando `/usr/lib/yp/ypinit -m`; en algunas distribuciones es necesario verificar que existe el archivo `/etc/networks`, que es imprescindible para este *script*. Si este archivo no existe, cread uno vacío con `touch/etc/networks`; este *script* nos solicitará cuáles serán los servidores NIS indicándonos la máquina local por defecto y deberemos terminar con Crtl+D. También se puede ejecutar el cliente `ypbind` sobre el servidor; así, todos los usuarios entran por NIS indicando en el archivo *./etc/default/nis* que exista `NISCLIENT=true`.

Considerad que a partir de este momento los comandos para cambiar la contraseña o la información de los usuarios, como `passwd`, `chfn` o `adduser`, no son válidos. En su lugar, se deberán utilizar comandos tales como `yppasswd`, `ypchsh` y `ypchfn`. Si se cambian los usuarios o se modifican los archivos men-

cionados, habrá que reconstruir las tablas de NIS ejecutando el comando `make` en el directorio `/var/yp` para actualizar las tablas.

La configuración de un servidor esclavo es similar a la del maestro, excepto si `NISSERVER = slave` en `/etc/default/nis`. Sobre el maestro se debe indicar que distribuya las tablas automáticamente a los esclavos, poniendo `NOPUSH = "false"` en el archivo `/var/yp/Makefile`. Ahora se debe indicar al maestro quién es su esclavo por medio de la ejecución de:

```
/usr/lib/yp/ypinit -m
```

e introduciendo los nombres de los servidores esclavos. Esto reconstruirá los mapas pero no enviará los archivos a los esclavos. Para ello, sobre el esclavo, ejecutad:

```
/etc/init.d/nis stop
/etc/init.d/nis start
/usr/lib/yp/ypinit -s nombre_master_server
```

Así, el esclavo cargará las tablas desde el maestro. También se podría poner en el directorio `/etc/cron.d` el archivo NIS con un contenido similar a (recordad hacer un `chmod 755 /etc/cron.d/nis`):

```
20 * * * * root /usr/lib/yp/ypxfr_1perhour >/dev/null 2>&1
40 6 * * * root /usr/lib/yp/ypxfr_1perday >/dev/null 2>&1
55 6,18 * * * root /usr/lib/yp/ypxfr_2perday >/dev/null 2>&1
```

Con ello, se garantizará que todos los cambios del maestro se transfieran al servidor NIS esclavo.

Iniciad el servidor ejecutando primero el comando `/etc/init.d/nis stop` y luego `/etc/init.d/nis start`. Esta sentencia iniciará el servidor (`ypserv`) y el *password daemon* (`yppasswdd`), cuya activación se podrá consultar con `ypwich -d domain`.

Actualización de las tablas NIS

Es recomendable que después de usar `adduser o useradd` para agregar un nuevo usuario sobre el servidor, ejecutéis `cd /var/yp; make` para actualizar las tablas NIS (y siempre que se cambie alguna característica del usuario, por ejemplo la palabra clave con el comando `passwd`, solo cambiará la contraseña local y no la de NIS).

Para probar que el sistema funciona y que el usuario dado de alta está en el NIS, podéis hacer `ypmatch userid passwd`, donde `userid` es el usuario dado de alta con `adduser` antes y después de haber hecho el `make`. Para verificar el funcionamiento del sistema NIS podéis usar el *script* de http://tldp.org/HOWTO/NIS-HOWTO/verification.html que permite una verificación más detallada del NIS.

También se puede probar en la misma máquina que un usuario se puede conectar a un domino NIS haciendo: `adduser usuario` y respondiendo a las preguntas, luego `cd /var/yp; make`. En este momento lo veremos tanto en */etc/passwd* como en NIS con `ypcat passwd`. Luego lo podemos borrar con el comando `userdel usuario` (no utilizar el comando `deluser` sino el `userdel`) que solo lo borrará del */etc/passwd* y no del NIS (hay que tener cuidado de no hacer un `make`, ya que este también lo borrará del NIS). Luego nos podemos conectar como `ssh usuario@localhost` y con ello podremos verificar que el NIS funciona, ya que el acceso solo se podrá hacer por NIS porque por */etc/passwd* no existe este usuario. Con relación a NIS+ su desarrollo fue parado en el 2012 dada la falta de interés/recursos de la comunidad, no así NIS/YP, que sigue actualizado y presente en todas las distribuciones (http://www.linux-nis.org/nisplus/)

1.3. Servicios de conexión remota: telnet y ssh

1.3.1. Telnet y telnetd

Telnet es un comando (cliente) utilizado para comunicarse interactivamente con otro *host* que ejecuta el *daemon* `telnetd`. El comando `telnet` se puede ejecutar como `telnet host` o interactivamente como `telnet`, el cual pondrá el *prompt* "telnet>" y luego, por ejemplo, `open host`. Una vez establecida la comunicación, se deberá introducir el usuario y la contraseña bajo la cual se desea conectar al sistema remoto. Se dispone de diversos comandos (en modo interactivo) como `open`, `logout` y `mode` (se deben definir las características de visualización), `close`, `encrypt`, `quit`, `set` y `unset` o se pueden ejecutar comando externos con "!". Se puede utilizar el archivo `/etc/telnetrc` para definiciones por defecto o `.telnetrc` para definiciones de un usuario particular (deberá estar en el directorio *home* del usuario).

El *daemon* `telnetd` es el servidor de protocolo telnet para la conexión interactiva. Generalmente, es el *daemon* `inetd` que pone en marcha `telned`; se recomienda incluir un `wrapper tcpd` (que utiliza las reglas de acceso en `host.allow` y `host.deny`) en la llamada al `telnetd` dentro del archivo `/etc/inetd.conf`. Para incrementar la seguridad del sistema se debería, por ejemplo, incluir una línea como:

```
telnet stream tcp nowait telnetd.telenetd /usr/sbin/tcpd /usr/bin/in.telnetd
```

En las distribuciones actuales, la funcionalidad de `inetd` se ha reemplazado por la de `xinetd`, el cual requiere configurar el archivo `/etc/xinetd.conf` que se ha explicado en el apartado de configuración de red de la asignatura *Administración GNU/Linux*. Si se quiere poner en marcha `inetd` a modo de pruebas, se puede usar la sentencia `/etc/init.d/inetd.real start`. Si el archivo `/etc/uissue.net` está presente, el `telnetd` mostrará su contenido al inicio de la sesión. También se puede usar `/etc/security/access.conf` para habilitar y deshabilitar *logins* de usuario, *host* o grupos de usuarios, según se conecten.

Se debe recordar que, si bien el par `telnet-telnetd` puede funcionar en modo *encrypt* en las últimas versiones (transferencia de datos encriptados, que

deben estar compilados con la opción correspondiente), es un comando que ha quedado en el olvido por su falta de seguridad (transmite el texto en claro sobre la red, lo que permite la visualización del contenido de la comunicación desde otra máquina, por ejemplo, con el comando `tcpdump`); no obstante, puede utilizarse en redes seguras o situaciones controladas.

Si no está instalado, se puede utilizar (Debian) `apt-get install telnetd` y después verificar que se ha dado de alta, bien en `/etc/inetd.conf`, bien en `/etc/xinetd.conf` (o en el directorio en que estén definidos los archivos; por ejemplo, `/etc/xinetd.d` según se indique en el archivo anterior con la sentencia `include /etc/xinetd.d`). El `xinetd.conf` o el archivo `/etc/xinetd.d/telnetd` deberán incluir una sección como*:

***Cualquier modificación en `xinetd.conf` deberá arrancar nuevamente el servicio con `service xinetd restart`.**

```
service telnet {
disable = no
flags = REUSE
socket_type = stream
wait = no
user = root
server = /usr/sbin/in.telnetd
log_on_failure += USERID
}
```

SSL telnet(d)

Se recomienda que en lugar de usar `telnetd` se utilice `SSL telnet(d)`, que reemplaza a `telnet(d)` utilizando encriptación y autentificación por SSL, o que se utilice SSH (el cual ya es un estándar en todas las distribuciones y entornos). El `SSLTelnet(d)` puede funcionar con el `telnet(d)` normal en ambas direcciones ya que al inicio de la comunicación verifica si el otro lado (*peer*) soporta SSL, de lo contrario, continúa con el protocolo telnet normal. Las ventajas con respecto al `telnet(d)` son que sus contraseñas y datos no circularán por la red en modo de texto plano y nadie que utilice el comando antes mencionado (`tcpdump`) podrá ver el contenido de la comunicación. También `SSL telnet` se puede utilizar para conectarse, por ejemplo, a un servidor web seguro (por ejemplo `https://servidor.web.org`) simplemente haciendo `telnet servidor.web.org 443`.

1.3.2. SSH, *Secure shell*

Un cambio aconsejable hoy en día es utilizar `ssh` (comando que ya está en todos los sistemas operativos incluido en Windows con `putty`* o `moabterm`**) en lugar de `telnet`, `rlogin` o `rsh`. Estos tres últimos comandos son inseguros (excepto `SSLTelnet`) por varias razones: la más importante es que todo lo que se transmite por la red, incluidos nombres de usuarios y contraseñas, es en texto plano (aunque existen versiones de telnet-telnetd encriptados, debe coincidir que ambos los sean), de modo que cualquiera que tenga acceso a esa red o a algún segmento de la misma puede obtener toda esta información y luego suplantar la identidad del usuario. La segunda es que estos puertos (telnet, rsh, etc.) son el primer lugar donde un *cracker* intentará conectarse. El protocolo `ssh` (en su versión OpenSSH) provee una conexión encriptada y comprimida mucho más segura que, por ejemplo, `telnet` (es recomendable utilizar la versión 2.0 o versiones superiores del protocolo). Todas las distribu-

***http://www.putty.org**
****http://mobaxterm.mobatek.net**

ciones actuales incorporan el cliente `ssh` y el servidor `sshd` por defecto. Es necesario tener actualizada la librería OpenSSL, utilizada por estos programas, ya que recientemente se encontró un problema de seguridad llamado *heartbleed* y que ha sido rápidamente solucionado en la mayoría de las distribuciones GNU/Linux (http://www.kriptopolis.com/recomendaciones-heartbleed).

ssh

Para ejecutar el comando, haced:

```
ssh -l login name host o ssh user@hostname
```

A través de SSH se pueden encapsular otras conexiones como X11 o cualquier otra TCP/IP. Si se omite el parámetro `-l`, el usuario se conectará con el mismo usuario local y en ambos casos el servidor solicitará la contraseña para validar la identidad del usuario. SSH soporta diferentes modos de autentificación (ved `man ssh`) basados en algoritmo RSA y clave pública. Si el cliente no está instalado, hacer `apt-get install openssh-client`.

Usando el comando `ssh-keygen -t rsa|dsa`, se pueden crear las claves de identificación de usuario. El comando crea en el directorio del `.shh` del usuario los ficheros* `id_rsa` y `id_rsa.pub`, las claves privada y pública, respectivamente. El usuario podría copiar la pública (`id_rsa.pub`) en el archivo `$HOME/.ssh/authorized_keys` del directorio `.ssh` del usuario de la máquina remota. Este archivo podrá contener tantas claves públicas como sitios desde donde se quiera conectar a esta máquina de forma remota. La sintaxis es de una clave por línea aunque las líneas tendrán un tamaño considerable. Después de haber introducido las claves públicas del usuario-máquina en este archivo, este usuario se podrá conectar sin contraseña desde esa máquina. También para realizar esta copia se puede utilizar el comando `ssh-copy-id máquina` que copiará las llaves en forma automática y es mucho más simple de utilizar.

***Por ejemplo, para el algoritmo de encriptación RSA.**

En forma normal (si no se han creado las claves), se le preguntará al usuario una contraseña, pero como la comunicación será siempre encriptada, nunca será accesible a otros usuarios que puedan escuchar sobre la red. Para mayor información, consultad `man ssh`. Para ejecutar remotamente un comando, simplemente haced:

```
ssh -l login name host_comando_remoto
Por ejemplo: ssh -l user localhost ls -al
```

sshd

El `sshd` es el servidor (*daemon*) para el `ssh` (si no están instalados, se puede hacer con `apt-get install openssh-client openssh-server`. Juntos reemplazan a `rlogin`, `telnet`, `rsh` y proveen una comunicación segura y encriptada entre dos *hosts* inseguros de la red. Este se arranca generalmente a través de los archivos de inicialización (`/etc/init.d` o `/etc/rc`) y espera conexiones de los clientes. El `sshd` de la mayoría de las distribuciones actuales soporta las versiones 1, 2 y 3 del protocolo SSH. Cuando se instala el paquete, se crea una clave RSA específica del *host* y cuando el *daemon* se inicia, crea otra, la RSA para la sesión, que no se almacena en el disco y que cambia cada hora. Cuando un cliente inicia la comunicación, genera un número aleatorio de 256 bits que está encriptado con las dos claves del servidor y enviado. Este número se utilizará durante la comunicación como clave de sesión para encriptar la comunicación que se realizará a través de un algoritmo de encriptación estándar. El usuario puede seleccionar cualquiera de los algoritmos disponibles ofrecidos por el servidor. Existen algunas diferencias (más seguro) cuando se utiliza la versión 2 (o 3) del protocolo. A partir de este momento, se inician algunos de los métodos de autentificación de usuario descritos en el cliente o se le solicita la contraseña, pero siempre con la comunicación encriptada. Para mayor información, consultad `man sshd`.

Tanto ssh como sshd disponen de un gran conjunto de parámetros que pueden ser configurados según se necesite en los archivos `/etc/ssh/ssh_config` y `/etc/ssh/sshd_config`. En el servidor probablemente algunas de las opciones más utilizadas son `PermitRootLogin yes|no` para permitir la conexión del usuario root o no, `IgnoreRhosts yes` para evitar leer los archivos `$HOME/.rhosts` y `$HOME/.shosts` de los usuarios, `X11Forwarding yes` para permitir que aplicaciones Xwindows en el servidor se visualicen en la pantalla del cliente (muy útil en la administración remota de servidores con el comando `ssh -X host_a_administrar`).

Túnel sobre SSH

Muchas veces tenemos acceso a un servidor `sshd` pero por cuestiones de seguridad no podemos acceder a otros servicios que no están encriptados (por ejemplo, un servicio de consulta de mail POP3 o un servidor de ventanas X11) o simplemente se quiere conectar a un servicio al cual solo se tiene acceso desde el entorno de la empresa. Para ello, es posible establecer un túnel encriptado entre la máquina cliente (por ejemplo, con Windows y un cliente ssh llamado `putty` de software libre) y el servidor con `sshd`. En este caso, al vincular el túnel con el servicio, el servicio verá la petición como si viniera de la misma máquina. Por ejemplo, si queremos establecer una conexión para POP3 sobre el puerto 110 de la máquina remota (y que también tiene un servidor `sshd`) haremos:

```
ssh -C -L 1100:localhost:110 usuario-id@host
```

Este comando pedirá la contraseña para el usuario-id sobre *host* y una vez conectado, se habrá creado el túnel. Cada paquete que se envíe desde la máquina local sobre el puerto 1100 se enviará a la máquina remota *localhost* sobre el puerto 110, que es donde escucha el servicio POP3 (la opción `-C` comprime el tráfico por el túnel).

Hacer túneles sobre otros puertos es muy fácil. Por ejemplo, supongamos que *solo* tenemos acceso a un *remote proxy server* desde una máquina remota (*remote login*), no desde la máquina local; en este caso, se puede hacer un túnel para conectar el navegador a la máquina local. Consideremos que tenemos *login* sobre una máquina pasarela (*gateway*), que puede acceder a la máquina llamada *proxy*, la cual ejecuta el *Squid Proxy Server* sobre el puerto 3128. Ejecutamos:

```
ssh -C -L 8080:proxy:3128 user@gateway
```

Después de conectarnos, tendremos un túnel escuchando sobre el puerto local 8080 que reconducirá el tráfico desde *gateway* hacia *proxy* al 3128. Para navegar de forma segura solo se deberá hacer `http://localhost:8080/`.

1.3.3. VPN SSL (via tun driver)

OpenSSH v4.3 introduce una nueva característica que permite crear *on-the-fly* VPN vía el túnel driver (tun) como un puente entre dos interfaces en diferentes segmentos de red a través de Internet y por lo cual un ordenador (B en nuestro caso) "quedará dentro" de la red del otro ordenador (A) a pesar de estar dentro de otra red. Para analizar este mecanismo, consideraremos que el ordenador A y B están conectados a la red vía ethernet y a su vez a Internet a través de una *gateway* que hace NAT. A tiene IP sobre su red (privada) 10.0.0.100, y B sobre su red (privada) 192.168.0.100 y como dijimos cada uno tiene acceso a internet NAT *gateway*.

A través de OpenSSH conectaremos B a la red de A vía una interfaz túnel ssh. Como hemos definido A ya tiene IP sobre la red A (10.0.0.100) y que se ve desde el gateway externo como dirección 1.2.3.4, es decir esta es la IP pública del servidor `ssh` de A (el router de A deberá hacer un forwarding de 1.2.3.4:22 a 10.0.0.100:22 para que externamente se pueda contactar con servidor `ssh` de A). B deberá tener una IP sobre la red A que será su interfaz tun0 y que le asignaremos 10.0.0.200. B debe también debe tener acceso al servidor `ssh` de A (o bien acceso directo o el puerto 22 debe ser forwarded sobre la red A en la dirección 1.2.3.4). Cuando el túnel esté creado B accederá directamente a la red de A lo cual significa que B "estará" dentro de la red de A).

Los pasos de configuración son los siguientes:

1) Activar el IP forwarding: `echo 1 >/proc/sys/net/ipv4/ip_forward`

2) Túnel: Sobre B `ssh -w 0:0 1.2.3.4` se puede utilizar como opciones además `-NTCf`. Esto crea un tunnel interface tun0 entre cliente (B) y el servidor (A) , recordar que A tiene dirección pública (1.2.3.4). Se deberá tener acceso de root a ambos sistemas para que puedan crear las interfaces (en A verificar que se tiene en `sshd_config PermitRootLogin yes` y `PermitTunnel yes`). Se recomienda utilizar SSH *keys* (PKI) y por lo cual se deberá cambiar PermitRootLogin without-password y si sobre el cliente no se quiere dar acceso de root se puede utilizar el comando `sudo` (ver `man sudoers`).

3) Verificar las interfaces: `ip addr show tun0`

4) Configurar las interfaces:
Ordenador A: `ip link set tun0 up ip addr add 10.0.0.100/32 peer 10.0.0.200 dev tun0`
Ordenador B: `ip link set tun0 up ip addr add 10.0.0.200/32 peer 10.0.0.100 dev tun0`

5) Probar: sobre B `ping 10.0.0.100` y sobre A `ping 10.0.0.200`

6) Routing directo: tenemos un link que conecta B en la red A pero es necesario inicializar el routing.
Sobre B: `ip route add 10.0.0.0/24 via 10.0.0.200` (para permitir enviar desde B a cualquier máquina de la red A).

7) Routing inverso: También para que los paquetes vuelvan a B sobre A debemos hacer `arp -sD 10.0.0.200 eth0`

El routing nos asegurará que otras máquinas que están en la red A puedan enviar paquetes a B a través de A. Podemos probar haciendo desde B `ping 10.0.0.123`. Si se desea que todos los paquetes de B vayan a través de A deberemos cambiar el default *gateway* de B pero esto no es simple ya que el propio túnel va por internet. Primero debemos crear un *host-based route* hacia la máquina A (todos los paquetes excepto los que crean el *link* deben ir por el túnel -pero obviamente no los que crean el túnel-). Sobre B: `ip route add 1.2.3.4/32 via 192.168.0.1` En este caso 192.168.0.1 es el *default gateway* para B, es decir el *gateway* sobre la red B que provee conectividad a internet y que nos servirá para mantener el túnel. Luego se podemos cambiar el *default gateway* sobre B: `ip route replace default via 10.0.0.1`. Con lo cual tendremos que 192.168.0.1 es el *default gateway* de la red B y 10.0.0.1 *default gateway* de la red A. Ya que la máquina B está conectada a la red de A le estamos indicando utilizar la red A como *default gateway* en lugar de *default gateway* sobre la red B (como sería habitual). Se puede verificar esto con el comando `tracepath google.com`.

1.3.4. Túneles encadenados

Muchas veces el acceso a redes internas solo es posible a través de un servidor SSH (que están en la DMZ) y desde esta es posible la conexión hacia servidores

SSH internos para luego llegar a la máquina SSH de nuestro interés (y si queremos copiar archivos o hacer el *forwarding* de X11 puede ser todo un reto). La forma de hacerlo es primero conectarnos a la M1, luego de esta a la M2 (a la cual solo es posible conectarse desde M1) y desde esta a la M3 (a la cual solo es posible conectarse desde M2). Una forma de facilitar la autentificación es utilizar el *agent forwarding* con el parámetro -A y poniendo nuestra llave pública en todos los *$HOME/.ssh/authorized_keys* así cuando vayamos ejecutando los diferentes comandos la petición de llaves puede ser *forwarded* hacia la máquina anterior y así sucesivamente. los comandos serán `ssh -A m1.org` y desde esta `ssh -a m2.org` y a su vez `ssh -a m3.org`. Para mejorar esto podemos automatizarlo con `ssh -A -t m1.org ssh -A -t m2.org ssh -A m2.org` (se ha incluido la opción -t para que `ssh` le asigne una pseudo-tty y solo veremos la pantalla final). El applet SSHmenu puede ayudar a automatizar todo esto (http://sshmenu.sourceforge.net/).

Una forma de gestionar efectivamente esta conexión "multisaltos" es a través de la opción `ProxyCommand` de `ssh` (ver `man ssh_config`) que nos permitirá conectarnos de forma más eficiente. Para ello definiremos los siguientes comandos en *$HOME/.ssh/config*:

```
Host m1
    HostName m1.org
Host m2
    ProxyCommand ssh -q m1 nc -q0 m2.org 22
Host m3
    ProxyCommand ssh -q m2 nc -q0 m3.org 22
```

Donde el primer comando (cuando hagamos `ssh m1`) nos conectará a m1.org. Cuando ejecutemos `ssh m2` se establecerá una conexión ssh sobre m1 pero el comando ProxyCommand utilizará en comando nc para extender la conexión sobre m2.org. Y el tercero es una ampliación del anterior por lo cual cuando ejecutemos `ssh m3` es como si ejecutáramos una conexión a m1 y luego ampliáramos esta a m2 y luego a m3.[23]

1.4. Servicios de transferencia de ficheros: FTP

El FTP (*File Transfer Protocol*) es un protocolo cliente/servidor (bajo TCP) que permite la transferencia de archivos desde y hacia un sistema remoto. Un servidor FTP es un ordenador que ejecuta el *daemon* `ftpd`.

Algunos sitios que permiten la conexión anónima bajo el usuario *anonymous* son generalmente repositorios de programas. En un sitio privado, se necesitará un usuario y una contraseña para acceder. También es posible acceder a un servidor ftp mediante un navegador y generalmente hoy en día los repositorios de programas se sustituyen por servidores web (p. ej. Apache) u otras tecnologías como Bittorrent (que utiliza redes *peer to peer*, P2P) o servidores web con módulos de WebDav o el propio comando `scp` (*secure copy*) que forma parte

del paquete openssh-client o el par `sftp/sftpd` que forman parte de los paquetes `openssh-client` y `openssh-server` respectivamente. No obstante, se continúa utilizando en algunos casos y Debian, por ejemplo, permite el acceso con usuario/contraseña o la posibilidad de subir archivos al servidor (si bien con servicios web-dav también es posible hacerlo).

El protocolo (y los servidores/clientes que lo implementan) de ftp por definición no es encriptado (los datos, usuarios y contraseñas se transmiten en texto claro por la red) con el riesgo que ello supone. Sin embargo, hay una serie de servidores/clientes que soportan SSL y por lo tanto encriptación.

1.4.1. Cliente ftp (convencional)

Un cliente ftp permite acceder a servidores ftp y hay una gran cantidad de clientes disponibles. El uso del ftp es sumamente simple; desde la línea de comando, ejecutad:

```
ftp nombre-servidor
```

O también `ftp` y luego, de forma interactiva, `open nombre-servidor`

El servidor solicitará un nombre de usuario y una contraseña (si acepta usuarios anónimos, se introducirá *anonymous* como usuario y nuestra dirección de correo electrónico como contraseña) y a partir del *prompt* del comando (después de algunos mensajes), podremos comenzar a transferir ficheros.

El protocolo permite la transferencia en modo ASCII o binario. Es importante decidir el tipo de fichero que hay que transferir porque una transferencia de un binario en modo ASCII inutilizará el fichero. Para cambiar de un modo a otro, se deben ejecutar los comandos `ascii` o `binary`. Los comandos útiles del cliente ftp son el `ls` (navegación en el directorio remoto), `get nombre_del_fichero` (para descargar ficheros) o `mget` (que admite `*`), `put nombre_del_fichero` (para enviar ficheros al servidor) o `mput` (que admite `*`); en estos dos últimos se debe tener permiso de escritura sobre el directorio del servidor. Se pueden ejecutar comandos locales si antes del comando se inserta un "`!`". Por ejemplo, `!cd /tmp` significará que los archivos que bajen a la máquina local se descargarán en `/tmp`. Para poder ver el estado y el funcionamiento de la transferencia, el cliente puede imprimir marcas, o *ticks*, que se activan con los comandos `hash` y `tick`. Existen otros comandos que se pueden consultar en la hoja del manual (`man ftp`) o haciendo `help` dentro del cliente. Contamos con numerosas alternativas para los clientes, por ejemplo en modo texto: `ncftp`, `lukemftp`, `lftp`, `cftp`, `yafc Yafc` o, en modo gráfico: `gFTP`, `WXftp`, `LLNL XFTP`, `guiftp`.

Enlace de interés

En la Wikipedia disponéis de una comparativa de diversos clientes FTP: http://en.wikipedia.org/wiki/Comparison_of_FTP_client_software

1.4.2. Servidores FTP

El servidor tradicional de UNIX se ejecuta a través del puerto 21 y es puesto en marcha por el *daemon* `inetd` (o `xinetd`, según se tenga instalado). En `inetd.conf` conviene incluir el *wrapper* `tcpd` con las reglas de acceso en `host.allow` y el `host.deny` en la llamada al `ftpd` por el `inetd` para incrementar la seguridad del sistema. Cuando recibe una conexión, verifica el usuario y la contraseña y lo deja entrar si la autentificación es correcta. Un FTP *anonymous* trabaja de forma diferente, ya que el usuario solo podrá acceder a un directorio definido en el archivo de configuración y al árbol subyacente, pero no hacia arriba, por motivos de seguridad. Este directorio generalmente contiene directorios `pub/, bin/, etc/` y `lib/` para que el *daemon* de ftp pueda ejecutar comandos externos para peticiones de `ls`. El *daemon* `ftpd` soporta los siguientes archivos para su configuración:

Ved también

El tema de la seguridad del sistema se estudia en el módulo "Administración de seguridad".

- **/etc/ftpusers**: lista de usuarios que no son aceptados en el sistema. Un usuario por línea.
- **/etc/ftpchroot**: lista de usuarios a los que se les cambiará el directorio base `chroot` cuando se conecten. Necesario cuando deseamos configurar un servidor anónimo.
- **/etc/ftpwelcome**: anuncio de bienvenida.
- **/etc/motd**: noticias después del *login*.
- **/etc/nologin**: mensaje que se muestra después de negar la conexión.
- **/var/log/ftpd**: *log* de las transferencias.

Si en algún momento queremos inhibir la conexión al ftp, se puede incluir el archivo `/etc/nologin`. El `ftpd` muestra su contenido y termina. Si existe un archivo `.message` en un directorio, el `ftpd` lo mostrará cuando se acceda al mismo. La conexión de un usuario pasa por cinco niveles diferentes:

1) tener una contraseña válida;

2) no aparecer en la lista de `/etc/ftpusers`;

3) tener un *shell* estándar válido;

4) si aparece en `/etc/ftpchroot`, se le cambiará al directorio `home` (incluido si es *anonymous* o ftp);

5) si el usuario es *anonymous* o ftp, entonces deberá tener una entrada en el `/etc/passwd` con usuario ftp, pero podrá conectarse especificando cualquier contraseña (por convención se utiliza la dirección de correo electrónico).

Es importante prestar atención a que los usuarios habilitados únicamente para utilizar el servicio ftp no dispongan de un *shell* a la entrada correspondiente de dicho usuario en `/etc/passwd` para impedir que este usuario tenga conexión, por ejemplo, por `ssh` o `telnet`. Para ello, cuando se cree el usuario, habrá que indicar, por ejemplo:

```
useradd -d/home/nteum -s /bin/false nteum
```
y luego: `passwd nteum`,

lo cual indicará que el usuario nteum no tendrá *shell* para una conexión interactiva (si el usuario ya existe, se puede editar el fichero `/etc/passwd` y cambiar el último campo por `/bin/false`). A continuación, se debe agregar como última línea `/bin/false` en `/ect/shells`. En [28] se describe paso a paso cómo crear tanto un servidor ftp seguro con usuarios registrados como un servidor ftp anonymous para usuarios no registrados. Dos de los servidores no estándares más comunes son el ProFTPD y Vsftpd (ambos incluidos en Debian por ejemplo).

Enlaces de interés

Para saber más sobre ProFTPD y Vsftpd podéis visitar las siguientes páginas http://www.proftpd.org https://security.appspot.com /vsftpd.html.

Para instalar el Proftpd sobre Debian, ejecutad: `apt-get install proftpd`. Una vez descargado `debconf`, preguntará si se quiere ejecutar por `inetd` o manualmente (es recomendable elegir la última opción). Si se debe detener el servicio (para cambiar la configuración, por ej.): `/etc/init.d/proftpd stop` y para modificar el fichero: `/etc/proftpd.conf`. Un servidor (Debian) muy interesante es el PureFtpd (`pure-ftpd`), que es muy seguro, permite usuarios virtuales, cuotas, SSL/TSL y unas características interesantes. El paquete virtual de Debian ftp-server (https://packages.debian.org/wheezy/ftp-server) muestra todos los paquetes de este tipo incluidos en la última versión.

Enlaces de interés

Para configurar un servidor FTP en modo encriptado (TSL) o para tener acceso *anonymous* podéis consultar: http://www.debian -administration.org /articles/228.
Por otro lado, para saber más sobre la instalación y configuración de PureFtpd podéis consultar: http://www.debian -administration.org /articles/383.

1.5. *Active Directory Domain Controller* con Samba4

Uno de los aspecto más importantes en la integración de sistemas, es la gestión de usuarios e identificación centralizada así como las autoridades de autentificación y los permisos. En mucho sitios basados en Windows esta tarea es realizada por un *Active Directory* (AD, servicio de directorio en una red distribuida de ordenadores -a veces también llamado PDC por *Primary Domain Controller*-) y es importante contar con herramientas de la banda de *Open Source* que permitan gestionar este tipo de entornos. Samba versión 4 es una nueva distribución de este popular software que permite la integración con sistemas Windows actuando como servidor de archivos y/o impresoras y además, en esta última versión y de forma estable, puede actuar como controlador de un dominio Windows aceptando clientes, gestionando usuarios y directorios en forma centralizada y totalmente compatible.[29, 30]

De los expertos de *Active Directory* (que generalmente son consultores especializados y con un alto coste) sabemos que es AD es una unión (que puede ser muy complicada de entender para los administradores que nos estén dedicados al mundo Windows) de diferentes servicios y tecnologías como DNS, Kerberos, LDAP y CIFS. Algunos incluyen DHCP también pero como veremos no es necesario en nuestra instalación. Estaríamos en un error si pensamos

que configurando cada uno de estos servicios tendríamos el resultado del conjunto, pero Samba4 presenta una capa de abstracción, estable y eficiente, que las integra para ofrecer un producto complejo pero que se puede configurar y administrar siguiendo un conjunto de pasos sin grandes dificultades (aunque no se debe pensar que es simple). El elemento crítico (base de los mayores problemas y errores) del AD es el DNS (en realidad Windows utilizará el AD como DNS) ya que este lo hará servir para 'agregar extraoficialmente' a la lista de nombres habituales en un DNS, los servidores AD para que los clientes puedan encontrarlos y tener acceso a ellos.

En relación Kerberos y LDAP los administradores de GNU/Linux saben de su potencialidad y complejidad, pero en el caso de AD están integrados en el paquete y si bien les otorga una cierta estandarización del sistema no son integrables con servidores u otros clientes, solo se utilizan para sus objetivos y particularmente cuando utilizamos Samba4, será este quien los configurará y gestionará en nuestro nombre con pequeñas modificaciones por parte del administrador. La versión actual de Samba (V4) no difiere de las anteriores en cuanto a compartición de archivos e impresoras (incluso se ha simplificado su gestión/administración) y además con la implementación de AD permitirá que aquellos administradores que utilicen Windows puedan continuar utilizando sus herramientas de gestión y administración de dominio solo con apuntar al servidor Samba. Toda la configuración de Samba pasará ahora por el archivo `smb.conf` (normalmente en */etc/samba/smb.conf*) con definiciones simplificadas pero permitiendo la gestión compleja de un dominio Windows a través de las herramientas (también complejas) de Windows como por ejemplo RSAT (*Remote Server Administration Tools*) y obviamente a través del sistema GNU/Linux y la CLI de Samba4 se tendrá acceso a toda la configuración y administración del AD (sin necesidad de utilizar Windows en ningún caso).[30, 33]

En nuestra instalación habitual utilizaremos Debian Wheezy como distribución y si bien Samba4 está en los repositorios *backport*, (en los de Wheezy solo existe la versión 3.x pero estará disponible en la nueva edición de Debian Jessie) optamos, en la facilidad a la hora de resolver las dependencias, por bajar el código fuente y compilarlo.

1.5.1. Pasos preliminares

Una de las primeras verificaciones que debemos hacer es que NPT esté instalado y funcionando correctamente (`apt-get install ntp` y probarlo que sincronza correctamente con `ntpq -p`) ya que Kerberos es muy exigente en cuanto a los valores del reloj. Un segundo aspecto a cuidar es definir los servidores con IP estáticas (al menos durante la prueba de concepto de la instalación y configuración del AD y óbviamente en un servidor en producción). Para ello definir */etc/network/interfaces* con:

```
auto eth0
iface eth0 inet static
    address 192.168.1.33
    netmask 255.255.255.0
    gateway 192.168.1.1
```

Luego deberemos verificar el */etc/hosts* (si es que no disponemos de un DNS que podamos modificar los registros A y PTR para incluir nuestro servidor de AD) para agregar (en nuestro caso) una entrada como: `192.168.1.33 sysdw.nteum.org sysdw`. Como DNS se podría utilizar cualquiera de los que se ha visto anteriormente, pero en este caso nos pareció más adecuado, para hacer una prueba total de Samba4, utilizar el que trae interno el paquete y que lo utilizará el AD. Es por ello que deberemos estar seguros de desactivar cualquier otro servicio de DNS que tengamos en esta máquina para que no existan conflictos (mirar por ejemplo con `netstat -anotup`). El otro aspecto importante es el nombre de dominio, y sin entrar en polémicas con relación a cómo se trata en GNU/Linux y cómo se 'ven' en Windows, hemos escogido por simplificar el del DNS (si bien algunos expertos indican que puede haber conflictos cuando tratemos con un DNS externo). Es decir un *AD Domain* no significa realmente un *DNS domain* sino que es una 'unión' híbrida de DNS y que Kerberos denomina `realm`. Es por ello que configuraremos el */etc/resolv.conf* apuntando como DNS a la propia máquina de Samba4 y con el dominio DNS de la misma (y como secundario uno público) -recordar que si se tiene instalado y activo el paquete `resolvconf` las modificaciones de */etc/resolv.conf* son dinámicas y las inclusión de los DNS se deberán hacer sobre */etc/network/interfaces* con los tags `dns-nameservers` y `dns-search`-:

```
domain nteum.org
search nteum.org
nameserver 192.168.1.33
nameserver 8.8.8.8
```

Además verificar que `/etc/hostname` coincide con el nombre indicado en `/etc/hosts`. Por último como Samba4 actuará como servidor de archivos y como PDC/AD, es necesario asegurar que el sistema de archivos que estemos exportando soporte ACL (*access control list*) y atributos extendidos, para ello modificaremos */etc/fstab* para indicarle al *file system* correspondiente los atributos `user_xattr,acl` incluyendo una línea como:

```
UUID=.... / ext4 user_xattr,acl,errors=remount-ro 0 2
```

Aquí lo hemos hecho sobre el directorio root ya que tenemos una partición solamente, pero esto solo se debe aplicar a aquellas particiones que se desee servir a los clientes.

Finalmente reiniciar la máquina para que todas las configuraciones se actualicen correctamente.

1.5.2. Compilar Samba4

En primer lugar debemos tener una serie de paquetes instalados (si bien muchos de ellos ya se encuentran instalados) -esta lista puede variar según las fuentes que se consulten- pero los requerimientos de Samba4 indican:

```
apt-get install build-essential libacl1-dev libattr1-dev \
   libblkid-dev libgnutls-dev libreadline-dev python-dev libpam0g-dev \
   python-dnspython gdb pkg-config libpopt-dev libldap2-dev \
   dnsutils libbsd-dev attr krb5-user docbook-xsl libcups2-dev acl
```

Otros expertos en Samba4 y AD también incluyen (algunas de ellas ya estarán incluidas o se resolverán por dependencias):

```
apt-get install  pkg-config libacl1 libblkid1 attr libattr1 libgnutls-dev  \
libncurses-dev libdm0-dev libfam0 fam libfam-dev xsltproc libnss3-dev \
docbook-xsl-doc-html docbook-xsl-ns python-dnspython libcups2-dev krb5-config
```

En nuestro caso usaremos como servidor de Kerberos nuestro servidor AD por lo cual es importante su configuración (paquetes `krb5-user` y `krb5-config`) y deberemos introducir durante su instalación el dominio (realm) que servirá en letras MAYÚSCULAS, que en nuestro caso será NTEUM.ORG y el *password* deberá tener cierta complejidad (como por ejemplo PaSSw0d2014) para que el programa lo considere válido. Para obtener y compilar la última versión de Samba4 haremos:

`wget http://www.samba.org/samba/ftp/stable/samba-4.x.y.tar.gz` (Reemplazar x.y con la última versión)

```
tar -xzf samba-4.x.y.tar.gz
cd samba-4.x.y
./configure --prefix=/opt/samba4
make
make install
```

Nosotros hemos escogido como lugar de instalación */opt/samba4* por lo cual deberemos actualizar la variable PATH con (poner en `.profile`):

```
export PATH=/usr/local/samba/bin:/usr/local/samba/sbin:$PATH
```

y para que la configuración quede como es habitual se crea un enlace a */etc/samba* con `ln -s /usr/local/samba/etc /etc/samba`.[32, 33]

Lo siguiente es aprovisionar el AD Domain:

```
samba-tool domain provision --use-rfc2307 --interactive
```

Aquí deberemos indicar el Realm (NTEUM.ORG que es el mismo que configuramos en krb5) y el Dominio (NTEUM en nuestro caso). Es importante que el *password* coincida con el que hemos introducido en krb5 (y tenga cierta complejidad). Asegurarse que el *DNS forwarder* apunte a el DNS server real que resolverá la peticiones de DNS e introducir la línea `allow dns updates = nonsecure` en los parámetros globales. El archivo */etc/samba/smb.conf* quedará como:

```
[global]
workgroup = NTEUM
realm = nteum.org
netbios name = SYSDW
server role = active directory domain controller
dns forwarder = 158.109.0.9
allow dns updates = nonsecure

[netlogon]
path = /opt/samba4/var/locks/sysvol/nteum.org/scripts
read only = No

[sysvol]
path = /opt/samba4/var/locks/sysvol
read only = No
```

A continuación debemos ajustar la configuración de Kerberos haciendo:

```
mv /etc/krb5.conf /etc/krb5.conf.org
cp  /opt/samba4/private/krb5.conf .
```

Cuyo contenido deberá quedar como:

```
[libdefaults]
default_realm = NTEUM.ORG
dns_lookup_realm = false
dns_lookup_kdc = true
```

Una vez reiniciado el sistema podemos poner en marcha el servidor simplemente ejecutando `samba` y mirar el */var/log/syslog* para verificar que se ha arrancado correctamente y si hay errores de que tipo son para solucionarlos (si ejecutamos `ps -edaf | grep samba` veremos aproximadamente 14 procesos samba si todo ha ido bien). Podremos verificar la ejecución del servidor haciendo:

```
smbclient -L localhost -U%
    Domain=[NTEUM] OS=[Unix] Server=[Samba 4.1.9]
       Sharename       Type      Comment
   ---------       ----      -------
   netlogon        Disk
   sysvol          Disk
   IPC$            IPC       IPC Service
    Domain=[NTEUM] OS=[Unix] Server=[Samba 4.1.9]
   Server               Comment
   ---------            -------
   Workgroup            Master
       ---------            -------

smbclient //localhost/netlogon -UAdministrator -c 'ls'      Nos solicitará el passwd de administrator
    Enter Administrator's password:
    Domain=[NTEUM] OS=[Unix] Server=[Samba 4.1.9]
    .                                   D        0  Wed Jul 23 10:11:33 2014
    ..                                  D        0  Wed Jul 23 10:11:39 2014

44541 blocks of size 262144. 7486 blocks available
```

Para verificar el DNS:

```
host -t SRV _ldap._tcp.nteum.org
    _ldap._tcp.nteum.org has SRV record 0 100 389 sysdw.nteum.org.

host -t SRV _kerberos._udp.nteum.org.
    _kerberos._udp.nteum.org has SRV record 0 100 88 sysdw.nteum.org.
```

```
host -t A sysdw.nteum.org
sysdw.nteum.org has address 158.109.65.67
```

También verificar que tenemos conexión externa con un `ping google.com` por ejemplo. Por último para verificar el funcionamiento de KRB5 (el realm debe ir en Mayúsculas):

```
kinit administrator@NTEUM.ORG
    Password for administrator@NTEUM.ORG:
    Warning: Your password will expire in 41 days on Wed Sep  3 17:10:52 2014
```

Y podríamos quitar (si bien en entornos de producción no sería recomendable) la caducidad del passwd con:

```
samba-tool user setexpiry administrator -noexpiry
```

Finalmente si todo funciona deberemos repetir el procedimiento de recrear el dominio pero agregando `-use-ntvfs` como parámetro, para ello podemos hacer:

```
mv /etc/samba/smb.conf /etc/samba/smb.conf.org
samba-tool domain provision --realm nteum.org --domain NTEUM --adminpass PaSSw0d2014 \
   --server-role=dc --use-ntvfs
```

El nuevo archivo smb.conf quedará (agregando nuevamente la línea `allow dns updates`):

```
[global]
workgroup = NTEUM
realm = nteum.org
netbios name = SYSDW
server role = active directory domain controller
dns forwarder = 158.109.0.9
server services = rpc, nbt, wrepl, ldap, cldap, kdc, drepl, winbind, ntp
_signd, kcc, dnsupdate, dns, smb
dcerpc endpoint servers = epmapper, wkssvc, rpcecho, samr, netlogon, lsa
rpc, spoolss, drsuapi, dssetup, unixinfo, browser, eventlog6, backupkey, dnsserv
er, winreg, srvsvc
allow dns updates = nonsecure
[netlogon]
path = /opt/samba4/var/locks/sysvol/nteum.org/scripts
read only = No

[sysvol]
path = /opt/samba4/var/locks/sysvol
read only = No
```

Y que /etc/krb5.conf es:

```
[libdefaults]
default_realm = NTEUM.ORG
dns_lookup_realm = false
dns_lookup_kdc = true
```

Reiniciar nuevamente y ejecutar `samba` (recordar que todavía no lo tenemos como servicio y que lo debemos ejecutar manualmente. Si queremos que Samba4 sirva los directorios *homes* deberemos agregar en *smb.conf* una sección con:

```
[home]
        path = /home/
        read only = no
```

Para que el administrador tenga los privilegios correctos en la gestión de las cuentas desde entornos Windows (y que Samba4 cree los directorios *home* automáticamente) deberemos ejecutar:

```
net rpc rights grant `NTEUM\Domain Admins` SeDiskOperatorPrivilege
-Uadministrator
```

Ahora deberíamos probar que podemos configurar un cliente Windows y agregarlo al directorio. Para ello (en nuestro caso) utilizaremos una máquina Windows8 donde en primer lugar modificaremos la interface de red para que el DNS apunte a nuestro servidor de AD (vamos a *Control Panel* > *Network and Internet* > *Network and Sharing Center* seleccionamos el dispositivo de red activo y en propiedades le cambiamos el DNS, dejando solo la IP de nuestro AD). Luego vamos a *Control Panel* > *System and Security* > *System* y seleccionamos *Advanced System Settings* y dentro *Computer Name* > *Change* e introducimos administrator y PaSSw0d2014 para agregar la máquina al dominio (en caso que no nos seleccione el dominio por defecto podemos poner en Username: NTEUM\administrator y luego el *passwd*), que nos dará un mensaje de confirmación si todo ha ido bien y nos pedirá hacer un *reboot* de la máquina. Cuando se inicie nuevamente, podremos seleccionar otro usuario/dominio y conectarnos a NTEUM\administrator (que es la cuenta previamente definida). Con todo ello se generarán una serie de pasos en el cliente y el servidor para aprovisionar en esa nueva máquina la configuración y directorio raíz.[32, 33]

Para administrar el sitio AD se pueden utilizar las herramientas de Windows (RSAT), las cuales se pueden obtener (sin cargo) desde el sitio del fabricante y con la guía indicada en [31] y ejemplos de configuración en [32]. También es posible utilizar la herramienta Swat (https://wiki.samba.org/index.php/SWAT2, última versión Noviembre de 2012) pero su instalación puede presentar algunos inconvenientes (sobre todo si Samba4 se ha instalado desde los fuentes). Por último en https://wiki.samba.org/index.php/Samba4/InitScript podremos encontrar el *script* para iniciar y apagar el servidor AD en forma automática.

1.6. Servicios de intercambio de información a nivel de usuario

1.6.1. El *Mail Transport Agent* (MTA)

Un MTA (*Mail Transport Agent*) se encarga de enviar y recibir los correos desde un servidor de correo electrónico hacia y desde Internet, que implementa el protocolo SMTP (*Simple Mail Transfer Protocol*). Todas las distribuciones incorporan diferentes MTA y por ejemplo las de Debian se pueden consultar en su paquete virtual *mail-transport-agent**. Una de las que se utilizan habitualmente es `exim`, ya que es más fácil de configurar que otros paquetes MTA, como son `postfix` o `sendmail` (este último es uno de los precursores). `Exim` presenta características avanzadas tales como rechazar conexiones de sitios de *spam* conocidos, posee defensas contra correo basura (*junk mails*) o bombardeo de

***https://packages.debian.org/wheezy/mail-transport-agent**

correo (*mail bombing*) y es extremadamente eficiente en el procesamiento de grandes cantidades de correos.

Su instalación es a través de `apt-get install exim4-daemon-heavy` (en este caso se optará por la instalación de la versión *heavy* que es la más completa y soporta lista de acceso (ACL) y características avanzadas, en instalaciones más simple se puede optar por *exim4-daemon-light*). Su configuración se realiza a través de `dpkg-reconfigure exim4-config` donde una respuesta típica a las preguntas realizadas es:

- *General type of mail configuration*: internet site; mail es enviado y recibido utilizado SMTP.
- *System mail name*: remix.world (nuestro dominio)
- *IP-addresses to listen on for incomming SMTP connections*: (dejar en blanco)
- *Other destinations for which mail is accepted*: remix.world
- *Domains to relay mail for*: (dejar en blanco)
- *Machines to relay mail for*: (dejar en blanco)
- *Keep number of DNS-queries minimal (Dial-on-Demand)*?: No
- *Delivery method for local mail: Maildir format in home directory*
- *Split configuration into small files*?: No

El servidor ya estará configurado y puede probarse con `echo test message | mail -s "test" adminp@SySDW.nteum.org` (por supuesto cambiando la dirección) y verificando que el mail ha llegado al usuario adminp (los errores se pueden encontrar en */var/log/exim4/mainlog*). La configuración será almacenada en */etc/exim4/update-exim4.conf.conf*. Para configurar autentificación por TLS, ACL y Spam Scaning consultar https://wiki.debian.org/Exim.

1.6.2. External SMTP

Cuando instalamos un nuevo sistema como servidores o estaciones de trabajo un aspecto relevante es el servidor de correo y podemos instalar grandes paquetes como los ya mencionados Postfix, Exim o Zimbra (en su version Community http://www.zimbra.com/) haciendo que los correos hacia dominios externos utilicen los servicios externos de SMTP (por ejemplo los de Google). Para máquinas virtuales, estaciones de trabajo o portátiles es un poco más complicado ya que generalmente tienen IP privadas o en redes internas por lo cual es necesario tener un servidor que haga de receptor de los correos externo a mi dominio, es decir un servidor que haga las funciones de *smarthost* por ejemplo el Google Apps SMTP. Para detalles de su configuración se puede seguir la documentación de https://wiki.debian.org/GmailAndExim4. De acuerdo a la información de Google* y para cuentas gratuitas el número máximo de destinatarios permitido por dominio y día es de 100 mensajes y que Gmail reescribirá la dirección del remitente. Para su configuración ejecutaremos `dpkg-reconfigure exim4-config` y seleccionaremos:

***https://support.google.com/a/answer/2956491?hl=es**

- *mail sent by smarthost; received via SMTP or fetchmail.*
- *System mail name*: localhost
- *IP-addresses to listen on for incomming SMTP connections*: 127.0.0.1
- *Other destinations for which mail is accepted*: (dejar en blanco)
- *Machines to relay mail for*: (dejar en blanco)
- *IP address or host name of the outgoing smarthost*: smtp.gmail.com::587
- *Hide local mail name in outgoing mail*?: No
- *Keep number of DNS-queries minimal (Dial-on-Demand)*?: No
- *Delivery method for local mail: mbox format in /var/mail*
- *Split configuration into small files*?: Yes

Esta es la configuración más adecuada si no se tiene un IP visible externamente. El envío sobre el puerto 587 de Gmail utiliza STARTTLS para asegurar la protección del passwd y para indicar el usuario y passwd de acceso a Gmail (utilizar una cuenta solo para este objetivo, no la cuenta habitual de Gmail) se debe editar el fichero */etc/exim4/passwd.client* y agregar la siguiente línea.

```
*.google.com:SMTPAccountName@gmail.com:y0uRpaSsw0RD
```

Luego ejecutar (para evitar que otros usuarios de la máquina puedan leer su contenido:

```
chown root:Debian-exim /etc/exim4/passwd.client
chmod 640 /etc/exim4/passwd.client
```

Gmail reescribirá la dirección del remitente automáticamente pero si no lo hiciera o enviamos a un *smarthost* que no lo hace deberíamos configurar el archivo */etc/email-addresses* con todas las combinaciones de direcciones posibles a utilizar (una por línea) y las dirección que se reescribirá (por ejemplo, nteum@remix.world: SMTPAccountName@gmail.com). Luego se deberá ejecutar:

```
update-exim4.conf
invoke-rc.d exim4 restart
exim4 -qff
```

Con ello se actualiza y recarga la configuración y se fuerza a enviar todos los correos que están pendientes. Como mostramos anteriormente en */var/log/exim4/mainlog* tendremos los errores si los hay.Si existen errores de autentificación sobre gmail verifique con el comando `host smtp.gmail.com` cuales son los *hosts* que devuelve y si estos concuerdan con el patrón incluido en */etc/exim4/passwd.client*. Si es diferente cámbielo para que coincida.

1.7. *Internet Message Access Protocol* (IMAP)

Este servicio permite acceder a los correos alojados en un servidor a través de un cliente de correo como por ejemplo Thunderbird del proyecto Mozilla.org.

Este servicio soportado por el *daemon* `imapd` (los actuales soportan el protocolo IMAP4rev1) permite acceder a un archivo de correo electrónico (*mail file*) que se encuentra en una máquina remota. El servicio `imapd` se presta a través de los puertos 143 (imap2), 220 (imap3) o 993 (imaps) cuando soporta encriptación por SSL. Si se utiliza `inetd`, este servidor se pone en marcha a través de una línea en `/etc/inetd.conf` como:

```
imap2 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
imap3 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
```

En este ejemplo se llama al *wrapper* `tcpd` que funciona con `hosts.allow` y `hosts.deny` para incrementar la seguridad. Las aplicaciones más populares son `courier-imap`, `cyrus-imapd`, `dovecot-imapd` entre otros. Para probar que el servidor `imap` funciona, se podría utilizar un cliente, por ejemplo Thunderbird/Icedove (Debian), Evolution, Squirremail, o cualquier otro cliente que soporte IMAP, crear una cuenta para un usuario local, configurarlo adecuadamente para que se conecte sobre la máquina local y verificar el funcionamiento de `imap`.

Se puede instalar `apt-get install dovecot-imapd` como prueba de concepto, que con las opciones por defecto permite una conexión encriptada por SSL y sobre buzones mailbox (o si queremos sobre buzones maildir deberemos cambiar la configuración de `/etc/dovecot/conf.d/10-mail.conf`). Dovecot es un servidor muy potente por lo cual permite una gran cantidad de opciones y configuraciones (consultar http://wiki2.dovecot.org/ y un resumen aplicado de ellas en Debian Wheezy*). Las pruebas se pueden completar configurando Evolution para que se conecte a nuestro servidor/usuario y leer los correos del servidor previamente configurado. Es importante notar que algunos clientes de correo/servidores de Imap solo soportan el formato MailBox y no Maildir y es por ello que se debe tener en cuenta cuando se utilicen los clientes/servidores de Imap. En las actividades que hemos realizado hasta ahora tanto `exim4` como `dovecot-imapd` soportan ambos formatos y se deben configurar durante la instalación.

***https://workaround.org/ispmail/wheezy/setting-up-dovecot**

1.7.1. Aspectos complementarios

Supongamos que como usuarios tenemos cuatro cuentas de correo en servidores diferentes y queremos que todos los mensajes que llegan a estas cuentas se recojan en una única, a la que podamos acceder externamente, y que haya también un filtro de correo basura (*antispam*). Primero se deben instalar `exim4 + imapd` y comprobar que funcionan.

Para recoger los mensajes de diferentes cuentas se utilizará `Fetchmail`, (que se instala con `apt-get install fetchmail`). A continuación, se debe crear el fichero `.fetchmailrc` en nuestro `$HOME` (también se puede utilizar la herramienta `fetchmailconf`) que deberá tener ser algo así como:

```
set postmaster “adminp”
set bouncemail
set no spambounce
set flush
poll pop.domain.com proto pop3
   user 'nteum' there with password 'MyPaSSwOrD' is 'nteum' here
poll mail.domain2.com
   user 'adminp' there with password 'MyPaSSwOrD' is 'adminp' here
   user 'nteum' there with password 'MyPaSSwOrD' is 'nteum' here
```

La acción `set` indica a `Fetchmail` que esta línea contiene una opción global (envío de errores, eliminación de los mensajes de los servidores, etc.). A continuación, se especifican dos servidores de correo: uno para que compruebe si hay correo con el protocolo POP3 y otro para que pruebe a usar varios protocolos con el fin de encontrar uno que funcione. Se comprueba el correo de dos usuarios con la segunda opción de servidor, pero todo el correo que se encuentre se envía al *spool* de correo de `adminp`. Esto permite comprobar varios buzones de diversos servidores como si se tratara de un único buzón. La información específica de cada usuario comienza con la acción `user`. El `Fetchmail` se puede poner en el `cron`* para que se ejecute automáticamente o ejecutarlo en modo *daemon* (poned `set daemon 60` en `.fetchmailrc` y ejecutadlo una vez, por ejemplo, en `autostart` de Gnome/KDE o en el `.bashrc` –se ejecutará cada 60 segundos–).

***Por ejemplo, en** `/var/spool/cron/crontabs/fetchmail` **agregando** `1 * * * * /usr/bin/fetchmail -s`

Para quitar el correo basura se utilizará SpamAssassin y en esta configuración se ejecutará a traves de Procmail, que es una herramienta muy potente en la gestión del correo (permite repartir el correo, filtrarlo, reenviarlo automáticamente, etc.). Una vez instalado (`apt-get install procmail`), se debe crear un fichero llamado `.procmailrc` en el `home` de cada usuario, que llamará al SpamAssassin:

Podéis instalar SpamAssassin mediante `apt-get install spamassassin`.

```
# Poned yes para mensajes de funcionamiento o depuración
VERBOSE=no
# Consideramos que los mensajes están en “~/.Maildir”), cambiar si es otro
PATH=/usr/bin:/bin:/usr/local/bin:.
MAILDIR=$HOME/Maildir
DEFAULT=$MAILDIR/

# Directorio para almacenar los ficheros
PMDIR=$HOME/.procmail
# Comentar si no queremos log de Procmail
LOGFILE=$PMDIR/log
# filtro de Smap
INCLUDERC=$PMDIR/spam.rc
```

El archivo `~/.procmail/spam.rc` contiene:

```
# si el SpamAssassin no está en el PATH, agregar a la variable PATH el directorio
:0fw: spamassassin.lock
| spamassassin -a

# La tres líneas siguientes moverán el correo Spam a un directorio llamado
# “spam-folder”. Si se quiere guardar el correo en la bandeja de entrada,
# para luego filtrarlo con el cliente, comentad las tres líneas.


:0:
```

```
* ^X-Spam-Status: Yes
spam-folder
```

El archivo `~/.spamassassin/user_prefs` contiene algunas configuraciones útiles para SpamAssassin (consultad la bibliografía).

```
# user preferences file. Ved man Mail::SpamAssassin::Conf

# Umbral para reconocer un Spam.
# Default 5, pero con 4 funciona un poco mejor
required_hits 4

# Sitios de los que consideraremos que nunca llegará Spam
whitelist_from root@debian.org
whitelist_from *@uoc.edu

# Sitios de los que siempre llega SPAM (separados por comas)
blacklist_from viagra@dominio.com

# las direcciones en Whitelist y Blacklist son patrones globales como:
# "amigo@lugar.com", "*@isp.net", o "*.domain.com".

# Insertad la palabra SPAM en el subject (facilita hacer filtros).
# Si no se desea comentar la línea.
subject_tag [SPAM]
```

Esto generará un *tag* `X-Spam-Status: Yes` en la cabecera del mensaje si se cree que el mensaje es *spam*. Luego se deberá filtrar y poner en otra carpeta o borrarlo directamente. Se puede usar el `procmail` para filtrar mensajes de dominios, usuarios, etc. Por último, se puede instalar un cliente de correo y configurar los filtros para que seleccionen todos los correos con `X-Spam-Status: Yes` y los borre o los envíe a un directorio. Después verificaremos los falsos positivos (correos identificados como basura pero que no lo son). Un aspecto complementario de esta instalación es que si se desea tener un servidor de correo a través de correo web (*webmail*, es decir poder consultar los correos del servidor a través de un navegador sin tener que instalar un cliente ni configurarlo, igual que consultar una cuenta de Gmail o Hotmail) es posible instalar Squirrelmail (`apt-get install squirrelmail`) para dar este servicio.

Enlace de interés

Para más información sobre `procmail` y el filtrado de mensajes, consultad: http://www.debian-administration.org/articles/242.

Enlace de interés

Sobre Squirrelmail en Debian, consultad: http://www.debian-administration.org/articles/200.

Enlace de interés

Hay otras posibilidades como instalar MailDrop en lugar de Procmail, Postfix en lugar de Exim, o incluir Clamav/Amavisd como antivirus (Amavisd permite vincular Postfix con SpamAssassin y Clamav). Para saber más sobre este tema podéis visitar la siguiente página web: http://www.debian-administration.org/articles/364.

1.8. Grupos de discusión

Las *news* o grupos de discusión son soportados a través del protocolo NNTP. Instalar un servidor de grupos de discusión es necesario cuando se desea leer *news* fuera de línea, cuando se quiere tener un repetidor de los servidores centrales o se quiere un propio servidor maestro de *news*. Los servidores más comunes son INN o CNEWS, pero son paquetes complejos y destinados a grandes servidores. Leafnode es un paquete USENET que implementa el servidor TNP, especialmente indicado para sitios con grupos reducidos de usuarios, pe-

ro donde se desea acceder a gran cantidad de grupos de noticias. Este servidor se instala en la configuración básica de Debian y se pueden reconfigurar con `dpkg-reconfigure leafnode` todos parámetros como los servidores centrales, el tipo de conexión, etc. Este *daemon* se pone en marcha desde `inetd` de forma similar al `imap` (o con `xinetd`). Leafnode soporta filtros a través de expresiones regulares indicadas (del tipo ^Newsgroups:. * [,] alt.flame$) en `/etc/news/leafnode/filters`, donde para cada mensaje se compara la cabecera con la expresión regular y, si existe coincidencia, el mensaje se rechaza.

La configuración de este servidor es simple y todos los archivos deben ser propiedad de un usuario de *news* con permiso de escritura (se debe verificar que dicho propietario existe en `/etc/passwd`). Todos los archivos de control, *news* y la configuración se encuentran en `/var/spool/news`, excepto la configuración del propio servidor que está en el fichero `/etc/news/leafnode/config`. En la configuración existen algunos parámetros obligatorios que se deben configurar (por ejemplo, para que el servidor pueda conectarse con los servidores maestros), como son `server` (servidor de *news* desde donde se obtendrán y enviarán las *news*) y `expire` (número de días a los que un hilo o sesión se borrará tras haber sido leído). Tenemos, asimismo, un conjunto de parámetros opcionales de ámbito general o específico del servidor que podrían configurarse. Para más información, consultad la documentación (`man leafnode` o `/usr/doc/leafnode/README.Debian`). Para verificar el funcionamiento del servidor, se puede hacer `telnet localhost nntp` y, si todo funciona correctamente, saldrá la identificación del servidor y se quedará esperando un comando. Como prueba, se puede introducir `help` (para abortar, haced `Ctrl+` y luego `Quit`).

1.9. *World Wide Web* (httpd)

Apache es uno de los servidores más populares y con mayores prestaciones de HTTP (*HyperText Transfer Protocol*). Apache tiene un diseño modular y soporta extensiones dinámicas de módulos durante su ejecución. Es muy configurable en cuanto al número de servidores y de módulos disponibles y soporta diversos mecanismos de autentificación, control de acceso, *metafiles*, *proxy caching*, servidores virtuales, etc. Con módulos (incluidos en Debian) es posible tener PHP3, Perl, Java Servlets, SSL y otras extensiones*.

*Podéis consultar la documentación en http://www.apache.org.

Apache está diseñado para ejecutarse como un proceso *daemon standalone*. En esta forma, crea un conjunto de procesos hijos que gestionarán las peticiones de entrada. También puede ejecutarse como *Internet daemon* a través de `inetd`, por lo que se pondrá en marcha cada vez que se reciba una petición pero no es recomendado. La configuración del servidor puede ser extremadamente compleja según las necesidades (consultad la documentación); sin embargo, aquí veremos una configuración mínima aceptable. Su instalación es simple, por ejemplo en Debian, `apt-get install apache2 apache2-doc apache2-utils`. La configuración del servidor estará en */etc/apache2* y por defecto el RootDirectory en */var/www*. Después de su instalación se pondrá

en marcha y llamando a través de un navegador veremos que funciona (nos mostrará el famoso **It works!**). Existen 5 comandos que deberán están en mente de todo administrador: `a2enmod|a2dismod` para habilitar/deshabilitar módulos, `a2ensite|a2dissite` para habilitar|deshabilitar sitios (virtuales) y `apachectl` para gestionar la configuración del servidor (start|stop|restart|graceful|graceful-stop|configtest|status|fullstatus|help).

La configuración de Apach2 en Debian es un poco diferente a la distribución general ya que intenta facilitar al máximo la configuración del servidor en cuanto a módulos, hosts virtuales y directivas de configuración (no obstante rápidamente se puede encontrar las equivalencias con otras distribuciones). Los principales archivos que se encuentran en el directorio */etc/apache2/* son *apache2.conf, ports.conf* y cinco directorios *mods-available|mods-enabled, sites-available|sites-enabled* y *conf.d*. Para información adicional leer */usr/share/doc/apache2.2** y en particular */usr/share/doc/apache2.2-common/README.Debian*.

1) *apache2.conf* es el archivo principal de configuración donde se define a nivel funcional las prestaciones del servidor y se llama a los archivos de configguración correspondientes (*ports, conf.d, sites-enabled*). Se recomienda poner como sufijo *.load* para los módulos que deban ser cargados y *.conf* para las configuraciones pero hay reglas más extensas en cuanto a los sufijos/nombres que pueden ampliarse en la documentación (p. ej., se ignoran todos los archivos que no comienzan por letra o número).

2) *ports.conf* (se incluye en el archivo de configuración global) define los puertos donde se atenderán la conexiones entrantes, y cuales de estos son utilizados en los *host* virtuales.

3) Los archivos de configuración en *mods-enabled/* y *sites-enabled/* son para los sitios activos y los módulos que desean ser cargado en el servidor. Estas configuraciones son activadas creando un link simbólico desde los directorios respectivos ***-available/* utilizando los comandos `a2enmod/a2dismod`, `a2ensite/a2dissite`.

4) Los archivos de *conf.d* son para configuraciones de otros paquetes o agregados pro el administrador y se recomienda que acaben con *.conf*.

5) Para que sea efectiva la configuración por defecto en estos directorios apache2 tendrá que ser gestionado a través de */etc/init.d/apache2* o `service` o `apache2ctl`.

6) El archivo *envvars* es el que contendrá la definición de las variables de entorno y es necesario modificar básicamente dos USER/GROUP que serán con las cuales se ejecutará y obtendrá los permisos. Por defecto se crea el usuario www-data y el grupo www-data (se pueden cambiar). Por lo cual deberá utilizarse `APACHE_RUN_USER=www-data` y `APACHE_RUN_GROUP=www-data`.

Apache también puede necesitar integrar diversos módulos en función de la tecnología que soporte y por lo cual se deberán agregar las librerías/paquetes correspondientes, por ejemplo:

1) Perl: `apt-get install libapache2-mod-perl2`

2) Rugby: `apt-get install libapache2-mod-ruby`

3) Python: `apt-get install libapache2-mod-python`

4) MySQL in Python: `apt-get install python-mysqldb`

5) PHP: `apt-get install libapache2-mod-php5 php5 php-pear php5-xcache`

6) PHP with MySQL: `apt-get install php5-mysql`

1.9.1. Servidores virtuales

Por servidores virtuales se entiende sitios aliados que serán servidos cada uno independiente del otro con sus propios archivos y configuración. En primer lugar deshabilitaremos el sitio por defecto con `a2dissite default`. Los sitios que crearemos serán remix.world y lucix.world que dispondrán de dos archivos de configuración en */etc/apache2/sites-available/* llamados como el dominio.

Contenido del archivo /etc/apache2/sites-available/remix.world.conf

```
<VirtualHost *:80>
    ServerAdmin adminpSySDW.nteum.org
    ServerName remix.world
    ServerAlias www.remix.world
    DocumentRoot /var/www/remix/
    ErrorLog /var/log/apache2/remix-error.log
    CustomLog /var/log/apache2/remix-access.log combined
    Options ExecCGI # habilitar Script en Perl
    AddHandler cgi-script .pl
</VirtualHost>
```

Contenido del archivo /etc/apache2/sites-available/lucix.world.conf

```
<VirtualHost *:80>
    ServerAdmin adminpSySDW.nteum.org
    ServerName lucix.world
    ServerAlias www.lucix.world
    DocumentRoot /var/www/lucix/
    ErrorLog /var/log/apache2/lucix-error.log
    CustomLog /var/log/apache2/lucix-access.log combined
    Options ExecCGI # habilitar Script en Perl
    AddHandler cgi-script .pl
</VirtualHost>
```

Esta configuración es básica y el estudiante deberá consultar la información detallada en [14]. Como se puede observar los directorios raíz para cada dominio estarán en */var/www/remix|lucix* y los archivos de log en /errores/accesos en */var/log/apache2/mmmm-error.log* y *var/log/apache2/nnnn-access.log/*. Para crear los directorios `mkdir -p /var/www/remix`; `mkdir -p /var/www/lucix` y en los cuales se podría poner un *index.html* con alguna identificación que mostrara que dominio se está cargando. por ejemplo para remix.world:

```
<html><body><h1>REMIX: It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Y lo mismo para lucix.world pero cambiando la línea en <h1></h1>. Para los log no debemos hacer nada ya que el directorio */var/log/apache2* ya existe y los archivos los creará el servidor. Finalmente debemos activar los sitios (para ello debemos crear el enlace desde *sites-available* a *sites-enabled*) con `a2ensite remix.world.conf`; `a2ensite lucix.world.conf` y reiniciar apache2 con `service apache2 reload`. Como no disponemos de los dominios en un DNS primario podemos editar */etc/hosts* y agregar para la IP de nuestro servidor (p. ej., 192.168.1.37) dos líneas:

```
192.168.1.37 remix.world
192.168.1.37 lucix.world
```

Luego desde un navegador podremos introducir la URL remix.world y el resultado será la visualización del index.html que nos dirá: **REMIX: It works!**

Una de las ventajas de apache es que puede agregar funcionalidad a través de módulos especializados y que se encontrarán en */etc/apache2/mods-available/*. Para obtener la lista de módulos en disponible para apache podemos hacer por ejemplo `apt-cache search libapache2*`, y para instalarlo `apt-get install [module-name]` los cuales estarán disponibles para su uso (recordad que puede ser necesario alguna configuración adicional en los archivos del sitio). Con `ls -al /etc/apache2/mods-available/` podemos mirar los disponibles e instalarlo con `a2enmod [module-name]`. Para listar los módulos cargados podemos hacer `/usr/sbin/apache2 -M` que nos listará con *shared* los cargados dinámicamente y con *static* los que se encuentran compilados con el servidor (estos se puede obtener también con `apache2 -l`). Los módulos en el directorio *mods-available* tienen extensiones *.load* (indica la librería a cargar) y *.conf* (configuración adicional del módulo) pero cuando utilizamos el comando `a2enmod` solo se debe indicar el nombre del módulo sin extensión. Para deshabilitar un módulo `a2dismod [module-name]`.

Como muestra de estas propiedades configuraremos un sitio seguro (https) bajo el dominio remix.world pero que redirigiremos al directorio */var/www/remix.ssl*. En primer lugar crearemos un certificado (autofirmado) para nuestro sitio con la orden `make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /etc/ssl/private/remix.crt` indicándole el dominio que queremos validar (remix.world en nuestro caso) -solo introducir el dominio y dejar los alias en blanco- y si no podemos ejecutar `make-ssl-crt` asegurarnos que tenemos el paquete ssl-cert. Luego activamos el módulo SSL con `a2enmod ssl`, creamos el directorio */var/www/remix.ssl* y modificamos el index.html como hicimos con los anteriores. A continuación modificamos creamos la configuración del sitio (podemos utilizar la que viene por defecto modificándola):

```
cd /etc/apache2/sites-available; cp default-ssl remix.world.ssl.conf
```

Editamos el archivo remix.world.ssl.conf (solo mostramos las líneas principales/modificadas):

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
    ServerAdmin adminpSySDW.nteum.org
    DocumentRoot /var/www/remix.ssl
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /var/www/remix.ssl>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>
# líneas igual que el archivo original...
    ErrorLog $APACHE_LOG_DIR/remix.world.ssl_error.log
    CustomLog $APACHE_LOG_DIR/remix.world.ssl_access.log combined

    SSLEngine on
    SSLCertificateFile /etc/ssl/private/remix.crt
    #SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
# líneas igual que el archivo original...
</VirtualHost>
</IfModule>
```

Finalmente nos queda activar el sitio (`a2ensite remix.world.ssl.conf`), reiniciar apache2 (`service apache2 reload`) y desde el navegador hacer *https://remix.world* que como el certificado es autofirmado nos hará una advertencia y aceptaremos el certificado y deberemos obtener **SSL - REMIX: It works!**.

Un aspecto interesante es la función del archivo *.htaccess** en los directorios de nuestro dominio. Este archivo se puede utilizar para control de acceso al sitio (p. ej., habilitar/restringir IP), control de acceso a carpetas, listados, redirecciones (p. ej., a otra página/site, a otra carpeta, a otro dominio, a https, ...), evitar el *hotlinking* (para evitar que nos hagan enlaces a ficheros -generalmente vídeos- y consuman ancho de banda de nuestro servidor), cambiar la página por defecto, crear URL amigables, favorecer el cache de nuestro sitio, etc. Como muestra de ello para evitar por ejemplo que una carpeta sea inaccesible solo basta poner un archivo *.htaccess* en ella con el contenido `deny from all`. Para permitir que una carpeta de nuestro sitio (p. ej., del dominio remix.com/valid-user) tenga acceso con un usuario y passwd deberemos crear dentro de ella un archivo *.htaccess* con el siguiente contenido (también podemos crear un index.html modificado dentro de esta carpeta para verificar el funcionamiento):

***http://httpd.apache.org/docs/2.2/howto/htaccess.htm**

```
AuthName “Restricted Area”
AuthType Basic
AuthUserFile /etc/apache2/htpasswd
AuthGroupFile /dev/null
require valid-user
```

Para crear el usuario ejecutamos `htpasswd -c /etc/apache2/htpasswd adminp` que nos pedirá el passwd para este usuario y los almacenará en el archivo indicado. Luego ponemos como URL *http://remix.world/valid-user/* nos

pedirá el usuario (adminp) y el passwd que almacenamos y veremos: **REMIX->Valid-User: It works!**. En caso contrario nos continuará pidiendo el usuario/passwd y si hacemos Cancel no indicará un mensaje de Authorization Required impidiendo el acceso.

1.9.2. Apache + PHP + Mysql + PhpMyAdmin

Una cuestión importante para los servidores web dinámicos es aprovechar las ventajas de Apache PHP (un lenguaje de programación usado generalmente para la creación de contenido para sitios web) y una base de datos como MySQL incluyendo un programa adminsitrador de MySQL como PHPMyAdmin, todo ello funcionando conjuntamente. Las distribuciones han evolucionado mucho y en Debian es sumamente fácil poner en marcha este conjunto (pero tampoco representa ninguna dificultad bajarse el software fuente, compilarlo e instalarlo si se desea, por ejemplo, tener las últimas versiones de los paquetes por algún motivo pero recordad que implicará más trabajo y dedicación).

En primer lugar instalamos PHP con `apt-get install php5` que nos indicará que cambiará el modo de trabajo de apache2 instalando otro. Esta cuestión es porque la configuración por defecto de apache trabaja con una herramienta llamada MPM-worker. Este es un módulo de multi-procesamiento que puede manejar múltiples peticiones rápidamente utilizando múltiples *threads* por proceso cliente. Sin embargo este no es compatible con alguna extensiones PHP y por ello el MPM-worker es reemplazado por MPM-prefork, que permite manejar todas las peticiones PHP (en modo compatibilidad) y evitar que si una petición falla pueda afectar a otras peticiones. Existe otro módulo llamado mpm-itk (http://mpm-itk.sesse.net/) que es similar prefork pero tiene mejores prestaciones y gestión de permisos (consultar la bibliografía en apache.org). Para verificar que PHP funciona creamos un fichero por ejemplo dentro de RootDirectory de remix.world llamado *test.php* con el siguiente contenido: *<?php phpinfo() ?>* y si en la URL introducimos *http://remix.world/test.php* deberemos ver una tabla con la versión e información sobre el paquete PHP instalado.

Para instalar los paquetes MySQL y PHPMyAdmin haremos `apt-get install mysql-server` (es importante recordar la contraseña de acceso que introduzcamos pero siempre podemos hacer `dpkg-reconfigure mysql-server` que perderemos todo lo que haya en la BD pero también hay otros métodos -menos agresivos- para recuperar la contraseña del root). Luego para instalar PHPMyAdmin haremos `apt-get install phpmyadmin` y prestar atención que nos pedirá la clave de acceso para entrar en la base de datos y crear una clave de acceso para entrar en la aplicación vía un navegador. Luego podremos poner en la URL de nuestro navegador *http://localhost/phpmyadmin*, nos solicitará el usuario (root generalmente) y el passwd que hemos introducido y ya podremos gestionar el servidor de bases de datos MySQL.

1.9.3. Otros servidores httpd

Lighttpd es un servidor web (con licencia BSD) diseñado para ser rápido, seguro, flexible, que implementa la mayoria de los estándares y está optimizado para entornos donde la velocidad es muy importante (consume menos CPU/-RAM que otros servidores) y es muy apropiado para cualquier servidor que tenga que dar soporte a grandes cargas. Entre sus principales características están la de Virtual hosting, redirecciones http y reescrituras de URL, dar soporte a CGI, SCGI y FastCGI, PHP, Ruby, Python entre otros y además con consumo de memoria constante.

Su instalación en Debian es `apt-get install lighttpd`, y si tenemos apache sobre el puerto 80 nos dará un error. Para ello debemos editar el archivo `/etc/lighttpd/lighttpd.conf` y cambiar la línea `server.port = 8080` y reiniciar `service lighttpd start`. Desde el navegador se puede hacer *http://localhost:8080index.lighttpd.html* y veremos la página inicial de lighttpd. Por defecto Lighttpd tiene su directorio raíz en */var/www* (en Debian) y el archivo de configuración en */etc/lighttpd/lighttpd.conf*. Configuraciones adicionales están en */etc/lighttpd/conf-available* y puede ser habilitadas con el comando `lighttpd-enable-mod` el cual crea enlaces entre *conf-enabled* y *conf-available*, las cuales se pueden deshabilitar con `lighttpd-disable-mod`.

Para habilitar el servidor de FastCGI con la intención de ejecutar PHP deberemos instalar PHP-FPM con `apt-get install php5-fpm php5` y sobre el archivo */etc/php5/fpm/php.ini* quitar el comentario a `cgi.fix_pathinfo=1`. Luego deberemos activar el servidor PHP-FPM, por lo cual haremos una copia del archivo original y lo modificaremos:

```
cd /etc/lighttpd/conf-available/
cp 15-fastcgi-php.conf 15-fastcgi-php-spawnfcgi.conf
```

Modificar *15-fastcgi-php.conf* con:

```
# -*- depends: fastcgi -*-

# Start an FastCGI server for php
fastcgi.server += ( ".php" =>
        ((
                "socket" => "/var/run/php5-fpm.sock",
                "broken-scriptfilename" => "enable"
        ))
)
```

Para habilitar fastcgi deberemos cargar los módulos `lighttpd-enable-mod fastcgi` y `lighttpd-enable-mod fastcgi-php` lo cual crea los enlaces correspondientes que podemos ver con `ls -l /etc/lighttpd/conf-enabled`. Luego podemos reiniciar `service lighttpd force-reload`. Para visualizar si el servidor y FastCGI funciona creamos un archivo */var/www/info.php* con el siguiente contenido `<?php phpinfo(); ?>` y podremos visualizar (http://localhost:8080/info.php) la página de configuración de PHP donde indica como Server API = FPM/FastCGI.

Otro servidor muy utilizado actualmente es **Nginx** (http://nginx.org/) programado en C y licencia BSD. Sus funciones principales son como servidor web/proxy inverso de muy alto rendimiento (puede soportar más de 10.000 conexiones simultáneas) y también puede funcionar como proxy para protocolos de correo electrónico (IMAP/POP3). Es un servidor utilizado por grandes instalaciones (WordPress, Netflix, Hulu, GitHub, y partes de Facebook entre otros) y entre sus principales características están (además de servidor de archivos estáticos, índices y autoindexado y proxy inverso con opciones de caché) el balanceo de carga, tolerancia a fallos, SSL, FastCGI, servidores virtuales, streaming de archivos (FLV y MP4.8), soporte para autenticación, compatible con IPv6 y SPDY. Su instalación básica es simple y para su configuración se puede consultar la wiki de nginx a http://wiki.nginx.org/Configuration.

1.10. Servidor de WebDav

El nombre webDAV son las siglas de *Web Based Distributed Authoring and Versioning* (también se refiere al grupo de trabajo de Internet Engineering Task Force) y es un protocolo que permite que la web se transforme en un medio legible y editable y proporciona funcionalidades para crear, cambiar y mover documentos en un servidor remoto (típicamente un servidor web). Esto se utiliza sobre todo para permitir la edición de los documentos que envía un servidor web, pero puede también aplicarse a sistemas de almacenamiento generales basados en la web y a los que se puede acceder desde cualquier lugar. En este subapartado instalaremos un servidor WebDav sobre Apache. El proceso es el siguiente:

Enlace de interés

Sobre la integración de WebDav con Apache podéis consultar el artículo "WebDAV on Apache2" disponible en: http://www.debian-administration.org/articles/285

1) Verificar que tenemos instalado apache2 y si no realizar su instalación como hemos visto anteriormente y verificar que funciona (`apt-get install apache2`).

2) Habilitar los módulos de Apache que son necesarios para WebDav: `a2enmod dav_fs` y `a2enmod dav`.

3) Crear el directorio para el directorio virtual (podemos hacer por ejemplo `mkdir -p /var/www/webdav`) y permitir que Apache sea el propietario del directorio `chown www-data /var/www/webdav/`.

4) Crear el archivo `/etc/apache2/sites-available/webdav.conf` para definir la funcionalidad del servidor (en esta configuración estamos configurando todo el servidor como WebDav pero podría ser un servidor virtual y en modo SSL para mayor seguridad):

```
<VirtualHost *:80>
    ServerAdmin adminpSySDW.nteum.org
    DocumentRoot /var/www/webdav/
    ErrorLog /var/log/apache2/webdav-error.log
    CustomLog /var/log/apache2/webdav-access.log combined
   <Directory /var/www/webdav>
        Options Indexes MultiViews
        AllowOverride None
        Order allow,deny
```

```
        allow from all
        DAV On
        AuthName “Restricted WeDav Area”
        AuthType Basic
        AuthUserFile /etc/apache2/htpasswd
        AuthGroupFile /dev/null
        require valid-user
    </Directory>
</VirtualHost>
```

Se puede comprobar que la configuración es correcta con la orden `apache2ctl configtest`.

5) Como se ha hecho anteriormente, se crean los usuarios (`htpasswd [-c] /etc/apache2/htpasswd` usuario) indicando el -c si es el primer usuario a crear.

6) Se reinicia Apache para que lea la configuración `/etc/init.d/apache2 reload` y ya nos podemos conectar a `http://localhost`, previa autentificación.

7) Desde un GNU/Linux podemos probar la funcionalidad del servidor abriendo el nautilus (gestor de ficheros) y desde el menu *File->Connect to Server* podemos seleccionar Servidor WebDav introduciendo los datos (IP, directorio, usuario, passwd) y tendremos acceso como si de una carpeta local se tratara.

8) Desde MacOS podemos utilizar el mismo procedimiento que el anterior desde el gestor de archivos o instalar un cliente específico (igualmente para Windows). El más recomendado para ello es CyberDuck (http://cyberduck.io/) que tiene licencia GPL y es una excelente aplicación (soporta múltiples protocolos) y muy fácil de configurar.

9) Otra forma de probarlo es con un cliente WebDav (en modo texto), por ejemplo Cadaver (http://www.webdav.org/cadaver), con `apt-get install cadaver`. Después, nos conectamos al servidor con `cadaver IP-nombre del servidor` y después de autentificarnos, podemos crear un directorio (`mkdir`), editar un archivo, listar un directorio (`ls`), cambiar de directorio (`cd`), cambiar los permisos de ejecución (`chexec`), borrarlo (`rm`), etc.

En muchas ocasiones, y dado que estaremos haciendo transferencias de archivos, es importante preservar la privacidad por lo cual sería adecuado trabajar con WebDav pero sobre SSL. Su configuración no implica mayores complicaciones y veremos una forma diferente de generar los certificados (serán autofirmados -hasta que podamos tener nuestra propia entidad certificadora-) para un dominio en particular, en nuestro caso webdav.world. Para ello hacemos:

1) Nos cambiamos al directorio donde almacenaremos los certificados: `cd /etc/ssl/private`. Hacemos la petición del certificado:

```
openssl req -config /etc/ssl/openssl.cnf -new -out webdav.csr
```

Este comando nos pedirá un *passwd* y una serie de información que quedará en el certificado pero la más importante es Common Name (CN) que será donde validará el certificado (en nuestro caso webdav.world). Podemos verificar la petición con:

```
openssl req -in /etc/ssl/private/webdav.csr -noout -text
```

2) Creamos la llave (*key*):

```
openssl rsa -in privkey.pem -out webdav.key
```

3) Firmamos:

```
openssl x509 -in webdav.csr -out webdav.crt -req -signkey webdav.key
-days 3650
```

Podemos verificarlo con:

```
openssl x509 -noout -in /etc/ssl/private/webdav.crt -text
```

4) Generamos en certificado en formato DER:

```
openssl x509 -in webdav.crt -out webdav.der.crt -outform DER
```

Se puede verificar con:

```
openssl x509 -in /etc/ssl/private/webdav.der.crt -inform der
-noout -text
```

5) Ahora generamos el archivo de configuración de apache a partir del anterior:

```
cd /etc/apache2/sites-available; cp webdav.conf webdav-ssl.conf
```

Modificamos para incluir las siguiente cuatro líneas al inicio y modificar el VirtualHost:

<VirtualHost *:443>
ServerName webdav.world
SSLEngine on
SSLCertificateFile /etc/ssl/private/webdav.crt
SSLCertificateKeyFile /etc/ssl/private/webdav.key
...

Solo nos resta activar el sitio (`a2ensite webdav-ssl.conf`), reiniciar apache (`service apache2 restart`) y verificar que funciona en la dirección https://webdav.world/ previa aceptación del certificado.

1.11. Servicio de *proxy*: Squid

Un servidor *proxy* (*proxy server*, PS) se utiliza para ahorrar ancho de banda de la conexión de red, mejorar la seguridad e incrementar la velocidad para obtener páginas de la red (*web-surfing*).

Squid es un *proxy caching server* para Web y da soporte a los protocolos HTTP, HTTPS, FTP, entre otros. Éste reduce el ancho de banda y mejora los tiempo de respuesta almacenado en caché y reutilizando las páginas más frecuentes. Squid tiene un extenso conjunto de reglas de control que permiten optimizar el flujo de datos ente el cliente y el servidor aportando seguridad y control y encaminando las peticiones correctamente, permitiendo su control y mejorando la utilización del ancho de banda de la red. Squid tiene diferentes modos de funcionamiento pero como los más importantes podemos mencionar: *Forward Proxy* (es el modo básico sobre el cual se configuran todo lo demás),

Transparent o *Interception Proxy* (permite incluir un proxy en una red sin que los clientes tengan que configurar nada) y *Reverse Proxy* -o *Accelerator-mode*- (permite ejecutar squid para mejorar la respuesta de una granja de servidores web).

Para instalar Squid como *proxy-cache* (http://www.squid-cache.org/) en Debian hacemos `apt-get install squid3` y editaremos el archivo de configuración */etc/squid3/squid.conf* para realizar una configuración básica. Se debe tener en cuenta que squid es muy potente pero esto se traduce en una configuración que puede ser compleja; como idea simplemente considerar que el archivo de configuración, que está muy bien explicado, tiene aproximadamente 5700 líneas (no obstante si ejecutamos

```
grep -v "^#" /etc/squid3/squid.conf | awk '$1 != "" {print $0}'
```

podremos ver que la configuración básica son unas 40 líneas mientras que el resto son comentarios y opciones comentadas).[16][17]

Definir la ACL (access control list) para habilitar la red/ip que deseamos hacer de proxy, en la línea 718 agregar: `acl lan src 192.168.1.0/24`

Permitir el acceso, en la línea 842 (aprox.) agregar: `http_access allow lan`
Cambiar el puerto de acceso (linea 1136), por defecto está `http_port 3128` y se puede dejar que es el estándar para squid o poner el que se prefiera (por ejemplo 8080).

Agregar las reglas de control, en la línea 3449 (aprox.):
request_header_access Referer deny all
request_header_access X-Forwarded-For deny all
request_header_access Via deny all
request_header_access Cache-Control deny all

Definimos el nombre visible, línea 3748 (aprox.): `visible_hostname remix.world`

Modificamos visibilidad de la IP, línea 5541 (aprox.): `forwarded_for off`

Finalmente reiniciamos el servidor: `service squid3 restart`
Nos dará un mensaje similar a:

[ok] Restarting Squid HTTP Proxy 3.x: squid3[....] Waiting.....done.
(tardará unos segundos...)

Con el servidor en marcha podemos configurar los navegadores para que utilicen el proxy, esto es por ejemplo en IceWeasel/Firefox en el apartado de *Preferences*/*Options->Advanced->Network->Proxy* e introducir los datos de nuestro servidor. Sobre Chrome por ejemplo en Windows, se debe ir a las *Settings->Advanced->Change proxy setting->Connections->Lan Settings* e introducir los datos de nuestro servidor.

Otra de las opciones que permite Squid es actuar como *Reverse Proxy* que es un tipo de *Proxy* donde los datos se recuperan de un/unos servidor/es y de devuelven a los clientes como si se originaran en el *proxy* quedando los servidores ocultos a los clientes como se muestra en la Wikipedia*. Esto permite hacer políticas de balanceo de carga y que las peticiones estén en un único dominio a pesar que internamente pueden estar distribuidas en diversos servidores. Para su configuración debemos hacer:

***http://upload.wikimedia.org /wikipedia/commons/6/67 /Reverse_proxy_h2g2bob.svg**

Especificar la dirección del servidor interno, en la línea 1136 modificar:
`http_port 80 defaultsite=192.168.1.33`
Agregar cache_peer, en la línea 1937 (aprox.) agregar:
`cache_peer 192.168.1.33 parent 80 0 no-query originserver`
Cambiar la acl para permitir cualquier conexión, en la línea 842 (aprox.) modificar `http_access allow all`
Finalmente reiniciamos el servidor: `service squid3 restart`

Cuando nos conectemos a la IP/dominio del servidor *Proxy* en realidad veremos las páginas enviadas por el servidor 192.168.1.33. Como prueba de concepto (si queremos hacer la prueba con una única máquina) podemos poner que en lugar del servidor 192.168.1.33 poner un servidor externo (p. ej., debian.org o su IP) y cuando pongamos como URL la de nuestro dominio visualizaremos la página de debian.org.

Otras de las configuraciones ampliamente utilizadas es como *interception proxy* (o *transparent proxy*) que intercepta la comunicación normal a la capa de red sin necesidad de configuraciones específicas en el cliente y por lo cual no sabrán que están detrás de un *proxy*. Generalmente un *transparent proxy* esta normalmente localizado entre el cliente e Internet con el *proxy* haciendo las funciones de *router* o *gateway*. Es habitual en las instituciones que desean filtrar algún tráfico de sus usuarios, ISP para hacer caché y ahorrar ancho de banda o países que controlan el acceso a determinados sitios por parte de sus ciudadanos. Para implementarlo sobre una institución lo habitual es disponer de una máquina con dos interfaces de red, una conectada a la red interna y otra hacia la red externa. Los pasos para su configuración serán básicamente los descritos en http://wiki.squid-cache.org/ConfigExamples/Intercept/LinuxDnat:

Sobre squid.conf:

Modificar la `acl/http_access` sobre las IP, IP/Mask permitidas como en el primer caso.

Configurar el puerto/modo como `http_port 3129 transparent` o en Squid 3.1+ se deberá utilizar http_port 3129 intercept para interceptar paquetes DNAT.

Sobre /etc/sysctl.conf modificar:

Permitir el packet forwarding: net.ipv4.ip_forward = 1

Controlar el source route verification: net.ipv4.conf.default.rp_filter = 0

: No aceptar source routing: net.ipv4.conf.default.accept_source_route = 0

Considerando que la IP de Proxy está en la variable SQUIDIP y el puerto esta en SQUIDPORT incluir las siguientes reglas de DNAT:

```
iptables -t nat -A PREROUTING -s $SQUIDIP -p tcp --dport 80 -j ACCEPT
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination
    $SQUIDIP:$SQUIDPORT
iptables -t nat -A POSTROUTING -j MASQUERADE
iptables -t mangle -A PREROUTING -p tcp --dport $SQUIDPORT -j DROP
```

1.11.1. Proxy SOCKS

SOCKS (abreviación de SOCKetS) es un protocolo de Internet (en el modelo OSI estaría en una capa intermedia entre la de aplicación y la de transporte) que permite a las aplicaciones en modo cliente-servidor atravesar en forma transparente un *firewall* de red. Los clientes que hay detrás de un *firewall*, los cuales necesitan acceder a los servidores del exterior, pueden conectarse en su lugar a un servidor *proxy* SOCKS. Tal servidor *proxy* controla qué cliente puede acceder al servidor externo y pasa la petición al servidor. SOCKS puede ser usado también de la forma contraria, permitiendo a los clientes de fuera del *firewall* (clientes externos) conectarse a los servidores de dentro del *firewall* (servidores internos).[26][27]

Se debe tener en cuenta que SOCKS solo sirve en modo cliente/servidor por o cual un usuario debe tener instalado un cliente SOCKS, ya sea en la aplicación (como Firefox, Chrome) o dentro de la pila TCP/IP desde donde el software del cliente redirige los paquetes en un túnel SOCKS. El procedimiento habitual comienza cuando el cliente SOCKS (p. ej., interno en una red privada) inicia una conexión a un servidor SOCKS (el protocolo SOCKS permite la autentificación y el registro de las solicitudes de conexión) y este (servidor SOCKS) actuará como cliente IP para la solicitud de conexión (del cliente interno en su nombre) lo cual significa que el servidor externo solo será consciente de las peticiones del servidor SOCKS (por lo cual actuará como *proxy forwarding*).

La pregunta habitual es si SOCKS es diferente a resolver el problema de acceso externo mediante NAT. La respuesta es: sí, es diferente, ya que los paquetes en NAT solo modifican las direcciones (p. ej., cambian las IP privadas por las pública del *router* como ocurre en un *router* ADSL) y el servidor recibe/contesta las peticiones. La sesión IP se establece directamente desde el cliente al servidor y el router/FW solo modifica/filtra el paquete pero no hay autenticación ni inspección del paquete/aplicación/datos (se podría hacer pero es complejo). La ventajas de SOCKS radican en que provee autentificación para protocolos que no lo permiten, puede traspasar el *routing* por defecto de una red interna, si bien HTTP y Telnet soportan autentificación por firewall (p. ej., utilizando *Authenticated Proxy* on a Cisco *firewall*) los protocolos encriptados nunca pueden ser autentificados por un FW en cambio SOCKS si puede hacerlo. No obstante existen desventajas ya que el cliente debe tener interface a SOCKS, el SO cliente debe tener interface a SOCKS (para interceptar el tráfico y reenviarlo al SOCKS *proxy*) y necesitaremos un servidor SOCKS específico para este fin.

Un caso de uso habitual es si consideramos que estamos en un punto de conexión inalámbrica abierta (p. ej., Wifi) y no se desea enviar datos de navegación sobre texto plano o se quiere acceder a una página web filtrada por router/FW perimetral. Una solución muy simple es utilizando SSH (que incluye un servidor SOCKS) que puede cifrar todo el tráfico de navegación por la web y redireccionarlo a través de un equipo de confianza cuando se está en algún otro punto de la red. Para ello deberemos disponer de un servidor SSH para que actúe como representante en un ordenador remoto (que le permita conectarse a él a través de SSH) y un cliente de SSH en el equipo que está utilizando. Lo que se hará con un proxy es la creación de un *middle-person* entre el usuario e Internet. El navegador hará las solicitudes de páginas web al servidor *proxy*, que controla la solicitud y obtiene la página desde internet y las devuelve al cliente. El sitio web en realidad piensa que la solicitud proviene del servidor *proxy*, no del equipo que la ha originado ocultando las dirección IP de origen. Además, la conexión entre el ordenador y el *proxy* que pasa a través de SSH es cifrada y esto evita que alguien pueda obtener los paquetes desde la Wifi (*sniffers* de wifi) en el sitio de la conexión.

Para su configuración desde donde nos conectamos debemos tenemos acceso a un servidor SSH y sobre el que crearemos un túnel que pasará el tráfico web entre nuestra máquina local y el proxy SOCKS sobre SSH. Para ello ejecutamos sobre nuestra máquina `ssh -ND 9999 login@remote-server.org` donde deberemos reemplazar login@remote-server.org con el usuario y nombre o IP del servidor remoto. Lo que está haciendo este comando es un *port forwarding* a través del puerto 9999 (puede ser cualquier otro pero conviene que sea superior a 1024 para evitar que solo lo pueda hacer root) y la conexión se reenvía a través de una canal seguro donde el protocolo de aplicación se utiliza para determinar dónde conectar desde la máquina remota. Actualmente OpenSSH soporta los protocolos SOCKS4-5 y por ello actuará como un servidor SOCKS. A continuación se solicitará el passwd y una vez autenticado no pasará NADA (el -N indica que no abra un prompt interactivo pero continuará funcionando). Si por el *firewall* solo podemos salir por e puerto 443 por ejemplo deberíamos configurar el ssh server para escuchar por el puerto 443 y en su lugar ejecutar `ssh -ND 9999 login@remote-server.org -p 443`. Ahora es necesario configurar el cliente para conectarse al proxy, Por ejemplo Firefox: *Options ->Advanced ->Network ->Connection* y seleccionar SOCKS, como nombre del servidor *localhost* (o su nombre real si tiene) y el puerto (9999) y guardar los ajustes y verificar que podemos navegar sin problemas. Se puede utilizar el plugin FoxyProxy (https://addons.mozilla.org/es/firefox/addon/foxyproxy-standard/) para Firefox que permite cambiar entre el proxy y la conexión directa en función del sitio o de un control. Como medida adicional (de anonimato) se puede configurar el servidor *proxy* para resolver peticiones DNS en lugar del método habitual haciendo en Mozilla Firefox poniendo como URL: `about:config` y modificando `network.proxy.socks_remote_dns=true`. También para conexiones lentas se puede utilizar la opción -C de ssh para utilizar la compresión de SSH por gzip. En Thunderbird u otros clientes la configuración es similar.

Si el túnel deja de funcionar (suele ocurrir en redes muy ocupadas) se puede utilizar el paquete `autossh` en lugar del `ssh` para establecer la conexión que se encargará de mantener el túnel abierto reiniciando automáticamente la conexión. Otro paquete interesante es `tsocks` (http://tsocks.sourceforge.net/) que se puede utilizar cuando el cliente que deseamos utilizar no soporta el protocolo SOCKS. `tsocks` monitoriza la llamada de inicio de sesión de una aplicación (*connect*) y redirecciona la comunicación hacia el *server* SOCKS sin que la aplicación tenga ninguna información. Para ello se debe instalar tsocks y configurar el proxy SOCKS que deberá utilizar en el fichero */etc/tsocks.conf* indicándole los valores (p. ej., server = 127.0.0.1, server_type = 5, server_port = 9999). Luego bastará con llamar a la aplicación con `tsocks aplicación` o simplemente el comando que abrirá una nueva shell redirigida al proxy y por lo cual todo lo que se ejecute allí será enviado al proxy SOCKS.

1.12. OpenLdap (Ldap)

LDAP significa *Lightweight Directory Access Protocol* y es un protocolo para acceder a datos basados en un servicio X.500. Este se ejecuta sobre TCP/IP y el directorio es similar a una base de datos que contiene información basada en atributos. El sistema permite organizar esta información de manera segura y utiliza réplicas para mantener su disponibilidad, lo que asegura la coherencia y la verificación de los datos accedidos-modificados.

El servicio se basa en el modelo cliente-servidor, donde existen uno o más servidores que contienen los datos; cuando un cliente se conecta y solicita información, el servidor responde con los datos o con un puntero a otro servidor de donde podrá extraer más información; sin embargo, el cliente solo verá un directorio de información global [28, 19]. Para importar y exportar información entre servidores `ldap` o para describir una serie de cambios que se aplicarán al directorio, el formato utilizado se llama LDIF (LDAP *Data Interchange Format*). LDIF almacena la información en jerarquías orientadas a objetos que luego serán transformadas al formato interno de la base de datos. Un archivo LDIF tiene un formato similar a:

```
dn: o = UOC, c = SP o: UOC
objectclass: organization
dn: cn = Remix Nteum, o = UOC, c = SP
cn: Remix Nteum
sn: Nteum
mail: nteumuoc.edu
objectclass: person
```

Cada entrada se identifica con un nombre indicado como `DN` (*Distinguished Name*). El DN consiste en el nombre de la entrada más una serie de nombres que lo relacionan con la jerarquía del directorio y donde existe una clase de objetos (`objectclass`) que define los atributos que pueden utilizarse en esta entrada. LDAP provee un conjunto básico de clases de objetos: **grupos** (inclu-

ye listas desordenadas de objetos individuales o grupos de objetos), **localizaciones** (tales como países y su descripción), **organizaciones y personas**. Una entrada puede, además, pertenecer a más de una clase de objeto, por ejemplo, un individuo es definido por la clase persona, pero también puede ser definido por atributos de las clases `inetOrgPerson`, `groupOfNames` y `organization`.

La estructura de objetos del servidor (llamado *schema*) determina cuáles son los atributos permitidos para un objeto de una clase (que se definen en el archivo `/etc/ldap/schema` como `inetorgperson.schema, nis.schema, opeldap.schema, corba.schema`, etc.). Todos los datos se representan como un par atributo = valor, donde el atributo es descriptivo de la información que contiene; por ejemplo, el atributo utilizado para almacenar el nombre de una persona es `commonName`, o cn, es decir, una persona llamada Remix Nteum se representará por cn: Remix Nteum y llevará asociado otros atributos de la clase persona como `givenname: Remix surname: Nteum mail: nteum@uoc.edu`. En las clases existen atributos obligatorios y optativos y cada atributo tiene una sintaxis asociada que indica qué tipo de información contiene el atributo, por ejemplo, `bin` (*binary*), `ces` (*case exact string*, debe buscarse igual), `cis` (*case ignore string*, pueden ignorarse mayúsculas y minúsculas durante la búsqueda), `tel` (*telephone number string*, se ignoran espacios y '-') y `dn` (*distinguished name*). Un ejemplo de un archivo en formato LDIF podría ser:

```
dn: dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit


dn: ou = groups, dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
ou: groups


dn: ou = people, dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
ou: people


dn: cn = Remix Nteum, ou = people, dc = UOC, dc = com
cn: Remix Nteum
sn: Nteum
objectclass: top
objectclass: person
objectclass: posixAccount
objectclass: shadowAccount
   uid:remix userpassword:{crypt}p1pss2ii(0pgbs*do& = )eksd uidnumber:104
   gidnumber:100
   gecos:Remix Nteum
   loginShell:/bin/bash
   homeDirectory: /home/remix
   shadowLastChange:12898
   shadowMin: 0
   shadowMax: 999999
   shadowWarning: 7
   shadowInactive: -1
   shadowExpire: -1
   shadowFlag: 0
```

```
dn:
cn = unixgroup, ou = groups, dc = UOC, dc = com
objectclass: top
objectclass: posixGroup
cn: unixgroup
   gidnumber: 200
   memberuid: remix
   memberuid: otro-usuario
```

Las líneas largas pueden continuarse debajo comenzando por un espacio o un tabulador (formato LDIF). En este caso, se ha definido la base DN para la institución `dc = UOC, dc = com`, la cual contiene dos subunidades: `people` y `groups`. A continuación, se ha descrito un usuario que pertenece a `people` y a `group`. Una vez preparado el archivo con los datos, este debe ser importado al servidor para que esté disponible para los clientes LDAP. Existen herramientas para transferir datos de diferentes bases de datos a formato LDIF [19]. Sobre Debian, se debe instalar el paquete `slapd` y `ldap-utils`, que es el servidor de OpenLdap y un conjunto de utilidades para acceder a servidores locales y remotos. Durante la instalación, solicitará un passwd para gestionar la administración del servicio, se generará una configuración inicial y se pondrá en marcha el servicio que se puede verificar con el comando `slapcat` que tomando la información disponible (FQDN de */etc/hosts*) generará algo como:

```
   dn: dc=nteum,dc=org
   objectClass: top
   objectClass: dcObject
   objectClass: organization
   o: nteum.org
   dc: nteum
...

   dn: cn=admin,dc=nteum,dc=org
   objectClass: simpleSecurityObject
   objectClass: organizationalRole
   cn: admin
   description: LDAP administrator     userPassword:: e1NTSEF9TU9jVE1qWllPVFBmd2FiZWJtSjcrY0pYd2wvaTk5aUc=
...
```

Con los comandos `ldapadd` y `ldapdelete` se pueden agregar/borrar registros de la tabla del servicio utilizando normalmente archivos de texto (sobre todo para el `add`) donde estén definidos los nuevos registros. Si bien no es un procedimiento complejo para gestionar el servidor puede ser más adecuado, en los primeros pasos, utilizar algunas de las aplicaciones que permiten gestionar el servidor de un forma más amigable como por ejemplo `phpldapadmin` (administración por medio apache+php), `jxplorer` (aplicación en java), `gosa` o `lat` (todos ellos en la mayoría de las distribuciones). En nuestro caso instalaremos `phpldapadmin` que combina simplicidad y permite gestionar la mayoría de las opciones del servidor Ldap.

Para instalarlo ejecutamos `apt-get install phpldapadmin` el cual ya reiniciará el servidor apache pero sino lo hace podemos reiniciarlo con la orden `/etc/init.d/apache2 restart`. Para conectarnos en un navegador introducimos `http://localhost/phpldapadmin/`. Saldrá la página de bienvenida y en la banda izquierda podremos hacer el *login* (si el *ServerName* de

apache no coincide con el LDAP no pedirá como usuario DN por lo cual le deberemos indicar el correcto cn=admin,dc=nteum,dc=org y la contraseña del servidor LDAP). Una vez conectado al servidor, seleccionamos la raíz (dc=nteum,dc=org) y seleccionamos entre los *templates* que aparecen una *Organizational Unit (ou)* a la que llamaremos `users`, repetimos los pasos (opción `create new entry` desde la raíz) y creamos otra *ou* llamada `groups`. Ahora solamente nos queda crear los usuarios y los grupos y asignar los usuarios a sus grupos. Dentro de la unidad organizativa `groups` crearemos los grupos `ventas` (gid=1001) y `compras` (gid=1002) y en la unidad organizativa `users` crearemos los usuarios `juan pirulo` (uid=1001, ventas, id=jpirulo) y `ana pirulo` (uid=1002, compras, id=apirulo). Para los grupos seleccionamos la OU `groups`, hacemos `create new child` y seleccionamos `Posix Group`. Creamos los dos grupos indicados pero deberemos modificar el `gid` ya que los asigna por defecto a partir de 1.000 y nosotros los queremos diferentes. Repetimos la operación en `users` seleccionando `User Account`. Aquí nos pedirá los datos de los usuarios (nombre, apellido, grupo, contraseñas, *home directory*, *shell*, etc.) y luego deberemos cambiar el `uid`, ya que los asigna por defecto. Luego podemos observar nuevamente el contenido ejecutando `slapcat` los nuevos datos generados. Phpldapadmin se puede configurar con el archivo */usr/share/phpldapadmin/config/config.php* pero en la mayoría de los casos la configuración por defecto ya es operativa (en algunos casos puede ser necesario adaptar la línea $servers->setValue('server','base',array('dc=nteum,dc=org')); para adecuar a los datos de nuestro servidor).

Para poder configurar un cliente tenemos que instalar los paquetes `slapd` y `ldap-utils` con `apt-get install slapd ldap-utils`, que nos solicitará una serie de información: URI del servidor (ldap://192.168.1.33), el DC (dc=nteum,dc=org), la versión (3), el usuario administrativo (cn=admin, dc=nteum,dc=org) y el `passwd`, nos informará que hagamos el cambio manual de nsswitch (ok), permitir a PAM cambiar los `password` locales (Yes), `enforces login` (No), `admin acount suffix` (cd=admin,dc=nteum, dc=org), y finalmente el `passwd` nuevamente.

A continuación deberemos modificar el archivo */etc/nsswitch.conf* con:

```
passwd: compat ldap
group: compat ldap
shadow: compat ldap
...
netgroup: ldap
```

Por último deberemos modificar el archivo */etc/pam.d/common-password* (quitando de la línea el 'use_authtok')

```
password [success=1 user_unknown=ignore default=die] pam_ldap.so
try_first_pass
```

y en */etc/pam.d/common-session* agregar al final (para crear el home directory automáticamente):

`session optional pam_mkhomedir.so skel=/etc/skel umask=077`.

Luego deberemos reiniciar la máquina (`shutdown -r now`) y ya nos podremos conectar con los usuarios que hemos creado (jpirulo o apirulo) desde la interfaz gráfica.

1.13. Servicios de archivos (NFS, *Network File System*)

El sistema NFS permite a un servidor exportar un sistema de archivo para que pueda ser utilizado de forma interactiva desde un cliente. El servicio se compone básicamente de un servidor (básicamente representado por `nfsd*` y un cliente (representado por `rpc.mountd`) que permiten compartir un sistema de archivo (o parte de él) a través de la red. En la última versión de NFSv4 se incluyen una serie de *daemons* más como *idmapd*, *statd*, además de uan serie de módulos para las nuevas funcionalidades de esta versión. En Debian, instalad `apt-get install nfs-common` (será necesario el paquete *rpcbin* que como se comentó anteriormente es el nuevo *portmap* y generalmente ya está instalado) para el cliente, mientras que el servidor necesita `apt-get install nfs-kernel-server`. El servidor (en Debian) se gestiona a través del *script* `/etc/init.d/nfs-kernel-server` o simplemente con `service nfs-kernel-server start|stop`. El servidor utiliza un archivo (`/etc/exports`) para gestionar el acceso remoto a los sistemas de archivo y el control de los mismos. Sobre el cliente (u otro usuario a través de `sudo`), el root puede montar el sistema remoto a través del comando:

```
mount -t nfs Ipserver:directorio-remoto directorio_local
```

y a partir de este momento, el directorio-remoto se verá dentro de directorio local (este debe existir antes de ejecutar el `mount`). Esta tarea en el cliente se puede automatizar utilizando el archivo de *mount* automático (`/etc/fstab`) incluyendo una línea; por ejemplo:

`remix.world:/home /home nfs defaults 0 0`.

Esta sentencia indica que se montará el directorio `/home` del *host* `remix.world` en el directorio local `/home`. Además, este sistema de archivo se montará con los parámetros por defecto (ver `man mount` apartado `mount options for ntfs` y man nfs para opciones específicas para NFSv4). Los últimos dos ceros indican que el sistema de archivos no debe ser `dumped` y que no se activará el `fsck` sobre él. El archivo `/etc/exports` sirve de ACL (lista de control de acceso) de los sistemas de archivo que pueden ser exportados a los clientes. Cada línea contiene un sistema de ficheros (*filesystem*) por exportar seguido de los clientes que lo pueden montar, separados por espacios en blanco. A cada

cliente se le puede asociar un conjunto de opciones para modificar el comportamiento (consultad *man exports* para ver un lista detallada de las opciones). Un ejemplo de esto podría ser:

```
# Ejemplo de /etc/exports
/           master(rw) trusty(rw,no_root_squash)
/projects   proj*.local.domain(rw)
/usr        .local.domain(ro) @trusted(rw)
/pub        (ro,insecure,all_squash)
/home       195.12.32.2(rw,no_root_squash) www.first.com(ro)
/user       195.12.32.2/24(ro,insecure)
/home       192.168.1.0/24(rw,sync,fsid=0,no_root_squash,no_subtree_check)
```

La primera línea exporta el sistema de archivos entero (`/`) a `master` y `trusty` en modo lectura/escritura. Además, para `trusty` no hay *uid squashing* (el root del cliente accederá como root a los archivos root del servidor, es decir, los dos root son equivalentes a pesar de ser de máquinas diferentes y se utiliza para máquinas sin disco). La segunda y tercera líneas muestran ejemplos de '*' y de *netgroups* (indicados por `@`). La cuarta línea exporta el directorio `/pub` a cualquier máquina del mundo, solo de lectura, permite el acceso de clientes NFS que no utilizan un puerto reservado para el NFS (opción `insecure`) y todo se ejecuta bajo el usuario *nobody* (opción `all squash`). La quinta línea especifica un cliente por su IP y en la sexta, se especifica lo mismo pero con máscara de red (/24) y con opciones entre paréntesis sin espacio de separación. Solo puede haber espacios entre los clientes habilitados. La última línea exporta el directorio */home* a todas las máquinas de la red 192.168.0.* en modo syncronizado, de lectura/escritura, con acceso del root remoto, fsid es la identificación de sistema de archivo y no_subtree_check indica que no se hará la verificación de la ruta/archivo en una petición sobre el servidor.

Dos comandos útiles para trabajar con el nfs son el `exportfs` (muestra y nos permite actualizar las modificaciones que se hayan realizado sobre */etc/exports*) y `nfsiostat/nfsstat` que nos permitirá obtener estadísticas de funcionamiento sobre NFS y observar su funcionamiento.

1.14. Servidor de wiki

Un (o una) **wiki** (del hawaiano *wiki wiki*, "rápido") es un sitio web colaborativo que puede ser editado por varios usuarios que pueden crear, editar, borrar o modificar el contenido de una página web, de forma interactiva, fácil y rápida; dichas facilidades hacen de una wiki una herramienta eficaz para la escritura colaborativa. La tecnología wiki permite que páginas web alojadas en un servidor público (las páginas wiki) sean escritas de forma colaborativa a través de un navegador, utilizando una notación sencilla para dar formato, crear enlaces, etc. y conservando un historial de cambios que permite recuperar de manera sencilla cualquier estado anterior de la página. Cuando alguien edita

una página wiki, sus cambios aparecen inmediatamente en la web, sin pasar por ningún tipo de revisión previa. Wiki también se puede referir a una colección de páginas hipertexto, que cualquier persona puede visitar y editar (definición de Wikipedia). Debian tiene su wiki en http://wiki.debian.org/ o también Apache en http://wiki.apache.org/general/ y ambas están basadas en **MoinMoin**. MoinMoin es una *Python WikiClone* que permite inicializar rápidamente su propia wiki y solo se necesitan un servidor de web y el lenguaje Python instalado. En la web de MoinMoin se encuentran las instrucciones detalladas para instalar MoinMoin, pero hay dos formas principales de hacerlo: instalación rápida e instalación de servidor.

Enlace de interés

Para saber más sobre MoinMoin podéis visitar su página web en: http://moinmo.in.
En concreto encontraréis las instrucciones detalladas para instalar MoinMoin en: http://master19.moinmo.in/InstallDocs.

1.14.1. Instalación rápida

1) Descargar el paquete desde http://moinmo.in/MoinMoinDownload que será, por ejemplo, para la versión 1.9 `moin-1.9.7.tar.gz`. Si se quiere verificar la integridad del paquete, se puede hacer `md5sum moin-x.x.x.tar.gz` y verificar que coincidan el *hash* generado con el que existe en la página de descarga.

2) Desempaquetar MoinMoin `tar xvzf moin-x.x.x.tar.gz`. Esto creará un directorio `moin-x.x.x` en el directorio actual con los archivos en su interior.

3) Dado que MoinMoin está escrita en Python, es necesario utilizar el intérprete de Python:

```
cd moin-x.x.x; python wikiserver.py
```

Esta orden mostrará por pantalla los mensajes de ejecución del servidor. Entre esta información se mostrará la dirección IP sobre la cual está corriendo el servidor, que podrá ser algo como `http://127.0.0.1:8080`. Esta opción usa un servidor web interno, será accesible desde `http://localhost:8080/` y funcionará hasta que se presione `Ctrl-C` en el terminal.

1.14.2. Instalación de servidor

MoinMoin es una aplicación WSGI (*Web Server Gateway Interface*), por lo tanto, el mejor entorno para ejecutar Moin Moin es uno que permita WSGI como, por ejemplo, Apache con mod_wsgi. En Debian podemos instalar el módulo instalando `apt-get install libapache2-mod-wsgi`.

Enlace de interés

Las instrucciones para instalar WSGI para Apache y configurar MoinMoin en este caso se pueden encontrar en la siguiente dirección: http://moinmo.in/HowTo/ApacheWithModWSGI.

Instalación de MoinMoin

Para instalar MoinMoin descargar la última versión y descompactar el archivo (p. ej., `tar xvzf moin-1.9.7.tar.gz`) y luego hacer una `cd moin-1.9.7/` y a continuación ejecutar:

```
python setup.py install --force –record=install.log --prefix='/usr/local'
```

Para hacer un test simple:

```
cd /usr/local/share/moin/server
python test.wsgi
```

En el navegador introducir como URL localhost:8000 y veremos la página de test de WSGI.

Copiar la configuración: `cd /usr/local/share/moin`

```
cp server/moin.wsgi .
cp config/wikiconfig.py .
```

Agregar un archivo en */etc/apache2/conf.d/moin.conf* con el siguiente contenido:

MoinMoin WSGI configuration
you will invoke your moin wiki at the root url, like http://servername/FrontPage:
WSGIScriptAlias / /usr/local/share/moin/moin.wsgi
create some wsgi daemons - use these parameters for a simple setup
WSGIDaemonProcess moin user=www-data group=www-data processes=5 threads=10
maximum-requests=1000 umask=0007
use the daemons we defined above to process requests!
WSGIProcessGroup moin

Modificar el archivo */usr/local/share/moin/moin.wsgi* agregando al final del párrafo a2: sys.path.insert(0, '/usr/local/share/moin')

Modificar los permisos de los directorios/páginas: `cd /usr/local/share; chown –R www-data:www-data moin; chmod –R ug+rwX moin; chmod –R o-rwx moin`

Verificar que tengamos un site por defecto en apache (sino hacer `a2ensite default`) y reiniciar apache (`service apache2 restart`).

Si nos conectamos a la URL localhost tendremos la página inicial de MoinMoin. Para configurar el nombre de la wiki y el usuario administrador podemos editar el archivo */usr/local/share/moin/wikiconfig.py*, quitamos el comentario de `page_front_page = u"FrontPage"` e indicamos el nombre del administrador, por ejemplo, `superuser = [u"WikiAdmin", ]` reiniciando apache nuevamente. Para configurar el lenguaje, debemos entrar como administrador (WikiAdmin) (si no tenemos un usuario seleccionamos *login*, seleccionamos 'you can create one now' y lo creamos -debe coincidir con el que introducimos como superuser-). Luego podremos configurar el idioma desde `http://localhost/LanguageSetup?action=language_setup` y a partir de esta acción ya podremos comenzar a crear nuestra primera wiki (podéis acceder a información adicional en http://moinmo.in/HowTo y particularmente en http://moinmo.in/HowTo/UbuntuQuick).

Para configurar múltiples wikis, primero debéis copiar `config/wikifarm/*` de la distribución en el directorio `moin/config/`. Luego se deben seguir las

instrucciones anteriores para cada una de las wikis de la colección (*farm*) teniendo en cuenta que:

1) es necesario tener `data_dir` y `data_underlay_dir` separados para cada wiki,

2) si buscáis que compartan alguna configuración, entonces esta debe estar en `farmconfig.py` y las específicas deben estar en `mywiki.py`.

1.15. Gestión de copias de respaldo (*backups*)

Las copias de seguridad o copia de respaldo (*backup*) se refiere a una copia de los datos originales que se realiza con el fin de disponer de un medio de recuperarlos en caso de pérdida total o parcial debido a fallos en los dispositivos físicos, borrados por accidente, ataques informáticos, infectados por virus u otras causas que hacen que la información no existe o no es la deseada. El proceso de copia de seguridad se complementa con un proceso de restauración de los datos (*restore*) que puede ser total o parcial/selectivo, que permite devolver el sistema informático al punto en en cual fueron los datos almacenados. Esto puede significar la pérdida de información entre el momento que se realiza la copia de seguridad y el momento que se detecta que los datos no existen o están corrompidos por lo cual la política de planificación de copias de seguridad debe ser una de las actividades importantes en todo administrador de sistemas.

Se debe tener en cuenta que la pérdida de datos es habitual (y no por ello sin consecuencia y en algunos casos fatales) ya que de acuerdo a estadísticas recientes más del 60 % de los usuarios de Internet declaran haber sufrido una seria pérdida de datos en alguna ocasión y según un estudio de la Universidad de Texas, solo el 6 % de empresas con pérdida catastrófica de datos sobrevivirá, frente a un 43 % que nunca reabrirá su negocio y un 51 % que tendrá que cerrar en un plazo de 2 años. Para reafirmar más aún la necesidad de copias de seguridad, aquellos sistemas que contengan datos de carácter personal y estén sujetos a la legislación del país (p. ej., en España la LOPD -Ley Orgánica de Protección de Datos-) entre una de las obligaciones que deben cumplir las empresas/instituciones/individuos, es tener copias de seguridad para preservar los datos que tienen almacenados y que están sujetos a esta normativa.

1.15.1. Programas habituales de copias de respaldo

Existen diversas opciones para hacer copias de respaldo con diferentes objetivos, prestaciones e interfaces en todas las distribuciones GNU/Linux (p. ej., backintime, bacula, backup2l, backuppc, bup, chiark, dejadup, dirvish, flexbackup, lucky, rdiff, vbackup entre otras). Una de las más potentes es **Bacula***

*http://www.bacula.org

(http://blog.bacula.org/), que es una colección de herramientas para realizar copias de seguridad en una red. Bacula se basa en una arquitectura cliente/servidor que resulta muy eficaz y fácil de manejar, ya que presenta un conjunto muy amplio de características y es eficiente tanto para un conjunto de ordenadores personales como para grandes instalaciones. El paquete está formado por diferentes componentes, entre los más importantes se pueden encontrar:

- **Bacula-director**, *daemon* que gestiona la lógica de los procesos de *backup*;
- **Bacula-storage**, *daemon* encargado de manejar los dispositivos de almacenamiento;
- **Bacula-file**, *daemon* por medio del cual Bacula obtiene los ficheros que necesita para hacer el respaldo y que se deberán instalar en las máquinas fuente de los ficheros a respaldar, y
- **Bacula-console**, que permite interactuar con el servicio de *backup*.

Bacula soporta discos duros, cintas, DVD, USB y también diferentes bases de datos (MySQL, PostgreSQL y SQLite) pero como contrapartida, es necesario disponer de todos los paquetes instalados y su instalación y puesta a punto pueden resultar complejas.

Otro paquete interesante es **BackupPC***, que permite hacer copias de seguridad de disco a disco con una interfaz basada en la web. El servidor se ejecuta en cualquier sistema Gnu/Linux y admite diferentes protocolos para que los clientes puedan escoger la forma de conectarse al servidor. Este programa no es adecuado como sistema de copia de seguridad de imágenes de disco o particiones ya que no soporta copias de seguridad a nivel de bloque de disco; sin embargo, es muy simple de configurar y la posible intrusión sobre la red de ordenadores en la cual se desea hacer el respaldo es mínima. Este servidor incorpora un cliente *Server Message Block* (SMB) que se puede utilizar para hacer copia de seguridad de recursos compartidos de red de equipos que ejecutan Windows.

***http://backuppc.sourceforge.net/info.html**

Su instalación es sumamente sencilla haciendo, en Debian, `apt-get install backuppc`. Nos indicará que seleccionemos el servidor web (apache2), y nos indicará el usuario y *passwd* que ha creado, no obstante este *passwd*/usuario se pueden cambiar con `htpasswd /etc/backuppc/htpasswd backuppc`. Luego en nuestro navegador ponemos como URL http://localhost/backuppc y con el usuario/*passwd* accederemos a la página principal de la aplicación donde nos dará el estado del servidor y las opciones.

Hay diversas formas de configurar los clientes para hacer las copias de respaldo y dependerá del método para hacerlo y del sistema operativo. Para ello consultar el apartado de documentación como se configurará cada una de las transferencias (http://backuppc.sourceforge.net/faq/BackupPC.html).

En el caso de un sistema (remoto) Gnu/Linux, desde la interfaz de administración editar la configuración del mismo (*host* y *xfer*) y confirmar que se ha

definido como método `rsync` y el directorio directorio a realizar la copia. Como los respaldos se realizan a través de `rsync` en combinación con `ssh` es necesario que el usuario *backuppc* del servidor pueda acceder como root sin clave a la máquina remota. Para ello adoptar la entidad del usuario *backuppc* (`su - backuppc`) y generar las llaves (sin *passwd*) `ssh-keygen -t dsa` y luego utilizar el comando `ssh-copy-id root@cliente` para copiar la llave. Verificar que se puede acceder al usuario root del cliente a través de ssh y sin *passwd*. A partir de este punto, solo bastará seleccionar al equipo remoto a realizar el *backup* desde la interfaz de administración e iniciar una primera copia seleccionando el botón Comenzar copia de seguridad completa.

En sistemas Windows, la forma más simple de realizar *backups* es a través del protocolo SMB por lo cual se deberá sobre el sistema Windows ingresar como administrador y configurar el sistema para compartir carpetas que se deseen hacer la copia de seguridad o bien el disco duro completo (por ejemplo C:) definiendo un usuario/*passwd* para compartir este recurso. Desde la interfaz de administración editar la configuración del host remoto con Windows a respaldar (por su IP por ejemplo), el usuario y el método smb (no olvidar de hacer Save después de haber modificado estos datos). Defina en la pestaña *Xfer* el nombre del recurso compartido a respaldar en el equipo remoto y el nombre del usuario y clave de acceso de recurso compartido del equipo Windows remoto. A partir de este punto, solo bastará seleccionar al equipo remoto a respaldar desde la interfaz de administración e iniciar un primer respaldo seleccionando el botón Comenzar copia de seguridad completa.

Con Backuppc se puede definir la frecuencia de los respaldos totales y respaldos incrementales. De modo predeterminado el valor para los respaldos totales es cada 7 días y para los incrementales es cada día. Se recomienda utilizar un valor ligeramente inferior a los días. Por ejemplo, 6.97 en lugar de 7 días y 0.97 en lugar de 1 día para mejorar la granularidad del respaldo (recordar siempre de hacer *Save* después de modificar cada opción).

1.15.2. rdiff-backup y rdiff-backups-fs

Para administradores o usuarios avanzados existe la opción de `rediff-backup` que es un comando para la copia de seguridad de un directorio a otro y que también puede ser a través de una red. El directorio de destino poseerá una copia del directorio fuente pero también información adicional (diffs) para gestionar mejor las copias incrementales (aún de archivos antiguos). La aplicación también preserva subdirectorios, enlaces no simbólicos (hard links), archivos dev, permisos, propiedad (uid/gid), fechas de modificación, atributos extendidos y ACL y puede operar de forma eficiente a través de un *pipe* a *rsync* por ejemplo o utilizar rdiff-backup + ssh para realizar backups incrementales en lugar remoto transmitiendo solo las diferencias. También es habitual (y muy simple) tener un proveedor de servicios (Gdrive, Dropbox, etc) con un directorio sobre el ordenador local que será sincronizado sobre el *cloud* del

proveedor y hacer la copia rdiff-backup sobre este directorio para que luego se transmita al *cloud* del proveedor. La forma habitual de trabajo (después de instalado el comando) es muy simple `rdiff-backup fuente destino` y en el caso remoto /algun/dir-local a /algun/dir-remoto sobre la máquina hostname.org será

```
rdiff-backup /algun/dir-local hostname.org::/algun/dir-remoto
```

pero puede ser a la inversa

```
rdiff-backup user@hostname.org::/remote-dir local-dir
```

y también podrían ser sobre dos máquinas remotas

```
rdiff-backup -v5 -print-statistics user1@host1::/source-dir user2@host2::/dest-dir
```

Para recuperar un directorio local es simplemente a través de la copia y si es remoto (más ejemplos en http://www.nongnu.org/rdiff-backup/examples.html)

```
rdiff-backup -r now hostname.org::/remote-dir/file local-dir/file
```

Uno de los problemas de recuperar la información previamente guardada es que acceder a los archivos de copia de seguridad más reciente es fácil (solo se debe introducir el directorio de copia de seguridad) pero es complicado si deseamos acceder y navegar a través de las versiones anteriores. `rdiff-backup` permite acceder a ellos por un conjunto de instrucciones, que requieren un conocimiento preciso de los nombres de archivo y los tiempos de copia de seguridad y que puede ser complicado de recordar o seleccionar. `rdiff-backup-fs` (https://code.google.com/p/rdiff-backup-fs/) permite crear un sistema de archivo en espacio de usuario basado en la información de rdiff. Para ello se utiliza FUSE (*Filesystem in userspac*e - http://fuse.sourceforge.net/) que permite que cualquier usuario pueda montar el sistema de archivo guardado por rdiff y navegar por cada incremento y copia realizada.

Una alternativa para las copias de resguardo de un sistema remoto por ssh es utilizar `sshfs` (http://fuse.sourceforge.net/sshfs.html) que permite acceso en el espacio de usuario a un directorio remoto mostrándolo localmente por lo cual luego aplicando rdiff se puede hacer copias incrementales de este recurso remoto.

1.16. *Public Key Infrastructure* (PKI)

Por PKI (*Public Key Infrastructure*) se entiende un conjunto de hardware y software, políticas y procedimientos de seguridad que permiten la ejecución con

garantías de operaciones como el cifrado, la firma digital o el no repudio de transacciones electrónicas. El término PKI se utiliza para referirse tanto a la autoridad de certificación y al resto de componentes si bien a veces, de forma errónea, se utiliza para referirse al uso de algoritmos de clave pública. En una operación que se utilice PKI intervienen, además de quien inicia la acción y su destinatario un conjunto de servicios que dan validez a la operación y garantizan que los certificados implicados son válidos. Entre ellas podemos contar con la autoridad de certificación, la autoridad de registro y el sistema de sellado de tiempo. La infraestructura PKI utiliza procedimientos basados en operaciones criptográficas de clave pública, con algoritmos de cifrado bien conocidos, accesibles y seguros y es por ello que la seguridad está fuertemente ligada a la privacidad de la clave privada y la políticas de seguridad aplicadas para protegerla. Los principales usos de la PKI son entre otros: autenticación de usuarios y sistemas (*login*), identificación, cifrado, firma digital, comunicaciones seguras, garantía de no repudio.[21][22][24]

Como se vio anteriormente (para apache+SSL) la generación de certificados digitales es extremadamente necesario dentro de las necesidades habituales de los administradores y usuarios de los sistemas de información, por ejemplo, para acceder a una página SSL o para firmar un correo electrónico. Estos certificados se pueden obtener de las entidades certificadoras privadas como StartComm (https://www.startssl.com/) o Verisign (http://www.verisign.es/) que tienen algunos productos gratuitos (por ejemplo para firmar correos) pero, en su mayoría, los productos tiene un costo elevado. También podemos recurrir a utilizar GNUPG (https://www.gnupg.org/) que es *Open-Source* y para determinados fines es correcto pero no para otros o bien considerar entidades de certificación institucionales, por ejemplo en Cataluña IdCat (http://idcat.cat/) que nos permitirán obtener certificados de ciudadano (gratuitos) para firmado, encriptación, autentificación, no repudio pero no para servidores (Idcat si que puede expedir otro tipo de certificados pero solo es para la administración pública y universidades del país). En el presente apartado instalaremos y configuraremos una entidad certificadora raíz (y sub-entidades CA para los diferentes dominios de control de esta CA) basada en la aplicación TinyCA (si bien existen otros paquetes como OpenCA -https://pki.openca.org/- que son más potentes y escalables pero son más complejos en su configuración) que se adapta muy bien para pequeñas y medianas instituciones/empresas. Una entidad certificadora es la base de la infraestructura PKI y emite certificados para dar garantías de autenticidad de una comunicación, un sitio o una información. Cuando instalamos Apache hemos auto-firmado el certificado digital (es decir nosotros hemos hecho todos los pasos: petición, generación y firma) que codifican la comunicación pero ello no da garantía si no se verifica la firma del certificado auto-firmado. Para solucionar este problema, y sin recurrir a un proveedor público/privado, crearemos nuestra propia estructura de CA y distribuiremos por canales seguros a nuestros usuarios/clientes el certificado personal/de servidores y el certificado raíz de la CA para que los instalen en sus navegadores (como ya están los de StartComm, Verisign, Catcert/Idcat u otras entidades de certificación). De esta forma que cuando un sitio web, por

ejemplo, le presente al navegador un certificado digital para codificar la comunicación SSL, el navegador reconozca el sitio por el certificado raíz que tiene instalado y confíe en él (y no salga la típica ventana de advertencia de sitio inseguro) y manteniendo la privacidad de las comunicaciones. La utilización de estos certificados creado por esta CA puede ser varios: para firmar/encriptar un mail, para validar nuestros servidores SSL o para configurar la VPN entre otros.

La práctica habitual es tener una CA y crear Sub-CA (una por cada dominio) para que el sistema sea escalable y por lo cual la CA firmará la creación de nuevas Sub-CA y luego estas (su responsable) tendrán capacidad para firmar sus certificados y todas tendrán el mismo certificado raíz de la RootCA. (guía muy detallada en [25]) Una vez instalada (`apt-get install tinyca`), ejecutamos `tinyca2` y nos presentará la pantalla principal y indicaciones para crear una nueva CA que será la rootCA. Completamos la pantalla con los datos, por ejemplo, *Name=ROOTCA-nteum.org, Data for CA=ROOTCA-nteum.org, SP, passwd, BCN, BCN, NTEUM, NTEUM, adminp@sysdw.nteum.org, 7300, 4096, SHA-1* para todos los campos respectivamente. Cuando le damos a OK, aparecerá una nueva pantalla para configurar la rootCA. Seleccionar *"Certificate Signing/CRL Signing"*, "*critical*" y en Netscape *Certificate Type="SSL CA, S/MIME CA, Object Signing CA*.", si se desea poner una URL para revocar los certificados (lo cual puede ser una buena idea) se pueden rellenar los campos correspondientes, luego finalmente OK. Con ello se crearán los certificados y aparecerán la pantalla principal de tinyCA con 4 tabuladores donde indica *CA* (información sobre la CA activa), *Certificates* (muestra los certificados creados por la CA) *Keys* (muestra las llaves de los certificados) y *Requests* (peticiones de certificados que esperan ser firmados por la CA). Por encima aparecen una serie de iconos para acceder a funciones directas pero Tinyca2 tiene un error y no muestra los nombres de los iconos.

El siguiente paso es crear una nueva sub-CA y para hacerlo verificar que estamos en el tab de la CA y haciendo click en el tercer icono desde la derecha que nos mostrará una ventana que como subtítulo tiene "*Create a new Sub CA*". Deberemos introducir el *passwd* que pusimos en la rootCA, darle un nombre (subca-sysdw.nteum.org), *Data-for-CA* (sysdw.nteum.org) y el resto de datos como hicimos anteriormente (el *passwd* no debe ser necesariamente el mismo de la rootCA) y cuando hacemos OK nos saldrá la ventana principal pero con la Sub-CA seleccionada, si queremos volver a la CA deberemos ir al menu *File->Open* y abrir la rootCA. Recordad que la Sub-CA será quien gestionará las peticiones y firmará los certificados para ese dominio por lo cual debemos seleccionar la adecuada en cada momento. La root-CA solo la utilizaremos para crear/revocar nuevas sub-CA y para exportar el certificado raíz de la rootCA (necesario para enviarles a nuestros clientes para que se lo instalen en sus navegadores).

Para crear un certificado que permita certificar nuestro web-server cuando utiliza SSL (https://sysdw.nteum.org), con nuestra Sub-CA seleccionada vamos al

tab de *Request* y con el botón derecho seleccionamos "*New request*" en la cual se abrirá una ventana donde deberemos introducir la URL de nuestro servidor (sysdw.nteum.org) como *CommonName* y rellenar el resto de los datos. Una vez creado, lo seleccionamos y con el botón derecho indicamos "*Sign request*" y seleccionamos "*Server request*" que nos mostrará una ventana con el *password* de la CA y la validez. Este procedimiento es el que genera confianza ya que estamos validando la información aportada en la petición y firmando el certificado (los cuales ya podremos ver en los tabs correspondientes.

Ahora deberemos exportar los certificados (del web y la rootCA), la *key* y configurar Apache para que los incorpore. Lo primero será exportar el rootCA e introducirlo en Firefox/Iceweasel. Para ello en la ventana principal *File->OpenCA* seleccionamos nuestra RootCA y seleccionando el segundo icono de la derecha que corresponde a "*Export CA Certificate*" indicamos un nombre de archivo y el formato (PEM es el adecuado) y salvamos el certificado en nuestro sistema de archivo (p. ej., /tmp/ROOTCA-nteum.org.pem). Es importante tener en cuenta de hacer un `chmod 644 /tmp/ROOTCA-nteum.org.pem` ya que se salvará como 600 y por lo cual otros usuarios no lo podrán importar. Desde Firefox/Iceweasel seleccionamos *Preferences/Setting->Advanced->Certificates->View Certificates->Authorities->Import* y seleccionamos el archivo antes creado marcando todas las "*trust settings*" que nos presenta en la ventana siguiente (3). Luego podremos ver con el nombre de que le dimos a "*Organization*" el certificado correspondiente.

A continuación en TinyCA2 abrimos nuestra sub-CA, seleccionamos el tab *Certificates* y sobre el certificado seleccionamos el botón derecho e indicamos "*Export certificate*" dando un nombre de archivo, p. ej, sysdw.nteum.org-cert.pem, luego repetimos el proceso con la *key* en el *Key* tab y haciendo "*Export key*" salvándolo como p. ej., sysdw.nteum.org-key.pem. Es importante decidir si le ponemos *passwd* o no ya que si le ponemos cada vez que arranque el servidor nos solicitará el *passwd*. Sobre el tab CA deberemos exportar ahora el "*certificate chain*" que es el primer icono desde la derecha y salvarlo como sysdw.nteum.org-chain.pem. Moveremos estos tres archivos al directorio */etc/ssl/private/* y pasaremos a configurar Apache modificando el archivo que configure nuestro sitio SSL, por ejemplo nosotros hemos utilizado */etc/apache2/sites-available/default-ssl* en el cual hemos modificado (solo se muestra las líneas principales):

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
    ServerName sysdw.nteum.org
...
    SSLEngine on
    SSLCertificateFile /etc/ssl/private/sysdw.nteum.org-cert.pem
    SSLCertificateKeyFile /etc/ssl/private/sysdw.nteum.org-key.pem
    SSLCertificateChainFile /etc/ssl/private/sysdw.nteum.org-chain.pem
...
</VirtualHost>
</IfModule>
```

Solo nos resta habilitar el sitio (*a2ensite default-ssl*), e reiniciar Apache (con la orden `service apache2 restart`) -nos pedirá el *passwd* si hemos lo hemos puesto en la *key*- y probar (en el navegador que tenemos instalado el certificado raíz de la rootCA) la URL https://sysdw.nteum.org que si todo está correcto cargará la página sin la típica ventana que el sitio no es seguro.[25]

Para crear certificados para una dirección de correo y distribuirlos a nuestros usuarios junto con el certificado de la rootCA podemos hacer en el tab *Certificates* de nuestra sub-CA, seleccionar "*New - Create Key and Certificate (Client)*" y entrar el nombre y la dirección de correo para la cual queremos validar así como el *passwd* para protegerlo hasta que llegue a su nuevo destinatario (y que luego el podrá/deberá cambiar). A continuación debemos exportarlo teniendo en cuenta de utilizar el formato PKCD#12 que incluye el certificado y la llave. Después de enviarle al usuario el archivo con el certificado más el de la root-CA, el podrá agregarlo a su gestor de correo de forma similar a root-CA pero como *personal certificates*. Luego podrá enviar correo firmados digitalmente y el destinatario (que deberá tener instalado el certificado de la rootCA) podrá verificar la firma del correo.

Como apunte final con relación a la PKI, todos nuestros usuarios/clientes/visitantes de nuestros servicios deberán tener instalado el certificado de la root-CA en sus navegadores/clientes para así validar los sitios/servicios que están bajo nuestro dominio/sub-dominios ya que los navegadores/clientes de correo no incorporan estos certificados raíz por defecto. Si nuestro dominio/servicios lo requiere podemos gestionar con Mozilla la inclusión de nuestro certificado en http://www.mozilla.org/en-US/about/governance/policies/security-group/certs/ pero no es un trámite fácil y es necesario cumplir con una serie de requisitos ya que las garantías del sistema radica en la inclusión de estos certificados.

Actividades

1. Configurad un servidor DNS como caché y con un dominio propio.

2. Configurad un servidor/cliente NIS con dos máquinas exportando los directorios de usuario del servidor por NFS.

3. Configurad un servidor SSH para acceder desde otra máquina sin contraseña.

4. Configurad un servidor Apache+ SSL+ PHP+ MySQL+ PHPAdmin para visualizar las hojas personales de los usuarios.

5. Configurad un servidor Apache + un Reverse Proxy para acceder al servidor web desde una máquina externa a través del Proxy.

6. Cread y configurad un sistema de correo electrónico a través de Exim, Fetchmail, SpamAssassin y un servidor IMAP para recibir correos desde el exterior y poder leerlos desde una máquina remota con el cliente Mozilla (Thunderbird).

7. Instalad la Wiki MoinMoin y cread un conjunto de páginas para verificar su funcionamiento.

8. Instalad el servidor de *backups* BackupPC y generad una copia de respaldo desde una máquina Windows y otra desde una máquina Linux. Escoged el método de comunicación con los clientes y justificar la respuesta.

9. Configurad una CA con tiny CA y generar/probar los certificados para una página web con SSL y para enviar correos firmados y verificarlos desde otra cuenta.

Bibliografía

Todas las URLs han sido visitadas por última vez en Junio de 2014.

[1] **Debian.org**. *Debian Home*.
<http://www.debian.org>

[2] *Server-World*.
<hhttp://www.server-world.info/en/note?os=Debian_7.0>

[3] **Langfeldt, N.** *DNS HOWTO*.
<http://tldp.org/HOWTO/DNS-HOWTO.html>

[4] **IETF**. Repositorio de Request For Comment desarrollados por Internet Engineering Task Force (IETF) en el Network Information Center (NIC). <http://www.ietf.org/rfc.html>

[5] **Instituto de Tecnologías Educativas**. *Redes de área local: Aplicaciones y Servicios Linux*.
<http://www.ite.educacion.es/formacion/materiales/85/cd/linux/indice.htm>

[6] **Trenholme, S.** *MaraDNS - A small open-source DNS server*.
<http://maradns.samiam.org/>

[7] **DNSMasq** *DNS forwarder and DHCP server*.
<https://wiki.debian.org/HowTo/dnsmasq>

[8] *Comparison of FTP client software*
<http://en.wikipedia.org/wiki/Comparison_of_FTP_client_software>

[9] *Servicios de red: Postfix, Apache, NFS, Samba, Squid, LDAP*
<http://debian-handbook.info/browse/es-ES/stable/network-services.html>

[10] *NIS HOWTO*.
<http://www.linux-nis.org/>

[11] **Kukuk, T.** *The Linux NIS(YP)/NYS/NIS+ HOWTO -obs:EOL-*.
<http://tldp.org/HOWTO/NIS-HOWTO/verification.html>

[12] *Exim*.
<http://www.exim.org/docs.html>

[13] *ProcMail*.
<http://www.debian-administration.org/articles/242>

[14] **Apache HTTP** *Server Version 2.2*.
<http://httpd.apache.org/docs/2.2/>

[15] *Apache2 + WebDav*.
<http://www.debian-administration.org/articles/285>

[16] **Squid** *Proxy Server*.
<http://www.squid-cache.org/ >

[17] **Squid** *Configuration Examples*.
<http://wiki.squid-cache.org/ConfigExamples >

[18] **Kiracofe, D.** *Transparent Proxy with Linux and Squid mini-HOWTO -obs:EOL pero interesante en conceptos-*.
<http://tldp.org/HOWTO/TransparentProxy.html#toc1>

[19] **Pinheiro Malère, L. E.** *Ldap. The Linux Documentation Project*.
<http://tldp.org/HOWTO/LDAP-HOWTO/>

[20] *Proftpd*.
<http://www.debian-administration.org/articles/228>

[21] **PKI** *Public-key cryptography*.
<http://en.wikipedia.org/wiki/Public_key_cryptography >

[22] *SSL/TLS Strong Encryption: An Introduction*.
<http://httpd.apache.org/docs/2.2/ssl/ssl_intro.html#cryptographictech>

[23] *Transparent Multi-hop SSH*.
<http://sshmenu.sourceforge.net/articles/transparent-mulithop.html>

[24] **Christof Paar, Jan Pelzl** *Introduction to Public-Key Cryptography*.
<http://wiki.crypto.rub.de/Buch/movies.php >

[25] **Magnus Runesson** (2007). *Create your own CA with TinyCA2*.
<http://theworldofapenguin.blogspot.com.es/2007/06/create-your-own-ca-with-tinyca2-part-1.html>

[26] **Greg Ferro** *Fast Introduction to SOCKS Proxy*.
<http://etherealmind.com/fast-introduction-to-socks-proxy/ >

[27] *Proxy SOCKS*.
<http://en.flossmanuals.net/bypassing-es/proxis-socks/ >

[28] **Mourani, G.** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.
<http://www.tldp.org/LDP/solrhe/Securing-Optimizing-Linux-The-Ultimate-Solution-v2.0.pdf>

[29] *Samba*.
<http://www.samba.org/ >

[30] *Wiki - Samba*.
<https://wiki.samba.org >

[31] *RSAT. Remote Server Administration Tools on a Windows workstation*.
<https://wiki.samba.org/index.php/Installing_RSAT_on_Windows_for_AD_Management>

[32] **M. López, C. Alonso***Samba 4: Controlador Active Directory*.
<http://waytoit.wordpress.com/2013/05/12/samba-4-controlador-active-directory-parte-1-de-3/ >

[33] **M. Rushing***Compiling Samba 4 on Debian Wheezy - Active Directory Domain Controllers*..
<http://mark.orbum.net/2014/02/22/compiling-samba-4-on-debian-wheezy-active-directory-domain-controllers-ho/ >

[34] *Guía Samba4 como Controlador de Dominio y Directorio Activo*.
<http://fraterneo.wordpress.com/2013/08/19/guia-samba4-como-controlador-de-dominio-y-directorio-activo-actualizacion/ >

Administración de datos

Remo Suppi Boldrito

PID_00212474

Universitat Oberta de Catalunya

Índice

Introducción 5

Objetivos 7

1. Administración de datos 9
1.1. PostgreSQL 10
1.1.1. Instalación de PostgreSQL 11
1.1.2. ¿Cómo se debe crear una DB? 13
1.1.3. ¿Cómo se puede acceder a una DB? 13
1.1.4. Usuarios de DB 14
1.1.5. Mantenimiento 15
1.2. El lenguaje SQL 16
1.2.1. Entornos de administración gráficos 18
1.3. MySQL 20
1.3.1. Instalación de MySQL 21
1.3.2. Postinstalación y verificación de MySQL 22
1.3.3. El programa monitor (cliente) mysql 22
1.3.4. Caso de uso: procesamiento de datos con MySQL 24
1.3.5. Administración de MySQL 26
1.3.6. Interfaces gráficas 27
1.4. MariaDB 28
1.4.1. Instalación MariaDB sobre Debian 29
1.5. SQLite 30
1.6. Source Code Control System 31
1.6.1. Concurrent Versions System (CVS) 32
1.6.2. Subversion 36
1.6.3. Git 39
1.6.4. Mercurial 43
1.7. Mantis Bug Tracker 45

Actividades 46

Bibliografía 47

Introducción

Según afirman algunos expertos, entre 2013-2018 se generarán tantos dados como en los últimos 5000 años. Este fenómeno conocido como *"Big Data"* es utilizado para referirse a situaciones donde el conjunto de datos a tratar supera las medidas habituales de datos gestionados hoy en día por los sistemas de información y que deberán ser obtenidos, almacenados y procesados en tiempos razonables. ¿Cómo se considera *Big Data*? Es un criterio dinámico, pero normalmente se puede considerar como un conjunto de datos único de entre decenas de *Terabytes* a decenas de *PetaBytes* (10 PBytes = 10.000 TBytes = 10 millones de Gigabytes) y donde las dificultades que se encontrarán estarán en su captura, almacenado, búsqueda, compartición, lectura, análisis, y visualización. Vinculadas a esta nueva tendencia, han recuperado fuerza tecnologías conocidas como *Data Minining* (Minería de Datos) en su versión más enfocada al procesamiento de datos sociales (*Social Data Mining*) o el *Datawarehouse* (Repositorio de Datos), que serán aspectos predominantes a tener en cuenta en el futuro cercano.[1]

Es por ello por lo que la gestión de datos es un aspecto muy importante en la utilización de la tecnología actual, y los sistemas operativos son la base donde se ejecutarán las aplicaciones que capturarán, almacenarán y gestionarán estos datos. Un manejo eficiente en el almacenamiento, gestión y procesamiento de los datos no puede verse, hoy en día, separado de una **Base de Datos** (*databases*, DB) y que serán el punto crítico para las nuevas tecnologías que se desarrollen a partir del *Big Data*. Una base de datos es un conjunto estructurado de datos que pueden organizarse de manera simple y eficiente por parte de un gestor de dicha base. Las bases de datos actuales se denominan relacionales, ya que los datos pueden almacenarse en diferentes tablas que facilitan su gestión y administración (ejemplos de ellas son MySQL/MariaDB, PostgreSQL). Para ello, y con el fin de estandarizar el acceso a las bases de datos, se utiliza un lenguaje denominado SQL (*Structured Query Language*), que permite una interacción flexible, rápida e independiente de las aplicaciones a las bases de datos.

También en necesario destacar que comienzan, con el tratamiento derivado del *BigData*, a desarrollarse tecnologías para gestionar datos no estructurados, en lo que se denominan bases de datos no estructuradas (o NoSQL o también "no solo SQL") y que definen sistemas de gestión de bases de datos que difieren de las tradicionales BD relacionales (RDBMS), en las que, por ejemplo, no usan SQL como el principal lenguaje de consultas, ya que no se realizan búsquedas exactas, la operación más típica es la búsqueda por similitud, los datos almacenados no requieren estructuras fijas como tablas y si utilizan categorías

como por ejemplo clave-valor. Las bases de datos NoSQL han crecido con el auge de grandes compañías de Internet (Google, Amazon, Twitter y Facebook entre otros), ya que necesitan encontrar formas de tratamiento de los datos producidos y que con las RDBMS no pueden o es muy complejo/ineficiente y donde el rendimiento/eficiencia en su procesamiento y sus propiedades de tiempo real son más importantes que la coherencia. Ejemplo de estas bases de datos son Hadoop-Hbase (utilizada por casi todas las grandes compañías de Internet), MongoDB (NoSQL orientado a documentos), Elasticsearch (*multitenancy*, *full-text search engine* con interfaz web RESTful y soporte para *schema-free JSON documents*) o Redis (motor de base de datos en memoria, basado en el almacenamiento en tablas de *hashes* -clave/valor-).

En este módulo se verán los gestores más importantes de bases de datos en entornos GNU/Linux, una breve reseña a SQL, así como diferentes formas de gestionar datos y repositorios dentro de un entorno distribuido y multiusuario.

Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Analizar las maneras de almacenar datos en forma masiva y uso eficiente.
2. Desarrollar los aspectos esenciales de bases de datos y su instalación y uso.
3. Trabajar con las técnicas de control de versiones y analizar sus ventajas.
4. Instalar y analizar las diferentes herramientas de gestión de datos y su integración para entornos de desarrollo.

1. Administración de datos

En la actualidad, la forma más utilizada para acceder a una base de datos es a través de una aplicación que ejecuta código SQL. Por ejemplo, es muy común acceder a una DB a través de una página web que contenga código PHP o Perl (los más comunes). Cuando un cliente solicita una página, se ejecuta el código PHP/Perl incrustado en la página, se accede a la DB y se genera la página con su contenido estático y el contenido extraído de la DB que posteriormente se envía al cliente. Dos de los ejemplos más actuales de bases de datos son los aportados por PostgreSQL y MySQL (o su *fork* MariaDB), que serán objeto de nuestro análisis.

También veremos algunos aspectos introductorios sobre SQLite, que es un sistema de gestión de bases de datos relacional (y compatible con ACID) contenido en una biblioteca (escrita en C) y que, a diferencia de los sistemas anteriores (cliente-servidor), el motor de SQLite no es un proceso independiente que se comunica con el programa sino que SQLite se enlaza con el programa pasando a ser parte del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a funciones, reduciendo el acceso a la base de datos y siendo mucho más eficientes. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos) son guardados en un solo archivo estándar en la máquina host y la coherencia de la BD se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

Por otro lado, cuando se trabaja en el desarrollo de un software, existen otros aspectos relacionados con los datos, como su validez y su ámbito (sobre todo si existe un conjunto de usuarios que trabajan sobre los mismos datos). Existen diversos paquetes para el control de versiones (revisiones), pero el objetivo de todos ellos es facilitar la administración de las distintas versiones de cada producto desarrollado junto a las posibles especializaciones realizadas para algún cliente específico. El control de versiones se realiza para controlar las distintas versiones del código fuente. Sin embargo, los mismos conceptos son aplicables a otros ámbitos y no solo para el código fuente sino para los documentos, imágenes, etc. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión (CVS, Subversion, GIT, Bazaar, Darcs, Mercurial, Monotone, Codeville, RCS, etc.).

En este módulo veremos **CVS (Control Version System)**, **Subversion**, **GIT** y **Mercurial** para controlar y administrar múltiples revisiones de archivos, automatizando el almacenamiento, la lectura, la identificación y la mezcla de

diferentes revisiones. Estos programas son útiles cuando un texto se revisa frecuentemente e incluye código fuente, ejecutables, bibliotecas, documentación, gráficos, artículos y otros archivos. Finalmente, también se analizará una herramienta para el seguimiento de incidencias y errores en entorno de desarrollo o de proyectos llamado Mantis.

La justificación de CVS y Subversion se puede encontrar en que CVS es uno de los paquetes tradicionales más utilizados y Subversion (también se lo conoce como `svn` por ser el nombre de la herramienta de línea de comandos) es un programa de control de versiones diseñado específicamente para reemplazar al popular CVS y que soluciona algunas de sus deficiencias. Una característica importante de Subversion es que, a diferencia de CVS, los archivos con versiones no tienen cada uno un número de revisión independiente. Por el contrario, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un determinado momento.

A continuación veremos dos grandes entornos en la gestión de repositorios dadas sus prestaciones y utilización en grandes proyectos: GIT y Mercurial. **GIT** es un software de control de versiones diseñado por Linus Torvalds, basado en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. **Mercurial** es un sistema de control de versiones (básicamente desarrollado en Python), y diseñado para obtener el mejor rendimiento y escalabilidad, con un desarrollo completamente distribuido (sin necesidad de un servidor) y que permite una gestión robusta de archivos (texto o binarios) y con capacidades avanzadas de ramificación e integración. Finalmente, para completar un mínimo círculo de herramientas para compartir y gestionar datos, se presenta una breve descripción de **Mantis Bug Tracker**, que es una herramienta de gestión de incidencias (*bug tracker*) de código abierto. Esta aplicación está escrita en PHP y requiere una base de datos y un servidor web (generalmente MySQL y Apache).

1.1. PostgreSQL

En el lenguaje de bases de datos, PostgreSQL utiliza un modelo cliente-servidor [3]. Una sesión de PostgreSQL consiste en una serie de programas que cooperan:

1) Un proceso servidor que gestiona los archivos de la DB acepta conexiones de los clientes y realiza las acciones solicitadas por estos sobre la DB. El programa servidor es llamado en PostgreSQL *postmaster*.

2) La aplicación del cliente (*frontend*) es la que solicita las operaciones que hay que realizar en la DB y que pueden ser de lo más variadas; por ejemplo: herramientas en modo texto, gráficas, servidores de web, etc.

Generalmente, el cliente y el servidor se encuentran en diferentes *hosts* y se comunican a través de una conexión TCP/IP. El servidor puede aceptar múltiples peticiones de diferentes clientes y activar para cada nueva conexión un proceso que lo atenderá en exclusiva de un modo transparente para el usuario. Existe un conjunto de tareas que pueden ser llevadas a cabo por el usuario o por el administrador, según convenga, y que pasamos a describir a continuación.

1.1.1. Instalación de PostgreSQL

Este paso es necesario para los administradores de la DB, ya que dentro de las funciones del administrador de DB se incluye la instalación del servidor, la inicialización y configuración, la administración de los usuarios y tareas de mantenimiento de la DB. La instalación de la base de datos se puede realizar de dos modos, a través de los binarios de la distribución, lo cual no presenta ninguna dificultad, ya que los *scripts* de distribución realizan todos los pasos necesarios para tener la DB operativa, o a través del código fuente, que será necesario compilar e instalar. En el primer caso (*pre-built binary packages*), se pueden utilizar los gestores de paquetes o desde línea de comandos, por ejemplo, en Debian el `apt-get`. Para el segundo caso, se recomienda ir siempre al origen (o a un repositorio espejo de la distribución original). Es importante tener en cuenta que después la instalación desde el código fuente quedará fuera de la DB de software instalado y se perderán los beneficios de administración de software que presenten por ejemplo `apt-cache` o `apt-get`.[3]

Instalación paso a paso

En este apartado se optará por la instalación y configuración de los paquetes de la distribución (tanto de la versión distribuida como la de la última versión de desarrollador) pero su instalación desde los fuentes no genera dificultad y es la adecuada si se desea tener la última versión de la BD. Los fuentes pueden obtenerse de http://www.postgresql.org/ftp/source/ y siguiendo las indicaciones desde http://www.postgresql.org/docs/9.3/interactive/installation.html.

La instalación es muy simple, ejecutando `apt-get install postgresql` instalará la versión 9.1 de la distribución de Debian Wheezy (junto con la instalación de otros paquetes adicionales como postgresql-client-9.1 postgresql-client-common postgresql-common). En el caso de que se desee instalar la última versión (9.3) desde los paquetes binarios, PostgreSQL facilita los paquetes para Debian (http://www.postgresql.org/download/linux/debian/) y se debe agregar el repositorio, p. ej., creando un archivo */etc/apt/sources.list.d/pgdg.list* que tenga la línea `deb http://apt.postgresql.org/pub/repos/apt/ wheezy-pgdg main`; luego debemos importar la llave del repositorio con

```
wget -quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc
| apt-key add -
```

y actualizar el repositorio `apt-get update`, finalmente podremos ejecutar `apt-get install postgresql-9.3`. Si se desea instalar el paquete que incluye diferentes contribuciones de la comunidad* deberemos hacer `apt-get install postgresql-contrib` (estas contribuciones incluyen *fuzzystrmatch, unaccent o citext* para búsquedas intensivas o difusas por ejemplo). Después de la instalación tendremos: cliente postgresql (`psql`), usuario del sistema por defecto: Debian ha creado el usuario **postgres** (no cambiar el passwd de este usuario), usuario por defecto de postgresql: es el *superuser* **postgres** (su passwd deberá ser cambiado desde dentro de la BD), usuario de la BD **postgres**, cluster por defecto: **main**, BD por defecto: **template1**, schema por defecto: **public**. Para cambiar el *passwd* del usuario **postgres** de la BD hacemos: `su -l root` a continuación `su - postgres` y luego `psql` para entrar en la consola (cliente o frontend). Después de los mensajes y delante del prompt *postgres=#* introducimos `\password postgres` y el *passwd* seleccionado dos veces. Para salir hacemos `\q`. A continuación se deberá hacer algunos ajustes como del archivo */etc/postgresql/9.1/main/postgresql.conf* cambiar la línea *# listen_addresses = 'localhost'* por *listen_addresses = 'localhost, 192.168.1.200'* donde la IP es la de servidor. También definir el método de autentificación en */etc/postgresql/9.1/main/pg_hba.conf* agregar por ejemplo

***http://www.postgresql.org /docs/9.3/static /contrib.html**

host all all 192.168.1.200/24 md5
host all all 192.168.1.100/24 md5

Donde 192.168.1.200 es la dirección de servidor y 192.168.1.100 la IP de gestión de la DB y deberemos activar, si lo tenemos instalado, `iptables` para aceptar comunicaciones en el puerto 5432. Finalmente deberemos reiniciar el servidor con `service postgresql restart` para que los cambios sean efectivos. Los archivos esenciales de PostgreSQL se encuentran en: configuración */etc/postgresql/9.1/main/*, binarios */usr/lib/postgresql/9.1/bin*, archivos de datos */var/lib/postgresql/9.1/main* y logs */var/log/postgresql/postgresql-9.1-main.log*. Para ver los *clusters* disponibles (PostgreSQL se basa en CLUSTERS que es un grupo de BD manejadas por un servicio común y donde cada una de estas BD, contiene uno o más SCHEMATA) podemos ejecutar `pg_lsclusters` que en nuestro caso es:

Version Cluster Port Status Owner Data directory Log file
9.1 main 5432 online postgres /var/lib/postgresql/9.1/main /var/log/postgresql/postgresql-9.1-main.log

Podéis ver la documentación de PostgreSQL en la dirección web siguiente: http://www.postgresql.org/docs/9.1/interactive/index.html.

1.1.2. ¿Cómo se debe crear una DB?

La primera acción para verificar si se puede acceder al servidor de DB es crear una base de datos. El servidor PostgreSQL puede gestionar muchas DB y es recomendable utilizar una diferente para cada proyecto.

Para crear una base de datos, se utiliza el comando `createdb` desde la línea de comandos del sistema operativo. Este comando generará un mensaje `CREATE DATABASE` si todo es correcto. Con el comando `!l` nos listará todas las BD definidas.

Es importante tener en cuenta que para llevar a cabo esta acción, debe haber un usuario habilitado para crear una base de datos, como hemos visto en el apartado anterior, en que existe un usuario que instala la base de datos y que tendrá permisos para crear bases de datos y crear nuevos usuarios que, a su vez, puedan crear bases de datos. Generalmente (y en Debian), este usuario es **postgres** por defecto. Por ello, antes de hacer el `createdb`, se debe hacer un `su postgres` (o previamente hacer `su -l root`), a continuación, se podrá realizar el `createdb`. Para crear una DB llamada *nteumdb*:

```
createdb nteumdb
```

Si la ejecución del comando da error, puede ser que no esté bien configurado el camino o que la DB esté mal instalada. Se puede intentar con el camino absoluto (`/usr/lib/postgresql/9.1/bin/createdb nteumdb`), donde la ruta dependerá de la instalación que se haya hecho (consultar referencias para la solución de problemas). Otros mensajes de error serían *could not connect to server*, cuando el servidor no está arrancado, o también el mensaje *CREATE DATABASE: permission denied*, cuando no se tienen privilegios para crear la DB. Para eliminar la base de datos, se puede utilizar `dropdb nteumdb`.

1.1.3. ¿Cómo se puede acceder a una DB?

Una vez creada la DB, se puede acceder a ella de diversas formas:

1) ejecutando un comando interactivo llamado `psql`, que permite editar y ejecutar comandos SQL (por ejemplo, `psql nteumdb`);

2) ejecutando una interfaz gráfica como PhpPgAdmin o alguna *suite* que tenga soporte ODBC para crear y manipular DB;

3) escribiendo una aplicación con algunos de los lenguajes soportados, como PHP, Perl o Java, entre otros (consultad *PostgreSQL Programmer's Guide*).

Nota

Para poder acceder a la DB, el servidor de base de datos deberá estar en funcionamiento. Cuando se instala Postgresql, se crean los enlaces adecuados para que el servidor se inicie en el arranque del ordenador. Para más detalles, consultad el apartado de instalación.

Por simplicidad, utilizaremos `psql` para acceder a la DB, por lo que se deberá introducir `psql nteumdb`: saldrán unos mensajes con la versión e información y un *prompt* similar a `nteumdb =#` o `nteumdb =>` (el primero si es el superusuario y el segundo si es un usuario normal). Se pueden ejecutar algunos de los comandos SQL siguientes:

```
SELECT version(); o también SELECT current_date;
```

`psql` también tienen comandos que no son SQL y comienzan por \, por ejemplo \h (enumera todos los comandos disponibles) o \q para terminar. Una lista de los comandos SQL que permite la podemos consultar en la dirección web http://www.postgresql.org/docs/9.1/static/sql-commands.html.

1.1.4. Usuarios de DB

Los usuarios de la DB son completamente distintos de los usuarios del sistema operativo. En algunos casos, podría ser interesante mantener una correspondencia, pero no es necesario. Los usuarios son para todas las DB que controla dicho servidor, no para cada DB.

Para crear un usuario, se puede ejecutar la sentencia SQL `CREATE USER` *`nombre`* y para borrar usuarios, `DROP USER` *`nombre`*.

También se puede llamar a los programas `createuser` y `dropuser` desde la línea de comandos. Existe un usuario por defecto llamado **postgres** (dentro de la DB), que es el que permitirá crear los restantes (para crear nuevos usuarios si el usuario de sistema operativo con el que se administra la DB no es postgres `psql –U usuario`).

Un usuario de DB puede tener un conjunto de atributos en función de lo que puede hacer:

- **Superusuario**: este usuario no tiene ninguna restricción. Por ejemplo, podrá crear nuevos usuarios: `CREATE USER nombre SUPERUSER;`
- **Creador de DB**: tiene permiso para crear DB. Para crear un usuario de estas características, utilizad el comando: `CREATE USER nombre CREATEDB;`
- **Contraseña**: solo es necesario si por cuestiones de seguridad se desea controlar el acceso de los usuarios cuando se conecten a una DB. Para crear un usuario con contraseña (`palabra_clave` será la clave para ese usuario): `CREATE USER nombre WITH PASSWORD 'palabra_clave';`

A un usuario se le pueden cambiar los atributos utilizando el comando `ALTER USER`.

También se pueden hacer grupos de usuarios que compartan los mismos privilegios con:
`CREATE GROUP NomGrupo;`

Para insertar usuarios en este grupo: `ALTER GROUP NomGrupo ADD USER Nombre1;`

Para borrar: `ALTER GROUP NomGrupo DROP USER Nombre1;`

Ejemplo: operaciones con grupo dentro de psql
```
CREATE GROUP NomGrupo;
ALTER GROUP NomGrupo ADD USER Nom1,...; ALTER GROUP NomGrupo DROP USER
Nom1,...;
```

Cuando se crea una DB, los privilegios son para el usuario que la crea (y para el *superusuario*). Para permitir que otro usuario utilice esta DB o parte de ella, se le deben conceder privilegios. Hay diferentes tipos de privilegios, como `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `RULE`, `REFERENCES`, `TRIGGER`, `CREATE`, `TEMPORARY`, `EXECUTE`, `USAGE` y `ALL PRIVILEGES` (se deben consultar las referencias para ver su significado).

Para asignar los privilegios, se puede utilizar `GRANT UPDATE ON objeto TO usuario` donde `usuario` deberá ser un usuario válido de PostgerSQL y `objeto`, una tabla, por ejemplo.

Este comando lo deberá ejecutar el superusuario o el dueño de la tabla. El usuario PUBLIC puede ser utilizado como sinónimo de todos los usuarios y ALL como sinónimo de todos los privilegios. Por ejemplo, para quitar todos los privilegios a todos los usuarios de objeto, se puede ejecutar:

```
REVOKE ALL ON objeto FROM PUBLIC;
```

1.1.5. Mantenimiento

Hay un conjunto de tareas que son responsabilidad del administrador de DB y que se deben realizar periódicamente:

1) Recuperar el espacio: para ello se deberá ejecutar periódicamente el comando `VACUUM`, que recuperará el espacio de disco de filas borradas o modificadas, actualizará las estadísticas utilizadas por el planificador de PostgreSQL y mejorará las condiciones de acceso.

2) Reindexar: en ciertos casos, PostgreSQL puede dar algunos problemas con la reutilización de los índices, por ello, es conveniente utilizar `REINDEX` periódicamente para eliminar páginas y filas. También hay la posibilidad de utilizar `contrib/reindexdb` para reindexar una DB entera (se debe tener en cuenta que dependiendo del tamaño de las DB, estos comandos pueden tardar un cierto tiempo).

3) Cambio de archivos de registro (*logs*): se debe evitar que los archivos de registro sean muy grandes y difíciles de manejar. Se puede hacer fácilmente cuando se inicia el servidor con `pg_ctl start | logrotate`, donde este último comando renombra y abre un nuevo archivo de registro y se puede configurar con `/etc/logrotate.conf`.

4) Copia de seguridad y recuperación (*backup* y *recovery*): existen dos formas de guardar los datos, con la sentencia SQL `dump` o guardando el archivo de la DB. El primero es `pg_dump ArchivoDB >ArchivoBackup`. Para recuperar, se puede utilizar `psql ArchivoDB <ArchivoBackup`. Para guardar todas las DB del servidor, se puede ejecutar `pg_dumpall >ArchivoBackupTotal`. Otra estrategia es guardar los archivos de las bases de datos a nivel del sistema operativo, p. ej. con `tar -cf backup.tar /var/lib/postgresql/9.1/main`. Existen dos restricciones que pueden hacer que este método sea poco práctico:

a) el servidor debe detenerse antes de guardar y de recuperar los datos y

b) deben conocerse muy bien todas las implicaciones a nivel archivo, donde están todas las tablas, transacciones y demás, ya que de lo contrario, una DB puede quedar inútil. Además, por lo general, el tamaño que se guardará será mayor que el realizado con los métodos anteriores, ya que por ejemplo, con el `pg_dumb` no se guardan los índices, sino el comando para recrearlos.

1.2. El lenguaje SQL

No es la finalidad de este subapartado hacer un tutorial sobre SQL, pero se analizarán unos ejemplos para ver las capacidades de este lenguaje. Son ejemplos que vienen con la distribución de los fuentes de PostgreSQL en el directorio `DirectorioInstalacion/src/tutorial`. Para acceder a ellos, cambiad al directorio de PostgreSQL (`cd DirectorioInstalación/src/tutorial`) y ejecutad `psql -s nteumdb` y después, ejecutad `\i basics.sql` dentro. El parámetro `\i` lee los comandos del archivo especificado (`basic.sql` en nuestro caso). PostgreSQL, como ya hemos mencionado, es una base de datos relacional (*Relational Database Management System, RDBMS*), lo cual significa que maneja los datos almacenados en tablas. Cada tabla tiene un número determinado de filas y de columnas, y cada columna tiene un tipo específico de datos. La tablas se agrupan en una DB y un único servidor maneja esta colección de DB (todo el conjunto se denomina agrupación o clúster de bases de datos, *database cluster*). Para crear, por ejemplo, una tabla con psql, ejecutad:

```
CREATE TABLE tiempo (
    ciudad  varchar(80),
    temp_min int,
    temp_max int,
    lluvia real,
    dia date
    );
```

El comando termina cuando se pone '`;`' y se pueden utilizar espacios en blanco y tabulaciones libremente. `Varchar(80)` especifica una estructura de datos que puede almacenar hasta 80 caracteres (en nuestro caso). El *point* es un tipo específico de PostgreSQL.

Para borrar la tabla:
```
DROP TABLE nombre_tabla;
```
Para introducir datos, se pueden utilizar dos formas: poner todos los datos de la tabla o indicar las variables y los valores que se desean modificar:
```
INSERT INTO tiempo VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19');
INSERT INTO tiempo (ciudad, temp_min, temp_max, lluvia, dia) VALUES
('Barcelona', 16, 37, 0.25, '2007-03-19');
```
Esta forma puede ser sencilla para unos pocos datos, pero cuando hay que introducir gran cantidad de datos, se pueden copiar desde un archivo con la sentencia:
`COPY tiempo FROM '/home/user/tiempo.txt';` Este archivo debe estar en el servidor, no en el cliente). Para mirar una tabla, podríamos hacer:
```
SELECT * FROM tiempo;
```
donde el * significa "todas las columnas".
Ejemplo: introducir datos en tabla. Dentro de psql:
```
INSERT INTO NombreTB (valorVar1, ValorVar2,...);
```
Datos desde un archivo. Dentro de psql:
```
COPY NombreTB FROM 'NombreArchivo';
```
Visualizar datos. Dentro de psql:
```
SELECT * FROM NombreTB;
```
Algunos ejemplos de comandos más complejos serían (dentro de psql). Visualiza la columna ciudad después de realizar la operación:
```
SELECT ciudad, (temp_max+temp_min)/2 AS temp_media, date FROM tiempo;
```
Visualiza todo donde se cumple la operación lógica:
```
SELECT * FROM tiempo WHERE city = 'Barcelona'AND lluvia > 0.0;
```
Unión de tablas:
```
SELECT * FROM tiempo, ciudad WHERE ciudad = nombre;
```
Funciones, máximo en este caso:
```
SELECT max(temp_min) FROM tiempo;
```
Funciones anidadas:
```
SELECT ciudad FROM tiempo WHERE temp_min = (SELECT max(temp_min) FROM
tiempo);
```
Modificación selectiva:
```
UPDATE tiempo SET temp_max = temp_max/2, temp_min = temp_min/2 WHERE
dia >'19990128';
```
Borrado del registro:
```
DELETE FROM tiempo WHERE ciudad = 'Sabadell';
```
Otros comandos de utilidad:
Login como usuario postgres (SuperUser) para trabajar con la base de datos: `# su - postgres`
Crear bases de datos: `$ createdb nombre_BD`
Eliminar base de datos: `$ dropdb nombre_BD`
Acceder a la base de datos: `$ psql nombre_BD`
Ayuda (dentro de la base de datos; por ejemplo, mynteum: `mynteum=# \h`
Salir del psql: `mynteum=# \q`
Guardar la base de datos: `$ pg_dump mynteum >db.out` Cargar la base de datos guardada anteriormente: `$ psql -d mynteum -f db.out`
Para guardar todas las bases de datos: `# su - postgres` y después `$ pg_dumpall >/var/lib/postgresql/backups/dumpall.sql`
Para restaurar una base de datos: `# su - postgres` y después
`$ psql -f /var/lib/postgresql/backups/dumpall.sql mynteum`
Para mostrar las bases de datos: `psql -l` o si no, desde dentro `mynteum=# \l;`
Mostrar los usuarios: desde dentro del comando `psql` ejecutar `mymydb=# SELECT * FROM "pg_user";`
Mostrar las tablas: desde dentro del comando `psql` ejecutar `mynteum=# SELECT * FROM "pg_tables";`
Cambiar la contraseña: desde dentro del comando `psql` `mynteum=# UPDATE pg_shadow SET passwd = 'new_password'where usename = 'username';` o también `ALTER USER nombre WITH PASSWORD 'password';`
Limpiar todas las bases de datos: `$ vacuumdb - -quiet - -all`

Para un usuario que se inicia en PostgreSQL es interesante obtener una base de datos ya construida y practicar con ella. Por ejemplo, en la dirección web

http://pgfoundry.org/projects/dbsamples/ podemos obtener la base World 1.0, que nos da una lista de 4079 ciudades del mundo de 239 países y su población. Una vez descargada (formato tar.gz) la descomprimimos con `tar xzvf world-1.0.tar.gz` y entramos en `psql`, desde el prompt hacemos `\i /tmp/world.sql`. Veremos cómo se crean las tablas y ya estaremos en condiciones de hacer consultas como:

```
SELECT * FROM "pg_tables" WHERE schemaname = 'public';     --Miramos las tablas
 schemaname |    tablename    | tableowner | tablespace | hasindexes | hasrules | hastriggers
------------+-----------------+------------+------------+------------+----------+-------------
 public     | city            | postgres   |            | t          | f        | t
 public     | country         | postgres   |            | t          | f        | t
 public     | countrylanguage | postgres   |            | t          | f        | t

SELECT * FROM "city" WHERE countrycode = 'ESP';         --Miramos las ciudades de un país
 id  |              name              | countrycode |      district      | population
-----+--------------------------------+-------------+--------------------+------------
 653 | Madrid                         | ESP         | Madrid             |    2879052
 654 | Barcelona                      | ESP         | Katalonia          |    1503451
 655 | Valencia                       | ESP         | Valencia           |     739412
...

SELECT count(*) FROM "city" WHERE countrycode = 'ESP';      --Contamos las ciudades de un país
 count
-------
    59

SELECT * FROM "city" WHERE name = 'Barcelona';              --Miramos las ciudades con un nombre
  id  |   name    | countrycode |  district  | population
------+-----------+-------------+------------+------------
  654 | Barcelona | ESP         | Katalonia  |    1503451
 3546 | Barcelona | VEN         | Anzoátegui |     322267

SELECT * FROM "city" WHERE name = 'Barcelona' and countrycode = 'ESP'; --Y solo las de un país
 id  |   name    | countrycode | district  | population
-----+-----------+-------------+-----------+------------
 654 | Barcelona | ESP         | Katalonia |    1503451

SELECT * FROM "city" WHERE population > '9500000';   --Miramos las ciudades con la población mayor
  id  |      name       | countrycode |   district    | population
------+-----------------+-------------+---------------+------------
  206 | Sao Paulo       | BRA         | Sao Paulo     |    9968485
  939 | Jakarta         | IDN         | Jakarta Raya  |    9604900
 1024 | Mumbai (Bombay) | IND         | Maharashtra   |   10500000
 1890 | Shanghai        | CHN         | Shanghai      |    9696300
 2331 | Seoul           | KOR         | Seoul         |    9981619

SELECT TRUNC(AVG(population),1) AS Total  FROM "city" WHERE population > '9500000';

                            --Obtenemos la media de las ciudades más pobladas
   total
------------
 9950260.8
```

Para más detalles sobre PostgreSQL y su utilización recomendamos [2][4].

1.2.1. Entornos de administración gráficos

Existen diversos entornos de administración gráficos, como el que hemos instalado en el apartado anterior, **Phppgadmin** y **Pgadmin** (disponible en la mayoría de las distribuciones, Debian incluida). Estos programas permiten ac-

ceder y administrar una base de datos con una interfaz gráfica ya sea mediante web o por interfaz propia. En el caso de interfaz propia, la forma más fácil de acceder es desde una terminal. El administrador de la DB (si no es el usuario postgresql) deberá hacer `xhost +`, lo cual permite que otras aplicaciones puedan conectarse al *display* del usuario actual y, a continuación, ejecutar el nombre de la aplicación.

Para instalar el paquete PhpPgAdmin deberemos ejecutar `apt-get install phppgadmin` y estará disponible en la URL *http://localhost/phppgadmin*. Para tenerlo disponible desde la dirección del propio servidor (o desde otra) deberemos modificar el archivo */etc/apache2/conf.d/phppgadmin* y comentar la línea `allow from 127.0.0.0/255.0.0.0 ::1/128` y descomentar la línea `allow from all` reiniciando el servidor luego. Cuando deseemos entrar en la base de datos en el menú de la izquierda nos pedirá un usuario/*passwd* que si introducimos "postgres" y el *passwd* del superusuario nos dará un error diciendo *Login disallowed for security reasons*, para habilitar su acceso iremos al archivo */etc/phppgadmin/config.inc.php* y cambiaremos la línea

```
$conf['extra_login_security'] = true;
```

por `$conf['extra_login_security'] = false;` a continuación deberemos recargar la página y reiterar el procedimiento y ya podremos acceder a la gestión de la base de datos.

Si se desea evitar trabajar con el usuario "postgres" (es recomendable) se puede crear un usuario con contraseña para acceder a la base de datos, para ello debemos hacer desde dentro de `psql`: `CREATE USER admin WITH PASSWD 'poner_passwd';`. Para ver que todo es correcto, se puede hacer `SELECT * FROM "pg_user";` y se verá el usuario creado con * en la contraseña. En Phppgadmin podremos seleccionar a la izquierda el servidor y configurar los parámetros indicando que el servidor es *localhost* y el usuario y la contraseña creados anteriormente. A partir de aquí, veremos las bases de datos, así como todos los parámetros y configuraciones.

Otra herramienta interesante es **Webmin**. Es una interfaz basada en la web para administrar sistemas para Unix que permite configurar usuarios, Apache, DNS, compartir archivos, servidores de bases de datos, etc. La ventaja de Webmin, sobre todo para usuarios que se inician en la administración, es que elimina la necesidad de editar manualmente los archivos de configuración y permite administrar un sistema de forma local o remota. La instalación para Debian se detalla en http://www.webmin.com/deb.html (Webmin fue retirado del repositorio Debian/Ubuntu en 2006 por retardos en la solución de errores y problemas en la política de mantenimiento del paquete https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=343897).

Enlaces de interés

Webmin no está disponible en algunas distribuciones y se debe bajar e instalar directamente de la web: http://www.webmin.com. Toda la documentación y la información de los módulos de Webmin disponibles se encuentra en: http://doxfer.webmin.com/Webmin.

1.3. MySQL

MySQL [7] es, según sus autores, la base de datos (DB) SQL abierta, es decir, de software libre (*Open Source*) más popular y utilizada por grandes compañías de Internet (Facebook, Google, Adobe, Twiter, Youtube, Zappos, entre otros...). La compañía que desarrolla y mantiene la BD y sus utilidades se denomina MySQL AB (en 2008 esta fue adquirida por Sun Microsystems y a su vez esta lo fue en 2010 por Oracle Corporation, por lo cual MySQL es subsidiaria de Oracle Co.) y desarrolla MySQL como software libre en un esquema de licencia dual. Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia (versión llamada MySQL Community Edition), pero para aquellas empresas que quieran incorporarlo en productos privados, deben comprar una licencia específica que les permita este uso (versiones MySQL Enterprise Edition y MySQL Cluster CGE básicamente). Es interesante analizar las diferentes situaciones (sobre todo en nuestra orientación hacia el Open Source) de licencias de MySQL. En la opción GPL, podemos utilizar libremente MySQL, es decir, sin coste alguno cuando la aplicación se desarrolla a nivel local y no se utiliza comercialmente. Por ejemplo, MySQL se pueden usar libremente dentro de un sitio web (si se desarrolla una aplicación PHP y se instala en un proveedor de servicios de Internet, tampoco se tiene que hacer que el código PHP esté disponible libremente en el sentido de la GPL).

Enlace de interés

Toda la documentación sobre MySQL se puede obtener desde la página web siguiente: http://dev.mysql.com /usingmysql/get_started.html.

Igualmente, un proveedor de servicios de Internet puede hacer MySQL disponible a sus clientes sin tener que pagar licencia de MySQL (siempre y cuando MySQL se ejecute exclusivamente en el equipo del ISP) o MySQL se puede utilizar de forma gratuita para todos los proyectos GPL o licencia libre comparable (por ejemplo, si hemos desarrollado un cliente de correo electrónico Open Source para GNU/Linux, y se desea almacenar datos de los correos en una base de datos MySQL) y todos los desarrollos/extensiones que se realicen sobre los productos MySQL debe ser abiertos/publicados y no comerciales. El uso de MySQL con una licencia comercial se aplica cuando desarrollamos productos comerciales y no están abiertos/publicados; es decir. no se puede desarrollar un producto comercial (p. ej.,. un programa de contabilidad con MySQL como base de datos) sin que el código esté disponible como código abierto. Si las limitaciones de la GPL no son aceptables para quien desarrolla la aplicación entonces se puede vender el producto (programa), junto con una licencia de MySQL comercial (más información en http://www.mysql.com/about/legal/). Considerando la licencia GPL, se puede obtener el código fuente, estudiarlo y modificarlo de acuerdo con sus necesidades sin pago alguno, pero estaremos sujetos a los deberes que impone GPL y deberemos publicar como código abierto lo que hemos desarrollado también. MySQL provee en su página web un conjunto de estadísticas y prestaciones en comparación con otras DB para mostrar al usuario cuán rápida, fiable y fácil es de usar. La decisión de elegir una DB se debe hacer cuidadosamente en función de las necesidades de los usuarios y del entorno donde se utilizará esta DB.

Al igual que PostgreSQL, MySQL es una base de datos relacional, es decir, que almacena los datos en tablas en lugar de en una única ubicación, lo cual permite aumentar la velocidad y la flexibilidad.

1.3.1. Instalación de MySQL

MySQL se puede obtener desde la página web* del proyecto o desde cualquiera de los repositorios de software. Se pueden obtener los binarios y los archivos fuente para compilarlos e instalarlos. En el caso de los binarios, utilizar la distribución de Debian y seleccionar los paquetes `mysql-*` (`client-5.5`, `server-5.5` y `common` son necesarios). La instalación, después de unas preguntas, creará un usuario que será el superusuario de la BD (p.ej., admin) y una entrada en `/etc/init.d/mysql` para arrancar o detener el servidor en el boot. También se puede hacer manualmente:

*http://www.mysql.com

```
/etc/init.d/mysql start|stop
```

Para acceder a la base de datos, se puede utilizar el monitor `mysql` desde la línea de comandos. Si obtiene los binarios (que no sean Debian ni RPM, para ellos simplemente utilizad las utilidades comunes `apt-get` y `rpm`), por ejemplo un archivo comprimido desde el sitio web de MySQL, deberá ejecutar los siguientes comandos para instalar la DB:

```
groupadd mysql
useradd -g mysql mysql
cd /usr/local
gunzip </path/to/mysql-VERSION-OS.tar.gz | tar xvf -
ln -s full-path-to-mysql-VERSION-OS mysql
cd mysql
scripts/mysql_install_db –user=mysql
chown -R root .
chown -R mysql data
chgrp -R mysql .
bin/mysqld_safe –user=mysql &
```

Esto crea el usuario/grupo/directorio, descomprime e instala la DB en el directorio `/usr/local/mysql`. En caso de obtener el código fuente, los pasos son similares:

```
groupadd mysql
useradd -g mysql mysql
gunzip <mysql-VERSION.tar.gz | tar -xvf -
cd mysql-VERSION
./configure --prefix=/usr/local/mysql
make
make install
cp support-files/my-medium.cnf /etc/my.cnf
cd /usr/local/mysql
bin/mysql_install_db –user=mysql
chown -R root .
```

```
chown -R mysql var
chgrp -R mysql .
bin/mysqld_safe –user=mysql &
```

Es importante prestar atención cuando se realiza la configuración, ya que `prefix=/usr/local/mysql` es el directorio donde se instalará la DB y se puede cambiar para ubicar la DB en el directorio que se desee.

1.3.2. Postinstalación y verificación de MySQL

Una vez realizada la instalación (ya sea de los binarios o del código fuente), se deberá verificar si el servidor funciona correctamente. En Debian se puede hacer directamente (el usuario es el que se ha definido durante la instalación a igual que su contraseña, indicados en los comandos por `-u` y `-p`, respectivamente):

```
/etc/init.d/mysql start     Inicia el servidor
mysqladmin -u user -p version     Genera información de versiones
mysqladmin -u user -p variables     Muestra los valores de las variables
mysqladmin -u root -p shutdown     Finaliza la ejecución del servidor
mysqlshow -u user -p     Muestra las DB predefinidas
mysqlshow mysql -u user -p     Muestra las tablas de la DB mysql
```

Si se instala desde el código fuente, antes de hacer estas comprobaciones se deben ejecutar los siguientes comandos para crear las bases de datos (desde el directorio de la distribución):

```
./scripts/mysql_install_db
cd DirectorioInstalacionMysql
./bin/mysqld_safe –user = mysql &
```

Si se instala desde binarios (RPM, Pkg ...), se debe hacer lo siguiente:

```
cd DirectorioInstalacionMysql
./scripts/mysql_install_db
./bin/mysqld_safe user = mysql &
```

El *script* `mysql_install_db` crea la DB `mysql` y `mysqld_safe` arranca el servidor `mysqld`. A continuación, se pueden probar todos los comandos dados anteriormente para Debian, excepto el primero, que es el que arranca el servidor. Además, si se han instalado los *tests*, se podrán ejecutar con `cd sql-bench` y después, con `run-all-tests`. Los resultados se encontrarán en el directorio `sql-bech/Results` para compararlos con otras DB.

1.3.3. El programa monitor (cliente) mysql

El cliente `mysql` se puede utilizar para crear y utilizar DB simples, es interactivo y permite conectarse al servidor, ejecutar búsquedas y visualizar los resultados. También funciona en modo *batch* (como un *script*) donde los comandos se le pasan a través de un archivo. Para ver todas las opciones del comando, se

puede ejecutar `mysql -help`. Podremos realizar una conexión (local o remota) con el comando mysql, por ejemplo, para una conexión por la interfaz de red, pero desde la misma máquina:

```
mysql -h localhost -u usuario -p [NombreDB]
```

Si no se pone el último parámetro, no se selecciona ninguna DB. Una vez dentro, el `mysql` pondrá un *prompt* (**mysql>**) y esperará a que le introduzcamos algún comando (propio y SQL), por ejemplo, *help*. A continuación, daremos una serie de comandos para probar el servidor (recordad poner siempre el ';' para terminar el comando):

```
mysql>SELECT VERSION(), CURRENT_DATE;
```
Se pueden utilizar mayúsculas o minúsculas.
```
mysql>SELECT SIN(PI()/4), (4+1)*5;
```
Calculadora.
```
mysql>SELECT VERSION(); SELECT NOW();
```
Múltiples comandos en la misma línea
```
mysql>SELECT
 ->USER()
 ->,
 ->CURRENT_DATE;
```
o en múltiples líneas.
```
mysql>SHOW DATABASES;
```
Muestra las DB disponibles.
```
mysql>USE test
```
Cambia la DB.
```
mysql>CREATE DATABASE nteum; USE nteum;
```
Crea y selecciona una DB llamada nteum.
```
mysql>CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
->species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```
Crea una tabla dentro de nteum.
```
mysql>SHOW TABLES;
```
Muestra las tablas.
```
mysql>DESCRIBE pet;
```
Muestra la definición de la tabla.
```
mysql>LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```
Carga datos desde `pet.txt` en pet. El archivo `pet.txt` debe tener un registro por línea separado por tabulaciones de los datos, de acuerdo con la definición de la tabla (fecha en formato AAAA-MM-DD)
```
mysql>INSERT INTO pet
->VALUES ('Marciano','Estela','gato','f','1999-03-30',NULL);
```
Carga los datos *in-line*.
```
mysql>SELECT * FROM pet;
```
Muestra los datos de la tabla.
```
mysql>UPDATE pet SET birth = "1989-08-31" WHERE name = "Browser";
```
Modifica los datos de la tabla.
```
mysql>SELECT * FROM pet WHERE name = "Browser";
```
Muestra selectiva.
```
mysql>SELECT name, birth FROM pet ORDER BY birth;
```
Muestra ordenada.
```
mysql>SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```
Muestra selectiva con funciones.
```
mysql>GRANT ALL PRIVILEGES ON *.* TO marciano@localhost ->IDENTIFIED
BY 'passwd'WITH GRANT OPTION;
```
Crea un usuario marciano en la DB. Lo debe hacer el root de la DB. También se puede hacer directamente con:
```
mysql>INSERT INTO user (Host,User,Password) ->
VALUES('localhost','marciano','passwd');
mysql>select user,host,password from mysql.user;
```
Muestra los usuarios, host de conexión, y passwd (también se pueden poner los comandos en minúsculas).
```
mysql>delete from mysql.user where user="";
```
Borra los usuarios anónimos (sin nombre en la info anterior).

1.3.4. Caso de uso: procesamiento de datos con MySQL

Consideraremos los datos del Banco Mundial de datos y concretamente los de Internet: ancho de banda internacional (bits por persona). Desde la dirección http://datos.bancomundial.org/indicador podemos descargar una hoja de datos que tendrá una información como (la información muestra el parcial de un total de 196 países y desde el año 1960):

```
País ... 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
...
España 297,14 623,61 1126,83 1938,19 2821,65 2775,71 6058,60 11008,05...
```

Se creará una base de datos en MySQL para importar estos datos y generar los listados que calculen la media por país y la media anual, y ordenen de mayor a menor el volumen para el año 2008; y un listado de los 10 países con mayor utilización de Internet por año. Sobre estos últimos datos, se debe hacer una clasificación de los cinco países con mayor utilización de Internet desde que existen datos (se muestran los comandos más relevantes y no para todos los datos).

```
mysql -u root -p
Welcome to the MySQL monitor. Commands end with ; or \g.
Server version: 5.5.35-0+wheezy1 (Debian)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database bancomundial;

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| bancomundial       |
| mysql              |
+--------------------+
3 rows in set (0.00 sec)

mysql> use bancomundial;
Database changed
mysql> create table paisbai(pais varchar(100), 1960
varchar(100), 1961 varchar(100), 1962 varchar(100), 1963
varchar(100), 1964 varchar(100), 1965 varchar(100), 1966
varchar(100), 1967 varchar(100), 1968 varchar(100), 1969
varchar(100), 1970 varchar(100), 1971 varchar(100), 1972
varchar(100), 1973 varchar(100), 1974 varchar(100), 1975
varchar(100), 1976 varchar(100), 1977 varchar(100), 1978
varchar(100), 1979 varchar(100), 1980 varchar(100), 1981
varchar(100), 1982 varchar(100), 1983 varchar(100), 1984
varchar(100), 1985 varchar(100), 1986 varchar(100), 1987
varchar(100), 1988 varchar(100), 1989 varchar(100), 1990
varchar(100), 1991 varchar(100), 1992 varchar(100), 1993
varchar(100), 1994 varchar(100), 1995 varchar(100), 1996
varchar(100), 1997 varchar(100), 1998 varchar(100), 1999
varchar(100), 2000 varchar(100), 2001 varchar(100), 2002
varchar(100), 2003 varchar(100), 2004 varchar(100), 2005
varchar(100), 2006 varchar(100), 2007 varchar(100), 2008
varchar(100), 2009 varchar(100), 2010 varchar(100) );

mysql>
CREATE TABLE 'bancomundial'.'paisbai' ('pais' VARCHAR(100) NOT
NULL, '1960' VARCHAR(100) NOT NULL, '1961' VARCHAR(100) NOT
```

```
NULL, '1962' VARCHAR(100) NOT NULL, '1963' VARCHAR(100) NOT
NULL, '1964' VARCHAR(100) NOT NULL, '1965' VARCHAR(100) NOT
NULL, '1966' VARCHAR(100) NOT NULL, '1967' VARCHAR(100) NOT
NULL, '1968' VARCHAR(100) NOT NULL, '1969' VARCHAR(100) NOT
NULL, '1970' VARCHAR(100) NOT NULL, '1971' VARCHAR(100) NOT
NULL, '1972' VARCHAR(100) NOT NULL, '1973' VARCHAR(100) NOT
NULL, '1974' VARCHAR(100) NOT NULL, '1975' VARCHAR(100) NOT
NULL, '1976' VARCHAR(100) NOT NULL, '1977' VARCHAR(100) NOT
NULL, '1978' VARCHAR(100) NOT NULL, '1979' VARCHAR(100) NOT
NULL, '1980' VARCHAR(100) NOT NULL, '1981' VARCHAR(100) NOT
NULL, '1982' VARCHAR(100) NOT NULL, '1983' VARCHAR(100) NOT
NULL, '1984' VARCHAR(100) NOT NULL, '1985' VARCHAR(100) NOT
NULL, '1986' VARCHAR(100) NOT NULL, '1987' VARCHAR(100) NOT
NULL, '1988' VARCHAR(100) NOT NULL, '1989' VARCHAR(100) NOT
NULL, '1990' VARCHAR(100) NOT NULL, '[...]
mysql> describe paisbai;
+-------------------------------------+
|Field | Type         |Null |K| Def. |E|
| pais | varchar(100) | YES | | NULL | |
| 1960 | varchar(100) | YES | | NULL | |
| 1961 | varchar(100) | YES | | NULL | |
| 1962 | varchar(100) | YES | | NULL | |
| 1963 | varchar(100) | YES | | NULL | |
| 1964 | varchar(100) | YES | | NULL | |
| 1965 | varchar(100) | YES | | NULL | |
...
| 2009 | varchar(100) | YES | | NULL | |
| 2010 | varchar(100) | YES | | NULL | |
+-------------------------------------+

1. Para cargar los datos, se puede entrar en http://localhost/phpmyadmin e
importar los datos desde un fichero a la base de datos bancomundial o desde la línea del mysql.
LOAD DATA LOCAL INFILE '/tmp/phpO5Dhpl' REPLACE INTO TABLE 'paisbai'
FIELDS TERMINATED BY ';'

ENCLOSED BY '"'

ESCAPED BY '\\'
LINES TERMINATED BY '\n';

2. Listado que permite calcular la media por país.
consulta SQL: SELECT 'pais',
('1960'+'1961'+'1962'+'1963'+'1964'+'1965'+'1966'+'1967'+'1968'+
'1969'+'1970'+'1971'+'1972'+'1973'+'1974'+'1975'+'1976'+'1977'+'
1978'+'1979'+'1980'+'1981'+'1982'+'1983'+'1984'+'1985'+'1986'+'1
987'+'1988'+'1989'+'1990'+'1991'+'1992'+'1993'+'1994'+'1995'+'19
96'+'1997'+'1998'+'1999'+'2000'+'2001'+'2002'+'2003'+'2004'+'200
5'+'2006'+'2007'+'2008'+'2009'+'2010')/51 as mediapais FROM
'paisbai' LIMIT 0, 30 ;
Filas: 30
pais        mediapais
Afganistán  0.0203921568627
Albania     7.3696078431373
Argelia     0.4425490196078

3. Generar un listado que visualice la media anual.
consulta SQL: SELECT AVG ('1989') AS '1989', AVG('1990') as
'1990', AVG('1991') as '1991', AVG('1992') as '1992',
AVG('1993') as '1993', AVG('1994') as '1994', AVG('1995') as
'1995', AVG('1996') as '1996', AVG('1997') as '1997',
AVG('1998') as '1998', AVG('1999') as '1999', AVG('2000') as
'2000', AVG('2001') as '2001', AVG('2002') as '2002',
AVG('2003') as '2003', AVG('2004') as '2004', AVG('2005') as
'2005', AVG('2006') as '2006', AVG('2007') as '2007',AVG('2008')
as '2008', AVG('2009') as '2009', AVG('2010') as '2010' FROM
'paisbai';
Filas: 1
1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000
0.0001 0.000 0.000 0.001 0.0019 0.005 0.044 0.158 0.6547 1.3870 27.483 126.9760 387.70

3. Generar un listado que visualice el orden de mayor a menor volumen para el año 2008.
SELECT 'pais', '2008' FROM 'paisbai' ORDER BY '2008' DESC LIMIT 0 , 30;
```

```
+--------------------+
| pais     | 2008    |
| Lituania | 9751.01 |
| Mongolia | 946.53  |
| Rumania  | 9110.51 |
| Zimbabwe | 9.71    |
...
| Bhután   | 65.52   |
| Hungría  | 5977.17 |
| Viet Nam | 580.72  |
+--------------------+

4. Generar un listado de los 10 países con mayor utilización de Internet por año.
Alternativa 1.
SELECT 'pais' , SUM( '1989' ) AS '1989' , SUM( '1990' ) AS
'1990' , SUM( '1991' ) AS '1991' , SUM( '1992' ) AS '1992' ,
SUM( '1993' ) AS '1993' , SUM( '1994' ) AS '1994' ,
SUM( '1995' ) AS '1995' , SUM( '1996' ) AS '1996' ,
SUM( '1997' ) AS '1997' , SUM( '1998' ) AS '1998' ,
SUM( '1999' ) AS '1999' , SUM( '2000' ) AS '2000' ,
SUM( '2001' ) AS '2001' , SUM( '2002' ) AS '2002' ,
SUM( '2003' ) AS '2003' , SUM( '2004' ) AS '2004' ,
SUM( '2005' ) AS '2005' , SUM( '2006' ) AS '2006' ,
SUM( '2007' ) AS '2007' , SUM( '2008' ) AS '2008' ,
SUM( '2009' ) AS '2009' , SUM( '2010' ) AS '2010'
FROM 'paisbai'
ORDER BY 'pais' DESC
LIMIT 0 , 10;

Alternativa 2:
SELECT 'pais' , MAX( '1989' ) AS '1989' , MAX( '1990' ) AS
'1990' , MAX( '1991' ) AS '1991' , MAX( '1992' ) AS '1992' ,
MAX( '1993' ) AS '1993' , MAX( '1994' ) AS '1994' ,
MAX( '1995' ) AS '1995' , MAX( '1996' ) AS '1996' ,
MAX( '1997' ) AS '1997' , MAX( '1998' ) AS '1998' ,
MAX( '1999' ) AS '1999' , MAX( '2000' ) AS '2000' ,
MAX( '2001' ) AS '2001' , MAX( '2002' ) AS '2002' ,
MAX( '2003' ) AS '2003' , MAX( '2004' ) AS '2004' ,
MAX( '2005' ) AS '2005' , MAX( '2006' ) AS '2006' ,
MAX( '2007' ) AS '2007' , MAX( '2008' ) AS '2008' ,
MAX( '2009' ) AS '2009' , MAX( '2010' ) AS '2010'
FROM 'paisbai'
GROUP BY 'pais'
LIMIT 0 , 10;
+----------------+------------------+
| país           | promedio         |
+----------------+------------------+
| Luxemburgo     |    284474.679628 |
| Hong Kong      | 21468.6420499333 |
| San Marino     | 5464.22342423529 |
| Países Bajos   | 3949.18559792941 |
| Dinamarca      |     3743.7899214 |
| Estonia        | 3118.98744675686 |
| Suecia         | 2967.78367829608 |
| Reino Unido    | 1902.25120777059 |
| Suiza          | 1897.13803142745 |
| Bélgica        | 1781.95881669216 |
+----------------+------------------+
```

1.3.5. Administración de MySQL

MySQL dispone de un archivo de configuración en `/etc/mysql/my.cnf` (en Debian), al que se pueden cambiar las opciones por defecto de la DB, como por ejemplo, el puerto de conexión, el usuario, la contraseña de los usuarios remotos, los archivos de registro (*logs*), los archivos de datos, si acepta conexiones externas, etc. Con respecto a la seguridad, se deben tomar algunas precauciones:

1) No dar a nadie (excepto al usuario root de Mysql) acceso a la tabla `user` dentro de la DB mysql, ya que aquí se encuentran las contraseñas de los usuarios que podrían utilizarse con otros fines.

2) Verificar `mysql -u root`. Si se puede acceder, significa que el usuario root no tiene contraseña. Para cambiarlo, se puede hacer:

```
mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('new_password')
-> WHERE user = 'root';
mysql> FLUSH PRIVILEGES;

Ahora, para conectarse como root: mysql -u root -p mysql
```

3) Comprobar la documentación* respecto a las condiciones de seguridad y del entorno de red para evitar problemas de ataques o intrusiones.

*http://dev.mysql.com/usingmysql/get_started.html

4) Para hacer copias de la base de datos, se puede utilizar el comando:

```
    mysqldump --tab = /DirectorioDestino --opt NombreDB
o también:
    mysqlhotcopy NombreDB /DirectorioDestino
```

Asimismo, se pueden copiar los archivos con extensión FRM, MYD, y MYI con el servidor parado. Para recuperar, se puede ejecutar mediante el comando `REPAIR TABLE` o `myisamchk -r`, lo cual funcionará en la mayoría de las veces. En caso contrario, se podrían copiar los archivos guardados y arrancar el servidor. Existen otros métodos alternativos en función de lo que se quiera recuperar, como la posibilidad de guardar/recuperar parte de la DB (consultad la documentación).

1.3.6. Interfaces gráficas

Para Mysql hay gran cantidad de interfaces gráficas, incluso propias del paquete MySQL. Una de ellas es **MySQL Workbench***, que es una aplicación potente para el diseño administración y control de bases de datos basadas en MySQL (incluye el *Database Administration* que reemplaza el anterior MySQL Administrator). Esta aplicación es un conjunto de herramientas para los desarrolladores y administradores de BD que integra el desarrollo, gestión, control y el mantenimiento de forma simple y en un mismo entorno de la BD. Las partes principales son: diseño y modelado de BD, desarrollo en SQL (reemplaza el MySQL Query Browser), administración de la BD (reemplaza MySQL Administrator) y migración de BD.

Enlace de interés

Se puede encontrar toda la documentación para la instalación y puesta en marcha de Mysql Administrator en http://dev.mysql.com/downloads/workbench/index.html.

*http://www.mysql.com/downloads/workbench

Para usuarios recién iniciados (o expertos) recomendamos, por su facilidad en el uso y simplicidad, **PhpMyAdmin** (incluida en la mayoría de las distribuciones incluida Debian), que es una herramienta de software libre escrito en PHP para gestionar la administración de MySQL a través de la web. PhpMyAdmin es compatible con un amplio conjunto de operaciones en MySQL, como

por ejemplo, gestión de bases de datos, tablas, campos, relaciones, índices, usuarios, permisos, etc. y también tiene la capacidad de ejecutar directamente cualquier sentencia SQL. Su instalación se puede hacer desde la gestión de programas de distribución de trabajo (`apt/rpm`, `yum/aptitude`, etc.) que durante el proceso de instalación pedirá con qué servidores de web se quiere hacer la integración y el usuario de la base de datos. Una vez instalada, se puede iniciar a través de `http://localhost/phpmyadmin`, que pedirá el usuario y la contraseña del servidor (indicados en los pasos anteriores).

Existen otras herramientas que permiten hacer tareas similares, como la que ya hemos comentado en la sección de PostgreSQL como **Webmin**, que permite gestionar y administrar bases de datos MySQL (incluyendo el módulo correspondiente). Si bien este paquete ya no se incluye con algunas distribuciones (p. ej., Debian|Ubuntu), se puede descargar desde http://www.webmin.com. Durante la instalación, el Webmin avisará de que el usuario principal será el root y usará la misma contraseña que el root del sistema operativo. Será posible conectarse, p. ej., desde un navegador `https://localhost:10000`, el cual solicitará aceptar (o denegar) la utilización del certificado para la comunicación SSL y, a continuación, mostrará todos los servicios que puede administrar, entre ellos Mysql Data Base Server.

1.4. MariaDB

MariaDB[9] es un sistema de gestión de bases de datos derivado de MySQL con licencia GPL que incorpora dos mecanismos de almacenamiento nuevos: Aria (que reemplaza con amplias ventajas a MyISAM) y XtraDB (que sustituye InnoDB el actual del Mysql). Como *fork* de MySQL mantiene una alta compatibilidad ya que posee los mismos comandos, interfaces, API y librerías con el objetivo que se pueda cambiar un servidor por otro. La motivación de los desarrolladores de hacer un *fork* de MySQL fue la compra de Sun Microsystems por parte de Oracle en el 2010 (Sun previamente había comprado MySQL AB que es la compañía que desarrolla y mantiene MySQL) y con el objetivo de mantener esta sobre GPL ya que estaban/están convencidos que el único interés de Oracle en MySQL era reducir la competencia que MySQL daba a su producto comercial (Oracle DB).

MariaDB es equivalente a las mismas versiones de MySQL (MySQL 5.5 = MariaDB 5.5, no obstante, la versión actual es MariaDB10.1) y existen una serie de ventajas con relación a MySQL que resumimos a continuación*:

***https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/**

1) Mecanismos de almacenamiento: Además de los estándar (MyISAM, Blackhole, CSV, Memory y Archive) se incluyen Aria (alternativa a MyISAM resistente a caídas), XtraDB (reemplazo directo de InnoDB), FederatedX (reemplazo directo de Federated), OQGRAPH, SphinxSE, Cassandra (NoSQL y en MariaDB 10.0) TokuDB (desde MariaDB 5.5) y se están trabajando en otras NoSQL, y las últimas CONNECT, SEQUENCE y Spider (solo a partir de MariaDB 10.0)

2) Facilidad de uso y velocidad: incluye nuevas tablas y opciones para generar estadísticas de índices y tablas, procesos, gestión del tiempo en microsegundos, características NoSQL (p. ej.,. HandlerSocket), columnas dinámicas y subqueries. Con relación a la velocidad de procesamiento se pueden ver las comparaciones con relación a MySQL en https://mariadb.com/blog/mariadb-53-optimizer-benchmark.

3) Test, errores y alertas: incluye un extenso conjunto de test en la distribución que permiten analizar diferentes combinaciones de configuración y sistema operativo. Estos tests han permitido reducir notablemente los errores y alertas por lo cual se traduce en una BD altamente estable y segura con relación a la ejecución y funcionamiento.

4) Licencia: TODO el código en MariaDB está bajo GPL, LPGL o BSD y no dispone de módulos cerrados como MySQL Enterprise Ed. (De hecho todo el código cerrado en MySQL 5.5 E.Ed. está en MariaDB como *open source version*). MariaDB incluye test para todos los errores solucionados, mientras que Oracle no hace lo mismo con MySQL 5.5 y todos los errores y planes de desarrollos son públicos y los desarrolladores pertenecen a la comunidad.

Es importante tener en cuenta que MariaDB es desarrollada por la comunidad y la responsabilidad de desarrollo y mantenimiento corren a cargo de SkySQL Corporation Ab, empresa fundada en 2010 por la mismas personas de MySQL AB (David Axmark, Michael 'Monty' Widenius y Kaj Arnö). Esta empresa se fusionó con el Monty Program en 2013 (programa fundado por Monty Widenius -desarrollador de MySQL junto con D. Axmark- después que vendió la compañía a Sun MS y decidió escribir MariaBD). En 2014 SkySQL crea otra marca llamada MariaDB AB y ofrece MariaDB Enterprise, MariaDB Enterprise Cluster y MaxScale bajo un modelo diferente de negocio que la licencia de software como MySQL, para dirigirse a un mercado empresarial garantizando fiabilidad y confiabilidad en aplicaciones de misión crítica y alto rendimiento.*https://mariadb.com/about*

1.4.1. Instalación MariaDB sobre Debian

La instalación es sumamente sencilla y tal como se indica en [10] (los paquetes estarán en el repositorio Debian a partir de la próxima version Jessie). Se selecciona la distribución, versión del SO y repositorio, y se ejecuta:

```
Si tenemos BD en MySQL hacer una copia de resguardo parando los servicios primero y luego volcando la BD:
        service apache2 stop; service mysql stop
        mysqldump -u root -p --all-databases > mysqlbackup.sql
Configuramos el repositorio:
        apt-get install python-software-properties
        apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
        add-apt-repository 'deb http://mirror.vpsfree.cz/mariadb/repo/10.1/debian wheezy main'
Luego de importada la llave actualizamos el repositorio e instalamos:
        apt-get update
        apt-get install mariadb-server
```

```
        service apache2 start
Podremos mirar la versión con: mysql  --version
        mysql  Ver 15.1 Distrib 10.1.0-MariaDB, for debian-linux-gnu (x86_64) using readline 5.1
O las bases de datos existentes con:
        mysql -u root -p -Be 'show databases'
Y si accedemos a phpmyadmin veremos en "Home"
        MySQL
        Server: Localhost via UNIX socket
        Server version: 10.1.0-MariaDB-1~wheezy
        Protocol version: 10
        User: root@localhost
        MySQL charset: UTF-8 Unicode (utf8)
```

Veremos que si tenemos instalado `mysql-server` lo desinstala y lo reemplaza por `mariadb`, haciendo las mismas preguntas de las bases de datos creadas (usuario y *passwd*).

1.5. SQLite

Como ya dijimos en la introducción, SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, contenida en una librería (escrita en C) bajo licencia *Public Domain* (http://www.sqlite.org/copyright.html). Como gran diferencia de los sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente del programa que desea realizar un acceso a la base de datos, sino una librería que se enlaza junto con el programa y pasa a ser parte del mismo como cualquier otra librería (p. ej., igual que la */lib/x86_64-linux-gnu/libc.so.6* en Debian para un programa en C). El programa que necesita la funcionalidad de SQLite simplemente llama subrutinas y funciones teniendo como ventaja substancial la reducción de la latencia de acceso a la base de datos ya que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados en un archivo fichero sobre la máquina *host* y la coherencia en las transacciones se obtiene bloqueando todo el fichero al principio de cada transacción. La biblioteca implementa la mayor parte del estándar SQL-92, y mantiene las características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción (ACID compliant) y su diseño permite que varios procesos o *threads* puedan acceder a la misma base de datos sin problemas donde los accesos de lectura pueden ser servidos en paralelo y un acceso de escritura solo puede ser servido si no se está haciendo ningún otro acceso en forma concurrente (lo dará un error o podrá reintentarse durante un cierto *timeout* programable).[11]

En Debian se incluye un paquete que implementa una interfaz en línea de comandos para SQlite2/3 llamado `sqlite` (versión2) o `sqlite3` (`apt-get install sqlite3`) para acceder y modificar BD SQLite además de un conjunto de librerías que permiten la interacción entre el lenguaje y la aplicación, por ejemplo, python-sqlite, ruby-sqlite, php5-sqlite como interfaces para SQLite desde phyton, ruby y php respectivamente. Además existen otras herra-

mientas para diseñar, crear o editar una base de datos compatible con SQLite, como por ejemplo SQLite Database Browser (GPL) (`apt-get install sqlitebrowser`) que permite en forma simple y visual acceder a los ficheros que contiene la base de datos [12]. Entre los principales usuarios de SQLite existen (http://www.sqlite.org/famous.html): Adobe (Photoshop Lightroom, AIR), Airbus (flight software A350), Apple (OSX, iPhone, iPod), Dropbox, Firefox, Thunderbird, Google (Android, Chrome), PHP, Python, Skype, Tcl/Tk entre otros.

Para instalar la librería SQLite en Debian, ejecutamos la instrucción `apt-get install libsqlite0-dev` (que incluye la librería y los archivos .h para desarrollar aplicaciones en C, si solo necesitamos la librería con *libsqlite0* hay suficiente). Para visualizar por ejemplo una de las bases de datos de Firefox hacemos:

```
Buscamos nuestro profile de firefox/iceweasel en el usuario de login: cd .mozilla/firefox/*.default
Abrimos, por ejemplo, la base de datos: sqlite3 places.sqlite
            SQLite version 3.7.13 2012-06-11 02:05:22
            Enter ".help" for instructions. Enter SQL statements terminated with a ";"
Miramos la tablas con   .tables
            moz_anno_attributes  moz_favicons         moz_items_annos
            moz_annos            moz_historyvisits    moz_keywords
            moz_bookmarks        moz_hosts            moz_places
            moz_bookmarks_roots  moz_inputhistory
Miramos la tabla moz_hosts:    SELECT * FROM moz_hosts;
            1|mozilla.org|140|0|
            2|sysdw.nteum.org|11180|1|
            3|127.0.0.1|2000|1|
            4|10.0.0.58|0|0|
Miramos la tabla moz_bookmarks (antes hacemos un bookmark, por ejemplo a SQLite.org: SELECT *
FROM moz_bookmarks;
            ...
            24|1|164|2|4|SQLite Home Page|||1404403846763242|1404403846777354|yfpoHwYrL7Ys
```

En http://www.sqlite.org/quickstart.html tendremos una guía de cómo acceder a través de las diferentes interfaces de programación a la BD y documentación para trabajar con la línea de comandos http://www.sqlite.org/cli.html. La documentación la podemos encontrar en http://www.sqlite.org/docs.html.

1.6. Source Code Control System

Los sistemas de Control de Revisiones (o también conocidos como control de versiones/código fuente) se refieren a la gestión de los cambios en los archivos, programas, páginas o cualquier información que es modificada y actualizada, ya sea un único usuario para llevar un control que ha hecho y cuando o por múltiples usuarios que trabajan simultáneamente en el mismo proyecto. Los cambios se identifican generalmente por un número o código, denominado "número de revisión", "nivel de revisión", "revisión" y el sistema de control permiten volver atrás, aceptar un cambio, publicarlo o saber quién lo ha realizado y qué ha hecho. El sistema trabaja con marcas de tiempos, repositorios

(para guardar las diferentes versiones) y usuarios y permitirá, en función de estos parámetros, comparar los repositorios/archivos, restaurarlos y/o fusionarlos. En la actualidad es una técnica aplicada en grandes aplicaciones informáticas (como por ejemplo, en Google Docs o Wikipedia -o en cualquier wiki-) pero existen diferentes aplicaciones que pueden trabajar como servidores o aplicaciones internas o en modo cliente servidor en función de las necesidades del proyecto/usuarios como por ejemplo CVS (unos de los iniciales junto con RCS), subversion (muy extendido), Git (muy utilizado en proyectos Open Source) o Mercurial (muy utilizado por su aspecto de distribuido -no es necesario un servidor-).

1.6.1. Concurrent Versions System (CVS)

El ***Concurrent Versions System*** (CVS) es un sistema de control de versiones que permite mantener versiones antiguas de archivos (generalmente código fuente), guardando un registro (*log*) de quién, cuándo y porqué fueron realizados los cambios.

A diferencia de otros sistemas, CVS no trabaja con un archivo/directorio por vez, sino que actúa sobre colecciones jerárquicas de los directorios que controla. El CVS tiene por objetivo ayudar a gestionar versiones de software y controla la edición concurrente de archivos fuente por múltiples autores. El CVS utiliza internamente otro paquete llamado **RCS** (*Revision Control System*) como una capa de bajo nivel. Si bien el RCS puede ser utilizado independientemente, esto no se aconseja, ya que CVS, además de su propia funcionalidad, presenta todas las prestaciones de RCS, pero con notables mejoras en cuanto a la estabilidad, el funcionamiento y el mantenimiento. Entre ellas, cabe destacar: funcionamiento descentralizado (cada usuario puede tener su propio árbol de código), edición concurrente, comportamiento adaptable mediante *shell scripts*, etc. [13].

En primer lugar, se debe instalar el Concurrent Versions System (CVS) desde la distribución y que deberemos instalar también OpenSSH, si se quiere utilizar conjuntamente con CVS para acceso remoto. Las variables de entorno `EDITOR CVSROOT` deben estar inicializadas, por ejemplo, en `/etc/profile` (o en `.bashrc` o `.profile`):

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
```

Obviamente, los usuarios pueden modificar estas definiciones utilizando ~/.bashrc y se debe crear el directorio donde estará el repositorio y configurar los permisos; como *root*, hay que hacer, por ejemplo:

```
export CVSROOT = /usr/local/cvsroot
groupadd cvs
```

```
useradd -g cvs -d $CVSROOT cvs
dfmkdir $CVSROOT
chgrp -R cvs $CVSROOT
chmod o-rwx $CVSROOT
chmod ug+rwx $CVSROOT
```

Para inicializar el repositorio y poner archivo de código en él:

```
cvs -d /usr/local/cvsroot init
```

`cvs init` tendrá en cuenta no sobrescribir nunca un repositorio ya creado para evitar pérdidas de otros repositorios. Luego, se deberán agregar los usuarios que trabajarán con el CVS al grupo cvs; por ejemplo, para agregar el usuario nteum:

```
usermod -G cvs,nteum
```

Ahora el usuario nteum deberá introducir sus archivos en el directorio del repositorio (`/usr/local/cvsroot` en nuestro caso):

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
export CVSREAD = yes
cd directorio_de_originales
cvs import NombreDelRepositorio vendor_1_0 rev_1_0
```

El nombre del repositorio puede ser un identificador único o también puede ser `usuario/proyecto/xxxx` si es que el usuario desea tener organizados sus repositorios. Esto creará un árbol de directorios en `CVSROOT` con esa estructura y añade un directorio (`/usr/local/cvsroot/NombreDelRepositorio`) en el repositorio con los archivos que a partir de este momento estarán en el repositorio. Una prueba para saber si se ha almacenado todo correctamente es almacenar una copia en el repositorio, crear después una copia desde allí y comprobar las diferencias. Por ejemplo, si los originales están en el `directorio_del_usuario/dir_org` y se desea crear un repositorio como `primer_cvs/proj`, se deberán ejecutar los siguientes comandos:

```
cd dir_org    Cambiar al directorio del código fuente original.
cvs import -m "Fuentes originales"primer_cvs/proj usuarioX vers0
```

Crea el repositorio en `primer_cvs/proj` con `usuarioX` y `vers0`.

```
cd..    Cambiar al directorio superior de dir_org.
cvs checkout primer_cvs/proj
```

Generar una copia del repositorio. La variable `CVSROOT` debe estar inicializada, de lo contrario, se deberá indicar todo el camino.

```
diff -r dir_org primer_cvs/proj
```

Muestra las diferencias entre uno y otro. No deberá haber ninguna excepto por el directorio `primer_cvs/proj/CVS` que ha creado el CVS.

```
rm -r dir_org
```

Borra los originales (realizad siempre una copia de resguardo por motivos de seguridad y para tener una referencia de dónde se inició el trabajo con el CVS).

```
Estructura de los directorios, archivos y ramas:
/home/nteum/primer_cvs/proj: a.c b.c c.c Makefile
            usuarioX, vers0.x  -> Trabajar solo con este directorio después del import
/home/nteum/dir_org/: a.c b.c c.c Makefile              Después del checkout, deberá borrarse

/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.1
/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.1.1  -> Branch 1.1

/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.2
/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.x
```

El hecho de borrar los originales no siempre es una buena idea, salvo en este caso, después de que se haya verificado que están en el repositorio, para que no se trabaje sobre ellos por descuido y para que los cambios no queden reflejados sobre el CVS. Sobre máquinas donde los usuarios quieren acceder (por `ssh`) a un servidor CVS remoto, se deberá hacer:

```
export CVSROOT = ":ext:user@CVS.server.com:/home/cvsroot";
export CVS_RSH = "ssh"
```

donde `user` es el *login* del usuario y `cvs.server.com`, el nombre del servidor en el que está CVS. CVS ofrece una serie de comandos (se llaman con `cvs comd opciones`...) para trabajar con el sistema de revisiones, entre ellos: `checkout`, `update`, `add`, `remove`, `commit` y `diff`.

Comandos de CVS

`cvs checkout` es el comando inicial y crea su copia privada del código fuente para luego trabajar con ella sin interferir en el trabajo de otros usuarios (como mínimo se crea un subdirectorio donde estarán los archivos).

`cvs update` se debe ejecutar del árbol privado cuando hay que actualizar sus copias de archivos fuente con los cambios que otros programadores han hecho sobre los archivos del repositorio.

`cvs add file` es un comando necesario cuando hay que agregar nuevos archivos en su directorio de trabajo sobre un módulo donde ya se ha hecho previamente un *checkout*. Estos archivos se enviarán al repositorio CVS cuando se ejecute el comando `cvs commit`.

`cvs import` se puede usar para introducir archivos nuevos en el repositorio.

`cvs remove file` se utilizará para borrar archivos del repositorio (una vez que se hayan borrado del archivo privado). Este comando debe ir acompañado de un `cvs commit` para que los cambios sean efectivos, ya que se trata del comando que transforma todas las peticiones de los usuarios sobre el repositorio.

`cvs diff file` se puede utilizar sin que afecte a ninguno de los archivos implicados si se necesita verificar las diferencias entre repositorio y directorio de trabajo o entre dos versiones.

`cvs tag -R "versión"` se puede utilizar para introducir un número de versión en los archivos de un proyecto y después hacer un `cvs commit` y un proyecto `cvs checkout -r 'version'` para registrar una nueva versión.

Una característica interesante del CVS es que puede aislar cambios de los archivos en una línea de trabajo separada llamada ramificación o rama (*branch*). Cuando se cambia un archivo sobre una rama, estos cambios no aparecen sobre los archivos principales o sobre otras ramas. Más tarde, estos cambios se pueden incorporar a otros ramas o al archivo principal (*merging*). Para crear una nueva rama, utilizad `cvs tag -b rel-1-0-patches` dentro del directorio de trabajo, lo cual asignará a la rama el nombre de `rel-1-0-patches`. La unión de ramas con el directorio de trabajo significa utilizar el comando `cvs update -j`. Consultad las referencias para mezclar o acceder a diferentes ramas.

Ejemplo de una sesión

Siguiendo el ejemplo de la documentación dada en las referencias, se mostrará una sesión de trabajo (en forma general) con CVS. Como CVS almacena todas los archivos en un repositorio centralizado, se asumirá que el mismo ya ha sido inicializado anteriormente. Consideremos que se está trabajando con un conjunto de archivos en C y un Makefile, por ejemplo. El compilador utilizado es gcc y el repositorio es inicializado a `gccrep`. En primer lugar, se debe obtener una copia de los archivos del repositorio a nuestra copia privada con:

```
cvs checkout gccrep
```

Esto creará un nuevo directorio llamado `gccrep` con los archivos fuente. Si se ejecuta `cd gccrep; ls`, se verá por ejemplo `CVS makefile a.c b.c c.c`, donde existe un directorio CVS que se crea para el control de la copia privada que normalmente no es necesario tocar. Después de esto, se podría utilizar un editor para modificar `a.c` e introducir cambios sustanciales en el archivo (consultad la documentación sobre múltiples usuarios concurrentes si se necesita trabajar con más de un usuario en el mismo archivo), compilar, volver a cambiar, etc. Cuando se decide que se tiene una versión nueva con todos los cambios introducidos en `a.c` (o en los archivos que sea necesario), es el momento de hacer una nueva versión almacenando `a.c` (o todos los que se han tocado) en el repositorio y hacer esta versión disponible para el resto de los usuarios:

```
cvs commit a.c
```

Utilizando el editor definido en la variable `CVSEDITOR` (o `EDITOR` si esta no está inicializada), se podrá introducir un comentario que indique qué cambios se han hecho para que sirva de ayuda a otros usuarios o para recordar qué es lo que caracterizó a esta versión y luego poder hacer un histórico.

Si se decide eliminar los archivos (porque ya se terminó con el proyecto o porque no se trabajará más con él), una forma de hacerlo es a nivel de sistema operativo (`rm -r gccrep`), pero es mejor utilizar el propio cvs fuera del directorio de trabajo (nivel inmediato superior): `cvs release -d gccrep`. El comando detectará si hay algún archivo que no ha sido enviado al repositorio y, si lo hay y se borra, preguntará si se desea continuar o no para evitar que se pierdan todos los cambios. Para mirar las diferencias, por ejemplo, si se ha modificado `b.c` y no se recuerda qué cambios se hicieron, se puede utilizar `cvs diff b.c` dentro del directorio de trabajo. Este utilizará el comando del sistema operativo `diff` para comparar la versión `b.c` con la versión que se tiene en el repositorio (siempre hay que recordar hacer un `cvs commit b.c` si se desea que estas diferencias se transfieran al repositorio como una nueva versión).

Múltiples usuarios

Cuando más de una persona trabaja en un proyecto de software con diferentes revisiones, se trata de una situación sumamente complicada porque habrá ocasiones en las que más de un usuario quiera editar el mismo fichero simultáneamente. Una posible solución es bloquear el fichero o utilizar puntos de verificación reservados (*reserved checkouts*), lo cual solo permitirá a un usuario editar el mismo fichero simultáneamente. Para ello, se deberá ejecutar el comando `cvs admin -l command` (consultad `man` para las opciones).

CVS utiliza un modelo por defecto de puntos no reservados (*unreserved checkouts*), que permite a los usuarios editar simultáneamente un fichero de su directorio de trabajo. El primero de ellos que transfiera sus cambios al repositorio lo podrá hacer sin problemas, pero los restantes recibirán un mensaje de error cuando deseen realizar la misma tarea, por lo cual, deberán utilizar comandos de cvs para transferir en primer lugar los cambios al directorio de trabajo desde el repositorio y luego actualizar el repositorio con sus propios cambios. Consultad las referencias para ver un ejemplo de aplicación y otras formas de trabajo concurrente con comunicación entre usuarios [13]. Como interfaces gráficas pueden consultar tkcvs* [34] desarrollada en Tcl/Tk y que soporta Subversion o la también muy popular, Cervisia [35] (ambas en Debian).

***http://www.twobarleycorns.net/tkcvs.html**

1.6.2. Subversion

Como idea inicial, **Subversion** sirve para gestionar un conjunto de archivos (repositorio) y sus distintas versiones. Es interesante remarcar que no nos importa cómo se guardan, sino cómo se accede a estos ficheros y que es común utilizar una base de datos. El repositorio es como un directorio del cual se quiere recuperar un fichero de hace una semana o 10 meses a partir del estado de la base de datos, recuperar las últimas versiones y agregar una nueva. A diferencia de CVS, Subversion hace las revisiones globales del repositorio, lo cual significa que un cambio en el fichero no genera un salto de versión únicamente en ese fichero, sino en todo el repositorio, el cual suma uno a la revisión. En Debian deberemos hacer `apt-get install subversion subversion-tools`, si deseamos publicar los repositorios en Apache2 deberemos tener instalado este, además del módulo específico `apt-get install libapache2-svn`. Además de paquete Subversion, existen en los repositorios (de la mayoría de distribuciones) paquetes complementarios, como por ejemplo, subversion-tools (herramientas complementarias), svn-load (versión mejorada para la importación de repositorios), svn-workbench (*Workbench* para Subversion), svnmailer (herramienta que extiende las posibilidades de las notificaciones del *commit*).

Enlace de interés

Además del libro disponible en http://svnbook.red-bean.com/nightly/es/index.html, consultad la documentación en http://subversion.tigris.org/servlets/ProjectDocumentList.

Primer paso: crear nuestro repositorio, usuario (consideramos que el usuario es *svuser*), grupo (*svgroup*) como root hacer:

```
mkdir -p /usr/local/svn
```

```
adduser svuser
addgroup svgroup
chown -R root.svgroup /usr/local/svn
chmod 2775 /usr/local/svn
addgroup svuser svggroup Agrego el usuario svuser al grupo svgroup
```

Nos conectamos como *svuser* y verificamos que estamos en el grupo *svgroup* (con el comando `id`).

```
svnadmin create /usr/local/svn/pruebas
```

Este comando creará un serie de archivos y directorios para hacer el control y gestión de las versiones. Si no se tiene permiso en /usr/local/svn, se puede hacer en el directorio local:

```
mkdir -p $HOME/svndir
svnadmin create $HOME/svndir/pruebas
```

Considerando que el repositorio lo tenemos en /usr/local/svn/pruebas/ a continuación, creamos un directorio temporal en nuestro directorio:

```
mkdir -p $HOME/svntmp/pruebas
```

Nos pasamos al directorio `cd $HOME/svntmp/pruebas` y creamos un archivo, por ejemplo:

```
echo Primer Archivo Svn ` date ` >file1.txt
```

Lo trasladamos al repositorio haciendo dentro del directorio:

```
svn import file:///usr/local/svn/pruebas/ -m "Ver. Inicial"
```

Si lo hemos creado en `$HOME/svndir/pruebas`, deberíamos reemplazar por *file:///$HOME/svndir/pruebas*. El `import` copia el árbol de directorios y el `-m` permite indicarle el mensaje de versión. Si no ponemos `-m`, se abrirá un editor para hacerlo (se debe poner un mensaje para evitar problemas). El subdirectorio `$HOME/svntmp/pruebas` es una copia del trabajo en repositorio y es recomendable borrarla para no tener la tentación o cometer el error de trabajar con ella en lugar de hacerlo con el repositorio (`rm -rf $HOME/svntmp/pruebas`).

Una vez en el repositorio, se puede obtener la copia local donde podremos trabajar y luego subir las copias al repositorio. Para ello hacemos:

```
mkdir $HOME/svn-work; cd $HOME/svn-work
svn checkout file:///home/svuser/svndir/pruebas
```

donde veremos que tenemos el directorio pruebas. Se puede copiar con otro nombre agregando al final el nombre que queremos. Para añadirle un nuevo fichero:

```
cd $HOME/svn-work/pruebas
echo Segundo Archivo Svn ` date ` >file2.txt
svn add file2.txt
svn commit -m "Nuevo archivo"
```

Es importante remarcar que una vez en la copia local (`svn-work`), no se debe indicar el camino (*path*). `svn add` marca para añadir el fichero al repositorio y realmente se añade cuando hacemos un `svn commit`.

Nos dará algunos mensajes indicándonos que es la segunda versión. Si agregamos otra línea, la `file1.txt` con `echo ` date ` >> file1.txt`, después podemos subir los cambios con: `svn commit -m "Nueva línea"`. Es posible comparar el archivo local con el del repositorio. Para hacerlo, podemos agregar una tercera línea a `file1.txt` con `echo ` date ` >> file1.txt` sin subirlo y si queremos ver las diferencias, podemos hacer: `svn diff`. Este comando nos marcará cuáles son las diferencias entre el archivo local y los del repositorio. Si lo cargamos con `svn commit -m "Nueva línea2"` (que generará otra versión), después el `svn diff` no nos dará diferencias.

También se puede utilizar el comando `svn update` dentro del directorio para actualizar la copia local. Si hay dos o más usuarios trabajando al mismo tiempo y cada uno ha hecho una copia local del repositorio y la modifica (haciendo un `commit`), cuando el segundo usuario vaya a hacer el `commit`

de su copia con sus modificaciones, le dará un error de conflicto, ya que la copia en el repositorio es posterior a la copia original de este usuario (es decir, ha habido cambios entremedias), con lo cual si el segundo usuario hace el `commit`, perderíamos las modificaciones del primero. Para ello deberemos hacer un `svn update` que nos indicará el archivo en conflicto y en el archivo en conflicto nos indicará cuál es el archivo y las partes del mismo que están en conflicto. El usuario deberá decidir con qué versión se queda y después podrá hacer un `commit`. Un comando interesante es el `svn log file1.txt`, que nos mostrará todos los cambios realizados en el fichero y sus correspondientes versiones.

Un aspecto interesante es que Subversion puede funcionar conjuntamente con Apache2 (y también sobre SSL) para acceder desde otra máquina o simplemente mirar el repositorio. La configuración de Apache2 y SSL se indicó en la parte de servidores, pero también se explica en Debian Administration. Para configurarlos, es necesario activar los módulos de WebDAV (`a2enmod dav dav_fs`) e instalar la librearía como se especificó antes `apt-get install libapache2-svn` verificando que el módulo dav_svn esté habilitado también (lo hará la instalación de la librería). A continuación creamos un archivo en */etc/apache2/conf.d/svn.conf* con el siguiente contenido:

Enlace de interés

Consultad los clientes para acceder a Subversion en: http://svnbook.red-bean.com.

```
<location "/svn">
DAV svn
SVNParentPath /usr/local/svn
SVNListParentPath On
   AuthType Basic
AuthName "Subversion"
AuthUserFile /etc/apache2/htpasswd
Require valid-user
</location>
```

Donde debemos apuntar hacia donde se encuentra nuestro repositorio con validación de usuario (debe ser un usuario válido creado con la instrucción `htpasswd /etc/apache2/htpasswd user`), reiniciamos Apache2 y ya tendremos este habilitado y disponible para acceder a través de la dirección URL: http://sysdw.nteum.org/svn/. Otra forma de acceder es a través de un cliente en forma local o en forma remota que es una opción útil para repositorios pequeños y medianos. Debian incluye una serie de ellos (algunos compatibles con CVS) como por ejemplo **rapidsvn**, **subcommander**, **svnkit** (en java), **tkcvs**, **viewvc** (estos dos últimos soportan también repositorios CVS) y **websvn** (en PHP). Si se desea trabajar en remoto se deberá poner en marcha el servidor de svn con `svnserve -d` y cambiar el archivo *svnserve.conf* para permitir el acceso y el modo (p.ej., quitando el comentario a la línea `anon-access = read`). Este archivo está en el directorio */Dir_Repo/conf* que es donde hemos inicializado el repositorio (en nuestro caso se encuentra en */usr/local/svn/pruebas/conf/svnserve.conf*).

Si deseamos trabajar con un repositorio gestionado via URL lo más práctico es crear uno y gestionarlo desde la consola para insertar los archivos y via web

para ponerlos a disposición del público. En este caso deberemos hacer algunas modificaciones con los permisos para que podamos trabajar con URLs. Crearemos un segundo repositorio para este ejemplo pero haremos que el usuario **svuser** genere sus copias dentro del repositorio:

Como root, hacemos:

```
  mkdir /var/svn Nuestro nuevo repositorio
  svnadmin create /var/svn Creo el repositorio
  chown -R www-data:www-data Para que Apache pueda acceder al directorio
  ls -s /var/svn
-rw-r-r-   1 www-data www-data   229 Jul 4 20:27 README.txt
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 conf
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 dav
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:28 db
-rw-r-r-   1 www-data www-data   2 Jul 4 20:27 format
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 hooks
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 locks
```

Para la autentificación, se utiliza `htpasswd` (por ejemplo, con la instrucción `htpasswd -c -m /etc/apache2/htpasswd user` donde el parámetro *-c* solo se debe poner la primera vez cuando se debe crear el archivo. A continuación, creamos un archivo en */etc/apache2/conf.d/svn2.conf* por algo como:

```
<location /svn2>
   DAV svn
   SVNPath /var/svn
   AuthType Basic
   AuthName "Subversion Repository"
   AuthUserFile /etc/apache2/htpasswd
   Require valid-user
</location>
```

Reiniciamos Apache y ya estamos listos para importar algunos archivos, por ejemplo desde la consola como svuser:

```
  svn import file1.txt http://sysdw.nteum.org/svn/file1.txt -m "Import
Inicial"
```

Nos pedirá la autentificación y nos dirá que el fichero `file1.txt` se ha añadido al repositorio. Desde la URL http://sysdw.nteum.org/svn2/ veremos el file1.txt.

1.6.3. Git

Git es un programa de control de versiones diseñado por Linus Torvalds basado en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones con un elevado número archivos de código fuente que, si bien se diseñó como un motor de bajo nivel sobre el cual se pudieran escribir aplicaciones de usuario (*front ends*), se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena a partir de su adopción como herramienta de revisión de versiones para el grupo de programación del núcleo Linux. Entre las características más relevantes encontramos:

1) Soporta el desarrollo no lineal y permite la gestión eficiente de ramificaciones y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no-lineal.

2) Gestión distribuida: permite a cada programador una copia local del historial del desarrollo entero y que los cambios se propaguen entre los repositorios locales.

3) Los repositorios pueden publicarse por HTTP, FTP, rsync o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado SSH, y permite emular servidores CVS para interactuar con clientes CVS preexistentes.

4) Los repositorios Subversion y svk se pueden usar directamente con git-svn.

Instalación de un repositorio de Git, Gitolite y Gitweb

Instalaremos un servidor de control de versiones Git sobre Debian Wheezy para que pueda gestionar los archivos de código fuente de un equipo de desarrollo. Para ello, utilizaremos Gitolite (como método de control de acceso de nuestros programadores a los repositorios de software), que es una capa de autentificación a Git basado en ssh y que permite especificar permisos no solo por repositorio, sino también por rama o etiquetas dentro de cada repositorio. Finalmente, instalaremos Gitweb para el acceso a los repositorios y adaptaremos una interfaz basada en http://kogakure.github.io/gitweb-theme/. [17, 18, 19]

Sobre el servidor instalamos los paquetes:

```
apt-get install git-core git-doc gitolite git-daemon-run gitweb
highlight
```

Creamos un usuario para el servicio git: `adduser git`

Adoptamos su identidad y configuramos:

```
su - git
git config --global user.name "git"
git config --global user.email git@sysdw.nteum.org
```

Se puede verificar con: `git config -l`

Se deben generar las llaves pública de los usuarios y enviar al servidor, por lo cual podemos hacer como usuario adminp:

```
ssh-keygen
scp .ssh/id_rsa.pub git@sysdw.nteum.org:/tmp/adminp.pub
```

Si el usuario adminp está en el mismo servidor utilizar `cp` en lugar de `scp`, pero se supone que es otra máquina.

En el servidor activamos la llave de adminp (y la de todos los usuarios), como usuario git hacemos `gl-setup /tmp/adminp.pub` Esto abrirá un editor (de archivo */home/git/.gitolite.rc*) con indicaciones en el cual deberemos cambiar la línea $REPO_UMASK = 0027; (en lugar de 77 cambiar por 27).

A continuación cambiar los permsisos:

```
chmod g+r /home/git/projects.list
chmod -R g+rx /home/git/repositories
```

Editar */etc/group* y agregar el usuario www-data al grupo git modificando la línea a: git:x:1001:www-data

En la máquina local y como usuario adminp (también se puede hacer en la misma máquina del servidor en la cuenta del usuario adminp) Cargaremos la configuración de gitolite y agregaremos dos repositorios:

```
git clone git@sysdw.nteum.org:gitolite-admin.git
cd gitolite-admin
```

Editamos el archivo *conf/gitolite.conf* con los repositorios:

```
repo    gitolite-admin
RW+     =   adminp

repo    testing
    RW+     =   @all

repo wiki
    RW+ = @all
    R = daemon
    R = gitweb

repo data
    RW+ = adminp remix
    R = daemon
    R = gitweb
```

Luego agregamos al repositorio: `git add *`

Y hacemos el commit: `git commit -m "Repositorios wiki y data con permiso de escritura para admin y remix"`

Lo subimos: `git push origin master`

Para la configuración de gitweb editamos */etc/gitweb.conf* modificando las líneas indicadas:

```
$projectroot = "/home/git/repositories";
$projects_list = "/home/git/projects.list";
```

Modificamos */etc/sv/git-daemon/run* (antes hacer una copia como run.org por ejemplo)

```
#!/bin/sh
  exec 2>&1
  echo 'git-daemon starting.'
  exec chpst -ugitdaemon:git \
       "$(git -exec-path)" /git-daemon -verbose -reuseaddr \
       -base-path=/home/git/repositories /home/git
```

Luego reiniciamos el daemon: `sv restart git-daemon`

También reiniciamos Apache2 y en la URL http://sysdw.nteum.org/gitweb/ tendremos los proyectos.

Para cambiar el aspecto de la página web con http://kogakure.github.io/gitweb-theme/. Podemos descargar el archivo tgz de esta página, lo descomprimimos y desde dentro del directorio copiamos el contenido

```
cp * /usr/share/gitweb/static/
```

Si recargamos el directorio borrando la caché (Crtl+F5) en el navegador deberíamos ver el nuevo aspecto. Un aspecto interesante es ver ahora cómo desde el usuario admin insertamos archivos en uno de los repositorios y los publicamos:

Creo un directorio *hello* con dos archivos uno archivo llamado *hello.c* y otro *library.h*

```
/* Hello.c */
#include <stdio.h>
#include "library.h"
    int main (void) {printf ("Hola UOC!\n");}
/* library.h */
#ifndef DEFINITIONS_H
#define DEFINITIONS_H 1
#endif /* DEFINITIONS_H */
```

Dentro del directorio ejecuto:

```
git init
git add .
git commit -m "initial commit"
git remote add origin git@sysdw.nteum.org:wiki.git
git push origin master
```

Actualizando la página web veremos que ya tenemos los archivos en el repositorio Si quisiéramos clonar el repositorio solo deberíamos hacer:

```
git clone git@sysdw.nteum.org:wiki.git
```

Y donde estamos creará un directorio wiki con los archivos hello.c library.h (además de otro .git que es donde se encuentra toda la información de Git).

1.6.4. Mercurial

Mercurial es un sistema de control de versiones (escrito en su mayor parte en python y algunas partes en C -diff-) diseñado para facilitar la gestión de archivos/directorios a los desarrolladores de software. El uso básico de Mercurial es en línea de comandos y existe un comando que es el que gestiona todo el paquete a través de opciones llamado `hg` (en referencia al símbolo químico del mercurio). Las premisas de diseño de mercurial son el rendimiento rendimiento y la escalabilidad a través de un desarrollo distribuido y sin necesidad de un servidor. Además presenta una gestión robusta de archivos tanto de texto como binarios y posee capacidades avanzadas de ramificación e integración y una interfaz web integradas.[21]

Su instalación es muy simple con `apt-get mercurial` y es interesante instalar otro paquete llamado Tortoisehg [23] (`apt-get install tortoisehg`), que es una extensión gráfica a Mercurial y actúa como *Workbench*, soportando múltiples repositorios, y permite resolver conflictos en una forma simple y gráfica. Después de la instalación podremos ejecutar el comando `hg version` y `hg debuginstall` donde este último nos dirá que debemos identificar al usuario e incluir una dirección de mail. Para ello, creamos el archivo more $HOME/.hgrc con el siguiente contenido (la segunda parte es para Tortoise):

```
[ui]
username = AdminP <adminp@sysdw.nteum.org>
[extensions]
graphlog =
```

Con ello ya estamos en condiciones de comenzar a trabajar en nuestro repositorio. Para usuario recién iniciados es muy recomendable seguir la guía

de [22] que muestra el procedimiento para gestionar un repositorio entre dos usuarios. Una secuencia de comandos podría ser:

Creo un repositorio: `hg init hello` veremos un directorio con un subdirectorio *.hg*
Dentro de *hello* creo un archivo: `echo "Hello World" > hello.txt`
Miro el estado pero veremos que no forma parte de repositorio: `hg status`
Lo agrego y si ejecutamos la orden anterior veremos que ya está dentro: `hg add hello.txt`
Para publicarlo en el repositorio: `hg commit -m "Mi primer hg-hello"`
La historia la podemos ver con: `hg log`
Modifico y miro las diferencias: `echo "Bye..." >hello.txt; hg diff`
Puedo revertir los cambios: `hg revert hello.txt`
Añado información: `echo "by Admin." >> hello.txt`
Publico los cambios: `hg commit -m "Added author."`
Pregunto a HG donde están estos: `hg annotate hello.txt`
Las anotaciones son de mucha ayuda para fijar los errores ya que nos permite ver que y quién cambió:
`hg log -r 1` y en forma espacial `hg glog` o mejor en forma gráfica utilizando Tortoise `hg glog`
Que nos mostrará que el cambio es hijo de la misma rama-autor.
Miro los parientes y vuelvo atrás (a una versión anterior): `hg parents` luego `hg update 0` si hacemos `hg parents` y miramos el archivo veremos que tenemos la versión original.
Pero no lo hemos perdido, si queremos mirar lo anterior: `hg cat -r 1 hello.txt`
Y recuperarlo: `hg update`
Si queremos trabajar entre dos usuarios que comparten sistema de archivos (podría ser por SSH o HTTP también), el usuario interesado podrá hacer una copia del repositorio: `hg clone ../adminp/hello`
Incluso se podría hacer por mail/USB utilizando `hg bundle` para generar un binario y llevárselo a otra máquina. Este nuevo usuario será independiente de lo que haga el otro usuario (adminp)
Este usuario (remix) podrá hacer:
`cd hello; echo "My hola" > hello2.txt; echo "by Remix." >> hello2.txt`
Lo agrega a su repositorio y publica: `hg add` y `hg commit -m "My hello."`
Luego modifico lo de Adminp: `echo "not by Remix" >> hello.txt`
Publico: `hg commit -m "Not my file."`
Y puedo preguntar si han habido cambios con respecto a adminp clone: `hg incoming`
Y preguntar que cambios no están en adminp: `hg outgoing`
Remix no tiene acceso al directorio de Adminp por lo cual no puede introducir los cambios pero Adminp si puede leer y agregarlos. Debe poner en su archivo *.hg/hgrc*

```
[paths]
default = /home/remix/hola
```

Y adminp puede observar los cambios con *hg incoming*
Y agregarlos con: *hg pull* es interesante verlos con Tortoise que lo mostrará gráficamente.
Es interesante que de una forma simple este repositorio lo podemos publicar via web haciendo en nuestro repositorio: `hg serve`
A partir de ello se podrá acceder en la URL http://sysdw.nteum.org:8000 (o en localhost)
Esta solución es simple (pero eficiente) y temporal. Repositorios más estables deberán ser publicados utilizando Apache con el script *hgweb.cgi/hgwebdir.cgi* indicado en el manual de instalación. [25][26]
También es interesante el comando `hg-ssh` cuando los repositorios se ofrecen a través de SSH.

Es interesante completar la información con un tutorial [24] y ver las extensiones que admite HG [28] y no menos interesantes son los paquetes (incluidos en Debian) **hgview** (*interactive history viewer*), **mercurial-git** (git *plugin* para mercurial), **mercurial-server** (*shared Mercurial repository service*), o **eclipse-mercurialeclipse** (integración con Eclipse).

1.7. Mantis Bug Tracker

Mantis Bug Tracker es un sistema (GPL) de seguimiento de errores a través de la web. El uso más común de MantisBT es hacer un seguimiento de errores o incidencias, pero también sirve como herramienta de seguimiento de gestión y administración de proyectos. Además del paquete de Mantis, es necesario tener instalado MySQL, Apache Web Server y el módulo PHP para Apache. La instalación de Mantis es muy simple `apt-get install mantis` y durante el proceso nos indicará que la contraseña para el usuario administrator (y el aviso de que es recomendable cambiarla) y pedirá alguna información más en función de los paquetes que tengamos instalados (Apache, MySQL, MariaDB, ...). A partir de ese momento, nos podremos conectar a `http://localhost/mantis/` introduciendo el usuario administrator y la contraseña root (se recomienda que la primera acción sea cambiar la contraseña en la sección `MyAccount`). Accediendo desde la interfaz web (*ítem manage*), se podrán crear nuevos usuarios con permisos de usuario por roles (*administrator, viewer, reporter, updater*, etc.), definir los proyectos y las categorías dentro de cada proyecto, gestionar los anuncios, informes, registros, tener una visión general de los proyectos y su estado, gestionar los documentos asociados, etc.. Se recomienda por crear nuevos usuarios, nuevas categorías y luego proyectos asignadas a estas categorías. A partir de ese momento se puede insertar incidencias, asignándolas y gestionándolas a través de la interfaz. Si tenemos instalada MariaDB nos podrá dar un error similar a *mysql_connect(): Headers and client library minor version mismatch* y deberemos actualizar los *headers y drivers* de PHP `apt-get install php5-mysqlnd`.

Hay un amplio conjunto de opciones que se pueden modificar en el archivo */usr/share/mantis/www/config_inc.php* [30], así, para cambiar la lengua del entorno, se edita y añade `$g_language_choices_arr = array( 'english', 'spanish'); $g_default_language = 'spanish';` y para que la página principal del Mantis Bug Tracker sea automáticamente el propio Bug Tracker y se permita un acceso anónimo a los proyectos públicos:

1) Crear una cuenta de usuario, por ejemplo anonymous o guest, dejando en blanco el `Real Name`, `Email=anonymous@localhost` (o dejar en blanco si se pone `$g_allow_blank_email = ON`), `Access Level = viewer` o `reporter` (dependiendo los que se quiera) `Enabled = true` y `Protected = true`.

2) Después se tiene que modificar en el archivo anterior (`config_inc.php`) poniendo las siguientes variables:

```
$g_allow_anonymous_login = ON;
$g_anonymous_account = 'anonymous'
```

y, opcionalmente, para dejar en blanco las direcciones de correo:

```
$g_allow_blank_email = ON
```

***http://www.mantisbt.org /wiki/doku.php /mantisbt:mantis_recipies Para más configuraciones o integraciones con otros paquetes, consultad el manual [29] o acceded a la página de la *wiki* del proyecto* [30].**

Actividades

1. Definid en PostgreSQL una base de datos que tenga al menos 3 tablas con 5 columnas (de las cuales 3 deben ser numéricas) en cada tabla. Generad un listado ordenado por cada tabla/columna. Generad un listado ordenado por el mayor valor de la columna X de todas las tablas. Cambiad el valor numérico de la columna Y con el valor numérico de la columna Z + el valor de la columna W/2.

2. Realizad el mismo ejercicio anterior, pero con MySQL o con MariaDB.

3. Utilizando las tablas de World http://pgfoundry.org/projects/dbsamples/ obtened la población promedio de las poblaciones de las ciudades entre 10.000 y 30.000 habitantes, y un porcentaje de cuál es la lengua más hablada y la menos hablada con relación a los habitantes.

4. Configurad el CVS para hacer tres revisiones de un directorio donde hay 4 archivos .c y un Makefile. Haced una ramificación (*branch*) de un archivo y luego mezcladlo con el principal.

5. Simulad la utilización de un archivo concurrente con dos terminales de Linux e indicar la secuencia de pasos para hacer que dos modificaciones alternas de cada usuario queden reflejadas sobre el repositorio CVS.

6. Realizad el mismo ejercicio anterior, pero uno de los usuarios debe conectarse al repositorio desde otra máquina.

7. Realizad nuevamente los tres ejercicios anteriores, pero en Subversion.

8. Cread un repositorio sobre Git y hacedlo visible a través del web, de tal modo que dos usuarios de la misma máquina puedan modificar los archivos y actualizar el repositorio.

9. Repetid el paso anterior con Mercurial.

10. Instalad Mantis, generad 3 usuarios, 2 categorías, 2 proyectos y 3 incidencias para cada usuarios, gestionándolas a través de la interfaz. Generad un usuario anónimo que pueda acceder a un proyecto público como reporter, pero no a uno privado.

Bibliografía

[1] *Big Data - Infografía*.
<http://i2.wp.com/unadocenade.com/wp-content/uploads/2013/05/Infograf%C3%ADa-Big-Data.jpg>

[2] **M. Gibert, O. Pérez**. *Bases de datos en PostgreSQL*.
<http://ocw.uoc.edu/computer-science-technology-and-multimedia/bases-de-datos/bases-de-datos/P06_M2109_02152.pdf>

[3] *PostgresSQL*.
<http://www.postgresql.org>

[4] **PostgreSQL**.*The SQL Language*.
<http://www.postgresql.org/docs/9.1/interactive/tutorial-sql.html>

[5] *PgAdmin*.
<http://www.pgadmin.org>

[6] *PgpPGAdmin*.
<http://phppgadmin.sourceforge.net/>

[7] *Documentación de MySQL*.
<http://dev.mysql.com/usingmysql/get_started.html>

[8] *Mysql Administrator*.
<http://www.mysql.com/products/workbench/>

[9] *MariaDB*.
<https://mariadb.org/>

[10] *Setting up MariaDB Repositories*.
<https://downloads.mariadb.org/mariadb/repositories/>

[11] *SQLite*.
<http://www.sqlite.org/>

[12] *SQLite Database Browser*.
<https://github.com/sqlitebrowser/sqlitebrowser>

[13] **Cederqvist**. *Version Management with CVS*.
<http://www.cvshome.org>

[14] *Libro sobre Subversion*.
<http://svnbook.red-bean.com/nightly/es/index.html>
(versión en español)

[15] *Subversion Documentation*.
<http://subversion.tigris.org/servlets/ProjectDocumentList>)

bibitemsub *Subversion*.
<http://subversion.apache.org/>

[16] *Multiples repositorios con Subversion*.
<http://www.debian-administration.org/articles/208>

[17] *Git. Fast Control Version system*.
<http://git-scm.com/>

[18] *Instalar Git sobre Debian*.
<http://linux.koolsolutions.com/2009/08/07/learn-git-series-part-1-installing-git-on-debian/>

[19] **L. Hernández**.*Servidor Git + Gitolite + Gitweb sobre Debian*.
<http://leninmhs.wordpress.com/2014/01/19/git-gitolite-gitweb/>

[20] **D. Mühlbachler**.*How-To: Install a private Debian Git server using gitolite and GitLabHQ*.
<http://blog.muehlbachler.org/2012/01/how-to-install-a-private-debian-git-server-using-gitolite-and-gitlabhq/>

[21] *Mercurial*.
<http://mercurial.selenic.com/>

[22] *Basic Mercurial*.
<http://mercurial.aragost.com/kick-start/en/basic/>

[23] *TortoiseHg Docs*.
<http://tortoisehg.bitbucket.org/docs.html>

[24] *HgInit: a Mercurial tutorial*.
<http://hginit.com/>

[25] *Publishing Repositories with hgwebdir.cgi*.
<http://mercurial.selenic.com/wiki/HgWebDirStepByStep>

[26] *Publishing Mercurial Repositories*.
<http://mercurial.selenic.com/wiki/PublishingRepositories>

[27] *Remote Repositories*.
<http://mercurial.aragost.com/kick-start/en/remote/>

[28] *Using Mercurial Extensions*.
<http://mercurial.selenic.com/wiki/UsingExtensions>

[29] *Documentación del proyecto Mantis*.
<http://www.mantisbt.org/documentation.php>

[30] *Mantis Bug Tracker Wiki*.
<http://www.mantisbt.org/wiki/doku.php>

[31] *Módulos de Webmin*.
<http://doxfer.webmin.com/Webmin>

[32] **Ibiblio.org** (2010). *Linux Documentation Center*.
<http://www.ibiblio.org/pub/Linux/docs/HOWTO/>

[33] **Mourani, G.** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.

[34] *TkCVS: Tcl/Tk-based graphical interface to the CVS*.
<http://www.twobarleycorns.net/tkcvs.html>

[35] *Cervisia - CVS Frontend*.
<http://www.kde.org/applications/development/cervisia/>

Administración de seguridad

Josep Jorba Esteve

PID_00212472

Universitat Oberta de Catalunya

Índice

Introducción 5

Objetivos 7

1. Tipos y métodos de los ataques 9
1.1. Técnicas utilizadas en los ataques 13
1.2. Contramedidas 20

2. Seguridad del sistema 26

3. Seguridad local 28
3.1. *Bootloaders* 28
3.2. Contraseñas y *shadows* 30
3.3. Bits *sticky* y *setuid* 31
3.4. Habilitación de *hosts* 32
3.5. Módulos PAM 33
3.6. Alteraciones del sistema 35
3.7. Recursos limitados, *cgroups* y *chroot* 36
3.8. Protección de ejecutables mediante Hardening-Wrapper 40

4. SELinux 43
4.1. Arquitectura 46
4.2. Crítica 49
4.3. Algunas alternativas 50

5. Seguridad en red 52
5.1. Cliente de servicios 52
5.2. Servidor: inetd y xinetd 52

6. Herramientas de seguridad: Detección de vulnerabilidades e intrusiones 56
6.1. OpenVAS 60
6.2. Denyhosts 63
6.3. Fail2ban 66

7. Protección mediante filtrado (*TCP wrappers* y cortafuegos) 69
7.1. Cortafuegos 70
7.2. Netfilter: iptables 71
7.3. Paquetes para gestión de cortafuegos iptables en las distribuciones 76

7.4. Netfilter: nftables 78
7.5. Consideraciones 81

8. Análisis de registros 83

9. Taller: análisis de la seguridad mediante herramientas 85

Resumen 93

Actividades 94

Bibliografía 95

Introducción

El salto tecnológico realizado desde los sistemas de escritorio aislados hasta los sistemas actuales integrados en redes locales e Internet, ha traído una nueva dificultad a las tareas habituales del administrador: el control de la seguridad de los sistemas.

La seguridad es un campo complejo, en el cual se mezclan técnicas de análisis con otras de detección o de prevención de los posibles ataques. Las técnicas a usar son tanto computacionales como relacionadas con otros campos, como el análisis de factores psicológicos, por lo que respecta al comportamiento de los usuarios del sistema o a las posibles intenciones de los atacantes.

Los ataques pueden provenir de muchas fuentes y afectar desde a una aplicación o servicio hasta a algún usuario, o a todos, o al sistema informático entero.

Los posibles ataques pueden cambiar el comportamiento de los sistemas, incluso "hacerlos caer" (es decir, inutilizarlos), o dar una falsa impresión de seguridad, que puede ser difícilmente detectable. Podemos encontrarnos con ataques de autentificación (obtener acceso por parte de programas o usuarios previamente no habilitados), escuchas (redirigir o pinchar los canales de comunicación y los datos que circulan) o sustitución (de programas, máquinas, comunicaciones o usuarios por otros, sin que se noten los cambios).

Seguridad absoluta

La seguridad absoluta no existe. Una falsa impresión de seguridad puede ser tan perjudicial como no tenerla. El área de la seguridad es muy dinámica y hay que mantener actualizados constantemente los conocimientos.

Una idea clara que hay que tener en mente es que es imposible poder conseguir una seguridad al 100 %.

Las técnicas de seguridad son un arma de doble filo, que fácilmente pueden darnos una falsa impresión de control del problema. La seguridad actual es un problema amplio, complejo y, lo que es más importante, dinámico. Nunca podemos esperar o asegurar que la seguridad esté garantizada en un determinado sistema, sino que con bastante probabilidad será una de las áreas a la cual el administrador tendrá que dedicar más tiempo y mantener actualizados sus conocimientos sobre el tema.

En este módulo examinaremos diferentes problemáticas con los que podemos encontrarnos, cómo podemos verificar y prevenir partes de la seguridad local y en entornos de red. Asimismo, examinaremos técnicas de detección de in-

trusiones y algunas herramientas básicas que nos pueden ayudar en el control de la seguridad.

También es preciso mencionar que en este módulo solo podemos hacer una mera introducción a algunos de los aspectos que intervienen en la seguridad de hoy en día. Para cualquier aprendizaje real con más detalle, se recomienda consultar la bibliografía disponible, así como los manuales asociados a los productos o herramientas comentados.

Objetivos

En este módulo mostraremos los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Conocer los principales tipos y métodos de ataques a la seguridad de los sistemas, así como algunas contramedidas básicas y herramientas útiles para su tratamiento.
2. Saber hacer un seguimiento de la seguridad local y en red en los sistemas GNU/Linux.
3. Conocer la metodología de uso de herramientas de detección de intrusiones.
4. Saber elaborar medidas de prevención mediante filtrado de servicios o paquetes, *wrappers* y cortafuegos (*firewalls*).
5. Conocer las herramientas de seguridad disponibles, para control de accesos y permisos, a nivel de MAC (*Mandatory Access Control*).

1. Tipos y métodos de los ataques

La seguridad computacional, en administración, puede ser entendida como el proceso que ha de permitir al administrador del sistema prevenir y detectar usos no autorizados del mismo. Las medidas de prevención ayudan a parar los intentos de usuarios no autorizados (los conocidos como **intrusos**) para acceder a cualquier parte del sistema. La detección ayuda a descubrir cuándo se produjeron estos intentos o, en el caso de llevarse a cabo, establecer las barreras para que no se repitan y poder recuperar el sistema si ha sido quebrantado o comprometido.

Los intrusos (también conocidos coloquialmente como *hackers*, *crackers*, "atacantes" o "piratas") normalmente desean obtener control sobre el sistema, ya sea para causar funcionamientos erróneos, corromper el sistema o sus datos, ganar recursos en la máquina o, simplemente, para lanzar ataques contra otros sistemas y así proteger su verdadera identidad y ocultar el origen real de los ataques. También existe la posibilidad de que el ataque se produzca para examinar o robar la información del sistema, el puro espionaje de las acciones del sistema o para causar daños físicos en la máquina, ya sea formatear el disco, cambiar datos, borrar o modificar software crítico, etc.

Con respecto a los intrusos, hay que establecer algunas diferencias, que no suelen estar muy claras en los términos coloquiales. Normalmente nos referimos a **hacker** [Him01] como aquella persona con grandes conocimientos en informática, más o menos apasionada por los temas de programación y seguridad informática y que, normalmente sin finalidad malévola, utiliza sus conocimientos para protegerse a sí mismo o a terceros, introducirse en redes para demostrar sus fallos de seguridad y, en algunos casos, para reconocimiento público de sus habilidades. Un ejemplo sería la propia comunidad GNU/Linux, que debe mucho a sus *hackers*, ya que hay que entender el término *hacker* como experto en unos temas computacionales, más que como intruso que afecta a la seguridad.

Por otro lado encontraríamos a los **crackers**. Aquí es cuando se utiliza el término, de manera más o menos despectiva, hacia aquellos que utilizan sus habilidades para corromper (o destrozar) sistemas, ya sea sólo por fama propia, por motivos económicos, por ganas de causar daño o simplemente por molestar, por motivos de espionaje tecnológico, actos de ciberterrorismo, etc.

Asimismo, se habla de *hacking* o *cracking*, respectivamente, cuando nos referimos a técnicas de estudio, detección y protección de la seguridad, o por el

contrario, a técnicas destinadas a causar daño rompiendo la seguridad de los sistemas.

Otros términos, más comunes actualmente para diferenciar a *hackers* (cuando el término se usa de manera genérica) en la comunidad de seguridad (debido al mal uso de los anteriores), son *Black hat* para referirse a un *hacker* que viola la seguridad informática por razones más allá de la malicia o para el beneficio personal. Son la personificación de lo que se entiende clásicamente por criminal informático. Los *hackers* de sombrero negro entran a redes seguras para destruir los datos o hacerlas inutilizables para aquellos que tengan acceso autorizado. Normalmente el proceso que siguen consta de la elección de los objetivos, la recopilación de información e investigación y, por último, la realización del ataque. Por otra parte, se conocen como *White hat* a aquellos *hackers* que rompen la seguridad por razones no maliciosas, quizá para poner a prueba sus propios sistemas, desarrollando software de seguridad o analizando la seguridad de sistemas u organizaciones bajo un acuerdo contractual. También se les suele denominar *hackers* éticos. Otros más curiosos son los denominados *Gray hat*, que actúan como los sombreros negros atacando sistemas, pero se ofrecen para arreglarlos a un módico precio. O los *Script kiddies*, que son inexpertos y aprovechan herramientas automatizadas empaquetadas realizadas por otros, generalmente sin ninguna (u poca) comprensión de los conceptos subyacentes y muchas veces sin ser apenas conscientes de los males que pueden provocar.

Desafortunadamente, obtener acceso a un sistema (ya sea desprotegido o parcialmente seguro) es bastante más fácil de lo que parece en primera instancia. Los intrusos descubren permanentemente nuevas **vulnerabilidades** (llamadas a veces "agujeros de seguridad"), que les permite introducirse en las diferentes capas de software (ya sean aplicaciones, servicios o partes del operativo). Así, definiremos una vulnerabilidad como el fallo de un programa (o capa de software) que permite, mediante su explotación, violar la seguridad de un sistema informático. Por otra parte, un *exploit* es un software, conjunto de datos o secuencia de comandos que, tomando ventaja de vulnerabilidades, permite causar comportamientos inesperados o no anticipados, a software, hardware o dispositivos electrónicos.

La complejidad cada vez mayor del software (y del hardware), hace que aumente la dificultad para testear de manera razonable la seguridad de los sistemas informáticos. El uso habitual de los sistemas GNU/Linux en red, ya sea en la propia Internet o en redes propias con tecnología TCP/IP como las *intranets*, nos lleva a exponer nuestros sistemas, como víctimas, a ataques de seguridad [Bur02, Fen02, Line].

Lo primero que hay que hacer es romper el mito de la seguridad informática: simplemente no existe. Lo que sí que podemos conseguir es un cierto grado de seguridad que nos haga sentirnos seguros dentro de ciertos parámetros.

Pero como tal, esto solo es una percepción de seguridad, y como todas las percepciones, puede ser falsa y de ello podemos darnos cuenta en el último momento, cuando ya tengamos nuestros sistemas afectados y comprometidos. La conclusión lógica es que la seguridad informática exige un esfuerzo importante de constancia, realismo y aprendizaje prácticamente diario.

Tenemos que ser capaces de establecer en nuestros sistemas unas políticas de seguridad que nos permitan prevenir, identificar y reaccionar ante los posibles ataques. Y tener presente que la sensación que podamos tener de seguridad no es más que eso, una sensación. Por lo tanto, no hay que descuidar ninguna de las políticas implementadas y hay que mantenerlas al día, así como nuestros conocimientos del tema.

Amenazas

Las amenazas afectan a la confidencialidad, integridad o accesibilidad de nuestros sistemas.

Los posibles ataques son una amenaza constante a nuestros sistemas y pueden comprometer su funcionamiento, así como los datos que gestionamos. Ante todo esto, siempre tenemos que definir una cierta política de requisitos de seguridad sobre nuestros sistemas y datos. Las amenazas que podemos sufrir podrían afectar a los aspectos siguientes:

- **Confidencialidad**: la información debe ser accesible sólo a aquellos que estén autorizados; estamos respondiendo a la pregunta: ¿quién podrá acceder a la información?

- **Integridad**: la información solo podrá ser modificada por aquellos que estén autorizados: ¿qué se podrá hacer con ella?

- **Accesibilidad**: la información tiene que estar disponible para quienes la necesiten y cuando la necesiten, si están autorizados: ¿de qué manera, y cuándo se podrá acceder a ella?

Pasemos a mencionar una cierta clasificación (no exhaustiva) de los tipos de ataques habituales que podemos padecer:

- **Autentificación**: ataques en los que se falsifica la identidad del participante, de manera que obtiene acceso a programas o servicios a los que en un principio no tenía acceso.

- **Intercepción** (o escucha): mecanismo por el cual se interceptan datos, por parte de terceros, cuando estos no estaban dirigidos a ellos.

- **Falsificación** (o reemplazo): sustitución de algunos de los participantes, ya sea máquinas, software o datos, por otros falsos.

- **Robo de recursos**: uso de nuestros recursos sin autorización.

- **Vandalismo**: después de todo, suele ser bastante común la existencia de mecanismos que permiten interferir en el funcionamiento adecuado del

sistema o de los servicios y causar molestias parciales, la parada o cancelación de recursos.

Los métodos utilizados y las técnicas precisas pueden variar mucho (es más, cada día se crean novedades) y nos obligan, como administradores, a estar en contacto permanente con el campo de la seguridad para conocer a qué nos podemos enfrentar diariamente.

Respecto a dónde se produce el ataque, hemos de tener claro qué puede hacerse o cuál será el objetivo de los métodos:

Finalidad de los ataques

Los ataques pueden tener finalidades destructivas, inhabilitadoras o de espionaje, de nuestros componentes, ya sea hardware, software o sistemas de comunicación.

- **Hardware**. A este respecto, la amenaza está directamente sobre la accesibilidad, ¿qué podrá hacer alguien que tenga acceso al hardware? En este caso normalmente necesitaremos medidas "físicas", como controles de seguridad para acceder a los locales donde estén situadas las máquinas, para evitar problemas de robo o rotura del equipo con el fin de eliminar su servicio. También puede comprometerse la confidencialidad y la integridad si el acceso físico a las máquinas permite utilizar algunos de sus dispositivos como las unidades de disco (extraibles o no), el arranque de las máquinas o el acceso a cuentas de usuario que podrían estar abiertas.

- **Software**. Si la accesibilidad se ve comprometida en un ataque, puede haber borrado o inutilización de programas, denegando el acceso. En caso de confidencialidad, puede provocar copias no autorizadas de software. En integridad podría alterarse el funcionamiento por defecto del programa, para que falle en algunas situaciones o bien para que realice tareas que puedan ser interesantes de cara al atacante, o podría simplemente comprometer la integridad de los datos de los programas: hacerlos públicos, alterarlos o simplemente robarlos.

- **Datos**, ya sean estructurados, como en los servicios de base de datos, gestión de versiones (como cvs) o simples archivos. Mediante ataques que amenacen la accesibilidad, estos datos pueden ser destruidos o eliminados, denegando así el acceso a los mismos. En el caso de la confidencialidad, estaríamos permitiendo lecturas no autorizadas y la integridad se vería afectada cuando se produjeran modificaciones o creación de nuevos datos.

- **Canal de comunicación** (en la red, por ejemplo). Para los métodos que afectan a la accesibilidad, nos puede provocar destrucción o eliminación de mensajes e impedir el acceso a la red. En confidencialidad, se puede dar la lectura y observación del tráfico de mensajes, desde o hacia la máquina. Y respecto a la integridad, podemos vernos afectados por cualquier modificación, retardo, reordenación, duplicación o falsificación de los mensajes entrantes o salientes.

1.1. Técnicas utilizadas en los ataques

Los métodos utilizados son múltiples y pueden depender de un elemento (hardware o software) o de la versión del mismo. Por lo tanto, hay que mantener actualizado el software para las correcciones de seguridad que vayan apareciendo y seguir las indicaciones del fabricante o distribuidor para proteger el elemento, además de las medidas de seguridad activa que podamos realizar.

Evolución de las técnicas de ataque

Las técnicas de los ataques son muy variadas y evolucionan constantemente en lo que a los detalles tecnológicos usados se refiere.

A pesar de ello, normalmente siempre hay técnicas o métodos "de moda", del momento actual. Algunas breves indicaciones de estas técnicas de ataque (actuales) son:

- **Explotación de agujeros (*bug exploits*)**: se trata de la explotación de agujeros o errores [Cerb, Ins, San] de un hardware, software, servicio, protocolo o del propio sistema operativo (por ejemplo, en el núcleo o *kernel*) y, normalmente de alguna de las versiones de estos en concreto. En general, cualquier elemento informático es más o menos propenso a errores en su concepción o simplemente a cosas que no se han tenido en cuenta o previsto. Periódicamente, se descubren agujeros (a veces se denominan *holes*, *exploits*, cuando existe software que puede aprovechar el agujero particular, o simplemente *bugs*), que pueden ser aprovechados por un atacante para romper la seguridad de los sistemas. Suelen utilizarse técnicas de ataque genéricas, como las que se explican a continuación, o bien técnicas particulares para el elemento afectado. Cada elemento tendrá un responsable, ya sea fabricante, desarrollador, distribuidor o la comunidad GNU/Linux, de producir nuevas versiones o parches para tratar estos problemas. Nosotros, como administradores, tenemos la responsabilidad de estar informados y de mantener una política de actualización responsable para evitar los riesgos potenciales de estos ataques. En caso de que no haya soluciones disponibles, también podemos estudiar la posibilidad de utilizar alternativas al elemento, o bien inhabilitarlo hasta que tengamos soluciones.

- **Virus**: se trata de programas normalmente anexos a otros y que utilizan mecanismos de autocopia y transmisión. Son habituales los virus adjuntos a programas ejecutables, a mensajes de correo electrónico, o incorporados en documentos o programas que permiten algún lenguaje de macros (no verificado). Son quizás la mayor plaga de seguridad de hoy en día en algunos sistemas.

  Los sistemas GNU/Linux están protegidos casi totalmente (existen diversos casos, así como pruebas de concepto), contra estos mecanismos víricos por varias razones: en los programas ejecutables, tienen un acceso muy limitado al sistema, en particular a la cuenta del usuario, con excepción del usuario root, cuenta que debería restringirse a los usos adecuados, li-

mitándose o evitándose completamente el uso como usuario del sistema. El correo no suele utilizar lenguajes de macros no verificados (como en el caso de Outlook y Visual Basic Script en Windows, que en su día fueron un aporte importante de agujeros de entrada para virus), mientras en el caso de los documentos, estamos en condiciones parecidas, ya que no soportan lenguajes de macros no verificados (como el VBA en Microsoft Office). Aunque hay que comentar que varios aspectos de este mismo tipo comienzan a tener cierta importancia, ya que algunas de las *suites* de ofimática para GNU/Linux utilizan ya lenguajes de macros, y los clientes de correo cada vez incorporan más soporte de html empotrado con soporte de JavaScript, una de las fuentes con más problemas, como veremos en algunos apartados siguientes. Que estos problemas no hayan sido explotados (o solo mínimamente) es solo una cuestión de cuán usada es una plataforma, como GNU/Linux. En la medida en que crece su uso, también se hace más atractiva para los atacantes.

En todo caso, habrá que prestar atención a lo que pueda pasar en un futuro, ya que podrían surgir algunos virus específicos para GNU/Linux aprovechando algunos *bugs* de los componentes del sistema. Un punto que sí que hay que tener en cuenta es el de los sistemas de correo, ya que si bien nosotros no generaremos virus, sí que podemos llegar a transmitirlos; por ejemplo, si nuestro sistema funciona como *router*, *relay*, o simplemente como servidor de correo, podrían llegar mensajes con virus que podrían ser enviados a otros. Aquí se puede implementar alguna política de detección y filtrado de virus, si en nuestros sistemas existen entornos Windows, en los que se pueden propagar virus externos que hayamos recibido. Otra forma muy importante de problemáticas que podría entrar dentro de la categoría de virus son los mensajes basura (*spam*), que si bien no suelen ser utilizados como elementos atacantes (aunque pueden combinarse con otras técnicas, como intentos de *phising*), sí que podemos considerarlos como un problema por su "virulencia" de aparición y por el coste económico que pueden causar (pérdidas de tiempo y recursos).

- **Gusanos (*worms*)**: normalmente se trata de un tipo de programas que aprovechan algún agujero del sistema para realizar ejecuciones de código sin permiso. Suelen ser utilizados para aprovechar recursos de la máquina, como el uso de CPU, cuando se detecta que el sistema no funciona o no está en uso o, si son malintencionados, con el objetivo de robar recursos o utilizarlos para parar o bloquear el sistema. También suelen utilizar técnicas de transmisión y replicacion.

Gusano Morris

Uno de los primeros gusanos, el conocido como Gusano Morris, fue muy importante porque hizo de la seguridad un tema importante, en unos momentos en que había cierta inocencia en estos temas. Véase http://en.wikipedia.org/wiki/Morris_worm.

- **Troyanos (*trojans* o *Trojan horses*)**: programas útiles que incorporan alguna funcionalidad pero ocultan otras que son las utilizadas para obtener información del sistema o comprometerlo. Un caso particular puede ser el de los códigos de tipo móvil en aplicaciones web, como los Java, JavaScript o ActiveX; estos normalmente piden su consentimiento para ejecutarse (ActiveX en Windows) o tienen modelos limitados de lo que pueden hacer

(Java, JavaScript). Pero como todo software, también pueden tener posibles agujeros, y son un método ideal para transmitir troyanos.

- **Puertas traseras (*back doors*)**: es un método de acceso escondido en un programa que puede utilizarse para otorgar acceso al sistema o a los datos manejados sin que lo sepamos. Otros efectos pueden ser cambiar la configuración del sistema o permitir la introducción de virus. El mecanismo usado puede ser desde venir incluidos en algún software común hasta en un troyano que produzca puertas traseras.

- **Bombas lógicas**: programa incrustado en otro, que comprueba que se den algunas condiciones (temporales, acciones del usuario, etc.), para activarse y emprender acciones no autorizadas.

- **Registradores de tecleo (*keyloggers*)**: se trata de un programa especial que se dedica a secuestrar las interacciones con el teclado y/o ratón del usuario. Pueden ser programas individuales o bien troyanos incorporados en otros programas. Normalmente, necesitarían introducirse en un sistema abierto al que se dispusiese de acceso (aunque cada vez más pueden venir incorporados en troyanos que se instalen). La idea es captar cualquier introducción de teclas, de manera que se capturen contraseñas (por ejemplo, las bancarias), la interacción con aplicaciones, los sitios visitados por la red, los formularios rellenados, etc.

- **Escaneo de puertos (*port scanning*)**: más que un ataque, sería un paso previo, que consistiría en la recolección de posibles objetivos. Básicamente, consiste en utilizar herramientas que permitan examinar la red en busca de máquinas con puertos abiertos, sean TCP, UDP u otros protocolos, que indican la presencia de algunos servicios. Por ejemplo, escanear máquinas buscando el puerto 80 TCP, indica la presencia de servidores web, de los cuales podemos obtener información sobre el servidor y la versión que utilizan para aprovecharnos de vulnerabilidades conocidas.

- **Husmeadores (*sniffers*)**: permiten la captura de paquetes que circulan por una red. Con las herramientas adecuadas podemos analizar comportamientos de máquinas: cuáles son servidores, cuáles son clientes, qué protocolos se utilizan y en muchos casos obtener contraseñas de servicios no seguros. En un principio, fueron muy utilizados para capturar contraseñas de telnet, rsh, rcp, ftp, etc., servicios no seguros que no tendrían que utilizarse (en su lugar deberían usarse las versiones seguras: SSH, scp, sftp). Tanto los husmeadores como los escaneadores de puertos no son necesariamente una herramienta de ataque, ya que también pueden servir para analizar nuestras redes y detectar fallos, o simplemente para analizar nuestro tráfico. Normalmente, tanto las técnicas de escaneo como las de los husmeadores suelen utilizarse por parte de un intruso con el objetivo de encontrar las vulnerabilidades del sistema, ya sea para conocer datos de un sistema desconocido (escaneadores) o para analizar la interacción interna (husmeadores).

- **Secuestro (*hijacking*)**: son técnicas que intentan colocar una máquina de manera que intercepte o reproduzca el funcionamiento de algún servicio en otra máquina a la que ha pinchado la comunicación. Suelen ser habituales los casos para correo electrónico, transferencia de ficheros o web. Por ejemplo, en el caso web, se puede capturar una sesión y reproducir lo que el usuario está haciendo, páginas visitadas, interacción con formularios, etc. En algunos casos puede ser a causa de secuestros a nivel de red, debido a que se capturan o escuchan paquetes de las comunicaciones, o puede producirse un secuestro *offline* mediante ejecución arbitraria de código de *scripts*. Por ejemplo, es común en sistemas de correo electrónico por webmail, o en aplicaciones web que incluyan el uso de html en algunas de sus ventanas, que el sistema permita inyectar código JavaScript o PHP (u otros lenguajes de *script* similares), que si no se controla o limita de forma correcta, puede provocar el robo de datos del usuario, ya sea sus identificadores de sesión o sus galletas (*cookies*) utilizadas, lo que permite al atacante reproducir o secuestrar *a posteriori* sus sesiones de correo o de aplicación web, sin necesidad de contraseña alguna, y así secuestrar la sesión y la identidad del usuario.

- **Desbordamientos (*buffer overflows*)**: técnica bastante compleja que aprovecha errores de programación en las aplicaciones. La idea básica es aprovechar desbordamientos (*overflows*) en *buffers* de la aplicación, ya sean colas, *arrays*, vectores, etc. Si no se controlan los límites, un programa atacante puede generar un mensaje o dato más grande de lo esperado y provocar fallos mediante la escritura por encima de los límites permitidos por la estructura de datos, y así sobreescribir memoria adyacente. Por ejemplo, en muchas aplicaciones C con *buffers* mal escritos, en *arrays*, si sobrepasamos el límite podemos provocar una sobreescritura del código del programa, lo que causa un funcionamiento incorrecto o la caída del servicio o máquina. Es más, una variante más compleja permite incorporar en el ataque trozos de programa (compilados en C o bien *shell scripts*) que pueden permitir la ejecución de cualquier código que el atacante quiera introducir.

  Algunas variaciones de esta técnica pueden aprovecharse para atacar de forma similar otras partes del código ejecutable de un programa, como por ejemplo la pila del programa (hablamos entonces de *stack overflows*) o la gestión de las peticiones dinámicas de memoria que realice el ejecutable (*heap overflows*). La alteración tanto de la pila como de la memoria dinámica tambien puede permitir al atacante la ejecución de código arbitrario añadido o la caída del ejecutable o servicio.

- **Denegación de servicio, *Denial of Service*, DoS**: este tipo de ataque provoca que la máquina caiga o que se sobrecarguen uno o más servicios, de manera que no sean utilizables. Otra técnica es la **DDoS (*Distributed DoS*, DoS distribuida)**, que se basa en utilizar un conjunto de máquinas distribuidas para que produzcan el ataque o sobrecarga de servicio. Este tipo de ataques se suelen solucionar con actualizaciones del software, y con una

correcta configuración de los parámetros de control de carga del servicio, ya que normalmente se ven afectados aquellos servicios que no fueron pensados para una carga de trabajo determinada y no se controla la saturación. Los ataques DoS y DDoS son bastante utilizados en ataques a sitios web o servidores DNS, que se ven afectados por vulnerabilidades de los servidores, por ejemplo, de versiones concretas de Apache o BIND. Otro aspecto que cabe tener en cuenta es que nuestro sistema también podría ser usado para ataques de tipo DDoS, mediante control, ya sea a través de una puerta trasera o de un troyano que forma parte de una botnet, por ejemplo, que dispone de una serie de programas en múltiples máquinas interconectadas, que pueden estar en un estado latente, y participar a *posteriori* en ataques DDoS bajo demanda.

Un ejemplo de este ataque (DoS) bastante sencillo es el conocido como SYN flood, que trata de generar paquetes TCP que abren una conexión, pero ya no hacen nada más con ella, simplemente la dejan abierta. Esto consume recursos del sistema en estructuras de datos del núcleo y recursos de conexión por red. Si se repite este ataque centenares o miles de veces, se consigue ocupar todos los recursos sin utilizarlos, de modo que cuando algunos usuarios quieran utilizar el servicio, les sea denegado porque los recursos están ocupados. Otro caso conocido es el bombardeo de correo (*mail bombing*) o, simplemente, el reenvío de correo (normalmente con emisor falso) hasta que se saturan las cuentas de correo y el sistema de correo cae o se vuelve tan lento que es inutilizable. Estos ataques son en cierta medida sencillos de realizar con las herramientas adecuadas, y no tienen una solución fácil, ya que se aprovechan del funcionamiento interno de los protocolos y servicios; en estos casos tenemos que tomar medidas de detección y control posterior, así como poner límites a los parámetros y la carga de servicios involucrados en estos problemas.

Enlaces de interés

Sobre SYN flood, véase: http://www.cert.org/advisories/CA-1996-21.html.
Sobre E-mail bombing y spam, véase: http://www.cert.org/tech_tips/email_bombing_spamming.html.

- **Falseamiento de identidad (*spoofing*)**: las técnicas de *spoofing* engloban varios métodos (normalmente complejos) para falsificar tanto la información como los participantes en una transmisión (origen y/o destino). Algunos métodos de *spoofing* son los siguientes:

– *IP spoofing*, falsificación de una máquina, permitiendo que genere tráfico falso o escuche tráfico que iba dirigido a otra máquina. Combinado con otros ataques, puede saltarse incluso protección de cortafuegos.

– *ARP spoofing*, técnica compleja (utiliza un DDoS), que intenta falsificar las direcciones de fuentes y destinatarios de una red, mediante ataques de las cachés de ARP que poseen las máquinas, de manera que se sustituyan las direcciones reales por otras en varios puntos de una red. Esta técnica permite saltarse todo tipo de protecciones, incluidos cortafuegos, pero no es una técnica sencilla.

– Correo electrónico, es quizás el más sencillo. Se trata de generar contenidos de correos falsos, tanto por el contenido como por la dirección de origen. En este tipo son bastante utilizadas las técnicas de lo que se denomina

ingeniería social, que básicamente intenta engañar de una forma razonable al usuario. Un ejemplo clásico son los correos falsos del administrador del sistema, o bien del banco donde tenemos nuestra cuenta corriente, en los que se mencionan problemas con la gestión de nuestras cuentas y nos piden enviar información confidencial o la contraseña anterior para solucionarlos, o se nos pide que la contraseña se cambie por otra concreta. Sorprendentemente, esta técnica (también conocida como pesca o *phishing*) consigue engañar a un número considerable de usuarios. Incluso con ingeniera social de métodos sencillos: algún *cracker* famoso comentaba que su método preferido era el teléfono. Como ejemplo, exponemos el caso de una empresa de certificación (Verisign), de la que los *crackers* obtuvieron la firma privada de software Microsoft con solo realizar una llamada mencionando que telefoneaban de parte de la empresa, que se les había presentado un problema y que volvían a necesitar su clave. Resumiendo, unos niveles altos de seguridad informática se pueden ir al traste por una simple llamada telefónica o un correo que un usuario malinterprete.

Enlace de interés

Véase el caso de Verisign-Microsoft en: http://www.computerworld.com/softwaretopics/os/windows/story/0,10801,59099,00.html.

- ***SQL injection***: es una técnica orientada a bases de datos, y a servidores web en particular, que se aprovecha generalmente de programación incorrecta de formularios web, en los cuales no se ha controlado correctamente la información que se proporciona. No se determina que la información de entrada sea del tipo correcto (fuertemente tipada respecto a lo que se espera) o no se controla qué tipo o caracteres literales se introducen. La técnica se aprovecha del hecho que los caracteres literales obtenidos por los formularios (por ejemplo web, aunque los ataques pueden sufrirse desde cualquier API que permita el acceso a una base de datos, por ejemplo php o perl) son utilizados directamente para construir las consultas (en SQL) que atacarán a una determinada base de datos (a la que en principio no se tiene acceso directo). Normalmente, si existen las vulnerabilidades y hay poco control de los formularios, se puede inyectar código SQL en el formulario, de manera que se construyan consultas SQL que proporcionen la información buscada. En casos drásticos podría obtenerse la información de seguridad (usuarios y contraseñas de la base de datos) o incluso tablas o la base de datos entera, y también podrían producirse pérdidas de información o borrados intencionados de datos. En particular, esta técnica en ambientes web puede ser grave (para la empresa proveedora de servicio, con fuertes consecuencias económicas), debido a las leyes que protegen la privacidad de datos personales, que se pueden ver comprometidos, hacerse públicos o ser vendidos a terceros si se obtienen por un ataque de este tipo. En este caso de ataque, más que una cuestión de seguridad del sistema, se trata de un problema de programación y control con tipado fuerte de los datos esperados en la aplicación, además del adecuado control de conocimiento de vulnerabilidades presentes del software usado (base de datos, servidor web, API como php, perl, etc.).

- ***Cross-site scripting*, XSS**: es otra problemática relacionada con ambientes web, en particular con alteraciones del código HTML y *scripts* cuando un usuario visualiza un determinado sitio web, que puede ser alterado diná-

micamente. Se aprovecha generalmente de los errores a la hora de validar código HTML (todos los navegadores tienen más o menos problemas, por la propia definición del HTML original, que permite leer prácticamente cualquier código HTML por incorrecto que sea). En algunos casos, la utilización de vulnerabilidades puede ser directa mediante *scripts* en la página web, pero normalmente los navegadores tienen buen control de los mismos. Por otra parte, indirectamente hay técnicas que permiten insertar código de *script*, bien mediante acceso a las galletas (*cookies*) del usuario desde el navegador, bien mediante la alteración del proceso por el que se redirecciona una página web a otra. También hay técnicas mediante marcos (*frames*), que permiten redirigir el HTML que se está viendo o colgar directamente el navegador. En particular, pueden ser vulnerables los motores de búsqueda de los sitios web, que pueden permitir la ejecución de código de *scripts*. En general, son ataques con diversas técnicas complejas, pero con el objetivo de capturar información como *cookies*, que pueden ser usadas en sesiones y permitir así sustituir a una determinada persona mediante redirecciones de sitios web u obtener información suya. Otra vez desde la perspectiva de sistema, es más una cuestión de software en uso. Hay que controlar y conocer vulnerabilidades detectadas en navegadores (y aprovechar los recursos que estos ofrecen para evitar estas técnicas) y controlar el uso de software (motores de búsqueda empleados, versiones del servidor web y API utilizadas en los desarrollos).

En general, algunas recomendaciones (muy básicas) para la seguridad podrían ser:

- Controlar un factor problemático: los usuarios. Uno de los factores que puede afectar más a la seguridad es la confidencialidad de las contraseñas, y esta se ve afectada por el comportamiento de los usuarios; esto facilita a posibles atacantes las acciones desde el interior del propio sistema. La mayoría de los ataques suelen venir de dentro del sistema, es decir, una vez el atacante ya ha ganado acceso al sistema.

- Entre los usuarios, está aquel que es un poco olvidadizo (o indiscreto), que olvida la contraseña cada dos por tres, la menciona en conversaciones, la escribe en un papel que olvida, o que está junto (o pegado) al ordenador o sobre la mesa de trabajo, o que simplemente la presta a otros usuarios o conocidos. Otro tipo es el que coloca contraseñas muy predecibles, ya sea su mismo identificador de usuario, su nombre, su DNI, el nombre de su novia, el de su madre, el de su mascota, etc., cosas que con un mínimo de información pueden encontrarse fácilmente (más en estos momentos de auge de las redes sociales, donde es fácil obtener este tipo de información desde diferentes redes en las que participe el usuario). Otro caso son los usuarios normales con un cierto conocimiento, que colocan contraseñas válidas, pero siempre hay que tener en cuenta que hay mecanismos que pueden encontrarlas (*cracking* de passwords, husmeadores, suplantación, etc.). Hay que establecer una cierta **cultura de seguridad** entre los usuarios

y, mediante diferentes técnicas, obligarles a que cambien las contraseñas, que no utilicen palabras típicas, que usen contraseñas largas (tener más de 6 o 8 cáracteres mínimo), etc. Últimamente, en muchas empresas e instituciones se está implantando la técnica de hacer firmar un contrato (o carta de compromisos) al usuario de manera que se le obliga a no divulgar la contraseña o cometer actos de vandalismo o ataques desde su cuenta (claro que esto no impide que otros lo hagan por él).

- No utilizar ni ejecutar programas de los que no podamos garantizar su origen. Normalmente, muchos distribuidores utilizan mecanismos de comprobación de firmas para verificar que los paquetes de software son tales, como por ejemplo las sumas md5 (comando `md5sum`) o la utilización de firmas GPG [Hatd] (comando `gpg`). El vendedor o distribuidor provee una suma md5 de su archivo (o imagen de CD/DVD), de modo que podemos comprobar su autenticidad. Últimamente, en las distribuciones se están usando tanto firmas para paquetes individuales como las firmas para los repositorios de paquetes, como mecanismo para asegurar la fiabilidad del proveedor.

- No utilizar usuarios privilegiados (como root) para el trabajo normal de la máquina, ya que cualquier programa (o aplicación) tendría los permisos para acceder a cualquier zona del sistema de archivos o a servicios en ejecución.

- No acceder remotamente con usuarios privilegiados ni ejecutar programas que puedan tener privilegios. Y más si no conocemos, o no hemos comprobado, los niveles de seguridad del sistema y sus conexiones. En particular, otro de los problemas actuales son las redes inalámbricas (como WiFi) inseguras, ya que, por ejemplo, algunos cifrados como el usado en redes WiFi WEP son muy débiles. No se deben usar conexiones críticas sobre redes inseguras.

- No utilizar elementos (programas o servicios) que no sabemos cómo actúan ni intentar descubrirlo a base de repetidas ejecuciones.

Estas medidas pueden ser poco productivas, pero si no hemos asegurado el sistema, no podemos tener ningún control sobre lo que puede pasar, y aun así, nadie asegura que no se pueda introducir algún programa malicioso que burle la seguridad si lo ejecutamos con los permisos adecuados. En general, hay que considerar que debemos tener una vigilancia activa con todo este tipo de actividades que supongan accesos y ejecución de tareas de formas más o menos privilegiadas y un uso de mecanismos de comunicación inseguros.

1.2. Contramedidas

Respecto a las medidas que se pueden tomar sobre los tipos de ataques presentados, podemos encontrar algunas preventivas y de detección de lo que sucede en nuestros sistemas.

Veamos algunos tipos de medidas que podríamos tomar en los ámbitos de prevención y detección de intrusos (se mencionan herramientas útiles, algunas las examinaremos más adelante):

- ***Password cracking***: en ataques de fuerza bruta para romper las contraseñas suele ser habitual intentar obtener acceso por *login* de forma repetida; si se consigue entrar, la seguridad del usuario ha sido comprometida y se deja la puerta abierta a otros tipos de ataques como, por ejemplo, las puertas traseras o, simplemente, a la destrucción de la cuenta. Para prevenir este tipo de ataques, hay que reforzar la política de contraseñas, pidiendo una longitud mínima y cambios de contraseña periódicos. Una cosa que hay que evitar es el uso de palabras comunes en las contraseñas: muchos de estos ataques se hacen mediante la fuerza bruta, con un fichero de diccionario (con palabras en el idioma del usuario, términos comunes, nombres propios, argot, etc.). Este tipo de contraseñas serán las primeras en caer. También puede ser fácil obtener información del atacado, como nombres, DNI o su dirección e información procedente de sus redes sociales, y usar estos datos para probar las contraseñas. Por todo ello tampoco se recomiendan contraseñas con DNI, nombres (propios o de familiares, etc.), direcciones, nombres de mascotas, etc. Una buena elección suele ser elegir contraseñas de entre 6 y 8 caracteres como mínimo y que contengan caracteres alfabéticos, numéricos y algún carácter especial. Aunque la contraseña esté bien elegida, puede ser insegura si se utiliza en servicios o en redes abiertas, no seguras. Por lo tanto, se recomienda reforzar los servicios mediante técnicas y servicios de cifrado que protejan las contraseñas y las comunicaciones. Por el contrario, debe evitarse (o no usar) todos aquellos servicios que no soporten cifrado y que, consecuentemente, sean susceptibles de ser atacados con métodos, por ejemplo, de husmeadores; entre estos podríamos incluir servicios como telnet, ftp, rsh y rlogin, entre otros , de los que existen ya de alternativas más seguras.

- **Explotación de agujeros**: se debe evitar disponer de programas que no se utilicen, sean antiguos o no se actualicen (por ser obsoletos). Hay que aplicar los últimos parches y actualizaciones disponibles, tanto para las aplicaciones como para el sistema operativo, probar herramientas que detecten vulnerabilidades y mantenerse al día de las vulnerabilidades que se vayan descubriendo. Aunque cabe señalar que en determinadas ocasiones las propias correcciones de algún fallo de seguridad pasado pueden traernos otros nuevos. Como caso particular, hay que estar en especial atento a las divulgaciones de vulnerabilidades de tipo *0day*, que suelen explotar vulnerabilidades no conocidas y que los desarrolladores no han tenido tiempo de arreglar. En algunos casos, la ventana de vulnerabilidad (tiempo entre el conocimiento de la vulnerabilidad y su solución), contra lo que podría parecer, podría alargarse semanas, o incluso años, y convertirse en un fallo de seguridad desconocido por largo tiempo para el público general, pero no así para cierto mercado, que puede involucrar desde *crackers* hasta agencias gubernamentales, que aprovechan estas *0day* para motivos políticos (ciberguerra), invasión de la privacidad o espionaje industrial.

Enlaces de interés

Sobre vulnerabilidades, una buena herramienta es Nessus. Para descubrir nuevas vulnerabilidades, véase CERT en: http://www.cert.org/advisories/, sitio antiguo y http://www.us-cert.gov/cas/techalerts/index.html.

Caso Snowden

Véase NSA *versus* Snowden: http://en.wikipedia.org/wiki/Edward_Snowden

- **Virus**: se deben utilizar mecanismos o programas antivirus, sistemas de filtrado de mensajes sospechosos, evitar la ejecución de sistemas de macros (que no se puedan verificar). No hay que minimizar los posibles efectos de los virus, cada día se perfeccionan más y técnicamente es posible realizar virus simples que pueden desactivar redes en cuestión de minutos (sólo hay que observar algunos de los virus de los últimos años en ambientes Windows). En especial, si en nuestros sistemas existen entornos Windows, sería interesante la posibilidad de examinar virus que puedan afectar a estos sistemas. Existen antivirus para UNIX y Linux, como ClamAV, que evitarán que se transmitan virus a nuestros sistemas internos. En sistemas de correo basados en GNU/Linux, podemos establecer combinaciones de productos antivirus y antispam para evitar estas transmisiones, como por ejemplo utilizar ClamAV y otros productos como SpamAssasin.

- **Gusanos**: hay que controlar el uso de nuestras máquinas o usuarios en horas no previstas, con patrones de comportamiento infrecuentes, así como el control del tráfico de salida y entrada.

- **Troyanos**: se debe verificar la integridad de los programas periódicamente, mediante mecanismos de suma o de firmas; detectar el tráfico anómalo de salida o entrada al sistema, y utilizar cortafuegos para bloquear tráfico sospechoso. Una versión bastante peligrosa de los troyanos la forman los *rootkits* (comentados más adelante), que realizan más de una función gracias a un conjunto variado de herramientas. Para la verificación de la integridad, podemos utilizar mecanismos de sumas, como md5 o gpg, o herramientas que automatizan este proceso, como Tripwire o AIDE.

- **Puertas traseras**: hay que obtener de los proveedores o vendedores del software la certificación de que no contiene ningún tipo de puerta trasera escondida no documentada y, por supuesto, aceptar el software proveniente solo de sitios que ofrezcan garantías. Cuando el software sea de terceros o de fuentes que podrían haber modificado el software original, muchos fabricantes (o distribuidores) integran algún tipo de verificación de software basado en códigos de suma o firmas digitales (tipo md5 o gpg) [Hatd]. Siempre que estas estén disponibles, sería útil verificarlas antes de proceder a la instalación del software. También puede probarse el sistema de forma intensiva, antes de colocarlo como sistema de producción. Otro problema puede consistir en la alteración del software *a posteriori*. En este caso pueden ser también útiles los sistemas de firmas o sumas para crear códigos sobre software ya instalado y controlar que no se produzcan cambios en software vital. También resultan útiles las copias de seguridad, con las que podemos hacer comparaciones para detectar cambios.

- **Bombas lógicas**: en este caso, suelen ocultarse tras activaciones por tiempo o por acciones del usuario. Podemos verificar que no existan en el sistema trabajos no interactivos introducidos de tipo `crontab`, `at` y otros procesos (por ejemplo, de tipo `nohup`), que dispongan de ejecución periódica o que estén en ejecución en segundo plano desde hace mucho tiempo (coman-

Enlaces de interés

Véanse parches y vulnerabilidades para el sistema operativo en: http://www.debian.org/security, https://fedoraproject.org/wiki/Security_Team

dos `w`, `jobs`). En cualquier caso, podrían utilizarse medidas preventivas que impidieran trabajos programados no interactivos a los usuarios (`crontab`) o que solamente los permitiesen a aquellos que realmente lo necesiten.

- **Registradores de tecleo *(keyloggers)* y *rootkits***: en este caso habrá algún proceso intermediario que intentará capturar nuestras pulsaciones de teclas y las almacenará o comunicará en algún lugar. Habrá que examinar situaciones donde aparezca algún proceso extraño perteneciente a nuestro usuario, o bien detectar si tenemos algún fichero abierto con el que no estemos trabajando directamente (por ejemplo, podría ser de ayuda `lsof`, ver man), o bien conexiones de red, si se tratase de un registrador de tecleo con envío externo. Para probar un funcionamiento muy básico de un registrador de tecleo muy sencillo, puede verse el comando de sistema `script` (ver man script). El otro caso, el *rootkit* (que suele incluir también algún registrador de tecleo) suele ser un paquete de unos cuantos programas con varias técnicas, y permite al atacante, una vez entra en una cuenta, utilizar diversos elementos como un registrador de tecleo, puertas traseras, troyanos (sustituyendo a comandos del sistema), etc., con tal de obtener información y puertas de entrada al sistema. Muchas veces se acompaña de programas que realizan limpieza de los registros, para eliminar las pruebas de la intrusión. Un caso particularmente peligroso lo forman los *kernel rootkits*, que se usan o vienen en forma de módulos de núcleo, lo que les permite actuar a nivel del mismo núcleo. Una herramienta útil para verificar los rootkits es `chkrootkit`, u otras alternativas como `rkhunter`.

Enlace de interés

La herramienta chkrootkit puede encontrarse en: http://www.chkrootkit.org.

- **Escaneo de puertos**: los escaneadores suelen lanzar sobre uno o más sistemas, bucles de escaneo de puertos conocidos, para detectar los que quedan abiertos y aquellos en los que los servicios estarán funcionando (y obtener, por tanto, información de las versiones de los servicios), y así conocer si podrían ser susceptibles de ataques.

- **Husmeadores**: deben evitarse intercepciones e impedir así la posibilidad de que se introduzcan escuchas. Una técnica es el uso de hardware específico para la construcción de la red, de manera que pueda dividirse en segmentos para que el tráfico solo circule por la zona que se va a utilizar, poner cortafuegos para unir estos segmentos y poder controlar el tráfico de entrada y salida. Usar técnicas de cifrado para que los mensajes no puedan ser leídos e interpretados por alguien que escuche la red. Para el caso tanto de escáneres como de husmeadores, podemos utilizar herramientas como `Wireshark` (antiguo `Ethereal`) y `Snort`, para realizar comprobaciones sobre nuestra red o, para el escaneo de puertos, *Nmap*. En el caso de los husmeadores, estos pueden ser detectados en la red mediante la búsqueda de máquinas en modo Ethernet promiscuo (están a la escucha de cualquier paquete que circula). Normalmente, la tarjeta de red solo debería capturar el tráfico que va hacia ella (o de tipo *broadcast* o *multicast*).

- **Secuestro**: algunas contramedidas en este caso son las siguientes: implementar mecanismos de cifrado en los servicios, requerir autentificación y,

si es posible, que esta autentificación se renueve periódicamente; controlar el tráfico entrante o saliente mediante cortafuegos, y monitorizar la red para detectar flujos de tráfico sospechosos. En casos de aplicaciones web o correo electronico, limitar al máximo la utilización de códigos de *script* o, sencillamente, eliminar la posibilidad de ejecución de *scripts* internos a ventanas HTML cuando provengan de fuentes no fiables.

- **Desbordamientos**: suelen ser comunes como *bugs* o agujeros del sistema y suelen (si han sido previamente detectados) solucionarse mediante una actualización del software o paquete afectado. En todo caso, pueden observarse, gracias a los registros del sistema, situaciones extrañas de caída de servicios que deberían estar funcionando. También pueden maximizarse los controles de procesos y accesos a recursos para aislar el problema cuando se produzca en entornos de acceso controlado, como el que ofrece SELinux. Existen otras altenativas, como Grsecurity o PaX, que son parches de los núcleos oficiales de Linux para obtener protecciones adicionales en este sentido, tanto problemas de desbordamiento de *buffers*, como de pila o *heap*.

Ved también

SELinux se trata en el apartado 4 de este módulo.

- **Denegación de servicio y otros como SYN flood o bombardeo de correo**: se deben tomar medidas de bloqueo de tráfico innecesario en nuestra red (por ejemplo, por medio de cortafuegos). En aquellos servicios que se pueda, habrá que controlar tamaños de *buffer*, número de clientes por atender, tiempos de espera (*timeouts*) de cierre de conexiones, capacidades del servicio, etc.

- **Falseamiento de identidad**: a) *IP spoofing*, b) *ARP spoofing*, c) correo electrónico. Estos casos necesitan un fuerte cifrado de los servicios, control por cortafuegos y mecanismos de autentificación basados en varios aspectos (por ejemplo, no basarse en la IP, si pudiera verse comprometida). Se pueden aplicar mecanismos que controlen las sesiones establecidas y que se basen en varios parámetros de la máquina a la vez (sistema operativo, procesador, IP, dirección Ethernet, etc.). También se pueden monitorizar sistemas DNS, cachés de ARP, *spools* de correo, etc., para detectar cambios en la información que invaliden a anteriores.

- **Ingeniería social**: no es propiamente una cuestión informática, pero también es necesaria para que las personas no empeoren la seguridad. Existen diversas medidas adecuadas, como aumentar la información o educar a los usuarios (para que no divulguen datos privados ni información que pueda ofrecer pistas sobre sus contraseñas, advertirlos sobre el uso que hagan de las redes sociales, etc.) y divulgar técnicas básicas para mejorar su seguridad. También deben tenerse en cuenta otros temas de seguridad: controlar qué personal dispondrá de información crítica de seguridad y en qué condiciones puede cederla a otros. Los servicios de ayuda y mantenimiento de una empresa pueden ser un punto crítico, y debe controlarse quién posee información de seguridad, y cómo la usa.

Respecto a los usuarios finales, hay que esforzarse por mejorar su cultura de seguridad, tanto contraseñas (y en la dejadez de su uso), como en la gestión del software que utilizan a diario, por tal de mantenerlo actualizado, así como proporcionarles servicios confiables.

2. Seguridad del sistema

Ante los posibles ataques, tenemos que disponer de mecanismos de prevención, detección y recuperación de nuestros sistemas.

Para la prevención local, hay que examinar los diferentes mecanismos de autentificación y permisos de acceso a los recursos para poderlos definir correctamente, de manera que se garantice la confidencialidad y la integridad de nuestra información.

Incluso en este caso, no estaremos protegidos de atacantes que hayan ya obtenido acceso a nuestro sistema, o bien de usuarios hostiles que quieran saltarse las restricciones impuestas en el sistema. En estos casos que podemos tener problemas con el acceso local, podríamos tomar medidas addicionales, como el cifrado de la información de nuestros sistemas, o permitirlo a nuestros usuarios.

Cifrado de sistemas de ficheros

En los sistemas GNU/Linux, se implementan diversas opciones para el cifrado de datos, por ejemplo destacar *Encfs*, un sistema de ficheros en espacio de usuario que nos permite cifrar los datos de usuario; véase: http://www.arg0.net/encfs

Respecto a la seguridad en red, tenemos que garantizar que los recursos que ofrecemos (si proporcionamos unos determinados servicios) tengan los parámetros de confidencialidad necesarios (más si estos están garantizados por leyes jurídicas nacionales, el incumplimiento de las cuales puede causar graves sanciones, así como la publicidad negativa indirecta por la falta de cumplimiento). Los servicios ofrecidos no pueden ser usados por terceros no deseados, de modo que un primer paso será controlar que los servicios ofrecidos sean los que realmente queremos y que no estamos ofreciendo, además, otros servicios que no tenemos controlados. En el caso de servicios de los que nosotros somos clientes, también habrá que asegurar los mecanismos de autentificación, en el sentido de acceder a los servidores correctos y que no existan casos de suplantación de servicios o servidores (normalmente bastante difíciles de detectar a nivel de usuario).

Respecto a las aplicaciones y a los mismos servicios, además de garantizar la correcta configuración de niveles de acceso mediante permisos y la autentificación de los usuarios permitidos, tendremos que vigilar la posible explotación de fallos en el software. Cualquier aplicación, por muy bien diseñada e implementada que esté, puede tener un número más o menos alto de errores que pueden ser aprovechados para, con ciertas técnicas, saltarse las restricciones impuestas. En este caso, practicamos una política de prevención que pasa por mantener el sistema actualizado en la medida de lo posible, de manera que,

o bien actualizamos ante cualquier nueva corrección, o bien somos conservadores y mantenemos aquellas versiones que sean más estables en cuestión de seguridad. Normalmente, esto significa verificar periódicamente unos cuantos sitios de seguridad, de conocida solvencia, para conocer los últimos fallos detectados en el software, tanto de aplicaciones como de sistema, y las vulnerabilidades que se derivan de ellos y que podrían exponer nuestros sistemas a fallos de seguridad, ya sea localmente o por red.

3. Seguridad local

Acceso local

Diversos ataques, aunque vengan del exterior, tienen como finalidad conseguir el acceso local.

La seguridad local [Pen, Hatb] es básica para la protección del sistema [Deb, Hatc], ya que, normalmente, tras un primer intento de acceso desde la red, es la segunda barrera de protección antes de que un ataque consiga hacerse con parte del control de la máquina. Además, la mayoría de los ataques acaban haciendo uso de recursos internos del sistema.

3.1. *Bootloaders*

Respecto a la seguridad local, ya en arranque se nos pueden presentar problemas debido al acceso físico que un intruso pudiera tener a la máquina.

Uno de los problemas ya se encuentra en el propio arranque del sistema. Si el sistema puede arrancarse desde dispositivos extraíbles o desde CD/DVD, un atacante podría acceder a los datos de una partición GNU/Linux (o también en entornos Windows) solo con montar el sistema de ficheros y podría colocarse como usuario root sin necesidad de conocer ninguna contraseña. En este caso, se necesita proteger el arranque del sistema desde la BIOS, por ejemplo, protegiendo el acceso por contraseña, de manera que no se permita el arranque desde CD/DVD (por ejemplo mediante un LiveCD), USB u otras conexiones externas.

También es razonable actualizar la BIOS, ya que también puede tener fallos de seguridad. Además, hay que tener cuidado, porque la mayoría de fabricantes de BIOS ofrecen contraseñas extras conocidas (una especie de puerta trasera), con lo cual no podemos depender de estas medidas en exclusiva.

El siguiente paso es proteger el *bootloader*, ya sea Lilo o Grub, para que el atacante no pueda modificar las opciones de arranque del núcleo o modificar directamente el arranque (caso de Grub). Cualquiera de los dos puede protegerse también por contraseñas adicionales.

En Grub-Legacy (Grub1), el fichero `/sbin/grub-md5-crypt` pide la contraseña y genera una suma md5 asociada. En algunas distribuciones en que no está disponible este comando, puede realizarse de manera alternativa durante el arranque: al cargar Grub, accedemos con la tecla `c` en *shell* de Grub en el momento de inicio e introducimos `md5crypt` para obtener el *hash* md5 asociado a una contraseña que propongamos.

Después, el valor obtenido se introduce en `/boot/grub/grub.conf`. Bajo la línea `timeout`, se introduce:

```
password --md5 suma-md5-calculada
```

Para LiLo se coloca, o bien una contraseña global con:

```
password=contraseña
```

o bien una en la partición que queramos:

```
image = /boot/vmlinuz-version
password = contraseña
restricted
```

En este caso `restricted` indica, además, que no se podrán cambiar los parámetros pasados al núcleo desde la línea de comandos. Hay que tener cuidado de poner el fichero `/etc/lilo.conf` protegido a solo lectura/escritura desde el root (`chmod 600`), si no cualquiera puede leer las contraseñas.

En el caso de Grub 2, es posible realizarlo mediante *menuentry*, disponible en la configuración, definiendo passwords para usuarios concretos, que después se pueden añadir a cada menuentry definida, además de existir la posibilidad de especificar un superusuario que dispone de acceso a todas las entradas. Un ejemplo simple de configuración sería el siguiente: un superusario que tiene acceso a todas las entradas, mientras que un segundo usuario solamente puede acceder a la segunda entrada.

```
set superusers=rootgrub
password rootgrub password1
password otheruser password2

menuentry "GNU/Linux" {
set root=(hd0,1)
linux /vmlinuz
}

menuentry "Windows" --users otheruser {
set root=(hd0,2)
chainloader +1
}
```

En este caso la configuración de Grub2 usa passwords en claro, y hay que asegurarse de que el fichero no disponga de acceso de lectura para otros usua-

rios, o en su lugar usar métodos alternativos para la creación de passwords, por ejemplo, mediante *grub-mkpasswd-pbkdf2*, que nos permitirá generar *hashes* del password a usar. En caso de querer dejar *menuentries* disponibles para todos, también puede usarse *–unrestricted* en la definición de la entrada.

En este último punto, cabe recordar una recomendación general: nunca los ficheros con información sensible deberían estar disponibles con permisos de lectura a terceros, y siempre que sea posible se deberá evitarlos o cifrarlos. En particular, para datos sensibles sería deseable algún tipo de sistema de ficheros que permitiese el cifrado de los ficheros presentes (o incluso el disco o sistema entero). Por ejemplo, podemos destacar Ecryptfs y EncFS, que ya está disponible en varias distribuciones, para el cifraje de los archivos de los directorios privados de los usuarios o el sistema de ficheros completo.

Sistemas de ficheros cifrados

Los sistemas de ficheros cifrados también son una buena opción para proteger los datos sensibles de usuarios, como por ejemplo, entre otros, Ecryptfs y EncFS. Véase, por ejemplo, Encfs: http://www.arg0.net/encfs. Otra opción son sistemas de cifrados de disco, como dm-crypt/LUKS: https://code.google.com/p/cryptsetup/wiki/DMCrypt

Otro tema relacionado con el arranque es la posibilidad de que alguien que tenga acceso al teclado, reinicie el sistema, ya que si se pulsa `CTRL+ALT+DEL` (ahora, en varias distribuciones, esta opción suele estar desactivada por defecto), se provoca una operación de cierre en la máquina. Este comportamiento viene definido (en un sistema con sysvinit) en `/etc/inittab`, con una línea como:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

Si se comenta, esta posibilidad de reiniciar quedará desactivada. O por el contrario, puede crearse un fichero `/etc/shutdown.allow`, que permite a ciertos usuarios poder apagar y/o reiniciar.

3.2. Contraseñas y *shadows*

Las contraseñas típicas de los sistema UNIX iniciales (y de primeras versiones GNU/Linux) estaban cifradas mediante unos algoritmos DES (pero con claves pequeñas, y una llamada de sistema se encargaba de cifrar y descifrar, en concreto `crypt`, ver su man).

Normalmente, se encontraban en el fichero `/etc/passwd`, en el segundo campo, por ejemplo:

```
user:sndb565sadsd:...
```

Pero el problema está en que este fichero es legible por cualquier usuario del sistema, con lo que un atacante podía obtener el fichero y utilizar un ataque de fuerza bruta, hasta que descifrase las contraseñas contenidas en el fichero, o bien mediante ataques de fuerza bruta un poco más inteligentes, por medio de diccionarios.

El primer paso es utilizar los ficheros `/etc/shadow`, donde se guarda ahora la contraseña. Este fichero es solo legible por el root. En este caso, en `/etc/passwd` aparece un asterisco (*****) donde antes estaba la contraseña cifrada. Por defecto, las distribuciones de GNU/Linux actuales usan contraseñas de tipo *shadow*, a no ser que se les diga que no las usen.

Un segundo paso es cambiar el sistema de cifrado de las contraseñas por uno más complejo y difícil de romper. Ahora, tanto Fedora como Debian ofrecen contraseñas por md5; normalmente nos dejan escoger el sistema en tiempo de instalación. Hay que tener cuidado con las contraseñas md5, ya que si usamos NIS, podríamos tener algún problema, dependiendo de las versiones; si no, todos los clientes y servidores usarán md5 para sus contraseñas. Las contraseñas se pueden reconocer en `/etc/shadow` porque vienen con un prefijo "`$id$`" con `id=1`. No son los únicos algoritmos de cifrado que pueden usarse, `id=5` es sha-256 o `id=6` es sha-512. De hecho, para utilizar contraseñas cifradas más fiables, puede usarse (en algunas distribuciones) el comando `mkpasswd`, al que puede aportarse el cifrado utilizado y la contraseña a cifrar (véanse los man de `mkpasswd` y `crypt`).

Otras posibles actuaciones consisten en obligar a los usuarios a cambiar la contraseña con frecuencia, imponer restricciones en el tamaño y el contenido de las contraseñas y validarlas con diccionarios de términos comunes (definiendo estas políticas por una combinación de parámetros en `/etc/passwd` o mediante control por módulos PAM).

Respecto a las herramientas, es interesante disponer de un *cracker* de contraseñas (o sea, un programa para probar y romper contraseñas), para comprobar la situación real de seguridad de las cuentas de nuestros usuarios y forzar así el cambio en las que detectemos inseguras. Dos de las más utilizadas por administradores son `john the ripper (paquete john)` y `crack`. Pueden funcionar también por diccionario, con lo cual, será interesante disponer de algún ASCII de español (pueden encontrarse en la red).

Una cuestión que debe tenerse en cuenta siempre es hacer estas pruebas sobre nuestros sistemas. No hay que olvidar que los administradores de otros sistemas (o el proveedor de acceso o ISP) tendrán sistemas de detección de intrusos habilitados y podemos ser objeto de denuncia por intentos de intrusión, ya sea ante las autoridades competentes (unidades de delitos informáticos) o en nuestro ISP para que se nos cierre el acceso. Hay que tener mucho cuidado con el uso de herramientas de seguridad, que están siempre al filo de la navaja entre ser de seguridad o de intrusión.

3.3. Bits *sticky* y *setuid*

Otro problema importante son algunos permisos especiales que son utilizados sobre ficheros o *scripts*.

El bit *sticky* se utiliza sobre todo en directorios temporales, donde queremos que en algunos grupos (a veces no relacionados), cualquier usuario pueda escribir, pero solo pueda borrar el propietario del directorio o bien el propietario del fichero que esté en el directorio. Un ejemplo clásico de este bit es el directorio temporal `/tmp`. Hay que vigilar que no haya directorios de este tipo, ya que pueden permitir que cualquiera escriba en ellos, por lo que habrá que comprobar que no haya más que los puramente necesarios como temporales. El bit se coloca mediante (`chmod +t dir`) y puede quitarse con `-t`. En un `ls` aparecerá como un directorio con permisos `drwxrwxrwt` (observad la última `t`).

El bit *setuid* permite a un usuario ejecutar (ya sea un programa binario ejecutable o un *shell script*) con los permisos de otro usuario. Esto en algún caso puede ser de utilidad, pero es potencialmente peligroso. Es el caso, por ejemplo, de programas con *setuid* de root: un usuario, aunque no tiene permisos de root, puede ejecutar un programa con *setuid* que puede disponer de permisos internos de usuario root. Esto es muy peligroso en caso de *scripts*, ya que podrían editarse y modificarse para realizar cualquier tarea. Por lo tanto, hay que tener controlados estos programas y, en caso de que no se necesite el *setuid*, eliminarlo. El bit se coloca mediante `chmod +s`, ya sea aplicándolo al propietario (se llama entonces *suid*) o al grupo (se llama bit *sgid*); puede quitarse con `-s`. En el caso de visualizar con `ls`, el fichero aparecerá con `-rwSrw-rw` (observad la `S`), si es sólo *suid*, en *sgid* la `S` aparecería tras la segunda `w`.

En caso de utilizar `chmod` con notación octal, se usan cuatro cifras, donde las tres últimas son los permisos clásicos `rwxrwxrwx` (recordad que se debe sumar en la cifra 4 para `r`, 2 `w`, y 1 para `x`) y la primera tiene un valor para cada permiso especial que se quiera (que se suman): 4 (para *suid*), 2 (*sgid*) y 1 (para *sticky*).

Sería deseable hacer un análisis periódico del sistema de ficheros para detectar los ficheros con *suid* y *sgid* en el sistema y determinar si estos son realmente necesarios o pueden provocar problemas de seguridad.

Tambien es posible detectar programas que se esten ejecutando con permisos suid, mediante por ejemplo:

```
# ps ax -o pid,euser,ruser,comm
```

si en este caso el usuario efectivo (euser) y el usuario real (ruser) no coinciden, seguramente se trate de un ejecutable o script con permisos suid.

3.4. Habilitación de *hosts*

En el sistema hay unos cuantos ficheros de configuración especiales que permiten habilitar el acceso a una serie de *hosts* para algunos servicios de red, pero

cuyos errores pueden permitir atacar después la seguridad local. Nos podemos encontrar con:

- `.rhosts` de usuario: permite que un usuario pueda especificar una serie de máquinas (y usuarios) que pueden usar su cuenta mediante comandos "r" (rsh, rcp...) sin necesidad de introducir la contraseña de la cuenta. Esto es potencialmente peligroso, ya que una mala configuración del usuario podría permitir entrar a usuarios no deseados, o que un atacante (con acceso a la cuenta del usuario) cambie las direcciones en `.rhosts` para poder entrar cómodamente sin ningún control. Normalmente, no se tendría que permitir crear estos archivos, e incluso habría que borrarlos completamente y deshabilitar los comandos "r". Un análisis periódico del sistema será útil para detectar la creación de estos ficheros.

- `/etc/hosts.equiv`: es exactamente lo mismo que los ficheros `.rhosts` pero a nivel de máquina, especificando qué servicios, qué usuarios y qué grupos pueden acceder sin control de contraseña a los servicios "r". Además, un error como poner en una línea de ese fichero un "+", permite el acceso a "cualquier" máquina. Normalmente, hoy en día tampoco suele existir por defecto este fichero creado, y siempre existen como alternativa a los "r" los servicios tipo SSH.

- `/etc/hosts.lpd`: en el sistema de impresión LPD se utilizaba para habilitar las máquinas que podían acceder al sistema de impresión. Hay que tener especial cuidado, si no estamos sirviendo impresión, de deshabilitar completamente el acceso al sistema, y en caso de que lo estemos haciendo, restringir al máximo las máquinas que realmente hacen uso del mismo. También se puede intentar cambiar a un sistema de impresión CUPS, por ejemplo, que tiene mucho más control sobre los servicios. El sistema de impresión LPD había sido un blanco habitual de ataques de tipo gusano o de desbordamiento, y están documentados varios *bugs* importantes. Hay que estar atentos si aún utilizamos este sistema y el fichero `hosts.lpd`.

3.5. Módulos PAM

Los módulos PAM [Pen, Mor03] son un método que permite al administrador controlar cómo se realiza el proceso de autentificación de los usuarios para determinadas aplicaciones. Las aplicaciones tienen que haber sido creadas y enlazadas a las bibliotecas PAM. Básicamente, los módulos PAM son un conjunto de bibliotecas compartidas que pueden incorporarse a las aplicaciones como método para controlar la autentificación de sus usuarios. Es más, puede cambiarse el método de autentificación (mediante la configuración de los módulos PAM) sin que sea necesario cambiar la aplicación.

Los módulos PAM (las bibliotecas) suelen encontrarse en `/lib/security` o `/lib64/security` (en forma de ficheros objeto cargables dinámicamente). La configuración de PAM está presente en el directorio `/etc/pam.d`, donde aparece un fichero de configuración de PAM por cada aplicación que está usando módulos PAM. Nos encontramos con la configuración de autentificación de aplicaciones y servicios como SSH, de *login* gráfico de X Window System, como *xdm, gdm, kdm, xscreensaver*, etc. o, por ejemplo, del *login* del sistema (la entrada por identificador de usuario y contraseña). En las antiguas versiones de PAM, se utilizaba un archivo de configuración general (típicamente en `/etc/pam.conf`), que era donde se leía la configuración PAM si el directorio `/etc/pam.d` no existía.

Enlace de interés

Para saber más sobre los módulos PAM, se puede ver la *The Linux-PAM System Administrators Guide* en: http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html

La línea típica de estos ficheros (en `/etc/pam.d`) tendría este formato (si se utilizaba el antiguo `/etc/pam.conf` habría que añadir el servicio a que pertenece como primer campo):

```
module-type control-flag module-path arguments
```

donde se especifica:

1) tipo de módulo: si es un módulo que requiere que el usuario se autentifique (`auth`) o es de acceso restringido (`account`); cosas que hay que hacer cuando el usuario entra o sale (`session`); o bien hay que actualizar la contraseña (`password`);

2) *flags* de control: especifican si es necesario (`required`), si es requisito previo (`requisite`), si es suficiente (`sufficient`) o si es opcional (`optional`). Esta es una de las posibles sintaxis, pero para este campo existe una alternativa más actual que trabaja en parejas valor y acción;

3) la ruta (*path*) del módulo;

4) argumentos que se pasan al módulo (dependen de cada módulo).

Debido a que algunos servicios necesitan diversas líneas de configuración comunes, hay posibilidad de operaciones de inclusión de definiciones comunes de otros servicios. Para ello solo hay que añadir una línea con:

```
@include servicio
```

Un pequeño ejemplo del uso de módulos PAM (en una distribución Debian), puede ser su uso en el proceso de *login* (se han listado también las líneas incluidas provenientes de otros servicios):

```
auth      requisite pam_securetty.so
auth      requisite pam_nologin.so
auth      required  pam_env.so
```

```
auth      required  pam_unix.so nullok
account   required  pam_unix.so
session   required  pam_unix.so
session   optional  pam_lastlog.so
session   optional  pam_motd.so
session   optional  pam_mail.so standard noenv
password  required  pam_unix.so nullok obscure min = 4 max = 8 md5
```

Esto especifica los módulos PAM necesarios para controlar la autentificación de usuarios en el *login* (entrada al sistema). Uno de los módulos `pam_unix.so` es el que realmente hace la verificación de la contraseña del usuario (examinando como datos ficheros `passwd`, `shadow`, etc.).

Otros módulos controlan su sesión para ver cuándo ha entrado por última vez, o guardan cuándo entra y sale (para el comando `lastlog`). También hay un módulo que se encarga de verificar si el usuario tiene correo por leer (también hay que autentificarse) y otro que controla que se cambie la contraseña (si está obligado a hacerlo en el primer *login* que haga) y que tenga de 4 a 8 letras. Puede utilizarse md5 para el cifrado de contraseñas.

En este ejemplo podríamos mejorar la seguridad de los usuarios: el `auth` y el `password` permiten contraseñas de longitud nula: es el argumento `nullok` del módulo. Esto permitiría tener usuarios con contraseñas vacías (fuente de posibles ataques). Si quitamos este argumento, ya no permitimos contraseñas vacías en el proceso de *login*. Lo mismo puede hacerse en el fichero de configuración de `passwd` (en este caso, el comando de cambio del contraseña), que también presenta el `nullok`. Otra posible acción es incrementar en ambos ficheros el tamaño máximo de las contraseñas, por ejemplo, con `max = valor`.

3.6. Alteraciones del sistema

Enlace de interés

Tenéis más información, a modo de checklist de seguridad, en: http://www.auscert.org.au/5816

Otro problema puede ser la alteración de comandos o configuraciones básicas del sistema, mediante la introducción de troyanos o puertas trasera en el sistema, por la simple introducción de software que sustituya o modifique ligeramente el comportamiento del software de sistema.

Un caso típico es la posibilidad de forzar al usuario root para que ejecute comandos falsos de sistema; por ejemplo, si el root incluyese el "." en su variable de PATH, esto permitiría la ejecución de comandos desde su directorio actual, lo que habilitaría la colocación de archivos que sustituyesen a comandos del sistema y que serían ejecutados en primer término antes que los de sistema. El mismo proceso puede hacerse con un usuario, aunque por ser más limitados sus permisos, puede no afectar tanto al sistema, cuanto más a la propia seguridad del usuario. Otro caso típico es el de las pantallas de *login* falsas, que

pueden sustituir el típico proceso de `login`, `passwd`, por un programa falso que almacene las contraseñas introducidas.

En caso de estas alteraciones, será imprescindible usar políticas de auditoría de cambios del sistema, ya sea por medio de cálculo de firmas o sumas (gpg o md5), o bien mediante algún software de control como `Tripwire` o `AIDE`. Para los troyanos podemos hacer diferentes tipos de detecciones, o bien utilizar herramientas como `chkrootkit` o `rkhunter`, si estos viniesen de la instalación de algún *rootkit* conocido.

Enlace de interés

Tenéis más información sobre `chkrootkit` y `rkhunter` en:
http://www.chkrootkit.org
http://rkhunter.sourceforge.net

3.7. Recursos limitados, *cgroups* y *chroot*

La gestión común de los procesos existentes de la máquina, ya sean perfectamente válidos como dañinos (intencionadamente o por descuido), nos puede provocar situaciones de agotamiento de los recursos, en forma de CPU disponible, memoria disponible, recursos de red o simplemente sesiones de usuario concurrentes y/o procesos ejecutándose.

Para realizar un control de este problema de agotamiento, podemos introducir límites a algunos de los recursos usados con el comando `ulimit`, aunque la configuración global de límites se mantiene en el fichero `/etc/security/limits.conf`. De hecho, estos límites, que pueden ser impuestos sobre tiempo de CPU, número máximo de procesos, cantidad de memoria y otras similares, son leídos por módulos PAM durante el proceso de *login* de los usuarios, y establecidos por defecto a partir de los límites impuestos a cada usuario.

La elección de los límites también se definirá por el perfil de funcionamiento del sistema: tenemos un gran número de usuarios al que hemos de limitar los recursos, algún servidor costoso en memoria, o por contra solo necesitamos unos pocos procesos para mantener un servidor simple. Si podemos perfilar la tipología de nuestro servidor, podríamos imponer límites razonables que nos permitiesen poner coto a situaciones problemáticas; puede suceder incluso que estas se estén dando por ataques de seguridad orientados al consumo de recursos para dejar inoperantes los servidores o el sistema en conjunto. Normalmente nos concentraremos en controlar, y limitar adecuadamente, los recursos empleados por el root y por los usuarios comunes para intentar evitar posibles ataques o comportamientos defectuosos de aplicaciones corrompidas en su funcionamiento y el uso de los recursos.

No establecer límites y usar el comportamiento por defecto puede hacer caer nuestro sistema con cierta facilidad, en procesos en mal funcionamiento (por mala programación o descuidos de sintaxis), tanto si son ejecutables de sistema como si son scripts; por ejemplo, es bastante fácil crear lo que se denomina una *Fork Bomb*. En sistemas UNIX (Linux incluido), la llamada *fork()* es la que permite crear un proceso derivado (hijo) a partir de la imagen de un proceso

inicial (proceso padre). Posteriormente, el proceso hijo puede especializar su código mediante otras llamadas a sistema y/o comunicarse con el proceso padre. De hecho, esta es la información de parentesco que se observa entre los procesos en *ps* con sus PID y PPID (Parent PID). Si este tipo de llamadas entran en bucle infinito (más bien finito, hasta que se agotan los recursos), ya sea por descuido, mala programación o intencionadamente, lo que conseguiremos en una máquina sin control de los límites es básicamente hacer caer el sistema directamente (o mantenerlo en unos extremos de servicio mínimos).

En el caso de `/etc/security/limits`, cada línea permite unos campos denominados:

```
dominio tipo item valor
```

donde el *dominio* es el login de un usuario o grupo. El tipo será *soft* o *hard* en función de si es un límite con cierta laxitud o por contra no se puede sobrepasar. El item es referido al tipo de límite (*fsize, nofile, cpu, nproc, maxlogins*), que se refieren a: tamaño máximo de fichero, número de ficheros abiertos, máxima CPU en minutos usada, número total de procesos del usuario y número máximo de *logins* del usuario, y finalmente el *valor* correspondiente estable. Normalmente lo más razonable es restringir a los usuarios el número máximo de procesos (para limitar problemas como los *fork bomb*), y a no ser que haya alguna razón explícita, podemos dejar sin uso otros límites. En algunos casos existen alternativas, por ejemplo, en el caso del uso de disco, los sistemas de ficheros podrían aportar control de cuotas de espacio de disco por usuario.

Si, por ejemplo, quisiésemos limitar a 64 los procesos máximos por usuario, la entrada de `limits.conf` sería:

```
* hard nproc 64
```

Podemos comprobar el funcionamiento saliendo y volviendo a entrar en la cuenta de usuario para observar la salida del comando `ulimit -a`.

Los límites por defecto en la mayoría de distribuciones suelen ser bastante holgados. En una Debian, por ejemplo, el caso por defecto (y el valor anterior propuesto, 64, contra el que exhibe por defecto la distribución, 16006 procesos máximos por usuario) sería:

```
# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
```

```
file size               (blocks, -f) unlimited
pending signals                 (-i) 16006
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 16006
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Con una correcta puesta en escena de límites razonables según el perfil de nuestro sistema y de nuestros usuarios, podemos mitigar de forma muy importante los peligros de caída de nuestros servidores.

En particular, para el establecimiento de límites, cabe señalar que también disponemos de una funcionalidad a nivel de kernel, introducida en las últimas ramas y usada mayoritariamente por el sistema *Systemd*, denominada *cgroups*, que permite controlar y solicitar recursos de sistema de diferentes tipos: CPU, memoria o acceso a la red. Podemos examinar la lista de los subsistemas soportados con *cat /proc/cgroups*. Se aporta también una librería (*libcgroup*) y un conjunto de utilidades administrativas (*libcgroup-tools*) que pueden ser usadas para controlar estos grupos, y monitorizar el uso de los recursos y los límites impuestos. *Cgroups* se utiliza a partir de un seudosistema de ficheros, *cgroupfs*, en el que se sitúan los recursos en grupos de control, a los cuales se pueden asociar procesos que los estén usando y limitar su uso. Por ejemplo, podemos averiguar qué grupos encontramos y dónde se encuentran montados, con tipo de *filesystem cgroupsfs*, dónde podemos observar los directorios y la información de control de cada grupo de recursos mantenida. También podemos observar la configuración en `/etc/cgconfig.conf` (aunque algunas de ellas las llevará a cabo *Systemd* en tiempo de arranque):

```
# cat /proc/cgroups
#subsys_name hierarchy num_cgroups enabled
cpuset 2 1 1
cpu 3 1 1
cpuacct 3 1 1
memory 4 1 1
devices 5 1 1
freezer 6 1 1
net_cls 7 1 1
blkio 8 1 1
perf_event 9 1 1
```

```
net_prio 7 1 1
hugetlb 10 1 1

# lsusbsys -am
cpuset  /sys/fs/cgroup/cpuset
cpu,cpuacct /sys/fs/cgroup/cpu,cpuacct
memory  /sys/fs/cgroup/memory
devices  /sys/fs/cgroup/devices
freezer  /sys/fs/cgroup/freezer
net_cls,net_prio /sys/fs/cgroup/net_cls,net_prio
blkio  /sys/fs/cgroup/blkio
perf_event  /sys/fs/cgroup/perf_event
hugetlb  /sys/fs/cgroup/hugetlb
```

Pongamos un ejemplo de control para una aplicación costosa de recursos que podríamos controlar con *cgroups* especificada en `/etc/cgconfig.conf`. Sea APP el ejecutable de la aplicación y $USER el usuario que la ejecutará:

```
# Prevenir que APP obtenga toda la memoria
#     y CPUs disponibles
group APP {
    perm {
        admin {
            uid = $USER;
        }
        task {
            uid = $USER;
        }
    }
# 4 cores usados
#(suponemos un sistema con más disponibles)
    cpuset {
        cpuset.mems="0";
        cpuset.cpus="0-3";
    }
    memory {
# 4 GiB limit
        memory.limit_in_bytes = 4294967296;
    }
}
```

Este *cgroup* permitirá a APP usar los *cores* 0 a 3 (4 en total) y limitar su uso de memoria a 4 GB, y la pondríamos en ejecución con el límite de recursos con (no olvidar substituir APP y $USER por el nombre de la aplicación y usuario):

cgroups info

consultar la documentación del kernel en: https://www.kernel.org /doc/Documentation/cgroups/

```
$ cgexec -g memory,cpuset:APP /opt/APPdir/APP -opciones
```

Respecto los *cgroups*, recientemente, en la versión de kernel 3.16 (Agosto 2014), se ha establecido un nuevo método de acceso que permite disponerlos como una única jerarquía.

Enlace de interés

Novedades cgroups en Kernel 3.16: http://lwn.net/Articles/601840

Examinamos una tercera posibilidad para el control de límites de recursos entregados a una determinada aplicación, servicio o usuario. Es lo que se conoce como creación de ambientes *chroot*, cuyo concepto es la creación de una especie de ambiente prisión (conocido como *jail*), donde ponemos unos límites muy claros a lo que puede realizarse sin afectar al resto del sistema.

Se basa en una llamada al sistema *chroot()* que básicamente redefine lo que el proceso hijo asociado percibe como directorio raíz (/), que, para el ambiente (o jaula) creado, puede asociarse en una determinada posición del sistema de ficheros, percibida por el proceso como raíz (/), y buscará a partir de allí todas las configuraciones, componentes del sistema, librerías, etc. a partir de esa falsa raíz.

Ya que se cambia la estructura del sistema de ficheros por una nueva, el proceso solamente funcionará si dispone de todos sus datos en la zona nueva. Por tanto, deberemos replicar todos los datos, ficheros y comandos necesarios.

Conviene recalcar que un ambiente de *chroot* protege contra el acceso a ficheros fuera de la zona (de jaula), pero no protege contra otros límites, como la utilización del sistema, accesos a memoria u otros similares.

Una vez escojamos el proceso que queremos restringir, podemos copiar sus ficheros de datos en la estructura adecuada de directorios y podemos examinar los ejecutables del proceso con *ldd ejecutable*, lo cual nos aporta información de las librerías necesarias para su ejecución, que pueden ser copiadas y situadas en los puntos de la estructura de ficheros adecuados.

Una vez dispongamos de la estructura simplemente con:

```
chroot /nueva/raiz ejecutable
```

le pasamos la ejecución a su nueva jaula (dentro de la raíz obtenida resolviendo el camino */nueva/raiz*).

3.8. Protección de ejecutables mediante Hardening-Wrapper

Con este nombre se conocen una serie de técnicas usadas para mitigar diversos problemas de seguridad relacionados con ataques llevados a cabo por ejecutables. En concreto, se trata de los problemas de desbordamiento de *stack* (pila), *heap* y protecciones contra el acceso a zonas de memoria de datos y código de los ejecutables.

Se utilizan estas técnicas en asociación con el compilador (por ejemplo, GNU *gcc*), mediante una serie de parámetros y opciones (*flags*) pasadas en tiempo de

compilación, ya sea para la compilación de aplicaciones de usuario, clientes de servicios o la parte *server* de servicios (*daemons*), que a partir de sus fuentes de código y el proceso de compilación son protegidas contra varias técnicas de ataque a los ejecutables.

En Debian podemos instalar las utilidades y ficheros complementarios para el proceso de compilación, además de disponer de la utilidad *hardening-check*, que nos permite comprobar las protecciones de que dispone un determinado comando, aplicación o *daemon*. Pongamos el ejemplo de la instalación de los paquetes en Debian y la comprobación de un *daemon* (*sshd* en este caso):

```
# apt-get install hardening-wrapper hardening-includes

# hardening-check /usr/sbin/sshd
/usr/sbin/sshd:
 Position Independent Executable: yes
 Stack protected: yes
 Fortify Source functions: yes (some protected functions found)
 Read-only relocations: yes
 Immediate binding: yes
```

Entre las comprobaciones que se realizan al ejecutable (y las relaciones con parámetros [*flags*] de compilación en GNU gcc, los que se mencionan pueden ser pasados por línea de comandos a gcc, o bien en el correspondiente fichero de Makefile que realice el proceso de compilación):

- *Position Independent Executable*, que permite al programa que su posición en memoria del sistema sea movible, sin disponer necesariamente de posición fija y incluso de forma aleatoria entre diferentes ejecuciones. En gcc conseguimos esta funcionalidad con los *flags* de compilación *-pie -fPIE* aplicados a todos los componentes del ejecutable. El kernel que maneja los procesos debe tener soporte de lo que se conoce como ASLR (*Address Space Layout Randomization*), que permite protección contra ataques de desbordamiento de *buffer* (*buffer overflows*). Para prevenir que un atacante pueda atacar a una determinada función en memoria, ASLR reordena de forma aleatoria áreas clave del proceso (la imagen del ejecutable en memoria), para cambiar posiciones de la base de la parte ejecutable, así como las posiciones del *stack*, el *heap* o las librerías dinámicas dentro del espacio de direcciones del proceso. Así se consigue que sea difícil predecir las posiciones de memoria cuando se intentan ataques con ese fin. Puede averiguarse si existe soporte en el kernel para ASLR examinando */proc/sys/kernel/randomize_va_space*, que debería dar un valor de 1 o 2 para opciones con ASLR activadas y 0 cuando están desactivadas.

- *Stack Protected*. En este caso se han ofrecido métodos adicionales para proteger los desbordamientos de *stack*, por ejemplo, mediante la compilación gcc con la opción (*-fstack-protector*).

- *Fortify Source functions*. En este caso, cuando durante el proceso de compilación (en la parte de link) los ejecutables son enlazados con las funciones de la librería estándar C (funciones de *glibc* en un sistema GNU/Linux), hay una cierta sustitución de algunas funciones por otras. Algunas funciones de *glibc* son conocidas por no ser seguras en varios aspectos, aunque tienen alternativas seguras en la misma librería, ya que tienen tests de seguridad incorporados, por ejemplo, frente a *buffer overflows*. En el caso de gcc, en el proceso de compilación se realiza con *-D_FORTIFY_SOURCE=2 -O1* (o superior nivel de optimización).

- *Read-only relocations*. Esto permite al linker, cuando detecta zonas de memoria que puede deducir que serán de solo acceso de lectura durante el proceso de compilación, marcarlas como tales antes de empezar la ejecución. De esta manera, se reduce la posibilidad de que un atacante pueda usar esas áreas para inyectar código o que aproveche *exploits* de corrupción de memoria. En compilación, pueden pasarse a gcc los *flags -W1,-z,relro* para esta opción.

- *Immediate binding*. Es una posible combinación con la anterior, que permite al linker, previamente a la ejecución, resolver movimientos de datos y ciertos enlaces a memoria que eviten que estos movimientos (*relocates*) se realicen *a posteriori*. Así, en combinación con la opción anterior, se reducen de forma sensible las zonas de memoria disponibles contra ataques de corrupción de memoria.

En el caso de gcc, la mayoría de estas técnicas pueden activarse por defecto antes de la compilación activando la variable siguiente (con valor igual a 1 para activar o 0 para desactivar):

```
$ export DEB_BUILD_HARDENING=1
$ gcc ....
```

Estas técnicas aportan una buena base para proteger los ejecutables del sistema contra ataques a su forma de proceso mientras están en ejecución, así como proteger sus datos y su código de cambios dinámicos que un atacante podría introducir.

Hardening-Wrapper

Se recomienda para mas información consultar las páginas man de hardening-wrapper y hardening-check. En el caso de Debian es recomendable: https://wiki.debian.org/Hardening

No obstante, cabe señalar algunos defectos, como que estas técnicas varían constantemente, que ciertos problemas en la implementación de las técnicas de protección pueden ser explotados, o que en determinados casos estas protecciones pueden dar falsos positivos.

4. SELinux

La seguridad tradicional dentro del sistema se ha basado en las técnicas DAC (*discretionary access control*, control de acceso discrecional), donde normalmente cada programa dispone de control completo sobre los accesos a sus recursos. Si un determinado programa (o el usuario lo permite) decide hacer un acceso incorrecto (por ejemplo, dejando datos confidenciales en abierto, ya sea por desconocimiento o por mal funcionamiento), no hay nada que se lo impida. Así en DAC, un usuario tiene completo control sobre los objetos que le pertenecen y los programas que ejecuta. El programa ejecutado dispondrá de los mismos permisos que el usuario que lo está ejecutando. Así, la seguridad del sistema dependerá de las aplicaciones que se estén ejecutando y de las vulnerabilidades que estas pudiesen tener, o del software malicioso que incluyesen, y en especial afectará a los objetos (otros programas, ficheros, o recursos) a los que el usuario tenga acceso. En el caso del usuario root, esto comprometería la seguridad global del sistema.

Por otro lado, las técnicas MAC (*mandatory access control*, control de acceso obligatorio) desarrollan políticas de seguridad (definidas por el administrador) en las que el sistema tiene control completo sobre los derechos de acceso que se conceden sobre cada recurso. Por ejemplo, con permisos (de tipo UNIX) podemos dar acceso a ficheros, pero mediante políticas MAC tenemos control adicional para determinar a qué ficheros explícitamente se permite acceso por parte de un proceso, y qué nivel de acceso queremos conceder. Se fijan contextos, en los cuales se indican en qué situaciones un objeto puede acceder a otro objeto.

SELinux [Nsab] es un componente de tipo MAC reciente incluido en la rama 2.6.x del núcleo, que las distribuciones van incluyendo progresivamente: Fedora/Red Hat lo traen habilitado por defecto (aunque es posible cambiarlo durante la instalación), y en Debian es un componente opcional. SELinux implementa políticas de seguridad de tipo MAC y permite disponer de un acceso de permisos más fino que los tradicionales permisos de archivo UNIX. Por ejemplo, el administrador podría permitir que se añadiesen datos a un archivo de registro, pero no reescribirlo o truncarlo (técnicas utilizadas habitualmente por atacantes para borrar las pistas de sus accesos). En otro ejemplo, podría permitirse que programas de red se enlazaran a un puerto (o puertos) que necesitaran y, sin embargo, denegar el acceso a otros puertos (esta podría ser una técnica que permitiese controlar y limitar el acceso de algunos troyanos o puertas traseras). SELinux fue desarrollado por la agencia NSA de Estados Unidos, con aportaciones de diversas compañías para sistemas UNIX

Enlaces de interés

En los siguientes enlaces tenéis algunos recursos sobre SELinux: http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide, http://www.nsa.gov/research/selinux/index.shtml, http://fedoraproject.org/wiki/SELinux, http://selinux.sourceforge.net/.

y libres, como Linux y BSD. Se liberó en el año 2000 y desde entonces se ha ido integrando en diferentes distribuciones GNU/Linux.

Disponemos en SELinux de un modelo de dominio-tipo, donde cada proceso corre en un denominado contexto de seguridad y cualquier recurso (fichero, directorio, *socket*, etc.) tiene un tipo asociado con él. Hay un conjunto de reglas que indican qué acciones pueden efectuarse en cada contexto sobre cada tipo. Una ventaja de este modelo contexto-tipo es que las políticas que se definan se pueden analizar (existen herramientas) para determinar qué flujos de información están permitidos; por ejemplo, para detectar posibles vías de ataque o si la política es lo suficientemente completa para cubrir todos los accesos posibles.

Se dispone de lo que se conoce como la base de datos de políticas de SELinux (*SELinux policy database*) que controla todos los aspectos de SELinux. Determina qué contextos puede utilizar cada programa para ejecutarse y especifica a qué tipos de cada contexto puede acceder.

En SELinux cada proceso del sistema tiene un contexto compuesto de tres partes: una identidad, un rol y un dominio. La identidad es el nombre de la cuenta del usuario, o bien `system_u` para procesos de sistema o `user_u` si el usuario no dispone de políticas definidas. El rol determina qué contextos están asociados. Por ejemplo `user_r` no tiene permitido tener el contexto `sysadm_t` (dominio principal para el administrador de sistema). Así un `user_r` con identidad `user_u` no puede obtener de ninguna manera un contexto `sysadm_t`. Un contexto de seguridad se especifica siempre por esta terna de valores, como por ejemplo:

```
root:sysadm_r:sysadm_t
```

que es el contexto para el administrador del sistema, define su identidad, su rol y su contexto de seguridad.

Por ejemplo, en una máquina con SELinux activado (una Fedora en este caso) podemos ver con la opción `-Z` del `ps` los contextos asociados a los procesos:

```
# ps -ax -Z

LABEL                        PID   TTY   STAT   TIME   COMMAND
system_u:system_r:init_t     1     ?     Ss     0:00   init
system_u:system_r:kernel_t   2     ?     S      0:00   [migration/0]
system_u:system_r:kernel_t   3     ?     S      0:00   [ksoftirqd/0]
system_u:system_r:kernel_t   4     ?     S      0:00   [watchdog/0]
system_u:system_r:kernel_t   5     ?     S      0:00   [migration/1]
```

```
system_u:system_r:kernel_t      6     ?       SN   0:00   [ksoftirqd/1]
system_u:system_r:kernel_t      7     ?       S    0:00   [watchdog/1]
system_u:system_r:syslogd_t     2564  ?       Ss   0:00   syslogd -m 0
system_u:system_r:klogd_t       2567  ?       Ss   0:00   klogd -x
system_u:system_r:irqbalance_t  2579  ?       Ss   0:00   irqbalance
system_u:system_r:portmap_t     2608  ?       Ss   0:00   portmap
system_u:system_r:rpcd_t        2629  ?       Ss   0:00   rpc.statd
user_u:system_r:unconfined_t    4812  ?       Ss   0:00   /usr/libexec/gconfd-2 5
user_u:system_r:unconfined_t    4858  ?       Sl   0:00   gnome-terminal
user_u:system_r:unconfined_t    4861  ?       S    0:00   gnome-pty-helper
user_u:system_r:unconfined_t    4862  pts/0   Ss   0:00   bash
user_u:system_r:unconfined_t    4920  pts/0   S    0:01   gedit
system_u:system_r:rpcd_t        4984  ?       Ss   0:00   rpc.idmapd
system_u:system_r:gpm_t         5029  ?       Ss   0:00   gpm -m /dev/input/mice -t exps2
user_u:system_r:unconfined_t    5184  pts/0   R+   0:00   ps ax -Z
user_u:system_r:unconfined_t    5185  pts/0   D+   0:00   Bash
```

y con `ls` con la opcion `-Z` podemos ver los contextos asociados a ficheros y directorios:

```
# ls -Z
drwxr-xr-x josep josep user_u:object_r:user_home_t Desktop
drwxrwxr-x josep josep user_u:object_r:user_home_t proves
-rw-r--r-- josep josep user_u:object_r:user_home_t yum.conf
```

y desde la consola podemos conocer nuestro contexto actual con:

```
$ id -Z
user_u:system_r:unconfined_t
```

Respecto al modo de funcionamiento, SELinux presenta dos modos denominados: *permissive* y *enforcing*. En *permissive* se permiten los accesos no autorizados, pero son auditados en los registros correspondientes (normalmente directamente sobre `/var/log/messages` o bien, dependiendo de la distribución, con el uso de `audit` en `/var/log/audit/audit.log`). En *enforcing* no se permite ningún tipo de acceso que no permitan las políticas definidas. También puede desactivarse SELinux a través de su fichero de configuración (habitualmente en `/etc/selinux/config`), colocando `SELINUX=disabled`.

Hay que tener cuidado con la activación y desactivación de SELinux, en especial con el etiquetado de contextos en los ficheros, ya que en periodos en que se active o desactive pueden perderse etiquetas (puesto que el sistema no estará activo) o simplemente no ser etiquetados. Asimismo, la realización de

copias de seguridad del sistema de ficheros tiene que tener en cuenta que se conserven las etiquetas de SELinux.

Otro posible problema a tener en cuenta es el gran número de reglas de política de seguridad que pueden llegar a existir y que pueden provocar limitaciones en el control de los servicios. Ante un tipo determinado de malfuncionamiento, cabe primero determinar que precisamente no sea SELinux el que está impidiendo el funcionamiento, por una limitación demasiado estricta de seguridad (véase el subapartado 4.2 de crítica a SELinux) o a causa de opciones que no esperábamos tener activadas (pueden requerir cambiar la configuración de los booleanos, como veremos).

Con respecto a la política usada, SELinux soporta dos tipos diferentes: *targeted* y *strict*. En *targeted* la mayoría de procesos operan sin restricciones, y solo servicios (ciertos *daemons*) específicos son puestos en diferentes contextos de seguridad que son confinados a la política de seguridad. En *strict* todos los procesos son asignados a contextos de seguridad y confinados a políticas definidas, de manera que cualquier acción es controlada por las políticas definidas. En principio estos son los dos tipos de políticas definidas generalmente, pero la especificación está abierta a incluir más.

Un caso especial de política es la MLS (*multilevel security*, seguridad multinivel), que es una política multinivel de tipo *strict*. La idea es definir, dentro de la misma política, diferentes niveles de seguridad y los contextos de seguridad tienen asociado un campo adicional de nivel de acceso. Este tipo de política de seguridad (como MLS) suele ser utilizada en organizaciones gubernamentales y militares, donde hay organizaciones jerárquicas con diferentes niveles de información privilegiada, niveles de acceso general y capacidades de acción diferentes en cada nivel. Para obtener algunas certificaciones de seguridad (concedidas o avaladas por ciertas organizaciones) para los sistemas operativos, se necesita disponer de políticas de seguridad de este tipo.

Se puede definir qué tipo de política se usará en `/etc/selinux/config`, variable `SELINUXTYPE`. La política correspondiente, y su configuración, normalmente estará instalada en los directorios `/etc/selinux/SELINUXTYPE/`. Por ejemplo, suele encontrarse en el subdirectorio `policy` el fichero binario de la política compilada (que es el que se carga en el núcleo, al inicializar SELinux).

4.1. Arquitectura

La arquitectura de SELinux está compuesta de los siguientes componentes:

- Código a nivel de núcleo
- La biblioteca compartida de SELinux
- La política de seguridad (la base de datos)
- Herramientas

Examinemos algunas consideraciones con respecto a cada componente:

- El código de núcleo monitoriza la actividad del sistema, y asegura que las operaciones solicitadas estén autorizadas bajo la configuración de políticas de seguridad de SELinux actual; así, no permite las operaciones no autorizadas y normalmente genera entradas en el registro de las operaciones denegadas. El código actualmente se encuentra integrado en los núcleos 2.6.x, mientras que en los anteriores se ofrece como serie de parches.

- La mayoría de utilidades y componentes SELinux no directamente relacionados con el núcleo hacen uso de la biblioteca compartida, llamada `libselinux1.so`, que facilita una API para interaccionar con SELinux.

- La política de seguridad es la que está integrada en la base de datos de reglas de SELinux. Cuando el sistema arranca (con SELinux activado), carga el fichero binario de política, que habitualmente reside en la siguiente ruta: `/etc/security/selinux` (aunque puede variar según la distribución).

  El fichero binario de políticas se crea a partir de una compilación (vía `make`) de los ficheros fuente de políticas y algunos ficheros de configuración.

  Algunas distribuciones (como Fedora) no instalan las fuentes por defecto, que suelen encontrarse en `/etc/security/selinux/src/policy` o en `/etc/selinux`. Normalmente, estas fuentes consisten en varios grupos de información:

  – Los ficheros relacionados con la compilación, `makefile` y *scripts* asociados.

  – Ficheros de la configuración inicial, usuarios y roles asociados.

  – Ficheros de `Type-enforcement`, que contienen la mayoría de las sentencias del lenguaje de políticas asociado a un contexto particular. Hay que tener en cuenta que estos ficheros son enormes, típicamente decenas de miles de líneas. Con lo cual puede aparecer el problema de encontrar fallos o definir cambios en las políticas.

  – Ficheros que sirven para etiquetar los contextos de los ficheros y directorios durante la carga o en determinados momentos.

- Herramientas: Incluyen comandos usados para administrar y usar SELinux, versiones modificadas de comandos estándar Linux y herramientas para el análisis de políticas y el desarrollo.

Veamos, de este último punto, las herramientas típicas de que solemos disponer, algunos de los comandos principales:

Nombre	Utilización
`chcon`	Etiqueta un fichero especifico, o un conjunto de ficheros con un contexto especifico.
`checkpolicy`	Realiza diversas acciones relacionadas con las políticas, incluyendo la compilación de las políticas a binario; típicamente se invoca desde las operaciones de `makefile`.
`getenforce`	Genera mensaje con el modo actual de SELinux (*permissive* o *enforcing*). O bien desactivado si es el caso.
`getsebool`	Obtiene la lista de booleanos, o sea la lista de opciones on/off para cada contexto asociado a un servicio, u opción general del sistema.
`newrole`	Permite a un usuario la transición de un rol a otro.
`runn_init`	Utilizado para activar un servicio (start, stop), asegurándose de que se realiza en el mismo contexto que cuando se arranca automáticamente (con `init`).
`setenforce`	Cambia de modo a SELinux: `0` *permissive*, `1` *enforcing*.
`setfiles`	Etiqueta directorios y subdirectorios con los contextos adecuados; es típicamente utilizado en la configuración inicial de SELinux.
`setstatus`	Obtiene el estado del sistema con SELinux.

Por otra parte, se modifican algunos comandos GNU/Linux con funcionalidades u opciones nuevas, como por ejemplo `cp`, `mv`, `install`, `ls`, `ps`, etc. por ejemplo modificando en los ficheros la etiqueta para asociar el contexto de seguridad (como pudimos comprobar con la opción `-Z` de `ls`). `Id` se modifica para incluir la opción de mostrar el contexto actual del usuario. Y en `ps` vimos que incluía una opción para visualizar los contextos de seguridad actuales de los procesos.

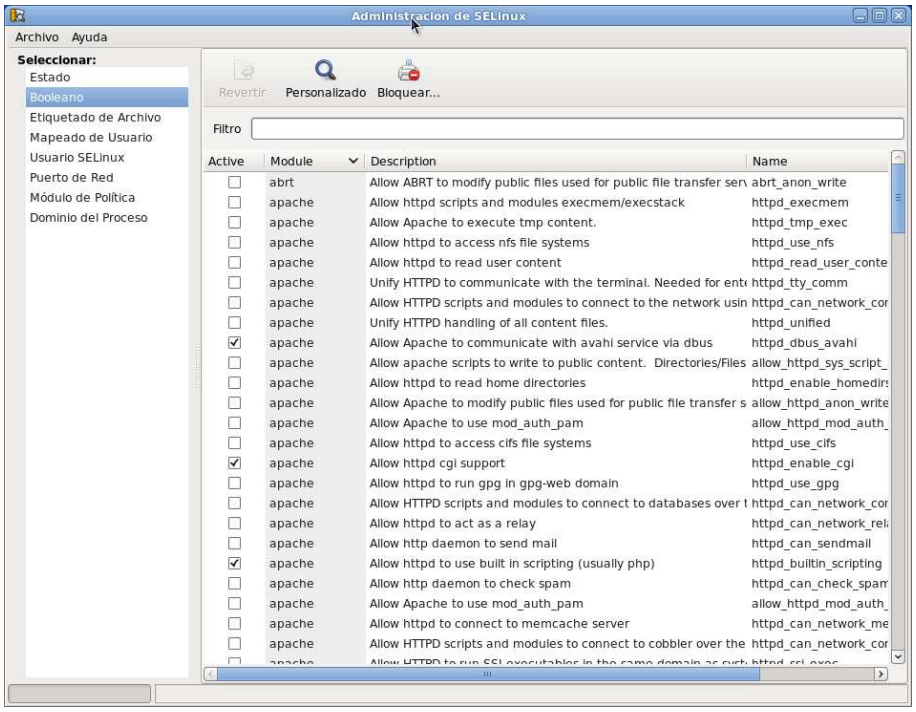
Además, se modifican algunos otros programas comunes para soportar SELinux como:

- cron: Modificado para incluir los contextos para los trabajos en ejecución por cron.
- login: Modificado para que coloque el contexto de seguridad inicial para el usuario cuando entra en el sistema.
- logrotate: Modificado para preservar el contexto de los logs cuando estén recopilados.
- pam: Modificado para colocar el contexto inicial del usuario y para usar la API SELinux y obtener acceso privilegiado a la información de contraseñas.
- ssh: Modificado para colocar el contexto inicial del usuario cuando entra en el sistema.
- Varios programas adicionales que modifican `/etc/passwd` o `/etc/shadow`.

En algunas distribuciones también se incluyen herramientas para la gestión de SELinux, como las `setools(-gui)`, que llevan varias herramientas de gestión y análisis de las políticas. Así, como alguna herramienta específica para controlar los contextos asociados a los diferentes servicios soportados por SE-

Linux en la distribución, la herramienta `system-config-securitylevel` en Fedora dispone de un apartado para la configuración de SELinux como podemos observar en la figura 1.

Figura 1. Interfaz en Fedora para configuración SELinux (examinando booleanos)



En la figura se observa la configuración de booleanos para diversos servicios y opciones genéricas, entre ellas el servidor web. También podemos obtener esta lista con `getsebool -a`, y con los comandos `setsebool`/`togglesebool` podemos activar/desactivar las opciones.

En Fedora, por ejemplo, encontramos soporte de booleanos para (entre otros): cron, ftp, httpd (apache), dns, grub, lilo, nfs, nis, cups, pam, ppd, samba, protecciones contra accesos incorrectos a la memoria de los procesos, etc.

La configuración de booleanos permite la personalización de la política SELinux en tiempo de ejecución. Los booleanos son utilizados en SELinux como valores condicionales de las reglas de la política usada, lo que permite modificaciones de la política sin necesidad de cargar una nueva política.

4.2. Crítica

Algunos administradores y expertos en seguridad han criticado especialmente a SELinux por ser demasiado complejo para configurar y administrar. Se argumenta que, por su complejidad intrínseca, incluso usuarios experimentados pueden cometer errores, lo que dejaría la configuración de SELinux no segura o inservible y el sistema, vulnerable. Esto es discutible hasta cierto punto, ya que aunque tuviésemos SELinux mal configurado, aún seguirían activos los

permisos UNIX y otras herramientas y SELinux no permitiría una operación que los permisos originales ya no permitieran. De hecho, podemos verlo como una etapa más de seguridad más estricta.

También hay factores de rendimiento que pueden verse afectados, debido al enorme tamaño de las políticas; pueden disminuir las prestaciones a causa del gran uso de memoria y al tiempo inicial de carga y, en algún caso, al procesamiento de las reglas. Cabe pensar que estamos hablando prácticamente de un sistema de más de 10.000 reglas en la política. Y aún puede ser un número mayor si escogemos una política de tipo *strict*, con la necesidad de especificar absolutamente todas las opciones a controlar. Normalmente, el procesamiento por política en formato binario y el uso de booleanos para inhabilitar reglas, permite un uso más eficiente del sistema.

Otro aspecto que suele molestar a los administradores es el problema adicional de determinar, ante un determinado malfuncionamiento, cuál es el origen o causa inicial. Es habitual que finalmente nos encontremos que el problema provenía de una configuración excesivamente restrictiva (quizás por desconocimiento del administrador) de SELinux para un determinado servicio. En última instancia, cabe señalar el amplio soporte para la seguridad que SELinux proporciona y que, como administradores, tenemos que ser conscientes de las capacidades y peligros de cualquier nueva técnica que utilicemos.

4.3. Algunas alternativas

Ante la complejidad de SELinux y algunas deficiencias del modelo (complejidad de políticas o mala configuración), se han producido algunas alternativas (o en algunos casos complementos) a la utilización de SELinux. Hemos de destacar en particular Grsecurity, una serie de parches para el núcleo Linux que intentan mejorar algunas deficiencias. También hay que destacar AppArmor, que algunas distribuciones están incluyendo como alternativa a SELinux.

grsecurity

http://grsecurity.net/.

Grsecurity, por contra, se ofrece como una serie de parches de núcleo publicados posteriormente al establecimiento de una versión estable del núcleo, y se pueden obtener de su página web para ciertas versiones de núcleo, además de paquetes ya preparados para algunas distribuciones.

Grsecurity incluye diferentes prestaciones, como un modelo de acceso a recursos mediante gestión de roles (tanto locales como remotos), gestionados mediante una herramienta denominada *gradm*; auditación de las operaciones internas al núcleo, y prevención de código arbitrario en el mismo (de hecho hace uso de un parche de núcleo denominado PaX, que ofrece algunas de estas protecciones). Ello evita posibles errores de tipo desbordamiento de *buffer*, *heap* o *stack* (técnicas aprovechadas para *exploits* del kernel). También incluye notificaciones de alertas de seguridad basadas en IP del usuario. Los parches

ofrecidos para el núcleo se aplican como parches al código fuente del núcleo y permiten activar, durante la configuración del núcleo previa a su compilación, diferentes opciones de protección del código del núcleo relacionadas con las prestaciones mencionadas. Además, una vez el núcleo está configurado para su soporte, la herramienta *gradm* permite gestionar los roles y las tareas o protecciones asociadas a cada rol y a los usuarios que incluyamos en cada uno de ellos.

Grsecurity es un buen complemento o sustituto de algunos puntos débiles de SELinux, ya que ofrece una interfaz más simple de usar e incluye algunas prestaciones de seguridad adicionales.

Enlace de interés

Sobre la administración de Grsecurity puede consultarse: http://en.wikibooks.org/wiki/Grsecurity.

Otra alternativa, que está surgiendo en algunas distribuciones como Ubuntu y SuSe, es AppArmor, que permite al administrador asociar a cada programa un perfil de seguridad que restrinja las capacidades del programa. Se puede establecer un perfil individual de permisos o bien puede usarse un modo de autoaprendizaje basado en violaciones de perfil registradas y habilitadas o no por los usuarios.

AppArmor en Ubuntu

Una alternativa a SELinux usada en Ubuntu es AppArmor: https://help.ubuntu.com/14.04/serverguide/apparmor.html.

Básicamente, se compone de un módulo de núcleo para la gestión de perfiles de aplicaciones, basado, al igual que SELinux, en LSM (*Linux Security Modules*), juntamente con diferentes utilidades. Mediante estas utilidades podemos establecer permisos de los programas para el acceso al sistema de ficheros y a los dispositivos. La mayor diferencia con SELinux es que las políticas de seguridad no se aplican con etiquetas a los ficheros, sino respecto a políticas aplicables a los *pathnames*. AppArmor se integra en el núcleo como módulo LSM disponible a partir de las versiones 2.6.36 del núcleo.

De hecho, en las ramas actuales del kernel se encuentran el framework denominado LSM (*Linux Security Modules*), que soporta una variedad de módulos de seguridad para evitar el favoritismo hacia alguno de los mencionados. Ahora mismo, en LSM, para las implementaciones de seguridad de tipo MAC (*mandatory access control*), están aceptados como parte del kernel, desde las ramas 2.6, los modelos de AppArmor, SELinux, Smack y TOMOYO (además de Yama, que podría incorporarse en el futuro).

Sobre Módulos LSM en el Kernel

Podéis leer comentarios de los diferentes modelos de LSM en: Overview of LSM.

5. Seguridad en red

5.1. Cliente de servicios

Clientes y servicios inseguros

Como clientes de servicios,tendremos que evitar el uso de servicios inseguros.

Como clientes de servicios básicamente tenemos que asegurar que no ponemos en peligro a nuestros usuarios (o se pongan en peligro ellos solos) por usar servicios inseguros. Hay que evitar el uso de servicios que no utilicen cifrado de datos y contraseñas (ftp, telnet, correo no seguro) y utilizar en este caso técnicas de conexión cifrada, como SSH y SSL/TSL. Así mismo, debemos asegurarnos de que encontremos en el sistema los programas clientes, y las librerías que estos usan (como las mencionadas previamente) en sus versiones más actualizadas. En algunos momentos críticos, delante de posibles vulnerabilidades de tipo *0day*, puede ser incluso necesario actualizar servicios, clientes o librerías, independientemente de la distribución, para asegurarnos que nuestro sistema no es vulnerable, mientras no se disponga de solución a las vulnerabilidades detectadas.

Caso HeartBleed

Un caso de vulnerabilidad crítica que afectó a las librerías openSSL durante 2014: http://heartbleed.com/

Otro punto importante se refiere a la posible sustitución de servidores por otros falsos, o bien a técnicas de secuestro de sesiones. En estos casos tenemos que disponer de mecanismos de autentificación robustos, que nos permitan verificar la autenticidad de los servidores (por ejemplo, SSH y SSL/TLS tienen algunos de estos mecanismos). Y también tendremos que verificar la red en busca de intrusos, potenciales peligros, intentos de intrusión fallidos o intentos de sustitución de servidores, además de políticas correctas de filtrado de paquetes mediante cortafuegos (*firewalls*), que nos permitan obtener salida a nuestros paquetes de datos válidos y usar los servidores adecuados, controlando los paquetes de entrada que recibamos como respuestas.

5.2. Servidor: inetd y xinetd

La configuración de los servicios de red [Mou02], tal como hemos visto, se realiza desde varios lugares [Ray01, Hat08, Pen]:

Nota Systemd

En nuevas distribuciones, el control de servicios se está migrando, en su mayoría, desde inetd y xinetd al concepto de targets basados en Systemd. Observar: http://0pointer.de/blog/projects/security.html

- En `/etc/inetd.conf` o el equivalente directorio `/etc/xinetd.d`: estos ficheros de configuración de servicios están asociados a lo que se conoce como *superservidores*, ya que controlan varios servicios hijos y sus condiciones de arranque. El servicio inetd es utilizado en Debian (aunque cada vez menos) y xinetd en Fedora (en Debian, xinetd puede ser instalado opcionalmente en sustitución de inetd).

- Servidores iniciados en arranque: según el *runlevel* tendremos una serie de servicios arrancados. El arranque se originará en el directorio asociado al

runlevel; por ejemplo, en Debian el runlevel por defecto es el 2, los servicios arrancados serán arrancados desde el directorio `/etc/rc2.d`, seguramente con enlaces a los *scripts* contenidos en `/etc/init.d`, en los cuales se ejecutará con los parámetros `start`, `stop`, `restart`, según corresponda.

- Otros servicios de tipo RPC: asociados a llamadas remotas entre máquinas se usan, por ejemplo, en NIS y NFS. Con el comando `rpcinfo -p` puede examinarse cuáles hay.

Otros ficheros de apoyo (con información útil) son: `/etc/services`, que está formado por una lista de servicios locales o de red conocidos, junto con el nombre del protocolo (tcp, udp u otros) que se utiliza en el servicio y el puerto que utiliza; `/etc/protocols` es una lista de protocolos conocidos, y `/etc/rpc` es una lista de posibles servidores RPC, junto con los puertos (rpc) usados. Estos ficheros vienen con la distribución y son una especie de base de datos que utilizan los comandos y herramientas de red para determinar los nombres de servicios, protocolos o rpc y sus puertos asociados. Cabe destacar que son ficheros más o menos históricos, que no tienen porqué contener todas las definiciones de protocolos y servicios. Pueden asimismo buscarse diferentes listas en Internet de puertos conocidos.

Una de las primeras acciones a realizar por el administrador será deshabilitar todos aquellos servicios que no esté utilizando o no tenga previsto utilizar, en este sentido habrá que documentarse sobre la utilización de los servicios [Mou02], y qué software puede necesitarlos [Neu].

En el caso de `/etc/inetd.conf`, solo tenemos que comentar la línea del servicio que hay que deshabilitar, colocando un "`#`" como primer carácter en la línea.

En el otro modelo de servicios, utilizado por defecto en Fedora (y opcional en Debian), el xinetd, la configuración reside en el fichero `/etc/xinetd.conf`, donde se configuran algunos valores por defecto de control de registros y, después, la configuración de cada servicio hijo se hace a través de un fichero dentro del directorio `/etc/xinetd.d`. En cada fichero se define la información del servicio, equivalente a la que aparece en el `inetd.conf`; en este caso, para desactivar un servicio solo hay que poner una línea `disable = yes` dentro del fichero del servicio. xinetd tiene una configuración más flexible que inetd, al separar la configuración de los diferentes servicios en diferentes archivos, y tiene bastantes opciones para limitar las conexiones a un servicio, en su número o su capacidad, todo lo cual permite un mejor control del servicio y, si lo configuramos adecuadamente, podemos evitar algunos de los ataques por denegación de servicio (DoS o DDoS).

Respecto al tratamiento de los servicios de los *runlevel* desde comandos de la distribución, ya mencionamos varias herramientas en la unidad de adminis-

tración local, que permitían habilitar o deshabilitar servicios. También existen herramientas gráficas como `ksysv` de KDE o el `system-config-services` y `ntsysv` en Fedora (en Debian son recomendables `sysv-rc-conf`, `rcconf` o `bum`). Y a un nivel inferior, podemos ir al nivel de *runlevel* que queramos (`/etc/rcx.d`) y desactivar los servicios que se desee cambiando las `S` o `K` iniciales del *script* por otro texto: un método sería, por ejemplo, cambiar `S20ssh` por `STOP_S20ssh` y ya no arrancará; la próxima vez, cuando volvamos a necesitarlo, quitamos el prefijo y lo volvemos a tener activo. Aunque este metodo es posible, es más recomendable utilizar una serie de herramientas simples que proporcionan las distribuciones para colocar, quitar o activar un determinado servicio. Por ejemplo, comandos como, `service` y `chkconfig` en Fedora, o similares en Debian como `update-rc.d` y `invoke-rc.d`).

Otro aspecto es el cierre de servicios no seguros. Tradicionalmente, en el mundo UNIX se habían utilizado servicios de transferencia de archivos como ftp, de conexión remota como telnet y comandos de ejecución (de login o de copia) remotos, muchos de los cuales comenzaban con la letra "r" (por ejemplo, rsh, rcp, rexec, etc.). Otros peligros potenciales son los servicios finger y rwhod, que permitían obtener información desde la red de los usuarios de las máquinas; aquí el peligro estaba en la información que podía obtener un atacante y que le podía facilitar mucho el trabajo. Todos estos servicios no deberían ser usados actualmente debido a los peligros potenciales que implican. Respecto al primer grupo:

- ftp, telnet: en las transmisiones por red no cifran las contraseñas, y cualquiera puede hacerse con las contraseñas de servicios o las cuentas asociadas (por ejemplo, mediante un husmeador).

- rsh, rexec, rcp: presentan, además, el problema de que bajo algunas condiciones ni siquiera necesitan contraseñas (por ejemplo, si se hace desde sitios validados en fichero `.rhosts`), con lo cual vuelven a ser inseguros, y dejan grandes puertas abiertas.

La alternativa es usar clientes y servidores seguros que soporten el cifrado de los mensajes y la autentificación de los participantes. Hay alternativas seguras a los servidores clásicos, pero actualmente la solución más usada es mediante la utilización del paquete OpenSSH (que puede combinarse también con OpenSSL para entornos web). OpenSSH ofrece soluciones basadas en los comandos SSH, scp y sftp, permitiendo sustituir a los antiguos clientes y servidores (se utiliza un *daemon* denominado sshd). El comando SSH permite las antiguas funcionalidades de telnet, rlogin y rsh, entre otros, y scp sería el equivalente seguro de rcp, y sftp, del ftp.

Respecto a SSH, también hay que tener la precaución de usar SSH en la versión 2. La primera versión tiene algunos *exploits* conocidos; hay que tener cuidado al instalar OpenSSH, y si no necesitamos la primera versión, instalar solo

soporte para SSH2 (véase la opción `Protocol` en el fichero de configuración `/etc/ssh/sshd_config`).

Además, la mayoría de servicios que dejemos activos en nuestras máquinas tendrían después que ser filtrados a través de cortafuegos, para asegurar que no fuesen utilizados o atacados por personas a los que no iban destinados.

6. Herramientas de seguridad: Detección de vulnerabilidades e intrusiones

Detección de intrusos

Los sistemas IDS nos permiten la detección a tiempo de intrusos usando nuestros recursos o explorando nuestros sistemas en busca de fallos de seguridad.

Con los sistemas de detección de intrusos [Hat08] (IDS) se quiere dar un paso más. Una vez hayamos podido configurar más o menos correctamente nuestra seguridad, el paso siguiente consistirá en una detección y prevención activa de las intrusiones.

Los sistemas IDS crean procedimientos de escucha y generación de alertas al detectar situaciones sospechosas, es decir, buscamos síntomas de posibles accidentes de seguridad.

Hay sistemas basados en la información local que, por ejemplo, recopilan información de los registros del sistema, vigilan cambios en los ficheros de sistema o bien en las configuraciones de los servicios típicos. Otros sistemas están basados en red e intentan verificar que no se produzcan comportamientos extraños, como por ejemplo, casos de suplantación, donde hay falsificaciones de direcciones conocidas; controlan el tráfico sospechoso y posibles ataques de denegación de servicio, detectando tráfico excesivo hacia determinados servicios y controlando que no hay interfaces de red en modo promiscuo (síntoma de *sniffers* o capturadores de paquetes).

Algunos ejemplos de herramientas IDS serían:

- Logcheck: verificación de logs.
- TripWire: estado del sistema mediante sumas md5 aplicadas a los archivos.
- Portsentry: protegen contra escaneos de puertos y detectan indicios de actividad sospechosa.
- AIDE: una versión libre de TripWire.
- Snort: IDS de verificación de estado de una red completa.

Algunas de las herramientas de seguridad pueden considerarse también herramientas de ataque. Por tanto, se recomienda probar estas herramientas sobre máquinas de nuestra red local o privada. Nunca realizar ataques de prueba con direcciones IP a terceros, ya que estos podrían tomarlo como intrusiones y pedirnos responsabilidades, pedírselas a nuestro ISP o avisar a las autoridades competentes para que nos investiguen o cierren nuestro acceso.

A continuación, comentamos brevemente algunas de las herramientas y algunos de los usos que se les puede dar:

1) Logcheck: una de las actividades de un administrador de red es verificar diariamente (más de una vez por día) los archivos log para detectar posibles ataques/intrusiones o eventos que puedan dar indicios sobre estas cuestiones. Esta herramienta selecciona (de los archivos log) información condensada de problemas y riesgos potenciales y luego envía esta información al responsable, por ejemplo, a través de un correo. El paquete incluye utilidades para ejecutarse de modo autónomo y recordar la última entrada verificada para las subsiguientes ejecuciones. La lista de archivos a monitorizar se almacena en `/etc/logcheck/logcheck.logfiles` y la configuración por defecto es adecuada (si se no modificó gran parte del archivo `/etc/syslog.conf`). Logcheck puede funcionar en tres modalidades: paranoid, server y workstation. El primero es muy detallado y debería limitarse a casos específicos, como por ejemplo, *firewalls*. El segundo (por defecto) es el recomendado para la mayoría de los servidores, y el último es el adecuado para estaciones de escritorio. Esta herramienta permite una configuración total de los filtros y de las salidas, si bien puede ser complicado reescribirlos. Las reglas se pueden clasificar en: intento de intrusión (*cracking*), almacenadas en `/etc/logcheck/cracking.d/`; las de alerta de seguridad, almacenadas en `/etc/logcheck/violations.d/`; y las que son aplicadas al resto de los mensajes. [Her13]

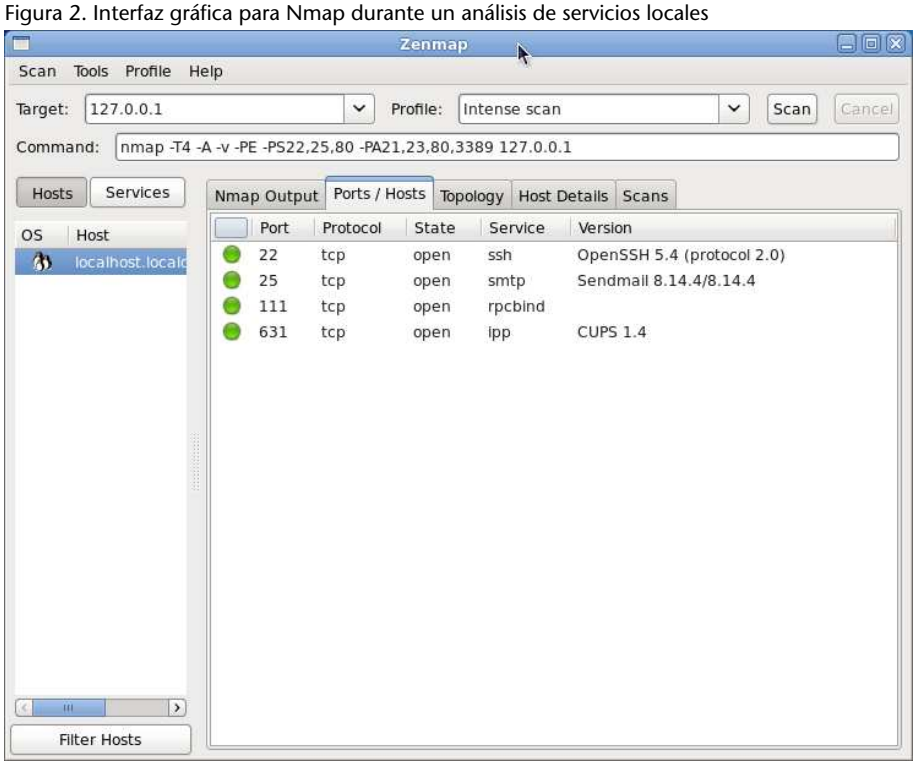
2) TripWire: esta herramienta mantiene una base de datos de sumas de comprobación de los ficheros importantes del sistema. Puede servir como sistema IDS preventivo. Nos sirve para realizar una foto del sistema, poder comparar después cualquier modificación introducida y comprobar que no haya sido corrompido por un atacante. El objetivo aquí es proteger los archivos de la propia máquina, evitar que se produzcan cambios como, por ejemplo, los que podría haber provocado un *rootkit*. Así, cuando volvamos a ejecutar la herramienta, podemos comprobar los cambios respecto a la ejecución anterior. Hay que elegir un subconjunto de archivos que sean importantes en el sistema o fuente de posibles ataques. Hay otras herramientas libres de funcionamiento equivalente, entre ellas una posibilidad es AIDE. Tripwire (http://sourceforge.net/projects/tripwire/) es una herramienta que ayudará al administrador notificando posibles modificaciones y cambios en archivos para evitar posibles daños (mayores). Esta herramienta compara las diferencias entre los archivos actuales y una base de datos generada previamente para detectar cambios (inserciones y borrado), lo cual es muy útil para detectar posibles modificaciones de archivos vitales, como por ejemplo, en archivos de configuración.

3) Portsentry: forma parte de un conjunto de herramientas* que proporcionan servicios de seguridad de nivel de host para GNU/Linux. PortSentry, LogSentry y Hostsentry protegen contra escaneos de puertos y detectan indicios de actividad sospechosa.

***http://sourceforge.net/projects/sentrytools/**

4) Nmap [Insb]: es una herramienta de escaneo de puertos para redes (figura 2). Puede escanear desde máquinas individuales a segmentos de red. Permite diversos tipos de escaneo de puertos, dependiendo de las protecciones que tenga el sistema. También tiene técnicas que permiten determinar el sistema operativo que usan las máquinas remotas. Puede emplear diferentes paquetes de TCP y UDP para probar las conexiones. Existen algunas interfaces gráficas para KDE y Gnome, y alguna con bastantes posibilidades, como zenmap.

Figura 2. Interfaz gráfica para Nmap durante un análisis de servicios locales



5) tcpdump: se trata de una herramienta muy potente que está en todas las distribuciones y que nos servirá para analizar los paquetes de red. Este programa permite el volcado del tráfico de una red y puede analizar la mayoría de los protocolos utilizados hoy en día (IPv4, ICMPv4, IPv6, ICMPv6, UDP, TCP, SNMP, AFS BGP, RIP, PIM, DVMRP, IGMP, SMB, OSPF, NFS, entre otros).

Dada la importancia de analizar los paquetes hacia adónde van y de dónde vienen, mostraremos algunos comandos habituales de tcpdump:

```
# tcpdump
parámetros por defecto -v o  -vv para el nivel de información mostrada, -q salida rápida
# tcpdump -D
interfaces disponibles para la captura
# tcpdump -n
muestra IP en lugar de direcciones
# tcpdump -i eth0
captura el tráfico de eth0
# tcpdump udp
solo los paquetes UDP
```

```
# tcpdump port http
solo los paquetes del puerto 80 (web)
# tcpdump -c 20
solo los 20 primeros paquetes
# tcpdump -w capture.log
envía los datos a un archivo
# tcpdump -r capture.log
lee los datos de un archivo
```

Los datos capturados no son texto, por lo que se puede utilizar esta misma herramienta para su lectura o bien usar otra, como wireshark, para realizar su lectura y decodificación. Con tcpdump host www.site.com solo examinamos los paquetes que contengan esta dirección.

```
tcpdump src 192.168.1.100 and dst 192.168.1.2 and port ftp
```

Para mostrar los paquetes ftp que vayan desde 192.168.1.100 a 192.168.1.2.

6) Wireshark (antes llamado Ethereal): es un analizador de protocolos y captura el tráfico de la red (actúa como *sniffer*). Permite visualizar el tráfico capturado, ver estadísticas y datos de los paquetes individuales y agruparlos, ya sea por origen, destino, puertos o protocolo. Puede incluso reconstruir el tráfico de una sesión entera de un protocolo TCP.

7) Snort [Sno]: es uno de los mejores sistema IDS en GNU/Linux, que permite realizar análisis de tráfico en tiempo real y guardar registros de los mensajes. Permite realizar análisis de los protocolos y búsquedas por patrones (protocolo, origen, destino, etc.). Puede usarse para detectar diversos tipos de ataques. Básicamente, analiza el tráfico de la red para detectar patrones que puedan corresponder a un ataque. El sistema utiliza una serie de reglas para anotar la situación (`log`), producir un aviso (`alert`) o descartar la información (`drop`).

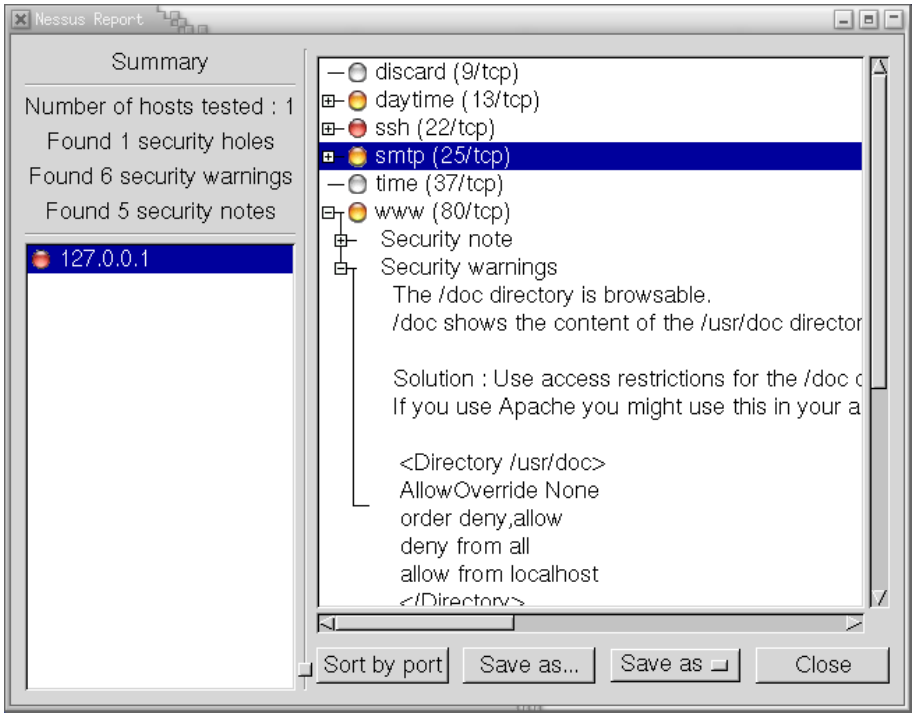
Instalación Snort

Snort es un sistema IDS altamente recomendable. Para la instalación detallada consultar las *Setup Guides* disponibles en https://www.snort.org /documents.

8) Nessus [Nes]: es un detector de vulnerabilidades conocidas, que prueba diferentes técnicas de intrusión y asesora sobre cómo mejorar la seguridad para las detectadas. Es un programa modular que incluye más de 11.000 conectores (*plugins*) para los diferentes análisis. Utiliza una arquitectura cliente/servidor, con un cliente gráfico que muestra los resultados y el servidor que realiza las diferentes comprobaciones sobre las máquinas. Tiene capacidad para examinar redes enteras y genera los resultados en forma de informes que pueden exportarse a diferentes formatos (por ejemplo, HTML). Hasta el año 2005 Nessus 2 era una herramienta libre, pero la compañía decidió convertirse en propietaria en la versión Nessus 3. Todavía en GNU/Linux se puede encontrar y seguir utilizando Nessus 2 (figura 3), que se mantiene con licencia GPL y una serie de conectores que se van actualizando. Nessus 3+ (y versiones posteriores) como herramienta propietaria para GNU/Linux es más potente, ampliamente utilizada y una de las herramientas más populares de seguridad. Normalmente se

mantiene libre de coste una versión con los conectores menos actualizados que la versión de pago. Y todavía se mantiene Nessus 2 en licencia GPL, actualizándola en repetidas ocasiones. También ha surgido un *fork* libre a partir de Nessus 2, denominado OpenVAS, que está disponible prácticamente para la mayoría de distribuciones GNU/Linux. Más adelante analizaremos alguna de sus posibilidades.

Figura 3. Cliente de Nessus 2 que muestra el informe de vulnerabilidades y las posibles soluciones



Podemos encontrar muchas más herramientas de seguridad disponibles. Un buen lugar para examinar es http://sectools.org, donde los creadores del Nmap mantuvieron una lista de herramientas populares votadas por los usuarios a lo largo de varios años. Ahora esta lista es un poco antigua, pero se mantienen actualizadas las referencias a las herramientas, las cuales, a día de hoy, son perfectamente útiles y de las más utilizadas.

Pasaremos ahora, en las secciones siguientes, a analizar diferentes herramientas con más detalle y algunos casos de uso. Algunas de ellas son bastante útiles en aspectos generales de detección de vulnerabilidades e intrusos.

6.1. OpenVAS

La Herramienta OpenVAS (*Open Vulnerability Assessment System*) es un conjunto de herramientas que trabajan al unísono para correr tests sobre máquinas clientes usando una base de datos de *exploits* conocidos. La idea es conocer en qué situación se encuentran los clientes probados contra una serie de ataques conocidos.

En este caso vamos a utilizar una distribución Debian como sistema base para probar los clientes correspondientes. Comenzamos instalando el repositorio necesario para una Debian 7.x (wheezy):

(NOTA: Vigilar los saltos de línea indicados en estas líneas de comandos con \, deberían ser una única línea sin este carácter, ni espacios añadidos)

```
# echo "deb http://download.opensuse.org/repositories/security:/OpenVAS\
:/UNSTABLE:/v6/Debian_7.0/ ./" >> /etc/apt/sources.list
# wget http://download.opensuse.org/repositories/security:/OpenVAS\
:/UNSTABLE:/v6/Debian_7.0/Release.key
# apt-key add ./Release.key
# apt-get update
```

Así instalamos el repositorio de OpenVAS para Debian, añadiéndolo como fuente de paquetes, importando su hash (del repositorio), y añadiéndola al sistema para que podamos autentificar los paquetes del repositorio. Seguidamente procedemos a un *update* del sistema de paquetes APT. Y procederemos a instalar los paquetes necesarios para los diferentes componentes de OpenVAS, paquetes necesarios para la generación de informes y algunos necesarios para la creación de certificados posteriores:

```
# apt-get -y install greenbone-security-assistant openvas-cli openvas-manager \
openvas-scanner openvas-administrator sqlite3 xsltproc rsync
# apt-get -y install texlive-latex-base texlive-latex-extra \
texlive-latex-recommended htmldoc
# apt-get -y install alien rpm nsis fakeroot
```

El siguiente bloque de comandos, todos como usuario root, se encargan de generar los certificados necesarios para el sitio OpenVAS, preparar las bases de datos que se utilizarán (sincronizándola por Internet con los servidores de OpenVAS y sus plugins disponibles), parar los *daemons* de OpenVAS, reiniciar el *daemon* principal de OpenVAS (se realiza la sincronización de datos necesarios) y añadir usuario como admin, por lo que nos preguntarán el password que deseamos. Posteriormente, se parara el *daemon* general de OpenVAS y se reiniciarán los servicios. Este bloque siguiente puede ejecutarse directamente como root sobre una Debian 7.x (con los pasos anteriores previamente realizados). Hay que tener en cuenta que la sincronización de datos con los servidores OpenVAS puede llevar un tiempo considerable:

```
test -e /var/lib/openvas/CA/cacert.pem  || openvas-mkcert -q
openvas-nvt-sync
test -e /var/lib/openvas/users/om || openvas-mkcert-client -n om -i
/etc/init.d/openvas-manager stop
/etc/init.d/openvas-scanner stop
openvassd
openvasmd --rebuild
openvas-scapdata-sync
openvas-certdata-sync
test -e /var/lib/openvas/users/admin || openvasad -c add_user -n admin -r Admin
killall openvassd
```

```
sleep 15
/etc/init.d/openvas-scanner start
/etc/init.d/openvas-manager start
/etc/init.d/openvas-administrator restart
/etc/init.d/greenbone-security-assistant restart
```

Una vez realizados todos estos pasos, podemos acceder a la herramienta en nuestro sistema arrancando el navegador web y abriendo la URL:

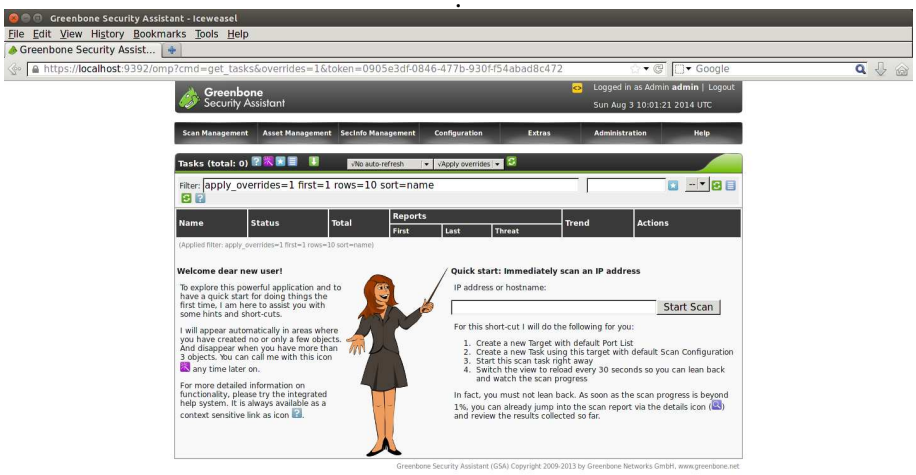
```
https://localhost:9392/
```

Cuando abramos la dirección mencionada, tendremos un proceso de *login*, que superaremos mediante el usuario login creado.

Al acceder tendremos la interfaz web disponible, donde mediante un wizard podremos directamente indicar la IP o DNS del sistema donde haremos el escáner de vulnerabilidades.

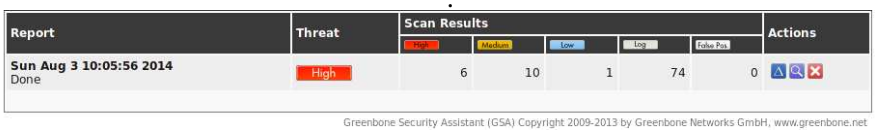
Como siempre, cabe señalar de nuevo que utilizaremos la herramienta para escanear un host propio y no contra terceros, que podrían tomarse el escaneo como un intento de ataque.

Figura 4. Interfaz web inicial OpenVAS para *quick start*



Una vez que el escáner haya progresado, podemos acceder al resultado a través de los iconos del apartado *actions* de la interfaz asociada.

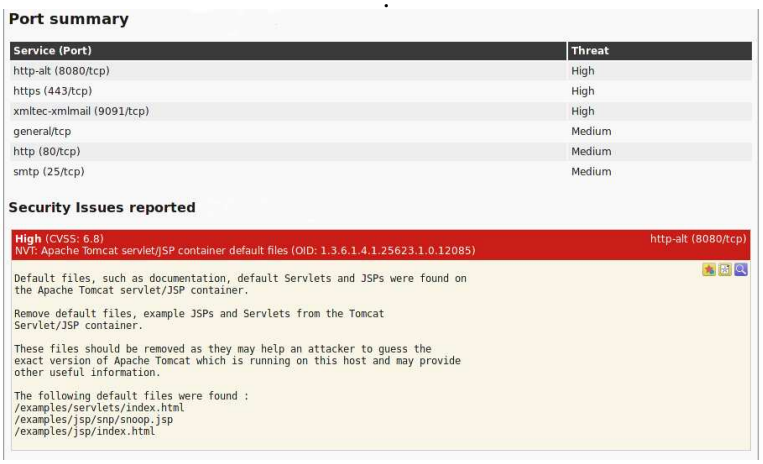
Figura 5. Vulnerabilidades detectadas con su nivel de riesgo



Al final dispondremos del informe en forma de una lista agregada de potenciales vulnerabilidades detectadas, con un nivel de *thread* y las vulnerabilidades asociadas a cada nivel de peligro. Podemos de nuevo examinar los detalles del informe (icono de lupa) para disponer de la posibilidad de visualizarlo o descargarlo (en varios formatos).

En la interfaz del informe (*Report Summary*) podemos filtrar los resultados por varios criterios, y podemos más abajo examinar cada incidencia según su nivel de peligro, verificando la explicación de la vulnerabilidad e indicaciones para su posible solución.

Figura 6. Una vulnerabilidad detectada



Así pues ya disponemos de una instalación básica de OpenVAS funcional para controlar vulnerabilidades de nuestros servidores, y podemos usarla como medio para perfeccionar problemas de seguridad de nuestro entorno.

Existen diversas opciones extras disponibles para perfeccionar la instalación, entre las que se encuentran tareas de programas de escaneo, generar informes automáticamente y enviar alertas por correo. Podéis explorar las opciones de interfaz para obtener una configuración más detallada y así optimizar mejor el uso de OpenVAS.

6.2. Denyhosts

En este caso tratamos con otro problema habitual. Se trata de la administración de los accesos a través de SSH y de la administración de cuentas remotas de los usuarios. El servicio de SSH es propenso a recibir ataques de varias tipologías, como:

1) Ataques controlados hacia puertos, típicamente el TCP 22, donde se aloja por defecto.

2) Ataques contra usuarios Linux predeterminados o de sistema, buscando cuentas de tipo invitado que estuvieran con passwords por defecto, o algunas contraseñas de usuarios asociados a servicios (por ejemplo, bases de datos, CMS, control de versiones u otros) que hayan estado instaladas con sus cuentas por defecto, o contraseñas provenientes de defecto.

3) Ataques contra usuario root, mediante diccionarios, guiados o simplemente por la fuerza bruta.

El servicio SSH en sí ofrece ciertas posibilidades (a través del fichero de configuración de su *daemon* `/etc/ssh/sshd_config`) para mitigar estos ataques, como cambiar el puerto por defecto del servicio, o en el plano de los usuarios, intentar mitigar los problemas mediante una selección correcta de sus contraseñas y promocionando políticas de cambio y uso de contraseñas no triviales. Al usuario root se le puede denegar, además de su acceso en SSH (opción *PermitRootLogin no*), el acceso remoto, lo que no impedirá que el atacante intente conseguir acceso local mediante algún otro usuario. También podremos complementar servicios como SSH con otras medidas: además de las que veremos en el capitulo siguiente 7, con *TCP wrappers*, con el uso de cortafuegos (*firewalls*), o también inhabilitar el uso de contraseñas para SSH y forzar así el uso de claves pública/privada para los usuarios de SSH.

Ante estos problemas, Denyhosts ofrece la posibilidad de monitorizar los logs de SSH e impedir (*ban*) el acceso a usuarios que detecte que posiblemente están realizando un ataque remoto. Puede configurarse para permitir una serie de intentos de conexión y los tiempos entre ellos. Para que Denyhosts funcione correctamente el *daemon* de SSHD ha de estar compilado con la opción de *TCP wrappers* (comentaremos esta técnica en el siguiente capitulo 7), aunque en el caso de los *daemons* instalados de paquetes de la distribución ya viene así por defecto.

En el caso de una distribución Debian (de forma semejante en una Fedora), podemos instalarlo con:

```
# apt-get install denyhosts
```

y podemos configurar los parámetros en el fichero `/etc/denyhosts.conf`, un ejemplo de valores de ciertos parámetros (existen muchas más opciones, en particular las relacionadas con correo de notificación mediante opciones *SMTP_*):

```
SECURE_LOG = /var/log/auth.log
BLOCK_SERVICE = sshd
DENY_THRESHOLD_INVALID = 5
```

```
DENY_THRESHOLD_VALID = 10
DENY_THRESHOLD_ROOT = 1
DENY_THRESHOLD_RESTRICTED = 1
HOSTNAME_LOOKUP=YES
```

Primero establecemos el lugar para examinar el log correspondiente a los intentos de SSH (en este caso en Debian es el fichero mencionado), que será el fichero que monitorizará Denyhosts. Controlaremos el bloqueo del servicio *sshd* (*daemon ssh*) si se dan las circunstancias (pueden controlarse otros servicios). En este ejemplo permitimos hasta 5 intentos fallidos de cuentas no válidas (no existentes en el sistema), 10 intentos fallidos de cuentas registradas en el sistema (en `/etc/passwd`), y 1 intento sobre root (esto tiene sentido, por ejemplo, si sabemos que no está permitido el acceso desde root); estos serán los intentos antes de bloquear al host que los ha producido. En el último se permite que se haga una búsqueda DNS para la IP que ha realizado el fallo por intentos excesivos.

Hay que tener cuidado con los valores de *Threshold* para asegurar que cumplimos unos mínimos de posibilidades, ya que intentos válidos desde hosts, por olvidos o fallos repetidos, pueden provocar que un host válido se bloquee.

El control anterior de los hosts que se permiten y no se realiza con los ficheros `/etc/hosts.deny` y `/hosts.allow`, para bloqueados y permitidos; por ejemplo, si nos queremos asegurar de que nuestro host habitual no se bloquee (siempre que dispongamos de IP estática), podríamos colocar (en `/etc/hosts.allow`):

```
ALL: nuestra_ip
```

Posteriormente, podemos reiniciar el servicio con:

```
# /etc/init.d/denyhosts restart
```

Los hosts que se bloqueen a lo largo del tiempo acabarán incluyéndose en `/etc/hosts.deny`. También pueden purgarse al cabo de un tiempo (hay opciones denominadas *PURGE_* en la configuración de denyhosts que lo controlan de forma automática); podrán purgarse explícitamente mediante:

```
# /etc/init.d/denyhosts stop
# denyhosts --purge
# /etc/init.d/denyhosts start
```

6.3. Fail2ban

Fail2Ban, disponible en Debian y Fedora, es similar en concepto a la herramienta previa *Denyhosts*, pero en lugar de enfocarse a SSH, en este caso puede ser configurada para monitorizar cualquier servicio que escriba intentos de login a un fichero de log, y en lugar de bloquear mediante poner el host culpable de ataques en `/etc/hosts.deny` (aunque también puede configurarse para que actúe así), realiza los bloqueos por medio de *iptables* (sistema de cortafuegos por defecto del kernel Linux, como veremos en el capitulo 7.

Instalamos con:

```
# apt-get install fail2ban
```

y disponemos posteriormente de su configuración para nuestro sistema en el directorio `/etc/fail2ban`, en el que encontraremos el comportamiento por defecto en el fichero `/etc/fail2ban/jail.conf`. En algunas versiones recientes se recomienda no editar este fichero y disponer la configuración local en el directorio `/etc/fail2ban/jail.d` o en `/etc/fail2ban/jail.local`. Consultad la página man dependiendo de la versión disponible en nuestra distribución.

Algunas de las configuraciones presentes (como ejemplos) se refieren a:

- ignoreip = 127.0.0.1/8 : si son IPs de máquinas conocidas que no queremos que fail2ban bloquee. En este caso el propio localhost, y direcciones asociadas.

- bantime = 600 : Tiempo en segundos en que una máquina es bloqueada si se detecta como problemática.

- findtime = 600 y maxretry=3 : Un host es bloqueado si antes de que llegue a *findtime* segundos ha generado *maxretry* intentos fallidos.

Después, para cada servicio encontramos varios campos (en la sección del fichero de configuracion `[servicio]`); normalmente por defecto solo se encuentra activado el servico de SSH:

- enabled: activado el control del servicio.

- port: puerto de servicio que se está utilizando, normalmente usando los nombres de `/etc/services`.

- filter: filtros aplicados de los disponibles en `/etc/fail2ban/filter.d`. Estos filtros sirven para parsear los mensajes de eventos de los logs examinados, para detectar si corresponden o no al servicio.

- action: la medida a tomar para el bloqueo, por ejemplo *iptables-allports*, que bloqueará los puertos que use el servicio para el host problemático. Las *actions* que tenemos disponibles las podemos encontrar en la ruta: `/etc/fail2ban/action.d/`.
- logpath: fichero de log que se comprueba para detectar logins fallidos.
- maxretry: si cambiamos en el servicio el número máximo de intentos fallidos.

Por ejemplo, en una Debian podríamos copiar `/etc/fail2ban/jail.conf` en `/etc/fail2ban/jail.local` y ajustar los parámetros locales generales como creamos conveniente, y dentro del fichero cada una de las secciones correspondientes al servicio.

Posteriormente reiniciamos el servicio:

```
# /etc/init.d/fail2ban restart
```

Y si por ejemplo tenemos *fail2ban* emitiendo logs a `/var/log/fail2ban.log`, podríamos observar salidas como:

```
2014-08-24 18:49:09,466 fail2ban.actions: WARNING [apache] Ban 1.2.3.4
2014-08-24 19:08:33,213 fail2ban.actions: WARNING [ssh] Ban 1.2.3.4
```

y observar con *iptables* si se han producido bloqueos (en este caso se visualiza el producido en el servicio SSH):

```
#iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
fail2ban-ssh  tcp  --  anywhere         anywhere            multiport dports ssh
........
Chain fail2ban-ssh (1 references)
target     prot opt source               destination
DROP       all  --  1.2.3.4             anywhere
RETURN     all  --  anywhere            anywhere
......
```

si disponemos de los hosts problemáticos bloqueados.

Algunos comandos útiles para examinar los logs, comprobar estatus general de un servicio particular o eliminar el bloqueo *iptables* de una ip determinada:

```
# tail /var/log/fail2ban.log
# fail2ban-client status
```

```
# fail2ban-client status ssh
# iptables -D fail2ban-<CHAIN_NAME> -s <IP> -j DROP
```

donde en el último caso se desbloquea una IP si el bloqueo se produjo por *iptables*, y donde normalmente (a no ser que hayamos cambiado el tipo de *action*), la *CHAIN_NAME* será la correspondiente al servicio, en el ejemplo anterior SSH (*fail2ban-ssh*).

7. Protección mediante filtrado (*TCP wrappers* y cortafuegos)

Los *TCP wrappers* [Mou02] son un software que actúa de intermediario entre las peticiones del usuario de servicio y los *daemons* de los servidores que ofrecen el servicio. Muchas de las distribuciones vienen ya con los *wrappers* activados y podemos configurar los niveles de acceso. Los *wrappers* se suelen utilizar en combinación con inetd o xinetd, de manera que protejan los servicios que ofrecen.

El *wrapper* básicamente sustituye al *daemon* (demonio) del servicio por otro que hace de intermediario (llamado tcpd, normalmente en `/usr/sbin/tcpd`). Cuando este recibe una petición, verifica el usuario y origen de la misma, para determinar si la configuración del *wrapper* del servicio permite o no utilizarlo. Además, incorpora facilidades de generar registros o bien informar por correo de los posibles intentos de acceso y después ejecuta el *daemon* adecuado asignado al servicio.

Por ejemplo, supongamos la siguiente entrada en inetd:

```
finger stream tcp nowait nobody /usr/etc/in.fingerd in.fingerd
```

Esta se cambia por:

```
finger stream tcp nowait nobody /usr/sbin/tcpd in.fingerd
```

de manera que, cuando llegue una petición, será tratada por el *daemon* tcpd que se encargará de verificar el acceso (para más detalles, véase la página *man tcpd*).

Wrappers

Los *wrappers* nos permiten controlar la seguridad mediante listas de acceso a nivel de servicios.

También existe un método alternativo de *wrapper TCP*, que consiste en que la aplicación original se compile con la biblioteca de *wrappers*. Entonces la aplicación no tiene porqué estar en inetd y podremos controlarla igual que el primer caso con la configuración que comentamos a continuación.

El sistema de *wrappers* se controla desde los ficheros `/etc/hosts.deny`, donde especificamos qué servicios denegamos y a quién, por medio de opciones, como un pequeño *shell* para guardar la información del intento, y el fichero `/etc/hosts.allow`, donde solemos colocar qué servicio vamos a utilizar, seguido de la lista de a quién dejamos entrar (más adelante, en el taller, ve-

remos un pequeño ejemplo). También existen los comandos `tcpdchk`, que comprueban la configuración de los ficheros *hosts* (véase *man hosts_access* y *hosts_options*), para comprobar que son correctos; es decir, comprueban la configuración. El otro comando útil es `tcpdmatch`, al cual damos un nombre de servicio y un posible cliente (usuario y/o *host*) y nos dice qué haría el sistema ante esta situación.

7.1. Cortafuegos

Un cortafuegos (*firewall*) es un sistema o grupo de sistemas que refuerza las políticas de control de acceso entre redes. El cortafuegos puede estar implementado en software, como una aplicación especializada ejecutándose en un computador individual, o bien puede tratarse de un dispositivo especial dedicado a proteger uno o más computadores.

En general dispondremos, o bien de la aplicación de cortafuegos para proteger una máquina concreta conectada directamente a Internet (ya sea directa o por proveedor), o bien podremos colocar en nuestra red una o varias máquinas dedicadas a esta función, de modo que protejan nuestra red interna.

Técnicamente, la mejor solución es disponer de un computador con dos o más tarjetas de red que aíslen las diferentes redes (o segmentos de red) conectadas, de manera que el software de cortafuegos en la máquina (o si fuese un hardware especial) se encargue de conectar los paquetes de las redes y determinar cuáles pueden pasar o no, y a qué red.

Este tipo de cortafuegos suele combinarse con un enrutador (*router*) para enlazar los paquetes de las diferentes redes. Otra configuración típica es la de cortafuegos hacia Internet, por ejemplo con dos tarjetas de red: en una obtenemos/proporcionamos tráfico a Internet y en la otra enviamos o proporcionamos el tráfico a nuestra red interna, pudiendo así eliminar el tráfico que no va destinado a nosotros, y también controlar el tráfico que se mueve hacia Internet, por si no queremos que se tenga acceso a algunos protocolos o bien sospechamos que hay posibilidades de fugas de información por algunos ataques. Una tercera posibilidad es la máquina individual conectada con una única tarjeta de red hacia Internet, directa o bien a través de un proveedor. En este caso, sólo queremos proteger nuestra máquina de ataques de intrusos, de tráfico no deseado o de que se vea comprometida al robo de datos.

Es decir, en estos casos podemos comprobar que el cortafuegos, dependiendo de si es software o no, de si la máquina tiene una o varias tarjetas de red o de si protege a una máquina individual o a una red, puede tener configuraciones y usos diferentes.

El cortafuegos en general permite definir al usuario una serie de políticas de acceso (cuáles son las máquinas a las que se puede conectar o las que pueden recibir información y el tipo de información a recibir) por medio del control de los puertos TCP/UDP permitidos de entrada (*incoming*) o de salida (*outcoming*). Algunas herramientas de gestión de cortafuegos vienen con una serie de políticas preconfiguradas, o solo dicen si se quiere un nivel de seguridad alto, medio o bajo, o, por último, permiten personalizar las opciones totalmente (máquinas, protocolos, puertos, etc.).

Seguridad a nivel de paquetes

Los cortafuegos permiten establecer seguridad a nivel de paquetes y conexiones de comunicación.

Otra técnica a veces relacionada es la NAT (*network address translation*, traducción de direcciones de red). Esta técnica proporciona una vía para ocultar las direcciones IP usadas en la red privada y las oculta de Internet, pero mantiene el acceso desde las máquinas. Uno de los métodos típicos es el denominado *masquerading*. Si se usa *NAT masquerading*, uno o varios dispositivos en la red pueden aparecer como una única dirección IP vistos desde fuera. Esto permite conectar varios computadores a un único dispositivo de conexión externa; por ejemplo, un caso de enrutador ADSL en el hogar permite conectar varias máquinas sin necesidad de que el proveedor nos proporcione diferentes direcciones IP. Los enrutadores ADSL suelen ofrecer algún tipo de *NAT masquerading* y también posibilidades de cortafuegos. Suele ser bastante común utilizar una combinación de ambas técnicas. En este caso, entra en juego, además de la configuración de la máquina del cortafuegos (los casos vistos antes), la configuración de la red privada interna que queremos proteger.

7.2. Netfilter: iptables

El núcleo Linux (a partir de versiones 2.4.x) proporciona un subsistema de filtrado denominado Netfilter [Net]. *Netfilter* es un conjunto de funcionalidaddes integradas en el kernel de Linux para interceptar y gestionar los paquetes de red, y que proporciona características de filtrado de paquetes y también NAT. Este sistema permite usar diferentes interfaces de filtrado, entre las cuales la más usada se denomina iptables. El comando principal de control es `iptables`. Anteriormente, se proporcionaba otro filtrado denominado ipchains en los núcleos 2.2 [Gre] y el sistema tenía una sintaxis diferente (aunque con similitudes conceptuales). En los núcleos 2.0 se utilizaba otro sistema denominado ipfwadm. Aquí (y en los ejemplos posteriores) vamos a tratar solo con Netfilter/iptables (es decir, con las posibilidades de cortafuegos en núcleos de las versiones 2.4/2.6/3.x). También comentaremos nuevas propuestas, como *nftables*, que será el próximo sustituto de Netfilter/iptables, a partir de algunas de las últimas ramas kernel 3.x, y seguramente el firewall por defecto en la rama 4.x y superiores. nftables tiene capas de compatibilidad con iptables, con lo que seguramente durante un tiempo veamos todavía mayoritariamente reglas de firewall compatibles con iptables.

Enlace de interés

Sobre Netfilter véase: http://www.netfilter.org y sobre nftables vease: http://netfilter.org /projects/nftables/

Respecto a Netfilter, como comentamos, su principal componente, en estos momentos es *iptables*, que funciona como una herramienta de cortafuegos

(*firewall*), con el soporte a nivel de kernel, permitiendo las acciones de filtraje de los paquetes, y gestión de traslación de direcciones de red (NAT).

Elementos de iptables

iptables aporta diferentes elementos como las tablas, *chains* y las propias reglas.

La interfaz del comando `iptables` permite realizar las diferentes tareas de configuración de las reglas que afectan al sistema de filtrado, ya sea generación de registros, acciones de pre y post enrutado de paquetes, NAT y reenvío (*forwarding*) de puertos.

El arranque del servicio se realiza con `/etc/init.d/iptables start`, si no estaba configurado ya en el *runlevel*.

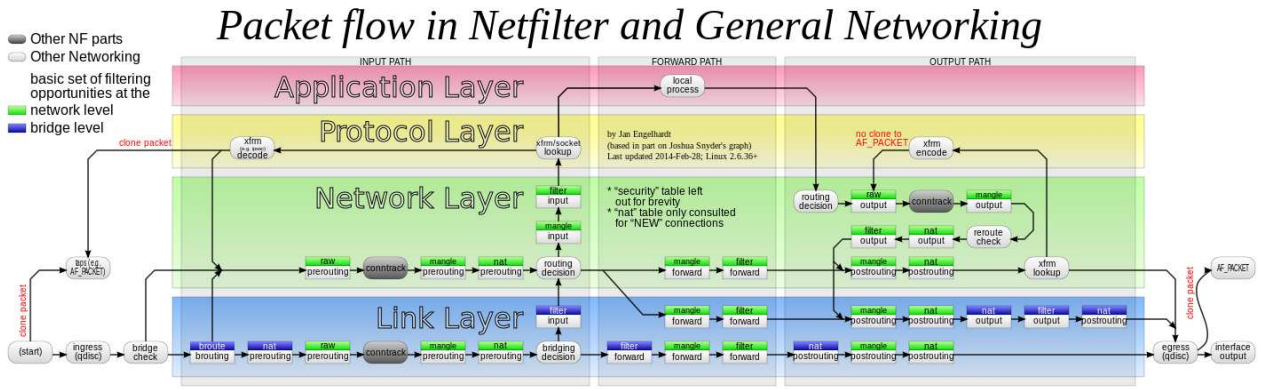
El comando `iptables -L` lista las reglas activas en ese momento en cada una de las cadenas (*chains*). Si no se han configurado previamente, suelen ser por defecto aceptar todos los paquetes de las cadenas de `INPUT` (entrada), `OUTPUT` (salida) y `FORWARD` (reenvío).

El sistema de iptables tiene como nivel superior las tablas. Cada una contiene diferentes cadenas, que a su vez contienen diferentes reglas. Las tres tablas que existen son: *Filter*, *NAT* y *Mangled*. La primera sirve para las propias normas de filtrado, la segunda para realizar traslación de direcciones dentro de un sistema que utilice NAT y la tercera, menos usada, sirve para especificar algunas opciones de control de los paquetes y cómo gestionarlos. Concretamente, si estamos con un sistema directamente conectado a Internet, utilizaremos, en general, tan solo la tabla Filter. Si el sistema está en una red privada que tiene que pasar por un enrutador, pasarela (*gateway*) o proxy (o combinación de estos), seguramente dispondremos de un sistema de NAT o IP *masquerading*; si estamos configurando precisamente la máquina que permite acceso externo, tendremos que tocar la tabla NAT y la Filter. Si la máquina está en un sistema de red privada, pero es una de las máquinas internas, será suficiente con la tabla Filter, a no ser que sea un servidor el que haga NAT a otro segmento de red.

Más detalle figura 7

Figura procedente de Wikipedia, puede observarse con más detalle en el articulo: http://en.wikipedia.org/wiki/Iptables

Figura 7. Flujos de los paquetes de red involucrados en la gestión de NetFilter/iptables.



Si un paquete llega al sistema (vease en la figura 7 la complejidad de la gestión de paquetes), en el cortafuegos implementado por iptables, se mirará primero

si existen reglas en la tabla NAT, por si hay que hacer traducciones de direcciones hacia la red interna (las direcciones normalmente no son visibles hacia el exterior); después se mirarán las reglas de la tabla Filter para decidir si se van a dejar pasar los paquetes o si no son para nosotros, y si tenemos reglas `forward` para saber hacia dónde los reenviamos. Por el contrario, cuando nuestros procesos generan paquetes, las reglas `output` de la tabla Filter controlan si los dejamos salir o no, y si hubiese sistema NAT, las reglas efectuarían la traducción de direcciones de manera que se enmascarasen. En la tabla NAT suele haber dos cadenas: `prerouting` y `postrouting`. En la primera, las reglas han de decidir si hay que hacer algún enrutamiento del paquete y cuál será la dirección de destino. En el segundo, se decide finalmente si el paquete se pasa o no hacia el interior (la red privada, por ejemplo). Y también existe una cadena `output` para el tráfico que se genere localmente de salida a la red privada, ya que `prerouting` no lo controla (para más detalles, se puede examinar *man iptables*).

A continuación comentaremos algunos aspectos y ejemplos de configuración de la tabla Filter*.

***Para las otras tablas, se puede consultar la bibliografía asociada.**

La configuración típica de la tabla Filter es de una serie de reglas que especifican qué se hace dentro de una determinada cadena, como las tres anteriores (`input`, `output` o `forward`). Normalmente, se especifica:

```
iptables -A chain -j target
```

donde `chain` es `input`, `output` o `forward` y `target`, el destino que se le va a dar al paquete que se corresponda con la regla. La opción `-A` añade la regla a las existentes.

Con esta fase de añadir reglas hay que tomar precauciones, ya que el orden importa. Hay que colocar las menos restrictivas al principio, puesto que, si primero ponemos una regla que elimine los paquetes, aunque haya otra regla, esta no será tenida en cuenta. La opción `-j` permite decidir qué haremos con los paquetes, típicamente `accept` (aceptarlos), `reject` (rechazarlos) o `drop` (simplemente perderlos). Es importante la diferencia entre `reject` y `drop`. Con el primero, rechazamos el paquete y normalmente informamos al emisario de que hemos rechazado el intento de conexión (normalmente por un paquete de tipo ICMP). Con el segundo (`drop`), simplemente perdemos el paquete como si nunca hubiese existido y no enviamos ningún tipo de respuesta. Otro `target` utilizado es `log`, para enviar el paquete al sistema de registros. Normalmente, en este caso hay dos reglas, una con el `log` y otra igual con `accept`, `drop` o `reject`, para permitir enviar al registro la información de qué paquetes han sido aceptados, rechazados o perdidos. Con las opciones de generarlos en el cortafuegos hay que controlar precisamente su uso, ya que son capaces, dependiendo del ambiente de red, de generar una enorme cantidad de información en los ficheros de registro.

Al colocar la regla, también puede usarse la opción `-I` (insertar) para indicar una posición, por ejemplo:

```
iptables -I INPUT 3 -s 10.0.0.0/8 -j ACCEPT
```

que nos dice que se coloque la regla en la cadena `input` en tercera posición; y se van a aceptar paquetes (`-j`) que provengan (con fuente, o *source*, `-s`) de la subred 10.0.0.0 con máscara de red 255.0.0.0. De forma parecida, con `-D` podemos borrar o un número de regla o la regla exacta, como se especifica a continuación, borrando la primera regla de la cadena o la regla que mencionamos:

```
iptables -D INPUT 1
iptables -D INPUT -s 10.0.0.0/8 -j ACCEPT
```

También hay reglas que permiten definir una política por defecto de los paquetes (opción `-P`); se va a hacer con todos los paquetes lo mismo. Por ejemplo, se suele colocar al inicio que se pierdan todos los paquetes por defecto, y se habilitan luego los que interesan, y muchas veces también se evita que haya reenvío de paquetes si no es necesario (si no actuamos de enrutador). Esto podría ponerse:

```
iptables -P INPUT DENY
iptables -P OUTPUT REJECT
iptables -P FORWARD REJECT
```

Todo lo cual establece unas políticas por defecto que consisten en denegar la entrada de paquetes, no permitir salir y no reenviar paquetes. Ahora se podrán añadir las reglas que conciernen a los paquetes que deseemos utilizar, diciendo qué protocolos, puertos y orígenes o destinos queremos permitir o evitar. Esto puede ser difícil, ya que tenemos que conocer todos los puertos y protocolos que utilicen nuestro software o servicios. Otra táctica sería dejar solo activos aquellos servicios que sean imprescindibles y habilitar con el cortafuegos el acceso de los servicios a las máquinas deseadas.

Algunos ejemplos de estas reglas de la tabla Filter podrían ser:

```
iptables -A INPUT -s 10.0.0.0/8 -d 192.168.1.2 -j DROP
iptables -A INPUT -p tcp --dport 113 -j REJECT --reject-with tcp-reset
iptables -I INPUT -p tcp --dport 113 -s 10.0.0.0/8 -j ACCEPT
```

Donde:

1) Perdemos los paquetes que vengan de 10.x.x.x con destino a 192.168.1.2.

2) Rechazamos los paquetes tcp con destino al puerto 113, emitiendo una respuesta de tipo tcp-reset.

3) Los mismos paquetes que en 2) pero que provengan de 10.x.x.x serán aceptados.

Respecto a los nombres de protocolos y puertos, el sistema iptables usa la información facilitada por los ficheros `/etc/services` y `/etc/protocols`, pudiéndose especificar la información (de puerto y protocolo) de forma numérica o por nombre (hay que tener cuidado, en este caso, de que la información de los ficheros sea correcta y que no hayan sido modificados, por ejemplo, por un atacante).

La configuración de iptables suele establecerse mediante llamadas consecutivas al comando `iptables` con las reglas. Esto crea un estado de reglas activas que pueden consultarse con `iptables -L`. Si deseamos salvarlas para que sean permanentes, podemos hacerlo en Fedora con:

```
/etc/init.d/iptables save
```

y se guardan en:

```
/etc/sysconfig/iptables
```

En Debian puede hacerse, si existe el anterior *script*:

```
/etc/init.d/iptables save nombre-reglas
```

Si no se da soporte directo para guardar o recuperar reglas, siempre se puede con los comandos `iptables-save` y `iptables-restore` rederigirlas a ficheros, para guardar o recuperar la configuración del cortafuegos.

En el primer caso hay que tener cuidado de que antes exista el directorio donde se guardaran los ficheros, `/var/lib/iptables` ; `nombre-reglas` será un fichero en el directorio.

Con `/etc/init.d/iptables load` podemos cargar las reglas (en Debian hay que dar el nombre del fichero de reglas o bien usar `iptables-restore`), aunque Debian soporta unos nombres por defecto de ficheros, que son `active` para las reglas normales (las que se utilizarán en un `start` del servicio) e `inactive` para las que quedarán cuando se desactive el servicio (se haga un `stop`). Otra aproximación comunmente usada es la de colocar las reglas en un fichero *script* con las llamadas iptables que se necesiten y llamarlas, por ejemplo, colocándolas en el *runlevel* necesario o con enlace hacia el *script* en `/etc/init.d`.

Por último citar un pequeño ejemplo de lo que podría ser un fichero completo de iptables (el # indica comentarios):

```
*filter
# Permitir todo el tráfico de loopback (lo0) y denegar el resto de 127/8
-A INPUT -i lo -j ACCEPT
-A INPUT ! -i lo -d 127.0.0.0/8 -j REJECT
# Aceptar todas la conexiones entrantes previamente establecidas
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# Aceptar todo el tráfico saliente
-A OUTPUT -j ACCEPT
# Permitir HTTP y HTTPS desde cualquier lugar
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
# Permitir las conexiones de SSH
# Normalmente utiliza el puerto 22, verificarlo en el archivo /etc/ssh/sshd_config.
-A INPUT -p tcp -m state --state NEW --dport 22 -j ACCEPT
# Responder al ping icmp
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
# Rechazar todo el tráfico restante de entrada.
-A INPUT -j REJECT
-A FORWARD -j REJECT
COMMIT
```

En este ejemplo solo aceptamos tráfico entrante de ping, http, htpps, ssh y bloqueamos todo el tráfico restante de entrada. En cuanto al tráfico de salida, se deja que salga todo sin restricciones. Se pueden salvar las reglas en un archivo, cargarlas y, por ejemplo, desde otra máquina probar la conectividad o ejecutar algún programa como nmap, que nos mostrará los puertos abiertos en la máquina configurada con iptables.

7.3. Paquetes para gestión de cortafuegos iptables en las distribuciones

Respecto a herramientas de configuración más o menos automatizadas por cortafuegos, existen varias posibilidades, pero hay que tener en cuenta que no suelen ofrecer las mismas prestaciones que la configuración manual de iptables (que en la mayoría de casos sería el proceso recomendado). Algunas herramientas son:

- *system-configure-firewall / Lokkit*: en la distribución Fedora/Red Hat, hasta hace poco era inicialmente un gestor de firewall muy básico y solo permite elegir al usuario el nivel de seguridad que desea (alto, medio o bajo). Después enseña los servicios que se ven afectados y podemos dejar pasar o no al servicio cambiando la configuración por defecto. En las últimas versiones ha evolucionado bastante y ya incluye posibilidades avanzadas de puertos a controlar para cada servicio. El mecanismo utilizado por debajo es iptables. Puede verse la configuración final de reglas que realiza `/etc/sysconfig/iptables` que, a su vez, es leído por el servicio iptables, que se carga en arranque o por parada o arranque mediante `/etc/init.d/iptables`, con las opciones `start` o `stop` (o las equivalen-

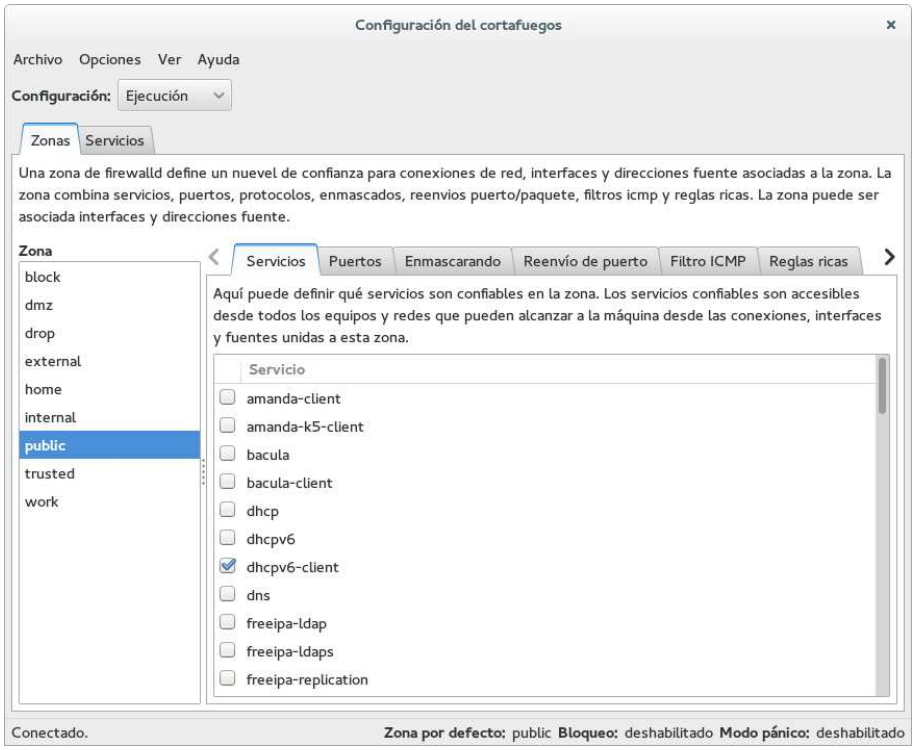
tes en Systemd). Se ofrece en versión gráfica o en textual llamada *system-config-firewall-tui*.

- En las versiones recientes de Fedora/Red Hat, se ha incluido un nuevo gestor por defecto de firewall, que encapsula el funcionamiento de iptables y que sustituye al previo (*system-config-firewall*), denominado *FirewallD*. Este proporciona control dinámico del firewall y permite definir múltiples zonas de firewall para gestionar las diversas redes a las que esté conectado el sistema. Permite mantener, de forma simultánea, configuraciones permanentes y en tiempo de ejecución. El antiguo control de Firewall de Fedora era estático y requería que, cada vez que las reglas cambiaban, se tuviera que reiniciar el firewall. FirewallD permite aplicar las nuevas reglas sin necesidad de reiniciar el firewall completo. Algunos comandos útiles de este caso que nos permiten obtener el estatus, los servicios soportados, habilitar un servicio, obtener las zonas y las zonas con servicios habilitados:

```
# firewall-cmd --state
# firewall-cmd --get-services
# firewall-cmd --add-service=http
# firewall-cmd --get-zones
# firewall-cmd --list-all-zones
```

Se dispone también de la herramienta gráfica *firewall-config* para realizar una configuración más cómoda. En entorno de terminal se recomienda ver el man correspondiente a *firewall-cmd*, debido al gran número de opciones disponibles.

Figura 8. Interfaz gráfica firewall-config para FirewallD



FirewallD

Se recomienda examinar: https://fedoraproject.org/wiki/FirewallD

- fwbuilder: una herramienta que permite construir las reglas del cortafuegos de forma gráfica. Se puede usar en varios sistemas operativos (GNU/Linux tanto Fedora como Debian, OpenBSD, MacOS), con gestión de diferentes tipos de cortafuegos (iptables incluido).

- firestarter: otra herramienta, ahora obsoleta, pero en distribuciones antiguas era usada como una herramienta gráfica (Gnome) para la creación del cortafuegos. Prácticamente maneja todas las posibilidades de iptables, pero asimismo dispone de asistentes que facilitan la construcción intuitiva del cortafuegos. Dispone, además, de un monitor en tiempo real para detectar las intrusiones.

- ufw (*Uncomplicated Firewall*): como su nombre indica, pretende ser un firewall de uso simple, que puede encontrarse en las distribuciones Debian y Ubuntu, basado en uso de línea de comandos, que permite un uso bastante sencillo (pero soportando gran parte de las posibilidades de iptables). Una pequeña sesión de ejemplo:

```
$ sudo ufw allow ssh/tcp
$ sudo ufw logging on
$ sudo ufw enable
$ sudo ufw status
Firewall loaded

To                         Action  From
--                         ------  ----
22:tcp                     ALLOW   Anywhere
```

  También se dispone de un paquete gráfico para su uso llamado *Gufw*.

ufw

Sitio y herramienta gráfica:
https://launchpad.net/ufw
http://gufw.org/

Normalmente, cada uno de estos paquetes utiliza un sistema de reglas que guarda en algún fichero propio de configuración, y que suele arrancar como servicio o como ejecución de *script* en el *runlevel* por defecto. Hay que tener especial cuidado con la consistencia de las reglas iptables del cortafuegos, ya que muchas de estas utilidades no soportan sincronización bidireccional de las reglas: o bien se actualiza el cortafuegos siempre mediante la herramienta de línea de comandos o gráfica, o bien con el comando `iptables`, pero si se realizan cambios de ambas maneras, estos cambios pueden no ser consistentes o puede perderse la configuración del cortafuegos. Se recomiendan copias de seguridad periódicas de la configuración del cortafuegos (en especial si este tiene cierto nivel de complejidad).

7.4. Netfilter: nftables

NFtables se presenta como un subproyecto de Netfilter para la siguiente generación de cortafuegos para Linux, con la intención de sustituir a la infraes-

tructura actual basada en iptables y proporcionar un nuevo framework para el filtrado de paquetes (incluido en el espacio de kernel, a partir de la versión 3.13), una nueva utilidad *nft* de usuario (en espacio de usuario) y una capa de compatibilidad con iptables.

Algunas diferencias destacables con iptables:

- Sintaxis diferente, más clara.
- Las tablas y cadenas son totalmente configurables por el usuario, a diferencia de iptables, que se basada en una estructura de tablas predefinida.
- Mayor control de la correspondencia de protocolos. En iptables algunas veces eran necesarias extensiones.
- Varias acciones en una sola regla. En iptables podían tenerse que generar varias.
- Los protocolos de red soportados pueden actualizarse realizando modificaciones en las aplicaciones clientes, sin necesidad de actualizar el kernel, evitando así problemáticas con versiones de kernel congeladas en ciertas distribuciones.

En cuanto a los caminos que siguen los paquetes, siguen teniendo la misma estructura (de *hooks*, en terminología Netfilter): cuando un paquete llega al sistema, pasa por un *prerouting*, accediendo después a una decisión de *routing* (¿es un paquete de entrada para el sistema o de reenvío hacia otros?); en caso de que hagamos *forwarding* en nuestro sistema (porque funcione de router, por ejemplo con: `echo 1 >/proc/sys/net/ipv4/ip_forward`), el paquete llegará al *postrouting* y saldrá del sistema, dependiendo de las opciones de *routing*. Si estamos en el caso contrario, y el paquete era para el sistema, pasará a `INPUT`, será procesado por el sistema, que puede generar paquetes de salida, `OUTPUT`, que después de una decisión de *routing* (¿es para el propio sistema o para envío externo?), llegará al *postrouting* para salir del sistema.

Para el uso de nftables, el kernel debe disponer del módulo *nf_tables*, que puede estar disponible en el kernel (3.13+) o puede llegarse a construir a partir de los fuentes para kernels anteriores. Así, también se necesita el módulo de la familia, por ejemplo *nf_tables_ipv4*. Si los tenemos disponibles, podemos proceder a su carga con *modprobe*.

Una vez dispongamos de los módulos idóneos, podemos proceder a realizar algunas funciones básicas (como hemos comentado, no hay tablas o cadenas [*chains*] predefinidas, por tanto las tendremos que crear):

```
Crear Tabla:
# nft add table ip filter

Listar tablas
# nft list tables ip

Listar cadenas asociadas a una tabla (en inicio no estarán creadas)
# nft list table ip filter

Borrar una tabla
# nft delete table filter

Flush de una tabla (liberar reglas de la tabla)
# nft flush table ip filter

Añadir una cadena (chain) base
(relacionada con un Netfilter Hook comentado antes),
esta cadena ve los paquetes de la pila TCP/IP de Linux,
ejemplos de input y output (si fuese necesario)
y tercer caso una cadena no base.
Con delete o flush en lugar de add, borramos o flush de las reglas.
# nft add chain ip filter input { type filter hook input priority 0 \; }
# nft add chain ip filter output { type filter hook output priority 0 \; }
# nft add chain ip filter test

Reglas: listarlas
(opciones -n desactiva resolución nombres, -nn resolución servicio),
una regla para un puerto concreto,
una regla a una posición previa concreta dentro de la cadena
guardar reglas de una cadena en un fichero
recuperarlas a la cadena
# nft list table filter
# nft add rule filter output tcp dport ssh counter
# nft add rule filter output position 8 ip daddr 127.0.0.8 drop
# nft list table filter > filter-table
# nft -f filter-table
```

Finalmente están las acciones disponibles, ya sea aceptando paquetes, perdiéndolos, rechazándolos, haciendo NAT, etc. Veamos un breve ejemplo de estas posibilidades. Para la protección simple de una máquina de escritorio, para ipv4 y ipv6, podría ser lo siguiente (este contenido es un fichero con la definición de las tablas y reglas básicas, que puede recuperarse con `nft -f fichero` para cargar el firewall.)

Acciones nftables

Para más detalles se recomienda consultar *Possible actions on packets* en: http://wiki.nftables.org /wiki-nftables/index.php /Main_Page

```
# IPv4 filtering
table filter {
        chain input {
                table filter hook input priority 0;
                ct state established accept
                ct state related accept
                meta iif lo accept
                tcp dport ssh counter packets 0 bytes 0 accept
                counter packets 5 bytes 5 log drop
        }

        chain output {
                table filter hook output priority 0;
                ct state established accept
                ct state related accept
                meta oif lo accept
                ct state new counter packets 0 bytes 0 accept
        }
}
#IPv6 filtering
table ip6 filter {
        chain input {
```

```
                table filter hook input priority 0;
                ct state established accept
                ct state related accept
                meta iif lo accept
                tcp dport ssh counter packets 0 bytes 0 accept
                icmpv6 type { nd-neighbor-solicit, echo-request, \
   nd-router-advert, nd-neighbor-advert } accept
                counter packets 5 bytes 5 log drop
        }

        chain output {
                table filter hook output priority 0;
                ct state established accept
                ct state related accept
                meta oif lo accept
                ct state new counter packets 0 bytes 0 accept
        }

}
```

7.5. Consideraciones

Niveles de seguridad

Nunca se debe confiar en un único mecanismo o sistema de seguridad. Hay que establecer la seguridad del sistema a diferentes niveles.

Aunque dispongamos de cortafuegos bien configurados, hay que tener presente que no son una medida de seguridad absoluta, ya que hay ataques complejos que pueden saltarse el control o falsear datos que creen confusión. Además, las necesidades de conectividad modernas obligan a veces a crear software que permita el *bypass* (paso a través) de los cortafuegos:

- Tecnologías como IPP, protocolo de impresión utilizado por CUPS, o el WebDAV, protocolo de autoría y actualización de sitios web, permiten pasar (o es necesario que pasen) las configuraciones de los cortafuegos.

- A menudo se utiliza (por ejemplo, los anteriores protocolos y otros) una técnica denominada *tunneling*, que básicamente encapsula protocolos no permitidos sobre la base de otros que sí que lo están; por ejemplo, si un cortafuegos permite solo paso de tráfico http (puerto 80 por defecto), es posible escribir un cliente y un servidor (cada uno a un lado diferente del cortafuegos) que hablen cualquier protocolo conocido por ambos, pero que en la red es transformado en un protocolo http estándar, con lo cual, el tráfico puede cruzar el cortafuegos. El `tunneling` por ssh también es utilizado para establecer diferentes tipos de túneles, para conexiones, usando los protocolos y puertos de ssh.

- Los códigos móviles por web (ActiveX, Java y JavaScript) cruzan los cortafuegos (via web) y, por lo tanto, es difícil proteger los sistemas si estos son vulnerables a los ataques contra agujeros descubiertos.

Así, aunque los cortafuegos son una muy buena solución a la mayor parte de amenazas a la seguridad, siempre pueden tener vulnerabilidades y dejar pasar tráfico que se considere válido, pero que incluya otras fuentes posibles de ataques o vulnerabilidades.

En seguridad nunca hay que considerar (y confiar en) una única solución y esperar que nos proteja absolutamente; hay que examinar los diversos problemas, plantear soluciones que los detecten a tiempo y políticas de prevención que nos curen en salud, por lo que pueda pasar.

8. Análisis de registros

Observando los ficheros de registro (*logs*) del sistema [Ray01, Fri02], podemos hacernos una idea rápida del estado global del sistema, así como de las últimas ocurrencias de sucesos, y detectar accesos (o intentos de acceso) indebidos. Pero también cabe tener presente que los registros, si ha sucedido una intrusión real, pueden haber sido limpiados o falseados. La mayoría de archivos de registro residen en el directorio `/var/log`.

Muchos de los servicios pueden poseer registros propios, que normalmente se establecen en la configuración (mediante el correspondiente fichero de configuración). La mayoría suelen utilizar las facilidades de registro incorporadas en el sistema Syslog, mediante el demonio Syslogd. Su configuración reside en `/etc/syslog.conf`. Esta configuración suele establecerse por niveles de mensajes: existen diferentes tipos de mensajes según su importancia. Normalmente suelen aparecer niveles `debug`, `info`, `err`, `notice`, `warning`, `crit`, `alert`, `emerg`, en que más o menos esta sería la ordenación de importancia de los mensajes (de menor a mayor). Normalmente, la mayoría de los mensajes se dirigen al registro `/var/log/messages`, pero puede definirse que cada tipo se distribuya a ficheros diferentes y también se puede identificar quién los ha originado: típicamente el núcleo, el correo, *news*, el sistema de autentificación, etc.

En consecuencia, cabe examinar (y en todo caso adaptar) la configuración de Syslog para conocer en qué registros podemos encontrar o generar la información. Otro punto importante es controlar su crecimiento, ya que según los que estén activos y las operaciones (y servicios) que se realicen en el sistema, los registros pueden crecer bastante. En Debian y Fedora se controla a través de `logrotated`, un demonio que se encarga periódicamente de hacer copias, y comprimirlas, de los registros más antiguos; se puede encontrar su configuración general en `/etc/logrotate.conf`, pero algunas aplicaciones hacen configuraciones específicas que se pueden encontrar en el directorio `/etc/logrotate.d`.

Comentamos en los siguientes puntos algunos ficheros de registro que habría que tener en cuenta (quizás los más utilizados):

Ficheros de registro

Hay que tener en cuenta que en la mayoría de ataques sofisticados exitosos se falsean o borran los datos de los ficheros de registro. Incluso así, dependiendo de la técnica usada, puede detectarse información útil sobre el ataque.

1) `/var/log/messages`: es el fichero de registro por defecto del demonio Syslogd, pero habría que revisar su configuración, no se hubiera dado el caso de que estuviera cambiado a otra parte o de que hubiera varios. Este fichero contiene una amplia variedad de mensajes de orígenes diversos (arranque,

diferentes demonios, servicios o el mismo núcleo); todo lo que resulte anómalo tendría que ser verificado. En caso de que haya habido una intrusión, se deberá mirar alrededor de las fechas de la intrusión en un determinado rango de tiempo, para detectar posibles intentos previos, la metodología usada, primeros avisos del sistema, etc.

2) `/var/log/utmp`: este fichero contiene información binaria para cada usuario que está actualmente activo. Es interesante para determinar quién está dentro del sistema. El comando `who` utiliza este fichero para proporcionar esta información.

3) `/var/log/wtmp`: cada vez que un usuario entra en el sistema, sale o la máquina reinicia, se guarda una entrada en este fichero. Es un fichero binario del cual el comando `last` obtiene información en que se menciona qué usuarios entraron o salieron del sistema, cuándo y dónde se originó la conexión. Puede ser útil para buscar dónde (en qué cuentas) se originó la intrusión o detectar usos de cuentas sospechosas. También existe una variación del comando, llamada `lastb`, que lista los intentos de *login* que no pudieron validarse correctamente y se usa el fichero `/var/log/btmp` (puede ser necesario crearlo si no existe). Estos mismos fallos de autentificación también suelen enviarse al registro `auth.log`. De modo parecido, el comando `lastlog` utiliza otro fichero `/var/log/lastlog` para verificar cuál fue la última conexión de cada uno de los usuarios.

4) `/var/log/secure`: suelen utilizarse en Fedora para enviar los mensajes de los *tcp wrappers* (o los cortafuegos). Cada vez que se establece una conexión a un servicio de inetd, o bien para xinetd (con su propia seguridad), se añade un mensaje de registro a este fichero. Pueden buscarse intentos de intrusos de servicios que no suelen usarse, o bien máquinas no familiares que se intentan conectar.

En el sistema de registros, otra cosa que habría que asegurar es que el directorio de registros `/var/log` solo pueda ser escrito por el usuario root (o *deamons* asociados a servicios). De otro modo, cualquier atacante podría falsificar la información de los registros. Aun así, si el atacante consigue acceso a root, puede, y suele, borrar las pista de sus accesos.

9. Taller: análisis de la seguridad mediante herramientas

Realizaremos a continuación algunos de los procesos descritos en los capítulos previos sobre una distribución Debian (aunque de forma equivalente puede realizarse en una Fedora/Red Hat, la mayoría de procesos y métodos son equivalentes) para configurar y auditar su seguridad (observaremos diferentes salidas de los comandos, dependiendo de la versión y sistema físico que utilicemos).

Podemos comenzar con algunas comprobaciones locales. No entramos en detalles, ya que muchas de ellas se han comentado previamente, y en algunos casos son procedimientos típicos de administración local (según examinamos en el material de *Introducción a la administración*):

- Comprobar que no existan usuarios o grupos problemáticos. Podemos analizar `/etc/passwd` y `/etc/group` en búsqueda de usuarios o grupos que no correspondan a usuarios de sistema, o sean usuarios de sistema o asociados a un determinado servicio. En este último caso también es una buena forma de detectar servicios que no teníamos identificados como en funcionamiento, y que posiblemente procedan de una instalación previa sin configuración o que han dejado de ser útiles. Sería un buen momento para desinstalar tales servicios, si se sabe que no se utilizarán, ya que probablemente estén deficientemente configurados en temas de seguridad.

- Comprobar que no existan usuarios con passwords vacíos. Recordar que si está funcionando `/etc/shadow`, el segundo parámetro en `/etc/passwd` debería ser `x`. Si simplemente está vacío: `::` o en `/etc/shadow`, si está vacío sin disponer de hash (el campo con ! o * tampoco permite login). En estas dos últimas comprobaciones también sería útil aprovechar la ocasión para inhabilitar cuentas de usuario que hayamos detectado como no usadas, además de verificar las últimas cuentas de usuario añadidas, por si detectamos alguna incoherencia con nuestra actividad previa de gestión de usuarios.

- Verificar también la existencia de usuarios y grupos con ID 0 (asociados a usuarios con permisos de root).

- Verificar configuración de PAM, para comprobar que no esté alterada, en especial en aquellos módulos relacionados con el proceso de login y/o restricciones de passwords que podamos haber impuesto.

- Verificar, con comandos como `lastlog`, las últimas conexiones de los usuarios para detectar los últimos usos interactivos y desde qué lugares se han realizado.

- Comprobación de mensajes de Syslog, Rsyslog o Systemd (journalctl), para la detección de errores de sistema y autentificación.

- Verificación de la existencia de archivos `Sticky Bit` y `suid/guid`. Podemos comprobar que no se hayan generado en el sistema archivos o directorios con estos permisos.

- Comprobación de firmas de binarios (y demás ficheros). Por ejemplo, si disponemos de sumas previas, con herramientas tipo tripware o AIDE.

- Verificación de los procesos en ejecución. Comandos como `ps`, `top`.

- Configuración de sudoers `/etc/sudoers`: qué usuarios disponen de permisos especiales.

- Comprobación de trabajos de cron: `/etc/cron.daily`, y otros periodos para comprobar que no hayan sido introducidas tareas no esperadas.

- Comprobar con el sistema de paquetes si se dispone de actualizaciones nuevas en el sistema, en especial correcciones de seguridad.

En cuanto a comprobaciones para la seguridad en red, detallaremos algunos pasos con mayor detalle, pero una serie de tareas típicas en la auditación y configuración de seguridad serían:

- Verificar configuración de red (*ifconfig*), rutas disponibles *route*.

- Verificación de runlevel activo (*runlevel*), estado `/etc/inittab` o target equivalente en arranque Systemd.

- Verificación del estado de los servicios de red. ¿Cuáles?, ¿estado?

- Verificación de las conexiones de red. Comandos `netstat`.

- Comprobar la configuración de SSH (`/etc/ssh/sshd_config`) para las opciones habituales de no login para root, SSH 2 activado, entre otras.

- Verificación del estado del firewall (iptables), comprobar reglas, ¿sigue activo?

- Verificación de algunos servicios problemáticos: Samba (¿está activo y lo necesitamos?). Servicios de NTP (nos permiten disponer del sistema sincronizado).

- Verificación de los logs específicos de algunos servidores, normalmente en `/var/log/servicio`, tanto si es un fichero como un directorio con los logs de acceso y error pertinentes, por ejemplo, de servidor web o bases de datos. Las herramientas comentadas como *logcheck* o *logwatch*, nos permiten también una mejor detección a través de sus resúmenes obtenidos de los logs de los servicios y del general del sistema.

Examinamos ahora con más detalle algunas de estas fases para controlar y auditar la seguridad de nuestro sistema. En este caso lo hacemos a partir de un sistema ejemplo con una distribución Debian (en particular una versión antigua que tenía algunos problemas de seguridad en la configuración por defecto).

Primero examinaremos qué ofrece, en cuanto a servicios, nuestra máquina a la red. Para ello, utilizaremos la herramienta *nmap* como escaneador de puertos. Con el comando (desde root):

```
nmap -sTU -O localhost
```

obtenemos:

```
root@maquina:# nmap -sUT -O localhost
starting nmap 5.21 ( nmap.org ) 11:31 CEST
Interesting ports on localhost (127.0.0.1):

(The 3079 ports scanned but not shown below are in state: closed)
Port  State Service
9/tcp   open discard
9/udp   open discard
13/tcp  open daytime
22/tcp  open ssh
25/tcp  open smtp
37/tcp  open time
37/udp  open time
80/tcp  open http
111/tcp open sunrpc
111/udp open sunrpc
113/tcp open auth
631/tcp open ipp
728/udp open unknown
731/udp open netviewdm3
734/tcp open unknown

Remote operating system guess: Linux kernel 2.6.X
Uptime 2.011 days
Nmap run completed 1 IP address (1 host up) scanned in 9.404 seconds
```

Podemos observar que nos ha detectado un gran número de servicios abiertos (dependiendo de la máquina podría haber más: telnet, ftp, finger, etc.), tanto en protocolos tcp como udp. Algunos servicios, como *discard*, *daytime* o *time*, pueden ser útiles en alguna ocasión, pero normalmente no tendrían que estar abiertos a red, ya que se consideran inseguros. SMTP es el servicio de reenvío y enrutamiento del correo electrónico; si actuamos como *host* o servidor de correo, tendría que estar activo, pero si solo leemos y escribimos correo mediante cuentas POP3 o IMAP, no tiene por qué estarlo.

Otra manera de detectar servicios activos sería mediante la búsqueda de puertos activos a la escucha, detectando las conexiones de red activas; esto puede hacerse con el comando `netstat -lut`.

El comando `nmap` también puede aplicarse con el nombre DNS o IP de la máquina; así vemos lo que se vería desde el exterior (con localhost vemos lo que puede ver la propia máquina) o, mejor incluso, podríamos utilizar una máquina de una red externa (por ejemplo, un PC cualquiera conectado a Internet) para examinar lo que verían de nuestra máquina desde el exterior. En los primeros casos no estaremos cortando las conexiones por el cortafuegos si este existiese, solo desde el exterior podremos comprobar si este efectivamente cierra los puertos esperados.

Vamos ahora a `/etc/inetd.conf` para desactivar estos servicios (si han aparecido o no en el escaneo previo, dependerá de la distribución GNU/Linux y de la configuración previa de estos servicios, o puede que `inetd` ya no esté activo en las últimas distribuciones). Buscamos líneas como:

```
discard stream tcp nowait root internal
smtp stream tcp nowait mail /usr/sbin/exim exim -bs
```

y les colocamos un # al principio (solo para aquellos servicios que queramos desactivar y sepamos qué hacen realmente –consúltese las páginas man– o se recomiende su desactivación). Otro caso de desactivación especialmente recomendada sería el de los servicios de ftp, telnet, finger, etc. y usar `ssh` para sustituirlos.

Ahora tenemos que reiniciar `inetd` para que vuelva a leer la configuración que hemos cambiado: `/etc/init.d/inetd restart`.

Volvemos a `nmap`:

```
22/tcp  open ssh
80/tcp  open http
111/tcp open sunrpc
111/udp open sunrpc
113/tcp open auth
631/tcp open ipp
728/udp open unknown
734/tcp open unknown
```

De lo que nos queda, tenemos el servicio ssh, que queremos dejar activo, y el servidor de web, que lo pararemos de momento:

```
/etc/init.d/apache2 stop
```

ipp es el servicio de impresión asociado a CUPS. En administración local vimos que había una interfaz web de CUPS que se conectaba al puerto 631. Si queremos tener una idea de a qué se dedica un puerto determinado, podemos mirar en `/etc/services`:

```
root@maquina:# grep 631 /etc/services
ipp 631/tcp # Internet Printing Protocol
ipp 631/udp # Internet Printing Protocol
```

Si no estamos actuando como servidor de impresión hacia el exterior, tenemos que ir a la configuración de CUPS y eliminar esa prestación (por ejemplo, colocando un *listen* 127.0.0.1:631, para que solo escuche la máquina local) o limitar el acceso a las máquinas permitidas.

Nos aparecen también algunos puertos como desconocidos, en este caso el 728 y 734; esto indica que nmap no ha podido determinar qué servicio está asociado al puerto. Vamos a intentar comprobarlo directamente. Para ello, ejecutamos sobre el sistema el comando `netstat`, que ofrece diferentes estadísticas del sistema de red, desde paquetes enviados y recibidos, y errores, hasta lo que nos interesa, que son las conexiones abiertas y quién las usa. Intentemos buscar quién está usando los puertos desconocidos:

```
root@maquina: # netstat -anp | grep 728
udp 0 0 0.0.0.0:728 0.0.0.0:* 552/rpc.statd
```

Y si hacemos lo mismo con el 734, observamos también que quien ha abierto el puerto ha sido rpc.statd, que es un *daemon* asociado a NFS (en este caso el sistema tiene un servidor NFS). Si hacemos este mismo proceso con los puertos 111 que aparecían como `sunrpc`, observaremos que el *daemon* que hay detrás es portmap, que se usa en el sistema de llamadas RPC. El sistema de llamadas RPC (*remote procedure call*) permite utilizar el mecanismo de llamadas remotas entre dos procesos que están en diferentes máquinas. portmap es un *daemon* que se encarga de traducir las llamadas que le llegan por el puerto a los números de servicios RPC internos que se tengan, y es utilizado por diferentes servidores, como NFS, NIS y NIS+.

Los servicios RPC que se ofrecen se pueden ver con el comando `rpcinfo`:

```
root@maquina:# rpcinfo -p
programa  vers proto  puerto
100000  2 tcp 111  portmapper
100000  2 udp 111  portmapper
100024  1 udp 731  status
```

```
100024  1 tcp 734  status
391002  1 tcp 39797 sgi_fam
391002  2 tcp 39797 sgi_fam
```

donde observamos los servicios RPC con algunos de los puertos que ya se habían detectado. Otro comando que puede resultar útil es `lsof`, que, entre otras funciones, permite relacionar puertos con los servicios que los han abierto (por ejemplo: `lsof -i | grep 731`).

El *daemon* portmap es un poco crítico con la seguridad, ya que, en principio, no ofrece mecanismos de autentificación del cliente, pues se supone que se delegan en el servicio (NFS, NIS, etc.). Por lo tanto, portmap podría ser víctima de intentos de DoS/DDoS que podrían provocar errores en los servicios o hacerlos caer. Normalmente, protegeremos portmap por medio de algún tipo de *wrapper* y/o cortafuegos. Si no utilizamos, y no tenemos previsto utilizar, servicios NFS y NIS, lo mejor es desactivar completamente portmap, quitándolo del *runlevel* en que se active. También podemos pararlos momentáneamente con los *scripts* (en Debian):

```
/etc/init.d/nfs-common
/etc/init.d/nfs-kernel-server
/etc/init.d/portmap
```

dándoles el parámetro `stop` para parar los servicios RPC (en este caso NFS). En todos estos casos sólo estamos parando los servicios para la ejecución actual; si queremos desactivar realmente estos servicios, hemos de ir al nivel de ejecución concreto (*runlevel*) y desactivar los servicios (como se ha comentado en el módulo), porque si no se reiniciarán en el siguiente arranque del sistema.

A continuación, controlaremos la seguridad en base a un *wrapper* sencillo. Vamos a suponer que queremos dejar paso a través de ssh de una máquina determinada, llamémosla 1.2.3.4 (dirección IP). Vamos a cerrar portmap al exterior, ya que no tenemos NIS, y de NFS tenemos servidor pero no estamos sirviendo nada (podríamos cerrarlo, pero lo dejaremos para futuros usos). Vamos a hacer un *wrapper* (suponemos que los *TCP wrappers* están ya instalados, pueden ser necesarios *tcpd y libwrap*), modificando los ficheros `hosts.deny` y `hosts.allow`. En `/etc/hosts.deny`:

```
ALL : ALL : spawn (/usr/sbin/safe_finger -l @%h \
| /usr/bin/mail -s "%c FAILED ACCESS TO %d!!" root) &
```

estamos denegando todos los servicios (cuidado, aquellos relacionados con inetd) (primer `all`) a todos (`all`), y la acción por tomar será averiguar quién

ha pedido el servicio (solo funcionará si la máquina soporta *finger*) y en qué máquina, y vamos a enviar un correo al root informando del intento. Podríamos también escribir un fichero de registro. Ahora, en `/etc/hosts.allow`:

```
sshd: 1.2.3.4
```

habilitamos el acceso por la máquina IP 1.2.3.4 en el servidor sshd (del ssh). Podemos colocar una lista de máquinas o bien subredes que puedan utilizar el servicio (véase `man hosts.allow`). Recordad que también tenemos los comandos `tcpdchk`, para comprobar que la configuración del *wrapper* sea correcta, y `tcpdmatch`, para simular qué pasaría con un determinado intento. Por ejemplo:

```
root@maquina:# tcpdmatch sshd 1.2.3.4

warning: sshd: no such process name in
   /etc/inetd.conf client: hostname maquina.dominio.es
client: address 1.2.3.4
server: process sshd
matched: /etc/hosts.allow line 13
access: granted
```

nos comenta que sería concedido el acceso. Un detalle es que nos explicita que sshd no está en el `inetd.conf` y, si lo verificamos, vemos que así es: no se activa por el servidor inetd, sino por *daemon* propio (sshd) en el *runlevel* en que estemos. Además (en Debian), este es un caso de un demonio que está compilado con las bibliotecas de *wrappers* incluidas. En Debian hay varios *daemons* que tienen este soporte de *wrappers* al compilarse con las librerías pertinentes como: rpc.statd, rpc.mountd, rpcbind, sshd entre otros. Esto permite asegurar estos *daemons* mediante *wrappers* por los ficheros *hosts* mencionados.

Otra cuestión por verificar son las conexiones actuales existentes. Con el comando `netstat -utp` podemos listar las conexiones tcp, u udp establecidas con el exterior, ya sean entrantes o salientes; así en cualquier momento podemos detectar los clientes conectados y a quién estamos conectados. Otro comando importante (de múltiples funciones) es `lsof`, que puede relacionar ficheros abiertos con procesos o conexiones por red establecidas mediante `lsof -i`, pudiéndose así detectar accesos indebidos a ficheros.

También podríamos utilizar un cortafuegos para procesos similares (o bien como mecanismo añadido). Comenzaremos viendo cómo están las reglas del cortafuegos en este momento (comando `iptables -L`):

```
root@aopcjj:# iptables -L
Chain INPUT (policy ACCEPT)
```

```
target prot opt source destination
Chain FORWARD (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

O sea, que el cortafuegos no está colocando ninguna restricción en este momento, y permite la entrada, salida y reenvío de todos los paquetes.

En este punto, podríamos añadir un cortafuegos que nos permitiese una gestión más adecuada de paquetes que recibimos y enviamos, y que sería un control previo para mejorar la seguridad. Dependiendo de nuestras necesidades, estableceríamos las reglas necesarias de modo parecido a las que comentamos en los ejemplos de cortafuegos de la unidad.

En caso de colocar activo algún cortafuegos, podemos considerar si usar este mecanismo como única garantía y quitar los *wrappers*: podría hacerse, ya que los cortafuegos (en este caso mediante iptables) ofrecen un mecanismo muy potente que nos permite seguir a un paquete por tipo, por protocolo y por lo que está haciendo en el sistema. Un buen cortafuegos podría ser más o menos suficiente, pero, por si acaso, más medidas de seguridad no vienen mal. Y en caso de que el cortafuegos no estuviese bien diseñado y dejase escapar algunos paquetes o *hosts*, el *wrapper* sería la medida, a nivel de servicio, para parar los accesos no deseados. Por poner una metáfora que se suele utilizar, si nos planteásemos nuestro sistema como la defensa de un castillo medieval, el foso y primeras murallas serían el cortafuegos, y la segunda muralla de contención, los *wrappers*.

Las siguientes medidas ya podrían venir directamente de cada servicio (web, correo, impresión, etc.), con las opciones de seguridad que ofrezca de forma adicional, ya sea mediante autentificación de sus clientes, mediante limitación de los accesos por perfiles o ACL o simplemente al ofrecer un subconjunto de funcionalidades requeridas. Veremos en diferentes módulos orientados a servicios concretos algunas de estas medidas a nivel servicio.

Como opciones adicionales para proteger a nuestro sistema, podriamos hacer intervenir a algunas de las herramientas vistas en capítulos anteriores, por ejemplo, para mediante *Denyhosts* y *Fail2ban*, con los procedimientos vistos en las secciones 63 y 6.3, proteger las conexiones SSH, y en el segundo caso otros servicios como servidor web, correo, samba, u otros que tengamos activos. Como siguiente paso podríamos colocar una herramienta de detección de vulnerabilidades, como OpenVAS (ver sección 6.1), para obtener un informe del estado final de nuestro servidor.

Resumen

En este módulo hemos examinado los conceptos básicos de seguridad aplicables a los sistemas GNU/Linux, y hemos identificado los posibles tipos de los ataques, tanto locales como a sistemas en red.

El conocimiento del proceso de los ataques nos permite tomar acciones de seguridad activa mediante herramientas de detección de intrusiones, así como de prevención de posibles situaciones problemáticas.

Sistemas como SELinux nos permiten políticas de seguridad, altamente especificadas, y nos ofrecen una amplia sintaxis de permisos, controles y prevención activa de la seguridad del sistema.

La seguridad debe examinarse tanto desde el punto de vista local al sistema, lo que incluye la seguridad física de acceso al mismo, como desde el punto de vista de los sistemas en red.

La utilización de herramientas de seguridad en las diferentes áreas de prevención, detección y actuación nos permite un control activo de seguridad, que nos puede evitar males mayores en nuestros sistemas.

También observamos las limitaciones de la seguridad de los sistemas informáticos y, en especial, las posibles sensaciones de una falsa seguridad total (difícil o imposible de obtener), que nos puede llevar a una confianza ciega, con peores resultados que no tenerla. La seguridad es un proceso activo, que necesita un seguimiento constante y participativo por parte del administrador de sistemas.

Actividades

1. Supongamos que colocamos en nuestra máquina un sitio web, por ejemplo con Apache. Nuestro sitio está pensado para diez usuarios internos, pero no controlamos este número. Más adelante nos planteamos poner en Internet este sistema, ya que creemos que puede ser útil para los clientes, y lo único que hacemos es poner el sistema con una IP pública en Internet. ¿Qué tipo de ataques podría sufrir este sistema?

2. ¿Cómo podemos detectar los ficheros con suid en nuestro sistema? ¿Qué comandos serán necesarios? ¿Y los directorios con SUID o SGID? ¿Por qué es necesario, por ejemplo, que `/usr/bin/passwd` tenga bit de SUID?

3. Los ficheros `.rhosts`, como hemos visto, son un peligro importante para la seguridad. ¿Podríamos utilizar algún método automático que comprobase su existencia periódicamente? ¿Cómo?

4. ¿Cómo crearíamos un ambiente chroot para `/bin/bash`? Describir los pasos necesarios.

5. Supongamos que queremos deshabilitar un servicio del cual sabemos que tiene el *script* `/etc/init.d/servicio` que lo controla: queremos desactivarlo en todos los *runlevels* en los que se presenta. ¿Cómo encontramos los *runlevels* en los que está? (por ejemplo, buscando enlaces al *script*) ¿Cómo lo haríamos en distribuciones que soportan Systemd?

6. Examinad los servicios en activo en vuestra máquina. ¿Son todos necesarios? ¿Cómo habría que protegerlos o desactivarlos?

7. Practicad el uso de algunas de las herramientas de seguridad descritas (nmap, chkrootkit, wireshark, OpenVas, etc.).

8. ¿Qué reglas iptables serían necesarias para una máquina en la que solo queremos acceso por SSH desde unas máquinas habituales concretas?

9. ¿Y si queremos únicamente un acceso al servidor web?

Bibliografía

[Aus] **CERT Australia**. *Australian CERT*. <http://www.auscert.org.au>

[Bur02] **Burgiss, H.** *Security QuickStart HOWTO for Linux*. The Linux Documentation Project. 2002.

[Cera] **CERT**. *CERT site*. <http://www.cert.org>

[Cerb] **CERT**. *CERT Vulnerability Database*. <http://www.kb.cert.org/vuls>

[Deb] **Debian**. *Sitio de seguridad de Debian*. <http://www.debian.org/security>

[Fbi] **Federal Bureau of Intelligence**. *Brigada del FBI para ciberdelitos*. <http://www.fbi.gov/about-us/investigate/cyber/cyber>

[Fen02] Kevin Fenzi. *Linux security HOWTO*. The Linux Documentation Project.

[Fri02] **Frisch, A.** *Essential System Administration* (3.ª ed.). O'Reilly, 2002.

[Gre] **Grennan, M.**. *Firewall and Proxy Server HOWTO*. The Linux Documentation Project.

[Hat08] **Hatch, B.** *Hacking Linux Exposed* 3rd Edition, McGraw-Hill, 2008.

[Hatb] **Red Hat** *Red Hat RHEL 7 Security Guide. 2014*. <https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/index.html>

[Hatc] **Red Hat** *Sitio de seguridad de Red Hat, Bases de datos de vulnerabilidades*. <https://access.redhat.com/security/cve/>

[Hatd] **Red Hat** *Utilización firmas GPG en Red Hat*. <https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html-single/System_Administrators_Guide/#s1-check-rpm-sig>

[Her13] **Hertzog, R; and Mas, R.** *El libro del administrador de Debian.*. 2013. Disponible en <http://debian-handbook.info/browse/es-ES/stable/index.html>

[Him01] **Himanen, P.** *La ética del hacker y el espíritu de la era de la información*. Destino. 2001.

[Incb] **Internet Storm Center**. *Analisis de Vulnerabilidades, y algunos incidentes. 2014.* <https://isc.sans.edu//>

[Ins] **Insecure.org** *Vulnerabilidades y exploits. 1998.* <http://www.insecure.org/sploits.html>

[Insa] **Insecure.org**. *Insecure.org site. 2014.* <http://www.insecure.org>

[Insb] **Insecure.org** *Nmap Home site. 2003.* <http://nmap.org/index.html>

[Line] **Linuxsecurity.com**. *Linux Security Reference Card. 2002.* <http://www.linuxsecurity.com/docs/QuickRefCard.pdf>

[Mor03] **Morill, D.** *Configuración de sistemas Linux*. Anaya Multimedia. 2003

[Mou02] **Mourani, G.** *Securing and Optimizing Linux: The Ultimate Solution. v2.0.*. The Linux Documentation Project. 2002.

[Nes] **Nessus.org**. *Nessus*. <http://www.nessus.org>

[Net] **Netfilter.org.** *Proyecto Netfilter/IPtables*. <http://www.netfilter.org>

[Neu] **Neufeld, C.**. *Setting Up Your New Domain MiniHOWTO*. The Linux Documentation Project.

[Nsaa] **NSA**. *NIST site*.
<http://csrc.nist.gov>

[Nsab] **NSA** *Security Enhanced Linux*. 2009.
<http://www.nsa.gov/research/selinux/index.shtml>

[Opv] **OpenVas**. *OpenVas vulnerability Scanner.*
<http://openvas.org>

[Pen] **Fernández-Sanguino Peña, J.** *Securing Debian Manual*. 2013.
<http://www.debian.org/doc/manuals/securing-debian-howto/>

[Ray01] **Ray, John**. *Maximum Linux Security: A Hacker's Guide to Protecting. 2nd Edition, Sams, 2011*.

[San] **Sans**. *Top20 de controles críticos de seguridad.* 2014.
<http://www.sans.org/critical-security-controls/>

[Sei] **Seifried, K.** *Securing Linux, Step by Step*. 2002.
<http://seifried.org/security/os/linux/20020324-securing-linux-step-by-step.html>

[Sno] **Snort.org**. *Snort*.
<http://www.snort.org>

[Usa] **The United States Department of Justice**. *División del Departamento de Justicia de Estados Unidos para el ciberdelito*.
<http://www.usdoj.gov/criminal/cybercrime/>

Sobre estas fuentes de referencia e información:

[Kerb] Sitio que proporciona un repositorio de las diversas versiones del núcleo Linux y sus parches.

[Kera] [lkm] Sitios web que recogen una parte de la comunidad del núcleo de Linux. Dispone de varios recursos de documentación y listas de correo de la evolución del núcleo, su estabilidad y las nuevas prestaciones que se desarrollan.

[Debk] Es un manual imprescindible sobre los procesos de compilación del núcleo en la distribución Debian. Suele actualizarse con los cambios producidos en la distribución. [Fedk] aporta una referencia similar para el caso Fedora.

[Ces06] Libro sobre el núcleo de Linux 2.4, que detalla los diferentes componentes y su implementación y diseño. Existe una primera edición sobre el núcleo 2.2 y una nueva actualización al núcleo 2.6. [Lov10] es un texto alternativo más actualizado. Para la rama 3.x estos libros siguen siendo útiles, ya que la mayoría de conceptos del kernel se mantienen inalterados.

[Lov10] Love, R. *Linux Kernel Development*. 3rd Edition. 2010. Addison-Wesley.

[Pra03] Artículo que describe algunas de las principales novedades de la rama 2.6 del núcleo Linux. [Pra11] es el equivalente para 3.0+.

[Pra11] Pranevich, J. (2011). *The Wonderful World of Linux 3.0*.
<http://www.kniggit.net/wwol30/>

[Ker] [Mur] Proyectos de documentación del núcleo, incompletos pero con material útil.

[Bac86] [Vah96] [Tan87] [Tan86] Algunos textos sobre los conceptos, diseño e implementación de los núcleos de diferentes versiones UNIX y Linux.

[Skoa][Zan01][Kan][Grub1][Grub2] Recursos para tener más información sobre los cargadores LiLo, Grub y Grub2.

[Gru2][Grub1] Sitios oficiales de Grub2 y el anterior original Grub (ahora conocido como Grub Legacy.)

Sintonización, optimización y alta disponibilidad

Remo Suppi Boldrito

PID_00212471

Universitat Oberta de Catalunya

Índice

Introducción 5

Objetivos 7

1. Sintonización, optimización y alta disponibilidad 9
1.1. Aspectos básicos 9
1.1.1. Monitorización sobre UNIX System V 10
1.1.2. Optimización del sistema 17
1.1.3. Optimizaciones de carácter general 20
1.1.4. Configuraciones complementarias 21
1.1.5. Resumen de acciones para mejorar un sistema 25
1.2. Monitorización 27
1.2.1. Munin 28
1.2.2. Monit 29
1.2.3. SNMP + MRTG 30
1.2.4. Nagios 33
1.2.5. Ganglia 35
1.2.6. Otras herramientas 37
1.3. Alta disponibilidad en Linux (High-Availability Linux) 38
1.3.1. Guía breve de instalación de Heartbeat y Pacemaker (Debian) 39
1.3.2. DRBD 43
1.3.3. DRBD + Heartbeat como NFS de alta disponibilidad 45

Actividades 48

Bibliografía 49

Introducción

Un aspecto fundamental, una vez que el sistema está instalado, es la configuración y adaptación del sistema a las necesidades del usuario y que las prestaciones del sistema sean lo más adecuadas posible a las necesidades que de él se demandan. GNU/Linux es un sistema operativo eficiente que permite un grado de configuración excelente y una optimización muy delicada de acuerdo a las necesidades del usuario. Es por ello que, una vez realizada una instalación (o en algunos casos una actualización), deben hacerse determinadas configuraciones vitales en el sistema. Si bien el sistema "funciona", es necesario efectuar algunos cambios (adaptación al entorno o sintonización) para permitir que estén cubiertas todas las necesidades del usuario y de los servicios que presta la máquina. Esta sintonización dependerá de dónde se encuentre funcionando la máquina y en algunos casos se realizará para mejorar el rendimiento del sistema, mientras que en otros (además), por cuestiones seguridad. Cuando el sistema está en funcionamiento, es necesario monitorizarlo para ver su comportamiento y actuar en consecuencia. Si bien es un aspecto fundamental, la sintonización de un sistema operativo muchas veces se relega a la opinión de expertos o gurús de la informática; pero conociendo los parámetros que afectan al rendimiento, es posible llegar a buenas soluciones haciendo un proceso cíclico de análisis, cambio de configuración, monitorización y ajustes.

Por otro lado con las necesidades de servicios actuales, los usuarios son muy exigentes con la calidad de servicio que se obtiene y es por ello que un administrador debe prevenir las incidencias que puedan ocurrir en el sistema antes de que las mismas ocurran. Para ello es necesario que el administrador "vigile" de forma continuada determinados parámetros de comportamiento de los sistemas que le puedan ayudar en la toma de decisiones y actuar "en avance" evitando que se produzca el error, ya que si esto pasara podría suponer la posibilidad de que el sistema dejara de funcionar con las posibles consecuencias económicas en algunos casos, pero casi siempre con el deterioro de la imagen del empresa/institución que esto conlleva (a nadie le agradará -o le debería agradar- que los usuarios le informen de un fallo en sus sistemas de información).

En resumen, un sistema de monitorización debe ayudar al administrador a reducir el MTTR (*Mean time to recovery*) que indica la media de tiempo que un sistema tarda en recuperarse de un fallo y que puede tener un valor dentro del contrato de calidad de servicio (normalmente llamado SLA *Service Level Agreement*) por lo cual también puede tener consecuencia económicas. Este valor a veces se le llama *"mean time to replace/repair/recover/resolve"* en función

de qué sistema se trate y qué SLA se haya acordado, pero en todos estamos hablando de la ventana de tiempo entre la cual se detecta un problema y las acciones para solucionarlo. Con la monitorización podremos tener indicios de estos problemas y ejecutar las acciones para que no lleguen a más y que si ocurren, tengan el MTTR más bajo posible.

Ved también

La seguridad se estudia en el módulo "Administración de seguridad".

En este módulo se verán las principales herramientas para monitorizar un sistema GNU/Linux, como son Munin, Monit, MRTG, Ganglia, Nagios, Cactis o Zabbix y se darán indicaciones de cómo sintonizar el sistema a partir de la información obtenida.

Es importante notar que si el MTTR es cercano a cero, el sistema tendrá que tener redundancia en los otros los sistemas y es un aspecto importante en la actualidad para los servidores de sistemas de la información, que se conoce como alta disponibilidad.

La alta disponibilidad (*high availability*) es un protocolo de diseño del sistema y su implementación asociada asegura un cierto grado absoluto de continuidad operacional durante períodos largos de tiempo. El término *disponibilidad* se refiere a la habilidad de la comunidad de usuarios para acceder al sistema, enviar nuevos trabajos, actualizar o alterar trabajos existentes o recoger los resultados de trabajos previos. Si un usuario no puede acceder al sistema se dice que está *no disponible*.

De entre todas las herramientas que existen para tratar estos aspectos (Heartbeat, ldirectord para LVS -Linux Virtual Server-, OpenSAF, Piranha, UltraMonkey, Pacemaker+Corosync, o Kimberlite (obs:EOL), etc.), en este módulo analizaremos cómo se desarrolla una arquitectura redundante en servicios utilizando Heartbeat+DRBD.

Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Analizar y determinar las posibles pérdidas de prestaciones de un sistema.
2. Solucionar problemas de sintonización del sistema.
3. Instalar y analizar las diferentes herramientas de monitorización y su integración para resolver los problemas de eficiencias/disponibilidad.
4. Analizar las herramientas que permiten tener un sistema en alta disponibilidad.

1. Sintonización, optimización y alta disponibilidad

1.1. Aspectos básicos

Antes de conocer cuáles son las técnicas de optimización, es necesario enumerar las causas que pueden afectar a las prestaciones de un sistema operativo [31]. Entre estas, se pueden mencionar:

1) Cuellos de botella en los recursos: la consecuencia es que todo el sistema irá más lento porque existen recursos que no pueden satisfacer la demanda a la que se les somete. El primer paso para optimizar el sistema es encontrar estos cuellos de botella y determinar por qué ocurren, conociendo sus limitaciones teóricas y prácticas.

2) Ley de Amdahl: según esta ley, "hay un límite de cuánto puede uno mejorar en velocidad una cosa si solo se optimiza una parte de ella"; es decir, si se tiene un programa que utiliza el 10 % de la CPU y se optimiza reduciendo la utilización en un factor 2, el programa mejorará sus prestaciones (*speedup*) en un 5 %, lo cual puede significar un tremendo esfuerzo no compensado por los resultados.

3) Estimación del *speedup*: es necesario estimar cuánto mejorará las prestaciones el sistema para evitar esfuerzos y costes innecesarios. Se puede utilizar la ley de Amdahl para valorar si es necesaria una inversión, en tiempo o económica, en el sistema.

4) Efecto burbuja: siempre se tiene la sensación de que cuando se encuentra la solución a un problema, surge otro. Una manifestación de este problema es que el sistema se mueve constantemente entre problemas de CPU y problemas de entrada/salida, y viceversa.

5) Tiempo de repuesta frente a cantidad de trabajo: si se cuenta con veinte usuarios, mejorar en la productividad significará que todos tendrán más trabajo hecho al mismo tiempo, pero no mejores respuestas individualmente; podría ser que el tiempo de respuesta para algunos fuera mejor que para otros. Mejorar el tiempo de respuesta significa optimizar el sistema para que las tareas individuales tarden lo menos posible.

6) Psicología del usuario: dos parámetros son fundamentales:

a) el usuario generalmente estará insatisfecho cuando se produzcan variaciones en el tiempo de respuesta; y

b) el usuario no detectará mejoras en el tiempo de ejecución menores del 20 %.

7) Efecto prueba: las medidas de monitorización afectan a las propias medidas. Se debe ir con cuidado cuando se realizan las pruebas por los efectos colaterales de los propios programas de medida.

8) Importancia de la media y la variación: se deben tener en cuenta los resultados, ya que si se obtiene una media de utilización de CPU del 50 % cuando ha sido utilizada 100, 0, 0, 100, se podría llegar a conclusiones erróneas. Es importante ver la variación sobre la media.

9) Conocimientos básicos sobre el hardware del sistema a optimizar: para mejorar una cosa es necesario "conocer" si es susceptible de mejora. El encargado de la optimización deberá conocer básicamente el hardware subyacente (CPU, memorias, buses, caché, entrada/salida, discos, vídeo, etc.) y su interconexión para poder determinar dónde están los problemas.

10) Conocimientos básicos sobre el sistema operativo a optimizar: del mismo modo que en el punto anterior, el usuario deberá conocer aspectos mínimos sobre el sistema operativo que pretende optimizar, entre los cuales se incluyen conceptos como procesos e hilos o *threads* (creación, ejecución, estados, prioridades, terminación), llamadas al sistema, *buffers* de caché, sistema de archivos, administración de memoria y memoria virtual (paginación, *swap*) y tablas del núcleo (*kernel*).

1.1.1. Monitorización sobre UNIX System V

El directorio `/proc` lo veremos como un directorio, pero en realidad es un sistema de archivos ficticio llamado procfs (y que se monta en tiempo de *boot* de la máquina), es decir, no existe sobre el disco y el núcleo lo crea en memoria. Se utiliza para proveer de información sobre el sistema (originalmente sobre procesos, de aquí el nombre), información que luego será utilizada por todos los comandos que veremos a continuación. El `/proc` actúa como interfaz a la estructura de datos internos de núcleo y puede ser utilizado para cambiar cierta información del kernel en tiempo de ejecución (sysctl). No debemos confundir **procfs** con el **sysfs** ya que este último exporta información sobre los dispositivos y sus controladores desde el modelo de dispositivos del núcleo hacia el espacio del usuario (y es utilizado por algunas partes importantes del sistema como el udev que es el que crea por compatibilidad el /dev) permitiendo obtener parámetros y configurar alguno de ellos (p.ej., saber el tamaño de un disco `cat /sys/block/sda/size` o que dispositivos tenemos en /sys/class). Una vista de este directorio es:

```
bus          locks           cgroups         meminfo
cmdline      misc            consoles        modules
cpuinfo      mounts          crypto          mtrr
devices      net             diskstats       pagetypeinfo
dma          partitions      dri             sched_debug
driver       self            execdomains     slabinfo
fb           softirqs        filesystems     stat
fs           swaps           interrupts      sys
iomem        sysrq-trigger   ioports         sysvipc
```

```
irq          timer_list      kallsyms        timer_stats
kcore        tty             keys            uptime
key-users    version         kmsg            vmallocinfo
acpi         kpagecount      vmstat          asound
kpageflags   zoneinfo        buddyinfo       loadavg

Además de una serie de directorios numéricos que corresponde a cada uno
de los procesos del sistema.
```

En el directorio `/proc` existen un conjunto de archivos y directorios con diferente información. A continuación veremos algunos de los más interesantes*:

***Consúltese la página del manual para obtener más información.**

`/proc/1`: un directorio con la información del proceso 1 (el número del directorio es el PID del proceso).
`/proc/cpuinfo`: información sobre la CPU (tipo, marca, modelo, prestaciones, etc.).
`/proc/devices`: lista de dispositivos configurados en el núcleo.
`/proc/dma`: canales de DMA utilizados en ese momento.
`/proc/filesystems`: sistemas de archivos configurados en el núcleo.
`/proc/interrupts`: muestra qué interrupciones están en uso y cuántas de ellas se han procesado.
`/proc/ioports`: ídem con los puertos.
`/proc/kcore`: imagen de la memoria física del sistema.
`/proc/kmsg`: mensajes generados por el núcleo, que luego son enviados a syslog.
`/proc/ksyms`: tabla de símbolos del núcleo.
`/proc/loadavg`: carga del sistema.
`/proc/meminfo`: información sobre la utilización de memoria.
`/proc/modules`: módulos cargados por el núcleo.
`/proc/net`: información sobre los protocolos de red.
`/proc/stat`: estadísticas sobre el sistema.
`/proc/uptime`: desde cuándo el sistema está funcionando.
`/proc/version`: versión del núcleo.

Estos archivos se construyen de forma dinámica cada vez que se visualiza el contenido y el núcleo del sistema operativo los provee en tiempo real. Es por ello que se denomina *sistema de archivos virtual* y el contenido de los archivos y directorios va cambiando en forma dinámica con los datos actualizados. De este modo se puede considerar el `/proc/` como una interfaz entre el núcleo de Linux y el usuario y es una forma sin ambigüedades y homogénea de presentar información interna y puede ser utilizada para las diversas herramientas/comandos de información/sintonización/control que utilizaremos regularmente. Es interesante, por ejemplo, ver la salida de comando `mount` y el resultado de la ejecución `more /proc/mounts`: es totalmente equivalente!

Ved también

En los siguientes subapartados se enseñará cómo obtener y modificar la información del núcleo de Linux trabajando con el sistema de archivos `/proc`.

Se debe tener en cuenta que estos archivos son visibles (texto), pero algunas veces los datos están "en crudo" y son necesarios comandos para interpretarlos, que serán los que veremos a continuación. Los sistemas compatibles UNIX SV utilizan los comandos `sar` y `sadc` para obtener estadísticas del sistema. En Debian es `atsar` (y `atsadc`), que es totalmente equivalente a los que hemos mencionado y posee un conjunto de parámetros que nos permiten obtener información de todos los contadores e información sin procesar del `/proc`. Debian también incluye el paquete **sysstat** que contiene los comandos `sar` (información general del la actividad del sistema), `iostat` (utilización CPU y de E/S), `mpstat` (informes globales por procesador), `pidstat` (estadísticas de procesos), `sadf` (muestra información del `sar` en varios formatos). El comando `atsar` lee contadores y estadísticas del fichero `/proc` y las muestra por

la salida estándar. La primera forma de llamar al comando es (ejecutarlo como root o agregar el usuario a la categoría correspondiente del `sudoers` para ejecutar con el `sudo`):

```
atsar opciones t [n]n
```

Donde muestra la actividad en `n` veces cada `t` segundos con una cabecera que muestra los contadores de actividad (el valor por defecto de $n = 1$). La segunda forma de llamarlo es:

```
atsar –opciones –s time –e time –i sec –f file –n day#
```

El comando extrae datos del archivo especificado por `–f` (que por defecto es `/var/log/atsar/atsarxx`, siendo `xx` el día del mes) y que fueron previamente guardados por `atsadc` (se utiliza para recoger los datos, salvarlos y procesarlos y en Debian está en `/usr/lib/atsar`). El parámetro `–n` puede ser utilizado para indicar el día del mes y `–s`, `–e` la hora de inicio y final, respectivamente. Para activar `atsadc`, por ejemplo, se podría incluir en `/etc/cron.d/atsar` una línea como la siguiente:

```
@reboot root test -x /usr/lib/atsadc && /usr/lib/atsar/atsadc /var/log/atsar/atsa'date +%d'
10,20,30,40,50 * * * * root test -x /usr/lib/atsar/atsa1 && /usr/lib/atsar/atsa1
```

La primera línea crea el archivo después de un reinicio y la segunda guarda los datos cada 10 minutos con el *shell script* `atsa1`, que llama al `atsadc`. En `atsar` (o `sar`), las opciones se utilizan para indicar qué contadores hay que mostrar y algunos de ellos son:

Opciones	Descripción
u	Utilización de CPU
d	Actividad de disco
I (i)	Número de interrupciones/s
v	Utilización de tablas en el núcleo
y	Estadísticas de utilización de ttys
p	Información de paginación y actividad de *swap*
r	Memoria libre y ocupación de *swap*
I (L)	Estadísticas de red
L	Información de errores de red
w	Estadísticas de conexiones IP
t	Estadísticas de TCP
U	Estadísticas de UDP
m	Estadísticas de ICMP
N	Estadísticas de NFS
A	Todas las opciones

Entre `atsar` y `sar` solo existen algunas diferencias en cuanto a la manera de mostrar los datos y `sar` incluye unas cuantas opciones más (o diferentes). A continuación se verán algunos ejemplos de utilización de `sar` (exactamente igual que con `atsar`, solo puede haber alguna diferencia en la visualización de los datos) y el significado de la información que genera:

Utilización de CPU: `sar -u 4 5`

```
Linux  debian  2.6.26-2-686  #1 SMP Thu May 28 15:39:35 UTC 2009  i686  11/30/2010
05:32:54  cpu %usr   %nice   %sys %irq %softirq    %wait %idle            _cpu_
05:33:05  all    3      0      8    0        0       88     0
05:33:09  all    4      0     12    0        0       84     0
05:33:14  all   15      0     19    1        0       65     0
...
05:41:09  all    0      0      1    0        0        0    99
```

`%usr` y `%sys` muestran el porcentaje de tiempo de CPU en el modo usuario con `nice=0` (normales) y en el modo núcleo. `idle` indica el tiempo no utilizado de CPU por los procesos en estado de espera (no incluye espera de disco). `wait` es el tiempo que la CPU ha estado libre cuando el sistema estaba realizando entrada o salida (por ejemplo de disco). `irq` y `sofirq` es el tiempo que la CPU ha dedicado a gestionar las interrupciones, que es un mecanismo de sincronización entre lo que hace la CPU y los dispositivos de entrada y salida. En el caso `idle=99 %` significa que la CPU está ociosa, por lo que no hay procesos por ejecutar y la carga es baja; si `idle` $\simeq$ y el número de procesos es elevado, debería pensarse en optimizar la CPU, ya que podría ser el cuello de botella del sistema. En el ejemplo podemos ver que hay poca utilización de CPU y mucho uso de entrada y salida, por lo cual se puede verificar que en este caso la carga del sistema la está generando el disco (para el ejemplo se habían abierto 5 copias del programa OpenOffice Writer).

Número de interrupciones por segundo: `sar -I 4 5`

```
Linux  debian  2.6.26-2-686  #1 SMP Thu May 28 15:39:35 UTC 2009  i686  11/30/2010
05:46:30  cpu  iq00 iq01 iq05 iq08 iq09 iq10 iq11 iq12 iq14 iq15       _intr/s_
05:46:34  all     0    0    0    0   33    0    0  134    4    5
05:46:37  all     0    0    0    0   54    1    0  227   38   13
05:46:42  all     0    0    0    0   41    0    0  167   10    8
```

Muestra la información de la frecuencia de interrupciones de los niveles activos que se encuentran en `/proc/interrupts`. Nos es útil para ver si existe algún dispositivo que está interrumpiendo constantemente el trabajo de la CPU. Consultando este archivo veremos que en el ejemplo las más activas son la 9 (acpi), 12 (teclado), 14-15 (ide) y muy poco la 10 (usb).

Memoria y *swap*: `sar -r 4 5`

```
Linux  debian  2.6.26-2-686  #1 SMP Thu May 28 15:39:35 UTC 2009  i686  11/30/2010
05:57:10  memtot memfree buffers   cached  slabmem      swptot swpfree  _mem_
05:57:17   1011M    317M    121M     350M      30M        729M    729M
05:57:21   1011M    306M    121M     351M      30M        729M    729M
05:57:25   1011M    300M    121M     351M      30M        729M    729M
```

En este caso `memtot` indica la memoria total libre y `memfree`, la memoria libre. El resto de indicadores es la memoria utilizada en *buffers*, la utilizada en

caché (de datos), `slabmem` es la memoria dinámica del núcleo y `swptot/free` es el espacio total/libre de *swap*. Es importante tener en cuenta que si `memfree` $\simeq$ 0 (no hay espacio), las páginas de los procesos irán a parar al *swap*, donde debe haber sitio teniendo en cuenta que esto permitirá la ejecución, pero todo irá más lento. Esto se debe contrastar con el uso de CPU. También se debe controlar que el tamaño de los *buffers* sea adecuado y esté en relación con los procesos que están realizando operaciones de entrada y salida. Es también interesante el comando `free`, que permite ver la cantidad de memoria en una visión simplificada:

```
             total       used       free     shared    buffers     cached
Mem:       1036092     711940     324152          0     124256     359748
-/+ buffers/cache:     227936     808156
Swap:       746980          0     746980
```

Esto indica que de 1 Gb casi las 3/4 partes de la memoria están ocupadas y que aproximadamente 1/3 son de caché. Además, nos indica que el *swap* no se está utilizando para nada, por lo que podemos concluir que el sistema está bien. Si quisiéramos más detalles deberíamos utilizar el comando `vmstat` (con más detalles que el `sar -r`) para analizar qué es lo que está causando problemas o quién está consumiendo tanta memoria. A continuación se muestra una salida de `vmstat 1 10`*****:

***Consúltese el manual para obtener una descripción de las columnas.**

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  1      0 324820 124256 359796    0    0    23    11   20  112  0  0 99  1
 0  0      0 324696 124256 359816    0    0     0    88    4   96  1  1 98  0
 0  0      0 324716 124256 359816    0    0     0     0  106  304  0  0 100  0
 0  0      0 324716 124256 359816    0    0     0     0  150  475  1  2 97  0
...
```

Utilización de las tablas del núcleo: `sar -v 4 5`

```
Linux  debian  2.6.26-2-686  #1 SMP Thu May 28 15:39:35 UTC 2009  i686  11/30/2010
06:14:02  superb-sz inode-sz     file-sz    dquota-sz    flock-sz      _curmax_
06:14:06     0/0     32968/36     3616/101976      0/0        13/0
06:14:10     0/0     32968/36     3616/101976      0/0        13/0
06:14:13     0/0     32984/36     3616/101976      0/0        13/0
06:14:17     0/0     32984/36     3616/101976      0/0        13/0
06:14:22     0/0     33057/36     3680/101976      0/0        13/0
```

En este caso, `superb-sz` es el número actual-máximo de *superblocks* mantenido por el núcleo para los sistemas de archivos montados; `inode-sz` es el número actual-máximo de *incore-inodes* en el núcleo necesario, que es de uno por disco como mínimo; `file-sz` es el número actual-máximo de archivos abiertos, `dquota-sz` es la ocupación actual-máxima de entradas de cuotas (para más información consúltese `man sar` –o `atsar`–). Esta monitorización se puede completar con el comando `ps -Af` (*process status*) y el comando `top`, que mostrarán la actividad y estado de los procesos en el sistema. A continuación, se muestran dos ejemplos de ambos comandos (solo algunas líneas):

```
debian:/proc# ps -Alw
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   525 -      ?        00:00:01 init
5 S     0     2     0  0  75  -5 -     0 -      ?        00:00:00 kthreadd
1 S     0     3     2  0 -40   - -     0 -      ?        00:00:00 migration/0
...
5 S     1  1601     1  0  80   0 -   473 -      ?        00:00:00 portmap
5 S   102  1612     1  0  80   0 -   489 -      ?        00:00:00 rpc.statd
...
4 S   113  2049  2012  0  80   0 - 31939 -      ?        00:00:03 mysqld
...
4 S     0  2654  2650  0  80   0 -  6134 -      tty7     00:00:49 Xorg
1 S     0  2726     1  0  80   0 -  6369 -      ?        00:00:00 apache2
0 S     0  2746     1  0  80   0 -   441 -      tty1     00:00:00 getty
...
```

Algunos aspectos interesantes para ver son la dependencia de los procesos (PPID=proceso padre) y, por ejemplo, que para saber el estado de los procesos se puede ejecutar con `ps -Alw` y en la segunda columna nos mostrará cómo se encuentra cada uno de los procesos. Estos parámetros reflejan el valor indicado en la variable del núcleo para este proceso, los más importantes de los cuales desde el punto de vista de la monitorización son: `F` *flags* (en este caso 1 es con superprivilegios, 4 creado desde el inicio *daemon*), `S` es el estado (`D`: no interrumpible durmiendo entrada/salida, `R`: ejecutable o en cola, `S`: durmiendo, `T`: en traza o parado, `Z`: muerto en vida, 'zombie'). `PRI` es la prioridad; `NI` es *nice*; `TTY`, desde dónde se ha ejecutado; `TIME`, el tiempo de CPU; `CMD`, el programa que se ha ejecutado y sus parámetros. Si se quiere salida con refresco (configurable), se puede utilizar el comando `top`, que muestra unas estadísticas generales (procesos, estados, carga, etc.) y, después, información de cada uno de ellos similar al `ps`, pero se actualiza cada 5 segundos por defecto (en modo gráfico está `gnome-system-monitor`):

```
top - 15:09:08 up 21 min,  2 users,  load average: 0.16, 0.15, 0.12
Tasks: 184 total,   2 running, 182 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  2.8 sy,  0.0 ni, 96.8 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1509992 total,   846560 used,   663432 free,   117304 buffers
KiB Swap:  1087484 total,        0 used,  1087484 free,   374076 cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4144 root      20   0  201m  36m 8448 S   9.6  2.5   0:39.35 Xorg
 4694 adminp    20   0  980m  64m  29m S   6.7  4.4   0:26.22 gnome-shell
 4730 adminp    20   0  363m  16m  10m S   2.3  1.1   0:04.04 gnome-terminal
 4221 root      20   0 69796 1776 1140 S   0.3  0.1   0:01.62 nmbd
 4655 adminp    20   0  571m  26m  13m S   0.3  1.8   0:01.00 gnome-settings-
 6287 root      20   0 15080 1520 1072 R   0.3  0.1   0:00.10 top
    1 root      20   0 10648  812  676 S   0.0  0.1   0:01.21 init
    2 root      20   0     0    0    0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0.0  0.0   0:00.93 ksoftirqd/0
```

Un comando interesante y que presenta la información de otra forma que puede servir para tener una panonrámica de todo el sistema es `atop` (se debe instalar `apt-get install atop`). A continuación unas líneas de este comando nos muestran su potencialidad:

```
ATOP - SySDW         2014/06/28  10:55:42         ------          17m58s elapsed
PRC | sys    2m06s | user   9.10s | #proc    184 | #zombie    0 | #exit      0 |
CPU | sys       7% | user      1% | irq       1% | idle    388% | wait      3% |
CPL | avg1    0.03 | avg5    0.07 | avg15   0.10 | csw  1116790 | intr  619733 |
MEM | tot     1.4G | free  659.0M | cache 369.7M | buff  100.2M | slab   64.5M |
SWP | tot     1.0G | free    1.0G |              | vmcom   2.1G | vmlim   1.8G |
DSK |          sda | busy      3% | read   27461 | write   2710 | avio 1.03 ms |
NET | eth0      0% | pcki     278 | pcko     260 | si    2 Kbps | so    0 Kbps |
NET | lo      ---- | pcki      12 | pcko      12 | si    0 Kbps | so    0 Kbps |
              *** system and process activity since boot ***
  PID  SYSCPU  USRCPU  VGROW  RGROW  RDDSK   WRDSK ST EXC S  CPU  CMD       1/27
 3865  67.00s   2.31s 199.6M 36676K 14196K    148K N-   - S   7%  Xorg
 4505  40.21s   4.14s 980.6M 68848K 25396K      8K N-   - R   4%  gnome-shell
 4543   4.14s   0.60s 427.4M 16048K  4756K    160K N-   - S   0%  gnome-terminal
```

También se pueden utilizar las herramientas del paquete `systat` para conocer el estado de los recursos, como por ejemplo `vmstat` (estadísticas de CPU, memoria y entrada/salida), `iostat` (estadísticas de discos y CPU) y `uptime` (carga de CPU y estado general).

Un resumen de los comandos más interesantes es:

Comando	Descripción
atop, top	Actividad de los preocesos
arpwatch	monitor de Ethernet/FDDI
bmon, bwm-ng, nload	monitor del ancho de banda
downtimed	Monitor del tiempo de caída, fuera de servicio
free	Utilización de memoria
iostat, iotop	Actividad de disco y E/S
ip monitor, rtmon, iptotal, iptraf	Monitor de dispositivos de red
mpstat	Estadísticas del procesador
netstat	Estadística de la red
nfswatch	Monitor de NFS
ps, pstree, god	Muestra estado y características de los procesos
/proc	Sistema de archivos virtual
sar, atsar	Recoge información del sistema
stat, iwatch	Estadísticas del sistema de archivos
strace	Eventos de las llamadas al sistemas y señales
tcpdump, etherape, sniffit	Volcado/monitor de paquetes de red
uptime, w	Carga media del sistema y tiempo desde el inicio
vmstat	Estadísiticas del uso de memoria
gnome-system-monitor, gkrellm, xosview, xwatch	Monitores gráficos del sistema
xconsole	Monitor de mensajes en el escritorio

También existen una serie de programas que miden las prestaciones del sistema (*benchmark*) o parte de él como por ejemplo: netperf, mbw (red y ancho de banda), iozone (E/S), sysbench, globs, gtkperf, hpcc (general), bonie++ (disco). Es importante destacar que el benchmark hpcc incluye el *High-Performance* LINPACK (HPL) *benchmark* que es el utilizado para medir las prestaciones y realizar el *ranking* de las máquinas más potentes del mundo*.

*http://www.top500.org/

1.1.2. Optimización del sistema

A continuación veremos algunas recomendaciones para optimizar el sistema en función de los datos obtenidos.

1) Resolver los problemas de memoria principal: Se debe procurar que la memoria principal pueda acoger un porcentaje elevado de procesos en ejecución, ya que si no es así, el sistema operativo podrá paginar e ir al *swap*; pero esto significa que la ejecución de ese proceso se degradará notablemente. Si se agrega memoria, el tiempo de respuesta mejorará notablemente. Para ello, se debe tener en cuenta el tamaño de los procesos (`SIZE`) en estado `R` y agregarle la que utiliza el núcleo. La cantidades de memoria se pueden obtener con el comando `free`, que nos mostrará (o con `dmesg`), por ejemplo (`total/used/free/buffers/cached`):

`1036092/723324/312768/124396/367472`,

que es equivalente a (en megabytes) `1011/706/305/121/358` y donde observamos, en este caso, que solo el 30 % de la memoria está libre y que en este momento no hay problemas, pero una carga mínima del sistema puede significar un problema. Es por ello que deberemos analizar si el sistema está limitado por la memoria (con `atsar -r` y `-p` se verá mucha actividad de paginación).

Las soluciones para la memoria son obvias: o se incrementa la capacidad o se reducen las necesidades. Por el coste actual de la memoria, es más adecuado incrementar su tamaño que emplear muchas horas para ganar un centenar de bytes al quitar, ordenar o reducir requerimientos de los procesos en su ejecución. Reducir los requerimientos puede hacerse reduciendo las tablas del núcleo, quitando módulos, limitando el número máximo de usuarios, reduciendo los *buffers*, etc.; todo lo cual degradará el sistema (efecto burbuja) y las prestaciones serán peores (en algunos casos, el sistema puede quedar totalmente no operativo).

Otro aspecto que se puede reducir es la cantidad de memoria de los usuarios gracias a la eliminación de procesos redundantes y cambiando la carga de trabajo. Para ello, se deberán monitorizar los procesos que están durmiendo (zombies) y eliminarlos, o bien aquellos que no progresan en su entrada/salida (saber si son procesos activos, cuánto de CPU llevan gastada y si los "usuarios están esperando por ellos"). Cambiar la carga de trabajo es utilizar planificación de colas para que los procesos que necesitan gran cantidad de memoria se puedan ejecutar en horas de poca actividad (por ejemplo, por la noche, lanzándolos con el comando `at`).

2) Mucha utilización de CPU: Básicamente nos la da el tiempo *idle* (valores bajos). Con `ps` o `top` se deben analizar qué procesos son los que "devoran

CPU" y tomar decisiones, como posponer su ejecución, pararlos temporalmente, cambiar su prioridad (es la solución menos conflictiva de todas y para ello se puede utilizar el comando `renice prioridad PID`), optimizar el programa (para la próxima vez) o cambiar la CPU (o agregar otra). Como ya se ha mencionado, GNU/Linux utiliza el directorio `/proc` para mantener todas las variables de configuración del núcleo que pueden ser analizadas y, en cierto caso, "ajustadas", para lograr prestaciones diferentes o mejores.

Para ello, se debe utilizar el comando `sysctl -a` para obtener todas las variables del núcleo y sus valores en el archivo*. Otros comandos alternativos son el `systune` y `systunedump`, que permiten descargar las variables en un archivo y modificarlas, para cargarlas nuevamente en el `/proc` (el comando `systune` guarda la configuración en `/etc/systune.conf`). En este caso, por ejemplo, se podrían modificar (se debe proceder con cuidado, porque el núcleo puede quedar fuera de servicio) las variables de la categoría `/proc/sys/vm` (memoria virtual) o `/proc/sys/kernel` (configuración del *core* del núcleo).

***Consúltese el manual para cambiar los valores y el archivo de configuración**
`/etc/sysctl.conf`

En este mismo sentido, también (para expertos o desesperados) se puede cambiar el tiempo máximo (*slice*) que el administrador de CPU (*scheduler*) del sistema operativo dedica a cada proceso en forma circular (si bien es aconsejable utilizar `renice` como práctica). Pero en GNU/Linux, a diferencia de otros sistemas operativos, es un valor fijo dentro del código, ya que está optimizado para diferentes funcionalidades (pero es posible tocarlo). Se puede "jugar" (a su propio riesgo) con un conjunto de variables que permiten tocar el *time slice* de asignación de CPU (`kernel-source-x.x.x/kernel/sched.c`).

3) Reducir el número de llamadas: Otra práctica adecuada para mejorar las prestaciones es reducir el número de llamadas al sistema de mayor coste en tiempo de CPU. Estas llamadas son las invocadas (generalmente) por el `shell` `fork()` y `exec()`. Una configuración inadecuada de la variable `PATH` con el directorio actual (indicado por `./`), puede tener una relación desfavorable de ejecución (esto es debido a que la llamada `exec()` no guarda nada en caché y perjudica esta ejecución) Para ello, siempre habrá que configurar la variable `PATH` con el directorio actual como última ruta. Por ejemplo, en `$HOME/.bashrc` hacer: `PATH=$PATH:.; export PATH` si el directorio actual no está en el *path* o, si está, rehacer la variable `PATH` para ponerlo como última ruta.

Se debe tener en cuenta que una alta actividad de interrupciones puede afectar a las prestaciones de la CPU con relación a los procesos que ejecuta. Mediante monitorización (`atsar -I`) se puede mirar cuál es la relación de interrupciones por segundo y tomar decisiones con respecto a los dispositivos que las causan. Por ejemplo, cambiar de módem por otro más inteligente o cambiar la estructura de comunicaciones si detectamos una actividad elevada sobre el puerto serie donde se encuentra conectado.

4) Mucha utilización de disco: Después de la memoria, un tiempo de respuesta bajo puede ser debido al sistema de discos. En primer lugar, se debe verificar que se disponga de tiempo de CPU (por ejemplo, `idle` >20 %) y que

el número de entrada/salida sea elevado (por ejemplo, superior a 30 entrada/salida/s) utilizando `atsar -u` y `atsar -d`. Las soluciones pasan por:

- En un sistema multidisco, planificar dónde se encontrarán los archivos más utilizados para equilibrar el tráfico hacia ellos (por ejemplo `/home` en un disco y `/usr` en otro) y que puedan utilizar todas las capacidades de entrada/salida con caché y concurrente de GNU/Linux (incluso, por ejemplo, planificar sobre qué *bus ide* se colocan). Comprobar luego que existe un equilibrio del tráfico con `atsar -d` (o con `iostat`). En situaciones críticas se puede considerar la compra de un sistema de discos RAID que realizan este ajuste de forma automática.

- Tener en cuenta que se obtienen mejores prestaciones sobre dos discos pequeños que sobre uno grande del tamaño de los dos anteriores.

- En sistemas con un solo disco, generalmente se realizan, desde el punto de vista del espacio, cuatro particiones de la siguiente manera (desde fuera hacia dentro): `/, swap, /usr, /home`, pero esto genera pésimas respuestas de entrada/salida porque si, por ejemplo, un usuario compila desde su directorio `/home/user` y el compilador se encuentra en `/usr/bin`, la cabeza del disco se moverá a lo largo de toda su longitud. En este caso, es mejor unir las particiones `/usr` y `/home` en una sola (más grande), aunque puede representar algunos inconvenientes en cuanto a mantenimiento.

- Incrementar los *buffers* de caché de entrada/salida (véase, por ejemplo, `/proc/ide/hd`...).

- Si se utiliza un `extfs`, se puede usar el comando `dumpe2fs -h /dev/hdx` para obtener información sobre el disco y `tune2fs /dev/hdx` para cambiar algunos de los parámetros configurables del mismo.

- Obviamente, el cambio del disco por uno de mayor velocidad (mayores RPM) siempre tendrá un impacto positivo en un sistema limitado por la entrada/salida de disco [31].

5) Mejorar aspectos de TCP/IP: Examinar la red con el comando `atsar` (o también con `netstat -i` o con `netstat -s | more`) para analizar si existen paquetes fragmentados, errores, *drops*, desbordamientos, etc., que puedan estar afectando a las comunicaciones y, con ello, al sistema (por ejemplo, en un servidor de NFS, NIS, ftp o web). Si se detectan problemas, se debe analizar la red para considerar las siguientes actuaciones:

- Fragmentar la red mediante elementos activos que descarten paquetes con problemas o que no sean para máquinas del segmento.

- Planificar dónde estarán los servidores para reducir el tráfico hacia ellos y los tiempos de acceso.

- Ajustar parámetros del núcleo (`/proc/sys/net/`). Por ejemplo, para obtener mejoras en el *throughput* debemos ejecutar la siguiente instrucción: `echo 600 >/proc/sys/net/core/netdev_max_backlog`*.

*Mínimo 300

6) Otras acciones sobre parámetros del núcleo: Existe otro conjunto de parámetros sobre el núcleo que es posible sintonizar para obtener mejores prestaciones, si bien, teniendo en cuenta lo que hemos tratado anteriormente, se debe ir con cuidado, ya que podríamos causar el efecto contrario o inutilizar el sistema. Consultad en la distribución del código fuente en el directorio `kernel-source-2.x/Documentation/sysctl` algunos archivos como por ejemplo `vm.txt`, `fs.txt` y `kernel.txt`. `/proc/sys/vm` controla la memoria virtual del sistema (*swap*) y permite que los procesos que no entran en la memoria principal sean aceptados por el sistema pero en el dispositivo de *swap*, por lo cual, el programador no tiene límite para el tamaño de su programa (obviamente debe ser menor que el dispositivo de *swap*). Los parámetros susceptibles de sintonizar se pueden cambiar muy fácilmente con `sysctl` (o también con `gpowertweak`). `/proc/sys/fs` contiene parámetros que pueden ser ajustados de la interacción núcleo-sistema de ficheros, tal como `file-max` (y exactamente igual para el resto de los archivos de este directorio).

7) Generar el núcleo adecuado a nuestras necesidades: La optimización del núcleo significa escoger los parámetros de compilación de acuerdo a nuestras necesidades. Es muy importante primero leer el archivo `readme` del directorio `/usr/src/linux`.

Una buena configuración del núcleo permitirá que se ejecute más rápido, que se disponga de más memoria para los procesos de usuario y, además, resultará más estable. Hay dos formas de construir un núcleo: **monolítico** (mejores prestaciones) o **modular** (basado en módulos, que tendrá mejor portabilidad si tenemos un sistema muy heterogéneo y no se desea compilar un núcleo para cada uno de ellos). Para compilar su propio núcleo y adaptarlo a su hardware y necesidades, cada distribución tiene sus reglas (si bien el procedimiento es similar).

Enlaces de interés

Es interesante consultar los siguientes libro/artículos:
www.redbooks.ibm.com sobre Linux Performance and Tuning Guidelines, http://people.redhat.com/alikins/system_tuning.html sobre información de optimización de sistemas servidores Linux y http://www.linuxjournal.com/article/2396 sobre *Performance Monitoring Tools for Linux*. El primero es un e-book abierto de la serie RedBooks de IBM muy bien organizado y con gran cantidad de detalles sobre la sintonización de sistemas Linux, los dos restantes son artículos que si bien tienen un cierto tiempo, los conceptos/metodología y algunos procedimientos continúan vigentes.

1.1.3. Optimizaciones de carácter general

Existen una serie de optimizaciones de índole general que pueden mejorar las prestaciones del sistema:

1) Bibliotecas estáticas o dinámicas: cuando se compila un programa, se puede hacer con una biblioteca estática (`libr.a`), cuyo código de función se incluye en el ejecutable, o con una dinámica (`libr.so.xx.x`), donde se

carga la biblioteca en el momento de la ejecución. Si bien las primeras garantizan código portable y seguro, consumen más memoria. El programador deberá decidir cuál es la adecuada para su programa incluyendo `-static` en las opciones del compilador (no ponerlo significa dinámicas) o `-disable-shared`, cuando se utiliza el comando `configure`. Es recomendable utilizar (casi todas las distribuciones nuevas lo hacen) la biblioteca estándar `libc.a` y `libc.so` de versiones 2.2.x o superiores (conocida como Libc6) que reemplaza a las anteriores. En gcc 4.X por defecto se utilizan bibliotecas dinámicas, pero se puede forzar (no recomendado) a utilizar estáticas incluso para la `libc` (opciones `-static -static-libgcc` en contraposición con las por defecto `-shared -shared-libgcc`).

2) Selección del procesador adecuado: generar código ejecutable para la arquitectura sobre la cual correrán las aplicaciones. Algunos de los parámetros más influyentes del compilador son:

a) `-march` (por ejemplo, `-march=core2` para el soporte de CPU Intel Core2 CPU 64-bit con extensiones MMX, SSE, SSE2, SSE3/SSSE3, o `-march=k8` para CPU AMD K8 Core con soporte x86-64) haciendo simplemente `gcc -march=i686`;

b) el atributo de optimización `-O1,2,3` (`-O3` generará la versión más rápida del programa, `gcc -O3 -march = i686`), y

c) los atributos `-f` (consultad la documentación para los diferentes tipos).

Paquetes `no-free`

Recordad que sobre Debian se debe activar el repositorio de paquetes `no-free` para poder instalar los paquetes de documentación del compilador `gcc-doc` y `gcc-doc-base`.

3) Optimización del disco: en la actualidad, la mayoría de ordenadores incluye disco UltraDMA (100) por defecto; sin embargo, en una gran cantidad de casos no están optimizados para extraer las mejores prestaciones. Existe una herramienta (`hdparm`) que permite sintonizar el núcleo a los parámetros del disco tipo IDE y SATA (aunque estos últimos cuentan también con una utilidad específica llamada sdparm). Se debe tener cuidado con estas utilidades, sobre todo en discos UltraDMA (hay que verificar en el BIOS que los parámetros para soporte por DMA están habilitados), ya que pueden inutilizar el disco. Consultad las referencias y la documentación ([4] y `man hdparm/sdparm`) sobre cuáles son (y el riesgo que comportan) las optimizaciones más importantes, por ejemplo: `-c3`, `-d1`, `-X34`, `-X66`, `-X12`, `-X68`, `-mXX`, `-a16`, `-u1`, `-W1`, `-k1`, `-K1`. Cada opción significa una optimización y algunas son de altísimo riesgo, por lo que habrá que conocer muy bien el disco. Para consultar los parámetros optimizados, se podría utilizar `hdparm -vtT /dev/hdX` (donde `X` es el disco optimizado) y la llamada a `hdparm` con todos los parámetros se puede poner en `/etc/init.d` para cargarla en el *boot*. Para consultar la información del disco se puede hacer, por ejemplo, `hdparm -i /dev/sdb`

1.1.4. Configuraciones complementarias

Existen más configuraciones complementarias desde el punto de vista de la seguridad que de la optimización, pero son necesarias sobre todo cuando el

sistema está conectado a una intranet o a Internet. Estas configuraciones implican las siguientes acciones [4]:

1) Impedir que se pueda arrancar otro sistema operativo: si alguien tiene acceso físico a la máquina, podría arrancar con otro sistema operativo preconfigurado y modificar el actual, por lo que se debe inhibir desde el BIOS del ordenador el *boot* por CD-ROM o USB y poner una contraseña de acceso (recordad la contraseña del BIOS, ya que, de otro modo, podría causar problemas cuando se quisiera cambiar la configuración).

2) Configuración y red: es recomendable desconectar la red siempre que se deseen hacer ajustes en el sistema. Se puede quitar el cable o deshabilitar el dispositivo con `/etc/init.d/networking stop` (`start` para activarla de nuevo) o con `ifdown eth0` (`ifup eth0` para habilitarla) para un dispositivo en concreto.

3) Modificar los archivos de `/etc/security`: de acuerdo a las necesidades de utilización y seguridad del sistema. En `access.conf` hay información sobre quién puede hacer un *login* al sistema; por ejemplo:

```
# Tabla de control de acceso. líneas con clomuna1=# es un comentario.
# El orden de la líneas es importante
# Formato: permission : users : origins
# Deshabilar todo los logins excepto root sobre tty1
-:ALL EXCEPT root:tty1
# User "root" permitido conectarse desde estas direcciones
+ : root : 192.168.200.1 192.168.200.4 192.168.200.9
+ : root : 127.0.0.1
# O desde la red
+ : root : 192.168.201.
# Impide el acceso excepto user1,2,3 pero el último solo desde consola.
-:ALL EXCEPT user1 user2 user3:console
```

También se debería, por ejemplo, configurar los grupos para controlar cómo y a dónde pueden acceder y también los límites máximos (`limits.conf`) para establecer los tiempos máximos de utilización de CPU, E/S, etc. y así evitar ataques por denegación de servicio (DoS).

4) Mantener la seguridad de la contraseña de root: utilizar como mínimo 8 caracteres, con uno, por lo menos, en mayúsculas o algún carácter que sea no trivial, como "-", ".", ",", etc.; asimismo, es recomendable activar el envejecimiento para forzar a cambiarlo periódicamente, así como también limitar el número de veces con contraseña incorrecta. También se puede cambiar el parámetro `min=x` de la entrada en `/etc/pam.d/passwd` para indicar el número mínimo de caracteres que se utilizarán en la contraseña (`x` es el número de caracteres). Utilizar algoritmos como SHA512 para la configuración de paswssd (en Debian viene configurado por defecto, ver */etc/pam.d/common-password*).

5) No acceder al sistema como root: si bien muchas distribuciones ya incorporan un mecanismo de este estilo (por ejemplo, Ubuntu), se puede crear una cuenta como sysadm y trabajar con ella. Si se accede remotamente, habrá

siempre que utilizar `shh` para conectarse al sysadm y, en caso de ser necesario, realizar un `su -` para trabajar como root o activar el `sudoers` para trabajar con el comando `sudo` (consultad la documentación para las opciones del comando y su edición).

6) Tiempo máximo de inactividad: inicializar la variable `TMOUT`, por ejemplo a 360 (valor expresado en segundos), que será el tiempo máximo de inactividad que esperará el *shell* antes de bloquearse; se puede poner en los archivos de configuración del *shell* (por ejemplo, `/etc/profile`, `.profile`, `$HOME/.bashrc`, etc.). En caso de utilizar entornos gráficos (KDE, Gnome, etc.), se puede activar el salvapantallas con contraseña, al igual que el modo de suspensión o hibernación.

7) Configuración del NFS en forma restrictiva: en el `/etc/exports`, exportar solo lo necesario, no utilizar comodines (*wildcards*), permitir solo el acceso de lectura y no permitir el acceso de escritura por root, por ejemplo, con `/directorio_exportado host.domain.com (ro, root_squash)`.

8) Evitar arranques desde el *bootloader* con parámetros: se puede iniciar el sistema como *linux single*, lo que arrancará el sistema operativo en modo de usuario único. Hay que configurar el sistema para que el arranque de este modo siempre sea con contraseña. Para ello, en el archivo `/etc/inittab` se debe verificar que existe la línea `S:wait:/sbin/sulogin` y que tiene habilitado el `/bin/sulogin`. Verificar que los archivos de configuración del *bootloader* (`/etc/ lilo.conf` si tenemos Lilo como gestor de arranque, o /etc/grub.d si trabajamos con Grub2) debe tener los permisos adecuados para que nadie lo pueda modificar excepto el root. Mediante los archivos de configuración de *boot* se permiten una serie de opciones que es conveniente considerar: `timeout` para controlar el tiempo de *boot*; `restricted` para evitar que se puedan insertar comandos en el momento del *boot* como `linux init = /bin/sh` y tener acceso como root sin autorización; en este caso, debe acompañarse de `password=palabra-de-password`; si solo se pone `password`, solicitará la contraseña para cargar la imagen del núcleo (consultar los manuales de Lilo/Grub para la sintaxis correcta).

Se puede probar el riesgo que esto representa haciendo lo siguiente (siempre que el Grub/Lilo no tenga contraseña), que puede ser útil para entrar en el sistema cuando no se recuerde la contraseña de usuario, pero representa un gran peligro de seguridad cuando es un sistema y se tiene acceso a la consola y el teclado:

a) se arranca el ordenador hasta que se muestre el menú de arranque,

b) se selecciona el núcleo que se desea arrancar y se edita la línea presionando la tecla "e" (edit),

c) buscamos la línea que comienza por `kernel ...` y al final de la línea borramos el parámetro `ro` e introducimos `rw init=/bin/bash` (lo cual indica acceso directo a la consola). Presionamos "F10"

d) Con esto se arrancará el sistema y pasaremos directamente a modo *root*, gracias a lo cual se podrá cambiar la contraseña (incluida la de root), editar el fichero `/etc/passwd` o el `/etc/shadow` o también crear un nuevo usuario y todo lo que deseemos.

9) Control de la combinación *Ctrl-Alt-Delete*: Para evitar que se apague la máquina desde el teclado, se debe insertar un comentario (#) en la primera columna de la línea siguiente: `ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now` del archivo `/etc/inittab`. Los cambios se activan con la orden `telinit q`.

10) Evitar peticiones de servicios no ofrecidos: se debe bloquear el archivo `/etc/services`, para no admitir servicios no contemplados, por medio de `chattr +i /etc/services`.

11) Conexión del root: hay que modificar el archivo `/etc/securetty` que contiene las TTY y VC (*virtual console*) en que se puede conectar el root dejando solo una de cada, por ejemplo, `tty1` y `vc/1` y, si es necesario, hay que conectarse como sysadm y hacer un `su`.

12) Eliminar usuarios no utilizados: se deben borrar los usuarios o grupos que no sean necesarios, incluidos los que vienen por defecto (por ejemplo, operator, shutdown, ftp, uucp, games, etc.) y dejar solo los necesarios (root, bin, daemon, sync, nobody, sysadm) y los que se hayan creado con la instalación de paquetes o por comandos (lo mismo con `/etc/group`). Si el sistema es crítico, podría considerarse el bloqueo (`chattr +i file`) de los archivos `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/gsahdow` para evitar su modificación (cuidado con esta acción, porque no permitirá cambiar posteriormente las contraseñas).

13) Montar las particiones en forma restrictiva: utilizar en `/etc/fstab` atributos para las particiones tales como `nosuid` (que impide suplantar el usuario o grupo sobre la partición), `nodev` (que no interpreta dispositivos de caracteres o bloques sobre esta partición) y `noexec` (que no permite la ejecución de archivos sobre esta partición). Por ejemplo: `/tmp /tmp ext2 defaults,nosuid,noexec 0 0`. También es aconsejable montar el `/boot` en una partición separada y con atributos `ro`.

14) Protecciones varias: se puede cambiar a 700 las protecciones de los archivos de `/etc/init.d` (servicios del sistema) para que solo el root pueda modificarlos, arrancarlos o pararlos y modificar los archivos `/etc/issue` y `/etc/issue.net` para que no den información (sistema operativo, versión, etc.) cuando alguien se conecta por telnet, ssh, etc.

15) SUID y SGID: un usuario podrá ejecutar como propietario un comando si tiene el bit `SUID` o `SGID` activado, lo cual se refleja como una '`s`' SUID (-rwsr-xr-x) y SGID (-r-xr-sr-x). Por lo tanto, es necesario quitar el bit (`chmod a-s file`) a los comandos que no lo necesitan. Estos archivos pueden buscarse con: `find / -type f -prem -4000 o -perm -2000 -print`. Se debe proceder con cuidado respecto a los archivos en que se quite el SUID-GUID, porque el comando podría quedar inutilizado.

16) Archivos sospechosos: hay que buscar periódicamente archivos con nombres no usuales, ocultos o sin un uid/gid válido, como "..." (tres puntos), ".. " (punto punto espacio), "..^G" o equivalentes. Para ello, habrá que utilizar: `find / -name=".*" -print | cat -v` o sino `find / -name ".." -print`.

Para buscar uid/gid no válidos, utilizad `find / -nouser` (o utilizad también `-nogroup` (cuidado, porque algunas instalaciones se hacen con un usuario que luego no está definido y que el administrador debe cambiar).

17) Conexión sin contraseña: no se debe permitir el archivo `.rhosts` en ningún usuario, a no ser que sea estrictamente necesario (se recomienda utilizar `ssh` con clave pública en lugar de métodos basados en `.rhosts`).

18) X Display manager: para indicar los *hosts* que se podrán conectar a través de XDM y evitar que cualquier *host* pueda tener una pantalla de *login* se puede modificar el archivo `/etc/X11/xdm/Xaccess`.

1.1.5. Resumen de acciones para mejorar un sistema

1) Observar el estado del sistema y analizar los procesos que consumen mucha CPU utilizando, por ejemplo, el comando `ps auxS -H` (o el comando `top`) y mirando las columnas `%CPU`, `%MEM` y `TIME`; se debe observar la jerarquía de procesos y prestar atención a cómo se está utilizando la CPU y la memoria y analizar el tiempo de ejecución de los procesos para encontrar procesos *zombies* mirando en la columna `STAT` aquellos que tengan el identificador `Z` (los cuales se podrán eliminar sin problemas). También se debe prestar especial atención a los que estén con `D`, `S` (que están haciendo entrada o salida) y `W` (que están utilizando el *swap*). En estos tres últimos utilizad el `atsar` y `free` (`sar` o `vmstat`) para verificar la carga de entrada y salida, ya que puede ser que estos procesos estén haciendo que las prestaciones del sistema bajen notablemente (generalmente por sus necesidades, en cuyo caso no podremos hacer gran cosa, pero en otros casos puede ser que el código no esté optimizado o bien escrito).

2) Analizar el estado de la memoria en detalle para descubrir dónde se está gastando la memoria. Recordad que todos los procesos que se deben ejecutar deben estar en memoria principal y, si no hay, el proceso paginará en *swap* pero con la consiguiente pérdida de prestaciones, ya que debe ir al disco y llevar zona de memoria principal. Es vital que los procesos más activos tengan memoria principal y esto se puede lograr cambiando el orden de ejecución o haciendo un cambio de prioridades (comando `renice`). Para observar el estado de la memoria en detalle utilizad el `vmstat 2` (o el `atsar`), por ejemplo, y observad las columnas `swpd`, que es la cantidad de memoria virtual (*swap*) utilizada, `free`, la cantidad de memoria principal libre (la ocupada se obtiene de la total menos la libre) y `si/so`, la cantidad de memoria virtual en lectura o escritura utilizada. Si tenemos un proceso que utiliza gran cantidad de *swap* (`si/so`) este proceso estará gastando mucho tiempo en gestión, retardará el

conjunto y veremos que la CPU tiene, por ejemplo, valores de utilización bajos. En este caso se deberían eliminar procesos de la memoria para hacer sitio o ampliar la memoria RAM del sistema si es que no se pueden quitar los procesos que hay, siempre y cuando el proceso bajo estudio no sea de ejecución ocasional.

3) Tened en cuenta que %CPU + % E/S + %Idle = 100 % por lo cual vemos que la E/S (I/O en `vmstat`) también afecta a un proceso.

```
  debian:/home/remo# vmstat 2
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 ...
 0  0      0 623624  29400 231596    0    0  7184     0  393 1389 10  8 10 73
 0  0      0 623596  29400 231612    0    0     0     0  416  800  0  2 98  0
 1  0      0 622540  29408 231604    0    0     0   276  212  549  2  2 94  2
 0  0      0 613544  29536 240620    0    0  4538     0  464 1597 10  8 29 54
 0  0      0 612552  29560 240824    0    0   112     0  412  850  1  2 85 12
```

En este caso podemos observar que hay una utilización muy grande de I/O (E/S) pero 0 de *swap* un y alto valor de CPU tanto en `wa` (*waiting*) como en `id` (*idle*) lo que quiere decir que la CPU está esperando a que algo que está en E/S, termine (en este caso es la ejecución de unas cuantas instancias del LibreOffice, lo que significa lectura de disco y carga de un ejecutable a memoria principal). Si esta situación se repite o es constante, se debería analizar cómo utilizan la memoria los procesos en espera de ejecución y cómo reducirla (por ejemplo, poniendo un disco más rápido o con más *buffer* de E/S).

4) Se debe tener en cuenta que el %CPU está constituido por la suma de dos valores "us" (User Time) y "sy" (System Time). Estos valores representan el tiempo empleado ejecutando código del usuario (*non-kernel code*) y el tiempo gastado ejecutando código del núcleo, respectivamente y pueden ser útiles cuando se desea optimizar el código de un programa con el fin de que consuma menos tiempo de CPU. Utilizaremos el comando `time`, que nos da el tiempo gastado en cada tipo de código, haciendo, por ejemplo, `time find /usr`, de modo que tendremos que nos da `real 1m41.010s, user 0m0.076s, sys 0m2.404s`; en cambio, si hacemos `time ls -R /usr` la salida es `real 0m5.530s user 0m0.160s sys 0m0.068s`. Como vemos, para la obtención de información equivalente (listado de archivos y directorios) un comando ha gastado 2,404 s en espacio de núcleo y el otro 0,06 s, por lo cual es interesante analizar qué comandos escogemos para hacer el trabajo. Otro aspecto interesante es que si ejecutamos, por ejemplo, `time find /var >/dev/null` (para no ver la salida) la primera vez obtenemos `real 0m23.900s, user 0m0.000s, sys 0m0.484s` pero una segunda vez obtenemos `real 0m0.074s, user 0m0.036s, sys 0m0.036s`. ¿Qué ha pasado? El sistema ha almacenado en las tablas de caché la información y la siguientes veces ya no tarda lo mismo, sino mucho menos. Si se desea utilizar el `time` en el formato avanzado o extendido, los usuarios que ejecuten *bash* como *shell* deberán ejecutar el `time` junto con el *path* donde se encuentre; por ejemplo `/usr/bin/time ls -R /usr`, para obtener los resultados deseados (consultad `man time` para más información).

5) Es interesante ver qué optimizaciones podemos generar en un código con modificaciones simples. Por ejemplo, observemos el código desarrollado por Bravo [3]:

```
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
int main(void)
{int x=1, y=2, z=3; long iter1=0,iter2=0;
struct timeval tv1,tv2;
gettimeofday(&tv1,NULL);
for(;;){
    x=(x*3+y*7+z*9)%11;
    y=(x*9+y*11+z*3)%29;
    z=(x*17+y*13+z*11)%37;
    iter1++;
    if(iter1==1000000){ iter2++; iter1=0;}
    gettimeofday(&tv2,NULL);
    if(tv2.tv_sec==tv1.tv_sec+5 && tv2.tv_usec>=tv1.tv_usec || tv2.tv_sec>tv1.tv_sec+5)
    break;}
    printf("Iteraciones: %ldM Resultado: %d %d %d\n",iter2,x,y,z);
    return 0;
}
```

El resultado de la ejecución es

```
time ./c: Iteraciones: 22M real 0m5.001s, user 0m1.756s, sys 0m3.240s
```

donde se puede observar que los 3,240 s han alcanzado para 22 millones de iteraciones. ¿En qué se gastan los 3,240 s? Pues en calcular la hora en cada iteración, ya que son múltiples las llamadas al núcleo. Si mejoramos el código y solo calculamos el `gettimeofday` cada millón de iteraciones, obtenemos `Iteraciones: 135M real 0m5.025s, user 0m4.968s, sys 0m0.056s` y vemos que se reduce notablemente el tiempo `sys` y obtenemos más tiempo para ejecutar el código del usuario, por lo cual sube el número de iteraciones (135 millones), teniendo en cuenta que hemos pasado de 22 millones de ejecución de la llamada `gettimeofday` a 135 millones de veces. ¿Cuál es la consecuencia? Que la finalización de ejecución de este ejemplo se obtiene por comparación de 5 s con el tiempo absoluto, por lo cual al calcular el tiempo absoluto menos veces se obtiene cierta "imprecisión" al determinar cuándo finaliza la ejecución (`real 0m5.001s` en el primer caso, mientras que en el segundo `0m5.025s`, una diferencia de 24 milésimas). Una solución optimizada para este caso sería no usar este tipo de llamadas al sistema para determinar cuándo debe finalizar un proceso y buscar alternativas, por ejemplo, con `alarm` y una llamada a `signal`. [3]

1.2. Monitorización

Un aspecto importante en el funcionamiento 24x7 de un sistema es que el administrador se debe anticipar a los problemas y es por ello que o bien está continuamente mirando su funcionamiento (lo cual es prácticamente imposible todo el tiempo) o bien se dispone de herramientas adecuadas que puedan prevenir la situación, generar alertas y advertir al responsable de que "algo

está pasando" y que este pueda realizar con antelación las acciones correctivas para evitar el fallo, disfunción o situación de fuera de servicio del sistema o recurso. Las herramientas que cumplen esta función se enmarcan dentro del grupo de herramientas de monitorización y permiten también obtener información del sistema con fines estadísticos, contables u otros que el usuario desee. Las herramientas más comunes permiten, mediante una interfaz web, conocer de forma remota los cinco factores principales (uso de CPU, memoria, E/S, red, procesos/servicios) que dan indicios de que "alguna cosa puede estar pasando"; las más sofisticadas generan alarmas por SMS para advertir de la situación al administrador. A continuación se describirán algunas de las herramientas más representativas (pero no son las únicas): Munin, Monit, MRTG, Nagios, Ganglia, Zabbix y Cacti.

1.2.1. Munin

Munin [8] produce gráficos sobre diferentes parámetros del servidor (load average, memory usage, CPU usage, MySQL throughput, eth0 traffic, etc.) sin excesivas configuraciones y presenta gráficos importantes para reconocer dónde y qué está generando problemas. Consideremos que nuestro sistema se llama `sysdw.nteum.org` y que ya la tenemos configurada con este nombre y con el DocumentRoot de Apache en `/var/www/`. Para instalar Munin sobre Debian hacemos, por ejemplo, `apt-get install munin munin-node`. Luego debemos configurar Munin (`/etc/munin/munin.conf`) con:

```
dbdir /var/lib/munin
htmldir /var/www/munin
logdir /var/log/munin
rundir /var/run/munin
tmpldir /etc/munin/templates
[debian.nteum.org]
    address 127.0.0.1
    use_node_name yes
```

Luego se crea el directorio, se cambian los permisos y se reinicia el servicio (caso de no existir).

```
mkdir -p /var/www/munin
chown munin:munin /var/www/munin
/etc/init.d/munin-node restart
```

Por último munin solo viene configurado para conectarse desde el localhost, si lo deseamos hacer desde otra máquina debemos cambiar entonces el archivo */etc/apache2/conf.d/munin* (que es un link a */etc/munin/apache.conf*) comentando la línea `#Allow from localhost 127.0.0.0/8 ::1` por `Allow from all` y reiniciar Apache (`service apache2 restart`).

Después de unos minutos se podrán ver los primeros resultados en la dirección web http://localhost/munin en el navegador (o también el dominio que tene-

mos asignado en */etc/hosts* por ejemplo en nuestro caso *sysdw.nteum.org*). Si se quiere mantener la privacidad de las gráficas basta con poner una contraseña para el acceso con Apache al directorio. Por ejemplo, se pone en el directorio `/var/www/munin/` el archivo `.htaccess` con el siguiente contenido:

```
AuthType Basic
AuthName "Members Only"
AuthUserFile /etc/munin/htpasswd
   require valid-user
```

Después se debe crear el archivo de contraseña en `/etc/munin/htpasswd` con el comando (como root): `htpasswd -c /etc/munin/htpasswd admin`. Cuando nos conectemos al `http://localhost/munin/` nos pedirá el usuario (admin) y la contraseña que hemos introducido después del comando anterior.

Munin viene con un conjunto de *plugins* instalados pero fácilmente se pueden habilitar otros haciendo, por ejemplo para monitorizar MySQL:

```
cd /etc/munin/plugins
ln -s /usr/share/munin/plugins/mysql_ mysql_
ln -s /usr/share/munin/plugins/mysql_bytes mysql_bytes
ln -s /usr/share/munin/plugins/mysql_innodb mysql_innodb
ln -s /usr/share/munin/plugins/mysql_isam_space_ mysql_isam_space_
ln -s /usr/share/munin/plugins/mysql_queries mysql_queries
ln -s /usr/share/munin/plugins/mysql_slowqueries mysql_slowqueries
ln -s /usr/share/munin/plugins/mysql_threads mysql_threads
```

1.2.2. Monit

Monit [7] permite configurar y verificar la disponibilidad de servicios tales como Apache, MySQL o Postfix y toma diferentes acciones, como por ejemplo reactivarlos si no están presentes. Para instalar Monit hacemos `apt-get install monit` y editamos `/etc/monit/monitrc`. El archivo por defecto incluye un conjunto de ejemplos, pero se deberá consultar la documentación para obtener más información*. A continuación presentamos un ejemplo de configuración típico sobre algunos servicios `/etc/monit/monitrc` [32]:

*http://mmonit.com/monit

```
# Monit control file example: /etc/monit/monitrc
# Solo se muestran las líneas cambiadas
    set daemon 120 # Poll at 2-minute intervals
    set logfile /var/log/monit.log
    set alert adminp@sysdw.nteum.org
# Se utiliza el servidor interno que dispone monit para controlar apache2 también
set httpd port 2812 and
    use address localhost  # only accept connection from localhost
    allow admin:monit      # require user 'admin' with password 'monit'
# Ejemplos de Monitores
check process sshd with pidfile /var/run/sshd.pid
   start program  "/etc/init.d/ssh start"
   stop program  "/etc/init.d/ssh stop"
   if failed port 22 protocol ssh then restart
   if 5 restarts within 5 cycles then timeout
```

```
check process mysql with pidfile /var/run/mysqld/mysqld.pid
   group database
   start program = "/etc/init.d/mysql start"
   stop program = "/etc/init.d/mysql stop"
   if failed host 127.0.0.1 port 3306 then restart
   if 5 restarts within 5 cycles then timeout
check process apache with pidfile /var/run/apache2.pid
   group www-data
   start program = "/etc/init.d/apache2 start"
   stop program  = "/etc/init.d/apache2 stop"
   if failed host sysdw.nteum.org port 80 protocol http
      and request "/monit/token" then restart
   if cpu is greater than 60% for 2 cycles then alert
   if cpu > 80% for 5 cycles then restart
check process ntpd with pidfile /var/run/ntpd.pid
   start program = "/etc/init.d/ntp start"
   stop  program = "/etc/init.d/ntp stop"
   if failed host 127.0.0.1 port 123 type udp then restart
   if 5 restarts within 5 cycles then timeout
```

Enlace de interés

Consultad el manual para obtener más detalles en http://mmonit.com/monit.

Para la monitorización de Apache hemos indicado que verifique un fichero que deberás estar en */var/www/monit/token* por lo cual se deberá crear con: `mkdir /var/www/monit; echo "hello" >/var/www/monit/token`. Para verificar que la sintaxis es correcta ejecutamos `monit -t` y para ponerlo en marcha ejecutamos `monit`. A partir de este momento se puede consultar en la dirección y puerto seleccionado en el archivo `/etc/monit/monitrc` (en nuestro caso `http://localhost:2812/`), que nos pedirá el usuario y contraseña también introducidos en el mismo archivo (admin y monit en nuestro caso). Se puede parar y arrancar el servicio con `service monit restart` (verificar el valor de la variable START en `/etc/default/monit`.

1.2.3. SNMP + MRTG

El **MRTG** (*Multi-Router Traffic Grapher*) [9] fue creado para mostrar información gráfica sobre datos de red, como se verá en el ejemplo que mostramos a continuación, para monitorizar la red Ethernet, pero se pueden usar otros datos para visualizar el comportamiento y para generar las estadísticas de carga (*load average*) del servidor. En primer lugar instalaremos SNMP o Protocolo Simple de Administración de Red (*Simple Network Management Protocol*) que es un protocolo de la capa de aplicación que facilita la obtención información entre dispositivos de red (p. ej., *routers*, *switches*, servidores, estaciones de trabajo, impresoras, ...) y que permite a los administradores supervisar el funcionamiento de la red y buscar/resolver sus problemas. Para ello hacemos `apt-get install snmp snmpd`. Las variables accesibles a través de SNMP están organizadas en jerarquías metadatos (tipo, descripción, ...) y están almacenadas en una tablas llamadas *Management Information Bases* (MIBs). Para instalarlas debemos agregar primero al repositorio de Debian en non-free y luego descargar el paquete:

Agregamos en /etc/apt/sources.list
deb http://ftp.debian.org/debian wheezy main contrib non-free

Luego actualizamos los repositorios e instalamos el paquete

```
apt-get update
apt-get install snmp-mibs-downloader
download-mibs
```

Luego debemos configurar el servicio `snmpd` y para ello editamos la configuración */etc/snmp/snmpd.conf* -solo hemos dejado las líneas más importantes a cambiar-:

```
agentAddress udp:127.0.0.1:161
rocomunity public
com2sec local localhost public
group MyRWGroup v1 local
group MyRWGroup v2c local
group MyRWGroup usm local
view all included .1 80
access MyRWGroup "" any noauth exact all all none
com2sec notConfigUser default mrtg
group notConfigGroup v1 notConfigUser
group notConfigGroup v2c notConfigUser
view systemview included .1.3.6.1.2.1.1
view systemview included .1.3.6.1.2.1.25.1.1
view systemview included .1 80
access notConfigGroup "" any noauth exact systemview none none
syslocation BCN
syscontact Adminp <adminp@sysdw.nteum.org>
```

A continuación en el archivo */etc/defaults/snmpd* hemos modificado la línea export `MIBS=/usr/share/mibs` para indicarle donde estaban las MIBs y se reinicia el servicio (`/etc/init.d/snmpd restart`). Podemos interrogar al servidor `snmpd` probar utilizando el comando `snmpwalk`, por ejemplo:

`snmpwalk -v1 -c public localhost` Dará una larga lista de información
`snmpwalk -v 2c -c public localhost` Idem anterior
O preguntarle por una variable específica de la MIB:
`snmpwalk -v1 -c mrtg localhost IP-MIB::ipAdEntIfIndex`
IP-MIB::ipAdEntIfIndex.127.0.0.1 = INTEGER: 1
IP-MIB::ipAdEntIfIndex.158.109.65.67 = INTEGER: 2

Con esto ya podemos instalar MRTG haciendo

```
apt-get install mrtg mrtg-contrib mrtgutils
```

Luego generamos la configuración con

```
cfgmaker public@localhost >/etc/mrtg.cfg
```

y debemos crear el directorio y cambiar las protecciones para el grupo de Apache: `/var/www/mrtg; chown www-data:www.data /var/www/mrtg`. Por último deberemos crear el índex.html con:

```
indexmaker --title="Localhost" --output /var/www/mrtg/index.html /etc/mrtg.cfg.
```

Antes de ejecutar mrtg existe un error en Debian Wheezy y IPv6 que podemos corregir con [10]:

> Cuando ejecutamos LANG=C /usr/bin/mrtg /etc/mrtg.cfg obtenemos el error:
> Subroutine SNMP_Session::pack_sockaddr_in6 redefined ...
> Solución: editar el archivo /usr/share/perl5/SNMP_Session.pm
> En la línea 149:
> Donde dice: import Socket6;
> Reemplazar por: Socket6->import(qw(inet_pton getaddrinfo));
> En la línea 609:
> donde dice: import Socket6;
> Reemplazar por: Socket6->import(qw(inet_pton getaddrinfo));

Para ejecutar `mrtg` inicialmente y verificar si todo está bien deberíamos hacer: `env LANG=C /usr/bin/mrtg /etc/mrtg.cfg` (y repitiéndola una serie de veces) podremos visualizar la actividad de red accediendo a la URL: *http://sysdw.nteum.org/mrtg/* y podremos ver que comienza a mostrar la actividad de eth0.

Finalmente si está todo bien deberemos configurar el `cron` para que se actualizacen las gráficas cada 5', para ello editamos el `crontab -e` insertando `*/5 * * * * root env LANG=C /usr/bin/mrtg /etc/mrtg.cfg`.

MRTG es muy potente para visualizar gráficos de variables ya sea SNMP o de scripts que podemos incluir. Veamos dos ejemplos donde el primero lo utilizaremos para medir la carga de CPU basado en una consulta SNMP al servidor y la segunda ejecutaremos un script para ver los procesos y los de root del sistema (con diferentes opciones de colores por ejemplo).

```
Para la carga de CPU agregar al final de /etc/mrtg:
    LoadMIBs: /usr/share/mibs/netsnmp/UCD-SNMP-MIB
    Target[localhost.cpu]:ssCpuRawUser.0&ssCpuRawUser.0:public@127.0.0.1+
    ssCpuRawSystem.0&ssCpuRawSystem.0:public@127.0.0.1+
    ssCpuRawNice.0&ssCpuRawNice.0:public@127.0.0.1
    RouterUptime[localhost.cpu]: public@127.0.0.1
    MaxBytes[localhost.cpu]: 100
    Title[localhost.cpu]: CPU Load
    PageTop[localhost.cpu]: <H1>CPU Load %</H1>
    Unscaled[localhost.cpu]: ymwd
    ShortLegend[localhost.cpu]: %
    YLegend[localhost.cpu]: CPU Utilization
    Legend1[localhost.cpu]: Active CPU in % (Load)
    Legend2[localhost.cpu]:
    Legend3[localhost.cpu]:
    Legend4[localhost.cpu]:
    LegendI[localhost.cpu]: Active
    LegendO[localhost.cpu]:
    Options[localhost.cpu]: growright,nopercent


Para el número de procesos agregar al final de /etc/mrtg:
    Title[procesos]: Processes
    Target[procesos]:`/usr/local/bin/proc.sh`
    PageTop[procesos]: <h1>Processes</h1>
    MaxBytes[procesos]: 200
    YLegend[procesos]: Processes
    ShortLegend[procesos]: Num.
    XSize[procesos]: 300
```

```
YSize[procesos]: 100
Options[procesos]: nopercent,gauge
Colours[procesos]: ORANGE\#FF7500,BLUE\#1000ff,DARK GREEN\#006600,VIOLET\#ff00ff
LegendI[procesos]: Processes Root
LegendO[procesos]: Total of Processes
```

Donde el script proc.sh es:

```
#!/bin/bash
sysname=`hostname`
p1=`ps -edaf | wc -l`
p1=`expr  $p1 - 1`\\
p2=`ps -edaf  | grep ^root \| wc -l`
p2=`expr $p2 - 2`

echo $p2
echo $p1
echo $sysname
```

Se debe tener en cuenta de ejecutar el

```
indexmaker --title="Localhost" --output /var/www/mrtg/index.html /etc/mrtg.cfg
```

y luego para verificar que todo esta bien `env LANG=C /usr/bin/mrtg /etc/mrtg.cfg`, recargando la URL http://sysdw.nteum.org/mrtg/ veremos las dos nuevas gráficas y su actividad.

En las siguientes referencias [34, 10, 11, 17] se puede encontrar más información sobre SNMP (configuración, solución de errores, etc).

1.2.4. Nagios

Nagios es un sistema de monitorización de redes y sistemas ampliamente utilizado por su versatilidad y flexibilidad sobre todo a la hora de comunicar alertas de comportamientos o tendencias de los sistemas monitorizados. Puede monitorizar servicios de red (SMTP, HTTP, SNMP...), recursos del sistema (carga del procesador, uso de los discos, memoria, estado de los puertos...) tanto locales como remotos a través de un *plugin* (llamado NRPE) y se puede extender su funcionalidad a través de estos *plugins*. El producto base de Nagios es llamado **Nagios Core** el cual es *Open Source* sirve de base a otros productos comerciales de Nagios como NagiosXI, IM, NA. Nagios tiene una comunidad muy activa (llamada Nagios Exchange) y en la página web de Nagios (http://www.nagios.org/) se puede acceder a las contribuciones, *plugins* y documentación. Nagios permite consultar prácticamente cualquier parámetro de interés de un sistema, y genera alertas, que pueden ser recibidas por los responsables correspondientes mediante diferentes canales como por ejemplo correo electrónico y mensajes SMS. La instalación básica es muy simple haciendo `apt-get install nagios3` que instalara todas las librerías y plugins para una primera configuración. Durante la

instalación se nos solicitará un *passwd* pero que también posteriormente se puede cambiar haciendo: `cd /etc/nagios3; htpasswd htpasswd.users nagiosadmin`. Luego podremos recargar nuevamente Apache2 y conectando a la URL *http://sysdw.nteum.org/nagios3/* se solicitará el acceso como *nagiosadmin* y el *passwd* que le hemos introducido y podremos ver la estructura de Nagios y observar los servicios monitorizados en cada uno de los apartados.

El archivo de configuración de nagios está en */etc/nagios3/nagios.cfg* y el cual incluye todos los archivos del directorio conf.d del mismo directorio. En ellos tendremos agrupados por archivos los diferentes aspectos a monitorizar, por ejemplo sistemas (localhost_nagios2.cfg) y servicios (services_nagios2.cfg). El fichero más importante probablemente es *localhost_nagios2.cfg* del cual haremos una breve descripción:

```
# definición de un host utilizando un template (generic-host)
define host{
        use                     generic-host
        host_name               localhost
        alias                   localhost
        address                 127.0.0.1
        }
# definición de un servicio -espacio de disco- utilizando un template
# (generic-service) ejecutando un cmd y con advertencia al 20% y error al 10%
# del espacio libre.
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Disk Space
        check_command                   check_all_disks!20%!10%
        }
# ídem para usuarios: advertencia 20 usuarios, crítico 50 usuario
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Current Users
        check_command                   check_users!20!50
        }
#ídem procesos:advertencia 250, crítico 400
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Total Processes
check_command                   check_procs!250!400
        }
# Carga de CPU.
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Current Load
check_command                   check_load!5.0!4.0!3.0!10.0!6.0!4.0
        }
```

Si quisiéramos agregar un servicio (por ejemplo ping al localhost) solo deberíamos agregar al final del archivo:

```
# Ping: advertencia cuando el 20% sea 100ms, crítico 60% sea 500ms.
define service{
        use                             generic-service
        host                            localhost
```

```
service_description             PING
check_command                   check_ping!100.0,20%!500.0,60%
}
```

Los *plugins* son incorporados por */etc/nagios3/nagios.cfg* y están definidos en */etc/nagios-plugins/config* donde se pueden ver las diferentes alternativas para monitorizar.

Dos complementos interesantes para Nagios son PNP4Nagios* y NagVis:

*http://docs.pnp4nagios.org/pnp-0.6/start

1) PNP4Nagios permite almacenar las información que proveen los *plugins* y almacenarla en una base de datos llamada RRD-database y llamar a la herramienta RRD Tool (http://oss.oetiker.ch/rrdtool/) que permite la visualización de estos datos incrustados en la página de Nagios. Por lo cual además del valor instantáneo de las variables a monitorizar también tendremos integrados estos en el tiempo y podremos ver su evolución.

2) NavVis (http://www.nagvis.org/) permite dibujar (sección map) la red de diferentes formas y aspectos para tener diferentes visualizaciones de la red monitorizada en forma activa, es decir viendo los valores de las variables seleccionadas sobre el gráfico.

Por último es interesante considerar **Icinga** (https://www.icinga.org/) que es una bifurcación (*fork*) de Nagios (del 2009) y ha evolucionado mucho en el último tiempo (para algunos ya ha superado a Nagios) y su objetivo es transformarse en una herramienta de referencia dentro del *Open Source* y en el ámbito de la monitorización de redes y sistemas. Icinga nace con la idea de superar las deficiencias en el proceso de desarrollo de Nagios y sus políticas, así como la voluntad de ser más dinámica y fácil agregar nuevas características, como por ejemplo una interfaz de usuario de estilo Web2.0, conectores de base de datos adicionales y una API REST que permita a los administradores integrar numerosas extensiones sin complicadas modificaciones del núcleo. Icinga está disponible en Debian y su configuración es prácticamente similar a Nagios.

1.2.5. Ganglia

Ganglia [12] es una herramienta que permite monitorizar de forma escalable y distribuida el estado de un conjunto de máquinas agrupadas bajo diferentes criterios (red, servicios, etc) o simplemente bajo una misma identificación que llamaremos clúster. La aplicación muestra al usuario las estadísticas de forma remota (por ejemplo, los promedios de carga de la CPU o la utilización de la red) de todas las máquinas que conforman este el clúster basándose en un diseño jerárquico y utiliza comunicaciones punto-a-punto o *multicast* para el intercambio de información entre los diferentes nodos que forman el clúster.

Ganglia utiliza XML para la representación de datos, XDR para el transporte compacto y portátil de datos y RRDtool (http://oss.oetiker.ch/rrdtool/) para almacenamiento de datos y visualización. El sistema se compone de dos *daemons* (gmond y gmetad), una página de visualización (ganglia-webfrontend) basada en PHP.

Gmond es un *daemon* multihilo que se ejecuta en cada nodo del clúster que se desea supervisar (no es necesario tener un sistema de archivos NFS o base de datos ni mantener cuentas especiales de los archivos de configuración). Las tareas de Gmond son: monitorizar los cambios de estado en el *host*, enviar los cambios pertinentes, escuchar el estado de otros nodos (a través de un canal *unicast* o *multicast*) y responder a las peticiones de un XML del estado del clúster. La federación de los nodos se realiza con un árbol de conexiones punto a punto entre los nodos determinados (representativos) del clúster para agregar el estado de los restantes nodos. En cada nodo del árbol se ejecuta el daemon **Gmetad** que periódicamente solicita los datos de los restantes nodos, analiza el XML, guarda todos los parámetros numéricos y exporta el XML agregado por un socket TCP. Las fuentes de datos pueden ser *daemons* gmond, en representación de determinados grupos, u otros *daemons* gmetad, en representación de conjuntos de grupos. Finalmente la web de Ganglia proporciona una vista de la información recogida de los nodos del clúster en tiempo real. Por ejemplo, se puede ver la utilización de la CPU durante la última hora, día, semana, mes o año y muestra gráficos similares para el uso de memoria, uso de disco, estadísticas de la red, número de procesos en ejecución y todos los demás indicadores de Ganglia.

Para la instalación de Ganglia sobre Debian (es similar para otras distribuciones):

1) Instalar los paquetes de Ganglia sobre el servidor web: `apt-get install ganglia-monitor gmetad ganglia-webfrontend`

2) Sobre todos los otros nodos solo se necesita tener instalado el paquete `ganglia-monitor`.

3) Los archivos de configuración están en */etc/ganglia* y en gmond.conf la línea más importante es `data_source "my cluster"localhost` donde indica cual será el nombre del cluster (para agregar todas las máquinas bajo el mismo tag y donde se recogerán los datos (en este caso en localhost). En gmond.conf tenemos las configuraciones generales y de nombres además de los canales de comunicación que por defecto son *multicast*. Si deseamos que sean *unicast* deberemos modificar las secciones udp_send|recv donde la IP de host será la del servidor (gmetad) y comentar las direcciones de *multicast*:

```
udp_send_channel {
#  mcast_join = 239.2.11.71
 host = IP_nodo_gmetad
  port = 8649
  ttl = 1
```

```
}

udp_recv_channel {
#  mcast_join = 239.2.11.71
  port = 8649
#  bind = 239.2.11.71
}
```

4) Por último debemos crear el link entre la configuración de Ganglia-frontend y Apache haciendo

```
ln -s /etc/ganglia-webfrontend/apache.conf /etc/apache2/conf.d/ganglia
```

Reiniciamos Apache (`service apache2 restart`) y nos conectamos a URL *http://sysdw.nteum.org/ganglia/* para visualizar una panorámica global de los sistemas monitorizados y haciendo click en cada imagen/botón podremos obtener más información sobre cada unos de los recurso monitorizados.

1.2.6. Otras herramientas

Otros paquetes interesantes a tener en cuenta para monitorizar un sistema son los siguientes:

- **Zabbix** [35] es un sistema de monitorización (*OpenSource*) que permite recoger el estado de diferentes servicios de red, servidores y hardware de red. Este programa utiliza una base de datos (MySQL, PostgreSQL, SQLite, etc) para mejorar las prestaciones y permite la instalación de agentes Zabbix sobre diferentes máquinas a monitorizar aspectos internos como por ejemplo carga de CPU, utilización de red, espacio en disco, etc. También es posible realizar esta monitorización a través de diferentes protocolos como SNMP, TCP y ICMP, IPMI, etc. y soporta una variedad de mecanismos de notificación en tiempo real, incluyendo XMPP. Su instalación no está disponible en los paquetes de Debian (por diferencias en las políticas de Debian y Zabbix desde 2012) pero se pueden obtener el paquete (.deb) desde la web de Zabbix y seguir la guía de instalación disponible en su web*.

***https://www.zabbix.com/documentation/2.0/manual/installation/install_from_packages**

- **Cacti** [16] es una solución gráfica diseñada para trabajar conjuntamente con datos de RRDTool. Cacti provee diferentes formas de gráficas, métodos de adquisición y características que puede controlar el usuario muy fácilmente y es una solución que se adapta desde una máquina a un entorno complejo de máquinas, redes y servidores.

Ved también

Cacti se describe en el módulo "Clúster, Cloud y Devops".

- **Frysk** [20] donde el objetivo del proyecto es crear un sistema de monitorización distribuido e inteligente para monitorizar procesos e hilos.

Existen un conjunto adicional de herramientas no menos interesantes (no se incluyen las ya mencionadas) que incorpora GNU/Linux para la monitorización de diferentes aspectos del sistema (se recomienda ver el `man` de cada herramienta para mayor información):

- **isag**: *Interactive System Activity Grapher*, para la auditoría de recursos hw/sw.
- **mon**: monitor de servicios de red.
- **diffmon**, **fcheck**: generación de informes sobre cambios en la configuración del sistema y monitorización del sistemas de ficheros para detectar intrusiones.
- **fam**: *file alteration monitor*, monitor de alteración de ficheros.
- **genpower**: monitor para gestionar los fallos de alimentación.
- **ksensors (lm-sensors)**: monitor de la placa base (temperatura, alimentación, ventiladores, etc.).
- **systune**: herramienta para retirar capacidades asignadas al núcleo en el fichero `/proc/sys/kernel`.
- **swatch**: monitor para la actividad del sistema a través de archivos de registro.
- **vtgrab**: monitorización de máquinas remotas (similar a VNC).
- **whowatch**: herramienta en tiempo real para la monitorización de usuarios.
- **wmnd**: monitor de tráfico de red y monitorización de un clúster por red.

1.3. Alta disponibilidad en Linux (High-Availability Linux)

Actualmente Linux es conocido como un sistema operativo estable; los problemas se generan cuando el hardware falla. En los casos en que un fallo de hardware provoca graves consecuencias, debido a la naturaleza del servicio (aplicaciones críticas), se implementan sistemas tolerantes a fallos (*fault tolerant*, FT) con los cuales se garantiza, con una determinada probabilidad (muy alta), que el servicio esté siempre activo. El problema de estos sistemas es que son extremadamente caros, suelen ser soluciones cerradas, totalmente dependientes de la solución integrada. Los sistemas de alta disponibilidad (*high availability*, HA) intentan obtener prestaciones cercanas a la tolerancia a fallos, pero a costes accesibles. La alta disponibilidad está basada en la replicación de elementos, por lo cual dejaremos de tener un servidor y necesitaremos tener un clúster de alta disponibilidad. Existen para Linux diferentes soluciones, como por ejemplo Heartbeat (elemento principal del Linux-HA), ldirectord y LVS (Linux Virtual Server), Piranha (solución basada en LVS de Red Hat), UltraMonkey (solución de VA Linux), o OpenAIS+Corosync+Pacemaker.

El proyecto Linux-HA (Linux de alta disponibilidad) [18] es una solución clúster de alta disponibilidad para Linux y otros sistemas operativos, como FreeBSD, OpenBSD, Solaris y MacOSX y que provee fiabilidad, disponibilidad y prestación discontinua de servicios. El producto principal del proyecto es **Heartbeat**, cuyo objetivo principal es la gestión de clústers con el objetivo de obtener alta disponibilidad. Sus más importantes características son: ilimitado número de nodos (útil tanto para pequeños clústers como para tamaños grandes), monitorización de recursos (estos se pueden reiniciar o desplazar

a otro nodo en caso de fallo), mecanismo de búsqueda para eliminar nodos con fallos del clúster, gestión de recursos basada en directivas o reglas con posibilidad de incluir el tiempo, gestión preconfigurada de recursos (Apache, DB2, Oracle, PostgreSQL, etc.) e interfaz gráfica de configuración. Para poder ser útiles a los usuarios, el *daemon* de Heartbeat tiene que combinarse con un administrador de recursos de clúster (CRM), que tiene la tarea de iniciar y detener los servicios (direcciones IP, servidores web, etc.) lo cual proporcionará la alta disponibilidad. Desde la versión 2.1.3 de Heartbeat se ha sustituido el código del gestor de recursos del clúster (CRM) por el componente Pacemaker. Pacemaker logra la máxima disponibilidad de sus servicios de clúster mediante la detección y recuperación de nodos y los fallos de nivel de servicio. Esto se logra mediante la utilización de las capacidades de mensajería y la pertenencia a la infraestructura proporcionada OpenAIS|Corosync + Pacemaker [25].

1.3.1. Guía breve de instalación de Heartbeat y Pacemaker (Debian)

***La información detallada se puede consultar en** `file:///usr/share/doc/heartbeat/GettingStarted.html`.

En este subapartado se dará una breve descripción de cómo construir un clúster de alta disponibilidad de dos nodos con Heartbeat* para por ejemplo disponer de un servidor Apache de alta disponibilidad. Es interesante (aunque un poco antiguo) el artículo [30] y la documentación del sitio web de Linux-HA [19]. Como punto inicial disponemos de dos ordenadores similares (no es necesario, pero si uno debe ocupar el lugar del otro es aconsejable). A los servidores los llamaremos NteumA (primario) y VteumB (secundario), con una interfaz a red que la utilizaremos para conectarse a la red tanto entre ellos como desde afuera. Para hacer esta prueba de concepto hemos utilizados dos máquinas virtuales (con la red configurada en modo *bridged* para que se vean en la red) pero que es la misma prueba con dos máquinas físicas. Nuestras máquinas están configuradas como NteumA: eth0 = 192.168.1.201 y VteumB: eth0 = 192.168.1.202, las dos tienen como netmask 255.255.255.0 y gateway 192.168.1.1. Y definimos además una dirección virtual donde prestaremos los servicios por ejemplo Apache 192.168.1.200. Configuramos las máquinas para que se vean entre ellas (a través de un ping) y solo es necesario que tengan una instalación mínima con `apache` y `sshd` y que estén configuradas tanto en nombre (`hostname`) como en */etc/hosts* con `IP FDQN alias` (por ejemplo que para cada máquina haya una línea como `192.168.1.201 nteuma.nteum.org nteuma` para todas las máquinas del clúster y deberemos ver el nombre de la máquina como la hemos configurado en */etc/hosts* en el nombre corto, por ejemplo con `uname -n`). Para instalar y configurar Apache2 + Heartbeat hacemos:

1) `apt-get install apache2`

2) modificamos la configuración de Apache agregando a *etc/apache2/ports.conf* la línea `NameVirtualHost 192.168.1.200:80` (dejando las restantes como están).

3) Para que Apache no esté arrancado desde el inicio (ya que queremos que lo haga Heartbeat) lo quitamos de los scripts rc*.d con `update-rc.d apache2 remove`

4) instalamos el paquete `chkconfig` que luego lo necesitará Heartbeat `apt-get install chkconfig`

5) Instalamos el paquete Heartbeat con `apt-get install heartbeat`. Todas estas acciones anteriores las realizamos en las dos máquinas.

6) sobre nteuma (primario) realizamos la configuración de heartbeat. Los ficheros de configuración están en */etc/ha.d/* y las plantillas para configurarlos están */usr/share/doc/heartbeat/*. Para el archivo ha.cf lo modificamos con lo siguiente:

```
logfile /var/log/cluster.log
logfacility local0
warntime 5
deadtime 30
initdead 120
keepalive 2
bcast eth0
udpport 694
auto_failback on
node nteuma
node vteumb
```

Donde *logfile* y *logfacility*: indican donde estará el archivo de log y el nivel de mensajes que queremos, *warntime* es el tiempo que transcurrirá antes que heartbeat nos avise, *deadtime* el tiempo tras el cual heartbeat confirmará que un nodo ha caído, *initdead* el tiempo máximo que heartbeat esperará a que un nodo arranque, *keepalive* el intervalo de tiempo para comprobar la disponibilidad. Además bcast la forma de comunicación (*broadcast*) y la interfaz y *node* los nodos que forma nuestro cluster HA.

7) El archivo authkeys es donde se configura la comunicación entre los nodos del clúster que deberá tener permisos solo de root (`chmod 600 /etc/ha.d/authkeys`):

```
auth 2
2 sha1 MyPaSsWoRd
```

8) el archivo */etc/ha.d/haresources* le indicamos el nodo primario y el servicio (apache2) a levantar:

```
nteuma IPaddr2::192.168.2.100/24/eth0 apache2
```

9) Ahora debemos propagar la configuración a los nodos del cluster (vteumb en nuestro caso pero se haría a todos los nodos indicados en ha.cf):

`/usr/share/heartbeat/ha_propagate`

10) Reiniciamos las máquinas (reboot) para asegurarnos que todo se inicia como deseamos.

11) Desde el *host* u otra máquina externa que vea las anteriores

Para comprobar cual es el servidor que está prestando el servicio hemos incluido en */var/www/index.html* el nombre de la máquina así cuando carguemos la página a través de la conexión a la IP 192.168.1.200 veremos cual máquina es la que presta el servicio. En primer lugar deberemos ver que es NteumA y si quitamos la red de esta (`ifdown eth0`) veremos que en unos segundos al refrescar la página será VteumB.

Consultar la documentación en la siguiente página web: http://www.linux-ha.org/doc/man-pages/man-pages.html. Como comandos para consultar y ver el estado de nuestro cluster podemos utilizar: `cl_status` para ver el estado de todo el sistema, `hb_addnode` envía un mensaje al cluster para añadir nuevos nodos o `hb_delnode` para quitarlos.

Se puede agregar en *ha.cf* la order `crm respawn` (que es el Cluster Resource Manager -crm- de LinuxHA) que iniciará el *daemon* crmd que nos brindará información sobre el estado del cluster y nos permitirá gestionar con una serie de comandos (crm_*) los recursos del cluster. Por ejemplo `crm_mon -1` nos dará información del estado de nuestro cluster.

Como podremos comprobar si agregamos esta línea nuestro simple cluster dejará de funcionar ya que el control de recurso ahora lo está haciendo Pacemaker (crm) y deberemos configurarlo para tal fin (o quitar la línea crm respawn y reiniciar los servicios). Para mayor información consultar [21, 22, 23, 24].

Para configurar el CRM junto con Hearbeat para un servicio activo-pasivo donde si un nodo falla el nodo pasivo adoptará su función (en nuestro caso Apache). En nuestro caso utilizaremos Heartbeat para mantener la comunicación entre los nodos y Pacemaker como Resource Manager que provee el control y administración de los recursos provistos por el clúster. Básicamente Pacemaker puede manejar diferentes tipos de recursos llamados LSB que serán los que provee GNU/Linux y que pueden gestionarse a través de */etc/init.d* y OCF que nos permitirá inicializar una dirección virtual, monitorizar un recurso, iniciar/parar un recurso, cambiar su orden de servicio, etc.

1) Partimos que ya hemos instalado heartbeat and pacemaker (este último se instala cuando se instala heartbeat) y sino podemos hacer `apt-get install heartbeat pacemaker`

2) A la configuración de */etc/ha.d/ha.cf* mencionada anteriormente agregamos `crm respawn` para iniciar pacemaker (veremos que hay un proceso lammado `crmd`)

3) */etc/ha.d/authkeys* lo dejamos como ya lo teníamos configurado.

4) Reiniciamos el servicio `service heartbeat restart`

5) Ahora podremos ver es estado del cluster con `crm status`

```
============
Last updated: Tue Jul 1 12:06:36 2014
Stack: Heartbeat
Current DC: vteumb (...) - partition with quorum
...
2 Nodes configured, unknown expected votes
0 Resources configured.
============
Online: [ veteumb nteuma ]
```

6) Deshabilitamos stonith que no lo necesitaremos: `crm configure property stonith-enabled=false`

7) Inicializamos los nodos para el quorum:

```
crm configure property expected-quorum-votes="2"
```

8) Para tener quorum, la mitad de los nodos del cluster debe ser online (Nro. de nodos/2)+1 pero en un cluster de 2 nodos esto no ocurre cuando falla un nodo por lo tanto necesitamos inicializar la política a *ignore*: `crm configure property no-quorum-policy=ignore`

9) Para prever el *failback* de un recurso: `crm configure rsc_defaults resource-stickiness=100`

10) Podemos listar los objectos OCF con `crm ra list ocf` y en nuestro caso utilizaremos IPaddr2. Para obteber más información sobre él: `crm ra info ocf:IPaddr2`

11) Agregamos una IP virtual (VIP) a nuestro cluster:

```
crm configure primitive havip1 ocf:IPaddr2 params ip=192.168.1.200
cidr_netmask=32 nic=eth0 op monitor interval=30s
```

12) Verificamos el recurso havip1 se encuentra en el primer nodo ejecutando `crm status` y nos dará una info similar a la anterior pero con una línea como *havip1 (ocf::heartbeat:IPaddr2): Started nteuma*

13) Agregamos el daemon a nuestro cluster: `crm ra info ocf:anything`

14) Agregamos apache a nuestro cluster

```
crm configure primitive apacheha lsb::apache2 op monitor interval=15s
```

15) Inicializamos el recurso VIP y apache en el mismo nodo: `crm configure colocation apacheha-havip1 INFINITY: havip1 apacheha` y veremos con `crm status` que *havip1 (ocf::heartbeat:IPaddr2): Started nteuma apacheha (lsb:apache2): Started nteuma*

16) Podemos configurar el orden de los servicios:

```
crm configure order ip-apache mandatory: havip1 apacheha
```

17) O migrar un servicio a otro nodo: `crm resource migrate apacheha vteumb` (con lo que si nos conectamos a 192.168.1.200 veremos que el servicio es provisto por un servidor o por otro). Este comando lo podemos usar para hacer mantenimientos transfiriendo el servicio de un sitio a otro sin interrumpirlo. Si ejecutamos la instrucción `crm status` veremos entonces *havip1 (ocf::heartbeat:IPaddr2): Started nteuma apacheha (lsb:apache2): Started vteumb*

18) La configuración de nuestro cluster la podemos ver con `crm configure show`:

19) Es interesante ejecutar el navegador con la URL 192.168.1.200 e ir desactivando los nodos o recuperándolos para ver como adopta el rol cada uno y mantienen el servicio (mirar el estado entre cambio y cambio con `crm status`)

Si comentamos la línea `crm respawn` en */etc/ha.d/ha.cf* volveremos a la gestión básica anterior sin el gestor de recursos Pacemaker.

1.3.2. DRBD

El **Distributed Replicated Block Device** (DRBD) es un almacenamiento distribuido sobre múltiples *hosts* donde, a igual que en RAID1, los datos son replicados sobre el sistema de archivo sobre los otros *hosts* y sobre TCP/IP.[26] En esta prueba de concepto utilizaremos las mismas máquinas utilizadas en Heartbeat a las cuales le hemos adicionado una unidad de disco más (*/dev/sdb*) y hemos creado una partición sobre ellas */dev/sdb1*). La instalación es:

1) Instalar en ambas máquinas DRBD: `apt-get install drbd8-utils` donde los archivos de configuración estarán en */etc/drbd.d/* (existe un archivo */etc/drbd.conf* pero que incluye a los archivos del directorio mencionado).

2) Crearemos la configuración del recurso como */etc/drbd.d/demo.res* con el siguiente contenido:

```
resource drbddemo {
   meta-disk internal;
   device /dev/drbd1;
   syncer {
       verify-alg sha1;
   }
   net {
       allow-two-primaries;
   }
   on nteuma {
       disk /dev/sdb1;
       address 192.168.1.201:7789;
   }
   on vteumb {
       disk /dev/sdb1;
       address 192.168.1.202:7789;
   }
}
```

3) Copiamos el archivo:

```
scp /etc/drbd.d/demo.res vteum:/etc/drbd.d/demo.res
```

4) Inicializamos en los dos nodos el dispositivo ejecutando:

```
drbdadm create-md drbddemo
```

5) Sobre nteum (verficar con `uname -n`) hacemos: `modprobe drbd` para cargar el módulo de kernel, luego `drbdadm up drbddemo` para levantar el dispositivo y lo podremos mirar con `cat /proc/drbd` que no indicará (entre otros mensajes) que está:

cs:WFConnection ro:Secondary/Unknown ds:Inconsistent/DUnknown C r—-

6) sobre vteumb hacemos: `modprobe drbd` para cargar el módulo del kernel, `drbdadm up drbddemo` para inicializar el dispositivo y hacemos `drbdadm – –overwrite-data-of-peer primary drbddemo` para poderlo configurar como primario. Si miramos la configuración con `cat /proc/drbd` veremos entre otros mensajes

1: cs:SyncSource ro:Primary/Secondary ds:UpToDate/Inconsistent C r—— y más abajo [>.....................] sync'ed: 0.4

7) En ambas máquinas veremos el dispositivo */dev/drbd1* pero solo desde aquel que es primario lo podremos montar (no desde el que es secundario). Para ello instalamos las herramientas para crear un *XFS filesystem* con la instrucción `apt-get install xfsprogs` y hacemos sobre el primario (vteumb) `mkfs.xfs /dev/drbd1` y luego podremos hacer montarlo `mount /dev/drb1 /mnt` y podremos copiar un archivo como

`cp /var/log/messages /mnt/test`.

8) Para ejecutar unos pequeños tests podemos hacer sobre vteum: `umount /mnt` y luego `drbdadm secondary drbddemo` y sobre nteum primero debemos instalar XFS `apt-get install xfsprogs` luego `drbdadm primary drbddemo` y finalmente `mount -t xfs /dev/drbd1 /mnt`. Podemos comprobar el contenido con `ls /mnt` y veremos los archivos que copiamos en vteum. Con `cat /proc/drbd` en ambas máquinas veremos el intercambio de roles primario por secundario y viceversa.

9) Un segundo test que podemos hacer es apagar el nodo vteum (secundario) para simular un fallo y veremos con `cat /proc/drbd` que el otro nodo no está disponible (*0: cs:WFConnection ro:Primary/Unknown*). Si copiamos hacemos `cp /mnt/test /mnt/test1` y ponemos en marcha vteum, cuando hace el boot sincronizará los datos y veremos sobre nteum *0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r—* y sobre vteum estarán los archivos sincronizados.

10) Si provocamos un fallo sobre el primario (apagándolo o desconectando la red) y tenemos montado el directorio (es lo que conocemos como *shutdown* no ordenado) no habrá problemas para ir al otro nodo, cambiarlo a primario y montarlo. Si recuperamos el nodo primario que que falló veremos que ambos quedan como secundarios cuando vuelve a estar activo por lo cual lo deberemos poner como primario y lo podremos montar nuevamente y en el secundario veremos que se ha sincronizado y luego en unos segundo volverá a su rol de secundario/primario.

11) Si han cambios durante el tiempo que un nod ha estado desconectado como primario obtendremos un mensaje de "split-brain" para recuperarlo deberemos ejecutar sobre el secundario `drbdadm secondary drbddemo`

y `drbdadm --discard-my-data connect drbddemo` y sobre el primario `drbdadm connect drdbdemo` para conectar los nodos y resolver el conflicto. [27]

1.3.3. DRBD + Heartbeat como NFS de alta disponibilidad

El objetivo de este apartado es mostrar como utilizar DRBD y Heartbeat para generar un clúster NFS que permita tener una copia del NFS y que entre en servicio cuando el servidor primario deje de funcionar. Para ello utilizaremos la misma instalación del apartado anterior pero sobre otro disco en cada nodo (sdc) sobre el cual hemos creado una partición (*/dev/sdc1*) y es importante verificar que tenemos los módulos de DRBD instalados (`lsmod | grep drbd`) [28, 29]. A continuación seguimos los siguientes pasos:

1) Creamos un archivo */etc/drbd.d/demo2.res* con el siguiente contenido:

```
resource myrs {
    protocol C;
    startup { wfc-timeout 0; degr-wfc-timeout 120; }
    disk { on-io-error detach; }
    on nteuma {
        device /dev/drbd2;
        disk /dev/sdc1;
        meta-disk internal;
        address 192.168.1.201:7788;
        }
    on vteumb {
        device /dev/drbd2;
        disk /dev/sdc1;
        meta-disk internal;
        address 192.168.1.202:7788;
        }
    }
```

Donde utilizamos la partición de cada disco sobre cada nodo y generaremos el dispositivo */etc/drbd2*. Es importante que los nombres de las máquinas sea exactamente el que nos da `uname -n` y modificar */etc/hosts*, */etc/resolv.conf* y */etc/hostname* para que las máquinas tengan conectividad entre ellas (y con el exterior) a través del nombre y de la IP. Esto se debe realizar sobre las dos máquinas incluyendo la copia del archivo anterior.

2) También sobre las dos máquinas deberemos ejecutar: `drbdadm create-md myrs` para inicializar el dispositivo, `drbdadm up myrs` para activarlo, `drbdadm syncer myrs` para sincronizarlo. Podremos ver el resultado con la instrucción `cat /proc/drbd` en en cual nos mostrará los dispositivos como "Connected" e inicializados.

3) Sobre el servidor primario ejecutamos

`drbdadm - -overwrite-data-of-peer primary myrs`

para indicarle que sea primario y visualizando el archivo */proc/drbd* veremos el resultado.

4) Finalmente ejecutamos/reiniciamos el servicio con la instrucción `service drbd start|restart` con lo cual tendremos un dispositivo */dev/drbd2* preparado para configurar el sistema de archivo.

5) En este caso utilizaremos LVM ya que permite más flexibilidad para gestionar las particiones pero se podría utilizar */dev/drbd2* como dispositivo de bloques simplemente. Instalamos LVM (`apt-get install lvm2`) y ejecutamos:

```
    pvcreate /dev/drbd2                  Creamos la particón LVM física
    pvdisplay                            Visualizamos
    vgcreate myrs /dev/drbd2             Creamos el grupo llamado myrs
    lvcreate -L 20 M -n web_files myrs   Creo una partición lógica web_files
    lvcreate -L 20 M -n data_files myrs  Creo otra partición lógica data_files
    lvdisplay                            Visualizamos

Con esto deberemos tener disponibles las particiones en /dev/myrs/web_files
y /dev/myrs/data_files.
Con ello creamos el sistema de archivos y lo montamos:
    mkfs.ext4 /dev/myrs/web_files
    mkfs.ext4 /dev/myrs/data_files
    mkdir /data/web-files                          Creamos los punto de montaje
    mkdir /data/data-files
    mount /dev/myrs/web_files /data/web-files      Montamos las particiones
    mount /dev/myrs/data_files /data/data-files
```

6) Ahora deberemos instalar y configurar el servidor NFS (sobre los dos servidores) para que pueda ser gestionado por Heartbeat y exportarlo a los clientes. Para ello ejecutamos (`apt-get install nfs-kernel-server`) y editamos el archivo */etc/exports* con el siguiente contenido:

```
/data/web-files 192.168.1.0/24(rw,async,no_root_squash,no_subtree_check,fsid=1)
/data/data-files 192.168.1.0/24(rw,async,no_root_squash,no_subtree_check,fsid=2)
```

Es importante el valor del parámetro 'fsid' ya que con él los clientes de sistema de archivo sobre el servidor primario sabrán que son los mismo que en el servidor secundario y si el primario queda fuera no se bloquearán esperando que vuelva a estar activo sino que continuarán trabajando con el secundario. Como dejaremos que Heartbeat gestione el NFS lo debemos quitar de *boot* con `update-rc.d -f nfs-common remove` y `update-rc.d -f nfs-kernel-server remove`.

7) Finamente debemos configurar Heartbeat y para ello modificamos el archivo */etc/ha.d/ha.cf* con:

```
autojoin none
auto_failback off
keepalive 2
warntime 5
deadtime 10
initdead 20
bcast eth0
node nteuma
node vteumb
logfile /var/log/ha-log
debugfile /var/log/had-log
```

Se ha indicado 'auto_failback = off' ya que no deseamos que vuelva al original cuando el primario retorne (lo cual podría ser deseable si el hw del primario es mejor que el del secundario en cuyo caso se debería poner a 'on'). 'deadtime' indica que considerá el servidor fuera de servicio luego de 10s, y cada 2s preguntará si están vivos. Dejamos el ficheros */etc/ha.d/authkeys* como ya lo teníamos definido y ejecutamos */usr/share/heartbeat/ha_propagate* para copiar los archivos en el otro servidor (también se podría hacer manualmente).

8) Para indicar una dirección virtual a los clientes NFS usaremos 192.168.1.200 así ellos siempre tendrán esta IP como referente independientemente de quien les esté prestado el servicio. El siguiente paso será realizar la modificación del archivo */etc/ha.d/haresources* para indicarle a Hearbeat cuales son los servicios a gestionar (IPV, DRBD, LVM2, Filesystems y NFS) los cuales los deberemos introducir en el orden que se necesiten:

```
nteum \
IPaddr::192.168.1.200/24/eth0 \
drbddisk::myrs \
lvm2 \
Filesystem::/dev/myrs/web_files::/data/web-files::ext4::nosuid,usrquota,noatime \
Filesystem::/dev/myrs/data_files::/data/data-files::ext4::nosuid,usrquota,noatime \
nfs-common \
nfs-kernel-server
```

Se debe copiar este archivo en los dos servidores (sin modificar) ya que este indica que nteuma es el servidor primario y cuando falle será vteumb tal y como lo hemos explicado en ha.cf.

9) Finalmente podremos iniciar|reiniciar Heartbeat (con la orden `service heartbeat start|restart`) y probar a montar el servidor en una misma red y hacer las pruebas de fallo correspondientes.

Actividades

1. Realizad una monitorización completa del sistema con las herramientas que consideréis adecuadas y haced un diagnóstico de la utilización de recursos y cuellos de botella que podrían existir en el sistema. Simular la carga en el sistema del código de `sumdis.c` dado en el módulo "Clúster, Cloud y DevOps". Por ejemplo, utilizad: `sumdis 1 2000000`.

2. Cambiad los parámetros del núcleo y del compilador y ejecutad el código mencionado en la actividad anterior (`sumdis.c`) con, por ejemplo: `time ./sumdis 1 1000000`.

3. Con la ejecución de las dos actividades anteriores extraed conclusiones sobre los resultados.

4. Con el programa de iteraciones indicado en este módulo, implementad un forma de terminar la ejecución en 5 segundos, independiente de las llamadas al sistema excepto que sea solo una vez.

5. Monitorizad todos los programas anteriores con Munin y Ganglia extrayendo conclusiones sobre su funcionalidad y prestaciones.

6. Registrad cuatro servicios con Monin y haced la gestión y seguimiento por medio del programa.

7. Instalad MRTG y monitorizad la CPU de la ejecución de los programas anteriores.

8. Instalad y experimentad con los sistemas descritos de alta disponibilidad (hearbeat, pacemaker, drbd).

9. Con Heartbeat + DRBD crear un sistema de archivos NFS redundante y hacer las pruebas de fallo de red sobre el servidor primario (deshabilitando/habilitando la red) para que el servidor secundario adquiera el control y luego devuelva el control a este cuando el primario recupere la red.

Bibliografía

[1] **Eduardo Ciliendo, Takechika Kunimasa** (2007). *Linux Performance and Tuning Guidelines*.
<http://www.redbooks.ibm.com/redpapers/pdfs/redp4285.pdf>

[2] **Nate Wiger** *Linux Network Tuning for 2013*.
<http://www.nateware.com/linux-network-tuning-for-2013.html>

[3] **Bravo E., D.** (2006). *Mejorando la Performance en Sistemas Linux/Unix*. GbuFDL1.2. <diegobravoestrada@hotmail.com>
<http://es.tldp.org/Tutoriales/doc-tut-performance/perf.pdf>

[4] **Mourani, G.** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.

[5] *Optimización de servidores Linux*.
<http://people.redhat.com/alikins/system_tuning.html>

[6] *Performance Monitoring Tools for Linux*.
<http://www.linuxjournal.com/article.php?sid=2396>

[7] *Monit*.
<http://mmonit.com/monit/>

[8] *Munin*.
<http://munin-monitoring.org/>

[9] *MRTG*.
<http://oss.oetiker.ch/mrtg/>

[10] **M. Rushing** *Fix for MRTG Generating SNMP_Session Error in Debian Wheezy*.
<http://mark.orbum.net/2013/06/07/fix-for-mrtg-generating-snmp_session-error-in-debian-wheezy-and-possibly-ubuntu/>

[11] *SNMP - Debian*.
<https://wiki.debian.org/SNMP>

[12] *Ganglia Monitoring System*.
<http://ganglia.sourceforge.net/>

[13] *Monitorización con SNMP y MRTG*.
<http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance>

[14] **D. Valdez** *Configuración rápida de MRTG*.
<http://sysnotas.blogspot.com.es/2013/06/mrtg-configuracion-rapida-para-debian.html>

[15] Howto *sobre instalación de MRTG en Debian*.
<http://preguntaslinux.org/-howto-instalacion-de-mrtg-monitoreo-debian-t-3061.html>

[16] *Cacti*.
<http://cacti.net/>

[17] *Net-SNMP - MIBs*.
<http://www.net-snmp.org/docs/readmefiles.html>

[18] *Linux-HA*.
<http://linux-ha.org/wiki/Main_Page>

[19] *Documentación de Linux-HA*.
<http://www.linux-ha.org/doc/users-guide/users-guide.html>

[20] *Frysk*.
<http://sources.redhat.com/frysk/>

[21] *Pacemaker documentation*.
<http://clusterlabs.org/doc/>

[22] *Pacemaker Cluster From Scratch*.
<http://clusterlabs.org/doc/en-US/Pacemaker/1.0/pdf/Clusters_from_Scratch/Pacemaker-1.0-Clusters_from_Scratch-en-US.pdf>

[23] **I. Mora Perez**.*Configuring a failover cluster with heartbeat + pacemaker*.
<http://opentodo.net/2012/04/configuring-a-failover-cluster-with-heartbeat-pacemaker/>

[24] **F. Diaz**.*Alta Disponibilidad con Apache2 y Heartbeat en Debian Squeeze*.
<http://www.muspells.net/blog/2011/04/alta-disponibilidad-con-apache2-y-heartbeat-en-debian-squeeze/>

[25] *Pacemarker*.
<http://clusterlabs.org/doc/en-US/Pacemaker/1.0/html/Pacemaker_Explained/s-intro-pacemaker.html>

[26] *DRBD*.
<http://www.drbd.org/home/what-is-drbd/>

[27] *DRBD tests*.
<https://wiki.ubuntu.com/Testing/Cases/UbuntuServer-drbd>

[28] **R. Bergsma**.*Redundant NFS using DRBD+Heartbeat*.
<http://blog.remibergsma.com/2012/09/09/building-a-redundant-pair-of-linux-storage-servers-using-drbd-and-heartbeat/>

[29] **G. Armer**.*Highly Available NFS Cluster on Debian Wheezy*.
<http://sigterm.sh/2014/02/highly-available-nfs-cluster-on-debian-wheezy/>

[30] **Leung, C. T**. *Building a Two-Node Linux Cluster with Heartbeat*.
<http://www.linuxjournal.com/article/5862>

[31] **Majidimehr, A.** (1996). *Optimizing UNIX for Performance*. Prentice Hall.

[32] *Monitor Debian servers with monit*.
<http://www.debian-administration.org/articles/269>

[33] *Monitorización con Munin y monit*.
<http://www.howtoforge.com/server_monitoring_monit_munin>

[34] *Monitorización con SNMP y MRTG*.
<http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance>

[35] *The Enterprise-class Monitoring Solution for Everyone*.
<http://zabbix.com>

Clúster, Cloud y DevOps

Remo Suppi Boldrito

PID_00212475

Universitat Oberta de Catalunya

Índice

Introducción 5

Objetivos 8

1. Clusterización 9
1.1. Virtualización 9
1.1.1. Plataformas de virtualización 10
1.2. Beowulf 12
1.2.1. ¿Cómo configurar los nodos? 14
1.3. Beneficios del cómputo distribuido 15
1.3.1. ¿Cómo hay que programar para aprovechar la concurrencia? 17
1.4. Memoria compartida. Modelos de hilos (*threading*) 19
1.4.1. Multihilos (*multithreading*) 19
1.5. OpenMP 23
1.6. MPI, *Message Passing Interface* 27
1.6.1. Configuración de un conjunto de máquinas para hacer un clúster adaptado a OpenMPI 28
1.7. Rocks Cluster 31
1.7.1. Guía rápida de instalación 32
1.8. FAI 33
1.8.1. Guía rápida de instalación 34
1.9. Logs 36
1.9.1. Octopussy 38
1.9.2. Herramientas de monitorización adicionales 39

2. *Cloud* 41
2.1. Opennebula 44

3. DevOps 49
3.1. Linux Containers, LXC 50
3.2. Docker 53
3.3. Puppet 55
3.3.1. Instalación 56
3.4. Chef 60
3.5. Vagrant 62

Actividades 65

Bibliografía 66

Introducción

Los avances en la tecnología han llevado, por un lado, al desarrollo de procesadores más rápidos, con más de un elemento de cómputo (núcleos o *cores*) en cada uno de ellos, de bajo coste y con soporte a la virtualización *hardware* y por otro, al desarrollo de redes altamente eficientes. Esto, junto con el desarrollo de sistemas operativos como GNU/linux, ha favorecido un cambio radical en la utilización de sistemas de procesadores de múltiple-cores y altamente interconectados, en lugar de un único sistema de alta velocidad (como los sistemas vectoriales o los sistemas de procesamiento simétrico SMP). Además, estos sistemas han tenido una curva de despliegue muy rápida, ya que la relación precio/prestaciones ha sido, y lo es cada día más, muy favorable tanto para el responsable de los sistemas TIC de una organización como para los usuarios finales en diferentes aspectos: prestaciones, utilidad, fiabilidad, facilidad y eficiencia. Por otro lado, las crecientes necesidades de cómputo (y de almacenamiento) se han convertido en un elemento tractor de esta tecnología y su desarrollo, vinculados a la provisión dinámica de servicios, contratación de infraestructura y utilización por uso -incluso en minutos- (y no por compra o por alquiler), lo cual ha generado una evolución total e importante en la forma de diseñar, gestionar y administrar estos centros de cómputo. No menos importante han sido los desarrollos, además de los sistemas operativos, para soportar todas estas tecnologías, en lenguajes de desarrollo, API y entornos (*frameworks*) para que los desarrolladores puedan utilizar la potencialidad de la arquitectura subyacente para el desarrollo de sus aplicaciones, a los administradores y gestores para que para que puedan desplegar, ofrecer y gestionar servicios contratados y valorados por minutos de uso o bytes de E/S, por ejemplo, y a los usuarios finales para que puedan hacer un uso rápido y eficiente de los sistemas de cómputo sin preocuparse de nada de lo que existe por debajo, dónde está ubicado y cómo se ejecuta.

Es por ello por lo que, en el presente capítulo, se desarrollan tres aspectos básicos que son parte de la misma idea, a diferentes niveles, cuando se desea ofrecer la infraestructura de cómputo de altas prestaciones. O una plataforma como servicio o su utilización por un usuario final en el desarrollo de una aplicación, que permita utilizar todas estas infraestructuras de cómputo distribuidas y de altas prestaciones. En sistemas de cómputo de altas prestaciones (*High Performance Computing -HPC*), podemos distinguir dos grandes configuraciones:

1) Sistemas fuertemente acoplados (*tightly coupled systems*): son sistemas donde la memoria es compartida por todos los procesadores (*shared memory systems*) y la memoria de todos ellos "se ve" (por parte del programador) como una única memoria.

2) Sistemas débilmente acoplados (*loosely coupled systems*): no comparten memoria (cada procesador posee la suya) y se comunican mediante mensajes pasados a través de una red (*message passing systems*).

En el primer caso, son conocidos como *sistemas paralelos de cómputo* (*parallel processing system*) y en el segundo, como *sistemas distribuidos de cómputo* (*distributed computing systems*).

En la actualidad la gran mayoría de los sistemas (básicamente por su relación precio-prestaciones) son del segundo tipo y se conocen como clústers donde los sistemas de cómputo están interconectados por una red de gran ancho de banda y todos ellos trabajan en estrecha colaboración, viéndose para el usuario final como un único equipo. Es importante indicar que cada uno de los sistemas que integra el clúster a su vez puede tener más de un procesador con más de un *core* en cada uno de ellos y por lo cual, el programador deberá tener en cuenta estas características cuando desarrolle sus aplicaciones (lenguaje, memoria compartida|distribuida, niveles de caché, etc) y así poder aprovechar toda la potencialidad de la arquitectura agregada. El tipo más común de clúster es el llamado Beowulf, que es un clúster implementado con múltiples sistemas de cómputo (generalmente similares pero pueden ser heterogéneos) e interconectados por una red de área local (generalmente Ethernet, pero existen redes más eficientes como Infiniband o Myrinet). Algunos autores denominan *MPP (massively parallel processing)* cuando es un clúster, pero cuentan con redes especializadas de interconexión (mientras que los clústers utilizan hardware estándar en sus redes) y están formados por un alto número de recursos de cómputo (1.000 procesadores, no más). Hoy en día la mayoría de los sistemas publicados en el TOP500 (http://www.top500.org/system/177999) son clústers y en la última lista publicada (2014) ocupaba el primer puesto el ordenador Tianhe-2 (China) con 3,1 millones de cores, 1 Petabyte de memoria RAM (1000 Tbytes) y un consumo de 17 MW.

El otro concepto vinculado a los desarrollos tecnológicos mencionados a las nuevas formas de entender el mundo de las TIC en la actualidad es el *Cloud Computing* (o cómputo|servicios en la nube) que surge como concepto de ofrecer servicios de cómputo a través de Internet y donde el usuario final no tiene conocimiento de dónde se están ejecutando sus aplicaciones y tampoco necesita ser un experto para contactar, subir y ejecutar sus aplicaciones en minutos. Los proveedores de este tipo de servicio tienen recursos (generalmente distribuidos en todo el mundo) y permiten que sus usuarios contraten y gestionen estos recursos en forma *on-line* y sin la mayor intervención y en forma casi automática, lo cual permite reducir los costes teniendo ventajas (además de que no se tiene que instalar-mantener la infraestructura física -ni la obra civil-) como la fiabilidad, flexibilidad, rapidez en el aprovisionamiento, facilidad de uso, pago por uso, contratación de lo que se necesita, etc.. Obviamente, también cuenta con sus desventajas, como lo son la dependencia de un proveedor y la centralización de las aplicaciones|almacenamiento de datos, servicio vinculado a la disponibilidad de acceso a Internet, cuestiones de seguridad, ya

que los datos 'sensibles' del negocio no residen en las instalaciones de las empresas y pueden generar riesgos de sustracción/robo de información, confiabilidad de los servicios prestados por el proveedor (siempre es necesarios firmar una SLA -contrato de calidad de servicio-), tecnología susceptible al monopolio, servicios estándar (solo los ofrecidos por el proveedor), escalabilidad o privacidad.

Un tercer concepto vinculado a estas tecnologías, pero probablemente más del lado de los desarrolladores de aplicaciones y *frameworks*, es el de DevOps. DevOps surge de la unión de las palabras *Development* (desarrollo) y *Operations* (operaciones), y se refiere a una metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de operaciones|administradores en las tecnologías de la información (IT). DevOps se presenta como una metodología que da respuesta a la relación existente entre el desarrollo de software y las operaciones IT, teniendo como objetivo que los productos y servicios software desarrollados por una entidad se puedan hacer más eficientemente, tengan una alta calidad y además sean seguros y fáciles de mantener. El término DevOps es relativamente nuevo y fue popularizado a través de *DevOps Open Days* (Bélgica, 2009) y como metodología de desarrollo ha ido ganando adeptos desde grande a pequeñas compañías, para hacer más y mejor sus productos y servicios y, sobre todo, debido a diferentes factores con una fuerte presencia hoy en día como: el uso de los procesos y metodologías de desarrollo ágil, necesidad de una mayor tasa de versiones, amplia disponibilidad de entornos virtualizados|*cloud*, mayor automatización de centros de datos y aumento de las herramientas de gestión de configuración.

En este módulo se verán diferentes formas de crear y programar un sistema de cómputo distribuido (clúster|*cloud*), las herramientas y librerías más importantes para cumplir este objetivo, y los conceptos y herramientas vinculadas a las metodológicas DevOps.

Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Analizar las diferentes infraestructuras y herramientas para el cómputo de altas prestaciones (incluida la virtualización) (HPC).
2. Configurar e instalar un clúster de HPC y las herramientas de monitorización correspondientes.
3. Instalar y desarrollar programas de ejemplos en las principales API de programación: Posix Threads, OpenMPI, y OpenMP.
4. Instalar un clúster específico basado en una distribución ad hoc (Rocks).
5. Instalar una infraestructura para prestar servicios en *cloud* (IasS).
6. Herramientas DevOps para automatizar un centros de datos y gestiones de la configuración.

1. Clusterización

La historia de los sistemas informáticos es muy reciente (se puede decir que comienza en la década de 1960). En un principio, eran sistemas grandes, pesados, caros, de pocos usuarios expertos, no accesibles y lentos. En la década de 1970, la evolución permitió mejoras sustanciales llevadas a cabo por tareas interactivas (*interactive jobs*), tiempo compartido (*time sharing*), terminales y con una considerable reducción del tamaño. La década de 1980 se caracteriza por un aumento notable de las prestaciones (hasta hoy en día) y una reducción del tamaño en los llamados microordenadores. Su evolución ha sido a través de las estaciones de trabajo (*workstations*) y los avances en redes (LAN de 10 Mb/s y WAN de 56 kB/s en 1973 a LAN de 1/10 Gb/s y WAN con ATM, *asynchronous transfer mode* de 1,2 Gb/s en la actualidad o redes de alto rendimiento como Infiniband -96Gb/s- o Myrinet -10Gb/s-), que es un factor fundamental en las aplicaciones multimedia actuales y de un futuro próximo. Los sistemas distribuidos, por su parte, comenzaron su historia en la década de 1970 (sistemas de 4 u 8 ordenadores) y su salto a la popularidad lo hicieron en la década de 1990. Si bien su administración, instalación y mantenimiento pueden tener una cierta complejidad (cada vez menos) porque continúan creciendo en tamaño, las razones básicas de su popularidad son el incremento de prestaciones que presentan en aplicaciones intrínsecamente distribuidas (aplicaciones que por su naturaleza son distribuidas), la información compartida por un conjunto de usuarios, la compartición de recursos, la alta tolerancia a los fallos y la posibilidad de expansión incremental (capacidad de agregar más nodos para aumentar las prestaciones y de forma incremental). Otro aspecto muy importante en la actualidad es la posibilidad, en esta evolución, de la virtualización. Las arquitecturas cada vez más eficientes, con sistemas *multicores*, han permitido que la virtualización de sistemas se transforme en una realidad con todas las ventajas (y posibles desventajas) que ello comporta.

1.1. Virtualización

La virtualización es una técnica que está basada en la abstracción de los recursos de un ordenador, llamada *Hypervisor* o VMM (*Virtual Machine Monitor*) que crea una capa de separación entre el hardware de la máquina física (*host*) y el sistema operativo de la máquina virtual (*virtual machine*, *guest*), y es un medio para crear una "versión virtual" de un dispositivo o recurso, como un servidor, un dispositivo de almacenamiento, una red o incluso un sistema operativo, donde se divide el recurso en uno o más entornos de ejecución. Esta capa de software (VMM) maneja, gestiona y administra los cuatro recur-

sos principales de un ordenador (CPU, memoria, red y almacenamiento) y los reparte de forma dinámica entre todas las máquinas virtuales definidas en el computador central. De este modo nos permite tener varios ordenadores virtuales ejecutándose sobre el mismo ordenador físico.

La máquina virtual, en general, es un sistema operativo completo que se ejecuta como si estuviera instalado en una plataforma de hardware autónoma.

Enlace de interés

Para saber más sobre virtualización podéis visitar: http://en.wikipedia.org/wiki/Virtualization.
Se puede consultar una lista completa de programas de virtualización en: http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines.

1.1.1. Plataformas de virtualización

Los ejemplos más comunes de plataforma de virtualización con licencias GPLs o similiares son Xen, KVM, Qemu (emulación), OpenVz, VirtualBox, Oracle VM (solo el servidor), y entre las plataforma propietarias (algunas como *free to use*) VMware ESX/i, Virtual PC/Hyper-V, Parallels, Virtuozzo. Existen diferentes taxonomías para clasificar la virtualización (y algunas plataformas pueden ejecutarse en más de una categorías) siendo la más conocida la siguiente:

1) Virtualización por HW: considerando como *Full Virtualization* cuando la máquina virtual simula todo el HW para permitir a un sistema *guest* ejecutarse sin modificaciones (siempre y cuando esté diseñado para el mismo set de instrucciones). Fue la 1.ª generación de VM para procesadores x86 y lo que hace es una captura de instrucciones que acceden en modo prioritario a la CPU y las transforma en una llamada a la VM para que sean emuladas por software. En este tipo de virtualización pueden ejecutarse Parallels, VirtualBox, Virtual Iron, Oracle VM, Virtual PC, Hyper-V, VMware por ejemplo.

2) Virtualización asistida por hardware, donde el hardware provee soporte que facilita la construcción y trabajo del VM y mejora notablemente las prestaciones (a partir de a partir de 2005/6 Intel y AMD proveen este soporte como VT-x y AMD-V respectivamente). Las principales ventajas, respecto a otras formas de virtualización, es que no se debe tocar el sistema operativo *guest* (como en paravirtualization) y se obtienen mejores prestaciones, pero como contrapartida, se necesita soporte explícito en la CPU, lo cual no está disponible en todos los procesadores x86/86_64.En este tipo de virtualización pueden ejecutarse KVM, VMware Fusion, Hyper-V, Virtual PC, Xen, Parallels, Oracle VM, VirtualBox.

3) Paravirtualization: es una técnica de virtualización donde la VM no necesariamente simula el HW, sino que presenta una API que puede ser utilizada por el sistema *guest* (por lo cual, se debe tener acceso al código fuente para reemplazar las llamadas de un tipo por otro). Este tipo de llamadas a esta API se denominan *hypercall* y Xen puede ejecutarse en esta forma.

4) Virtualización parcial: es lo que se denomina *Address Space Virtualization*. La máquina virtual simula múltiples instancias del entorno subyacente del hardware (pero no de todo), particularmente el *address space*. Este tipo de virtualización acepta compartir recursos y alojar procesos, pero no permite instancias separadas de sistemas operativos *guest* y se encuentra en desuso actualmente.

5) Virtualización a nivel del sistema operativo: en este caso la virtualización permite tener múltiples instancias aisladas y seguras del servidor, todas ejecutándose sobre el mismo servidor físico y donde el sistema operativo *guest* coincidirá con el sistema operativo base del servidor (se utiliza el mismo kernel). Plataformas dentro de este tipo de virtualización son FreeBSD jails (el pionero), OpenVZ, Linux-VServer, LXC, Virtuozzo.

La diferencia entre instalar dos sistemas operativos y virtualizar dos sistemas operativos es que en el primer caso todos los sistemas operativos que tengamos instalados funcionarán de la misma manera que si estuvieran instalados en distintos ordenadores, y necesitaremos un gestor de arranque que al encender el ordenador nos permita elegir qué sistema operativo queremos utilizar, pero solo podremos tener funcionando simultáneamente uno de ellos. En cambio, la virtualización permite ejecutar muchas máquinas virtuales con sus sistemas operativos y cambiar de sistema operativo como si se tratase de cualquier otro programa; sin embargo, se deben valorar muy bien las cuestiones relacionadas con las prestaciones, ya que si el HW subyacente no es el adecuado, podremos notar muchas diferencias en las prestaciones entre el sistema operativo instalado en base o el virtualizado.

Entre las principales ventajas de la virtualización podemos contemplar:

1) Consolidación de servidores y mejora de la eficiencia de la inversión en HW con reutilización de la infraestructura existente.

2) Rápido despliegue de nuevos servicios con balanceo dinámico de carga y reducción de los sobredimensionamientos de la infraestructura.

3) Incremento de *Uptime* (tiempo que el sistema está al 100 % en la prestación de servicios), incremento de la tolerancia a fallos (siempre y cuando exista redundancia física) y eliminación del tiempo de parada por mantenimiento del sistema físico (migración de las máquinas virtuales).

4) Mantenimiento a coste aceptables de entornos software obsoletos pero necesarios para el negocio.

5) Facilidad de diseño y test de nuevas infraestructuras y entornos de desarrollo con un bajo impacto en los sistemas de producción y rápida puesta en marcha.

6) Mejora de TCO (*Total Cost of Ownership*) y ROI (*Return on Investment*).

7) Menor consumo de energía que en servidores físicos equivalentes.

Como desventajas podemos mencionar:

1) Aumenta la probabilidad de fallos si no se considera redundancia/alta disponibilidad (si se consolidan 10 servidores físicos en uno potente equivalente, con servidores virtualizados, y dejan de funcionar todos los servidores en él, dejarán de prestar servicio).

2) Rendimiento inferior (posible) en función de la técnica de virtualización utilizada y recursos disponibles.

3) Proliferación de servicios y máquinas que incrementan los gastos de administración/gestión (básicamente por el efecto derivado de la 'facilidad de despliegue' se tiende a tener más de lo necesarios).

4) Infraestructura desaprovechada (posible) ya que es habitual comprar una infraestructura mayor que la necesaria en ese momento para el posible crecimiento futuro inmediato.

5) Pueden existir problemas de portabilidad, hardware específico no soportado, y compromiso a largo término con la infraestructura adquirida.

6) Tomas de decisiones en la selección del sistema anfitrión puede ser complicada o condicionante.

Como es posible observar, las desventajas se pueden resolver con una planificación y toma de decisiones adecuada, y esta tecnología es habitual en la prestación de servicios y totalmente imprescindible en entornos de *Cloud Computing* (que veremos en este capítulo también).

1.2. Beowulf

Beowulf [3, 1, 2, 4] es una arquitectura multiordenador que puede ser utilizada para aplicaciones paralelas/distribuidas. El sistema consiste básicamente en un servidor y uno o más clientes conectados (generalmente) a través de Ethernet y sin la utilización de ningún hardware específico. Para explotar esta capacidad de cómputo, es necesario que los programadores tengan un modelo de programación distribuido que, si bien es posible mediante UNIX (Sockets, RPC), puede implicar un esfuerzo considerable, ya que son modelos de programación a nivel de *systems calls* y lenguaje C, por ejemplo; pero este modo de trabajo puede ser considerado de bajo nivel. Un modelo más avanzado en

esta línea de trabajo son los **Posix Threads**, que permiten explotar sistemas de memoria compartida y *multicores* de forma simple y fácil. La capa de software (interfaz de programación de aplicaciones, API) aportada por sistemas tales como ***Parallel Virtual Machine*** (PVM) y ***Message Passing Interface*** (MPI) facilita notablemente la abstracción del sistema y permite programar aplicaciones paralelas/distribuidas de modo sencillo y simple. Una de las formas básicas de trabajo es la de maestro–trabajadores (*master-workers*), en que existe un servidor (maestro) que distribuye la tarea que realizarán los trabajadores. En grandes sistemas (por ejemplo, de 1.024 nodos) existe más de un maestro y nodos dedicados a tareas especiales, como por ejemplo entrada/salida o monitorización. Otra opción no menos interesante, sobre todo con el auge de procesadores *multicores*, es **OpenMP** que es una API para la programación multiproceso de memoria compartida. Esta capa de software permite añadir concurrencia a los programas sobre la base del modelo de ejecución *fork-join* y se compone de un conjunto de directivas de compilador, rutinas de biblioteca y variables de entorno, que influencian el comportamiento en tiempo de ejecución y proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas y arquitecturas de CPU que puedan ejecutar más de un hilo de ejecución simultáneo (hyperthreading) o que dispongan de más de un núcleo por procesador accediendo a la misma memoria compartida.

Existen dos conceptos que pueden dar lugar a dudas y que son Cluster Beowulf y COW (*Cluster of Workstations*). Una de las principales diferencias es que Beowulf "se ve" como una única máquina donde se accede a los nodos remotamente, ya que no disponen de terminal (ni de teclado), mientras que un COW es una agrupación de ordenadores que pueden ser utilizados tanto por los usuarios de la COW como por otros usuarios en forma interactiva, a través de su pantalla y teclado. Hay que considerar que Beowulf no es un software que transforma el código del usuario en distribuido ni afecta al núcleo del sistema operativo (como por ejemplo, Mosix). Simplemente, es una forma de agrupación (un clúster) de máquinas que ejecutan GNU/Linux y actúan como un super-ordenador. Obviamente, existe una gran cantidad de herramientas que permiten obtener una configuración más fácil, bibliotecas o modificaciones al núcleo para obtener mejores prestaciones, pero es posible construir un clúster Beowulf a partir de un GNU/Linux estándar y de software convencional. La construcción de un clúster Beowulf de dos nodos, por ejemplo, se puede llevar a cabo simplemente con las dos máquinas conectadas por Ethernet mediante un concentrador (*hub*), una distribución de GNU/ Linux estándar (Debian), el sistema de archivos compartido (NFS) y tener habilitados los servicios de red, como por ejemplo `ssh`. En estas condiciones, se puede argumentar que se dispone de un clúster simple de dos nodos. En cambio, en un COW no necesitamos tener conocimientos sobre la arquitectura subyacente (CPU, red) necesaria para hacer una aplicación distribuida en Beowulf pero sí que es necesario tener un sistema operativo (por ejemplo, Mosix) que permita este tipo de compartición de recursos y distribución de tareas (además de un API específica para programar la aplicación que es necesaria en los dos

sistemas). El primer Beowulf se construyó en 1994 y en 1998 comienzan a aparecer sistemas Beowulf/linux en las listas del Top500 (Avalon en la posición 314 con 68 cores y 19,3 Gflops, junio'98).

1.2.1. ¿Cómo configurar los nodos?

Primero se debe modificar el archivo `/etc/hosts` para que contenga la línea de `localhost` (con 127.0.0.1) y el resto de los nombres a las IP internas de los restantes nodos (este archivo deberá ser igual en todos los nodos). Por ejemplo:

```
    127.0.0.1 localhost
    192.168.0.254   lucix-server
Y añadir las IP de los nodos (y para todos los nodos), por ejemplo:
    192.168.0.1 lucix1
    192.168.0.2 lucix2
...
Se debe crear un usuario (nteum) en todos los nodos, crear un grupo y añadir este usuario al grupo:
    groupadd beowulf
    adduser nteum beowulf
    echo umask 007 >> /home/nteum/.bash_profile
```

Así, cualquier archivo creado por el usuario nteum o cualquiera dentro del grupo será modificable por el grupo beowulf. Se debe crear un servidor de NFS (y los demás nodos serán clientes de este NFS) y exportar el directorio */home* así todos los clientes verán el directorio *$HOME* de los usuarios. Para ello sobre el servidor se edita el */etc/exports* y se agrega una línea como `/home lucix*(rw,sync,no_root_squash)`. Sobre cada nodo cliente se monta agregando en el */etc/fstab* para que en el arranque el nodo monte el directorio como `192.168.0.254:/home /home nfs defaults 0 0`, también es aconsejable tener en el servidor un directorio */soft* (donde se instalará todo el software para que lo vean todos los nodos así nos evitamos de tener que instalar este en cada nodo), y agregarlos al *export* como `/soft lucix*(rw,async,no_root_squash)` y en cada nodo agregamos en el *fstab* `192.168.0.254:/soft /soft nfs defaults 0 0`. También deberemos hacer la configuración del */etc/resolv.conf* y el `iptables` (para hacer un NAT) sobre el servidor si este tiene acceso a Internet y queremos que los nodos también tengan acceso (es muy útil sobre todo a la hora de actualizar el sistema operativo).

A continuación verificamos que los servicios están funcionando (tened en cuenta que el comando `chkconfig` puede no estar instalado en todas las distribuciones) (en Debian hacer `apt-get install chkconfig`):

```
chkconfig --list sshd
chkconfig --list nfs
```

Que deben estar a "on" en el nivel de trabajo que estemos (generalmente el nivel 3 pero se puede averiguar con el comando `runlevel`) y sino se deberán hacer las modificaciones necesarias para que estos *daemons* se inicien en este nivel (por ejemplo, `chkconfig name_service on; service`

`name_service start`). Si bien estaremos en un red privada, para trabajar de forma segura es importante trabajar con `ssh` y nunca con `rsh` o `rlogin` por lo cual deberíamos generar las claves para interconectar en modo seguro las máquinas-usuario nteum sin `passwd`. Para eso modificamos (quitamos el comentario #), de `/etc/ssh/sshd_config` en las siguientes líneas:

```
RSAAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
```

Reiniciamos el servicio (`service sshd restart`) y nos conectamos con el usuario que deseamos y generamos las llaves:

```
ssh-keygen -t rsa
```

En el directorio *$HOME/.ssh* se habrán creado los siguientes archivos: `id_rsa` y `id_rsa.pub` en referencia a la llave privada y pública respectivamente. Podemos de forma manual copiar `id_rsa.pub` en un archivo con nombre `authorized_keys` en el mismo directorio (ya que al estar compartido por NFS será el mismo directorio que verán todas los nodos y verificamos los permisos: 600 para el directorio *.ssh*, *authorized_keys* e *id_rsa* y 644 para *id_rsa.pub*. Es conveniente también instalar NIS sobre el clúster así evitamos tener que definir cada usuario en los nodos y un servidor DHCP para definir todos los parámetros de red de cada nodo que que cada nodo solicite estos durante la etapa de *boot*, para ello consultad en el capítulo de servidores y el de red de la asignatura de Administración GNU/Linux donde se explica con detalle estas configuraciones.

A partir de aquí ya tenemos un clúster Beowulf para ejecutar aplicaciones con interfaz a MPI para aplicaciones distribuidas (también puede ser aplicaciones con interfaz a PVM, pero es recomendable MPI por cuestiones de eficiencia y prestaciones). Existen sobre diferentes distribuciones (Debian, FC incluidas) la aplicación `system-config-cluster`, que permite configurar un clúster en base a una herramienta gráfica o `clusterssh` o `dish`, que permiten gestionar y ejecutar comandos en un conjunto de máquinas de forma simultánea (comandos útiles cuando necesitamos hacer operaciones sobre todos los nodos del clúster, por ejemplo, apagarlos, reiniciarlos o hacer copias de un archivo a todos ellos).

Enlace de interés

Información Adicional:
https://wiki.debian.org
/HighPerformanceComputing

1.3. Beneficios del cómputo distribuido

¿Cuáles son los beneficios del cómputo en paralelo? Veremos esto con un ejemplo [3]. Consideremos un programa para sumar números (por ejemplo, 4 + 5 + 6 + ...) llamado `sumdis.c`:

```
#include <stdio.h>

int main (int argc, char** argv){
long inicial, final, resultado, tmp;
    if (argc < 2) {
   printf (" Uso: %s N° inicial N° final\n",argv[0]);
     return (4); }
    else {
   inicial = atoll(argv[1]);
   final = atoll(argv[2]);
   resultado = 0;}
for (tmp = inicial; tmp <= final; tmp++){resultado += tmp; };
    printf("%lu\n", resultado);
    return 0;
}
```

Lo compilamos con `gcc -o sumdis sumdis.c` y si miramos la ejecución de este programa con, por ejemplo,

```
 time ./sumdis 1 1000000
500000500000

    real 0m0.005s
    user 0m0.000s
    sys     0m0.004s
```

se podrá observar que el tiempo en una máquina Debian 7.5 sobre una máquina virtual (Virtualbox) con procesador i7 es 5 milésimas de segundo. Si, en cambio, hacemos desde 1 a 16 millones, el tiempo real sube hasta 0,050s, es decir, 10 veces más, lo cual, si se considera 1600 millones, el tiempo será del orden de unos 4 segundos.

La idea básica del cómputo distribuido es repartir el trabajo. Así, si disponemos de un clúster de 4 máquinas (lucix1–lucix4) con un servidor y donde el archivo ejecutable se comparte por NFS, sería interesante dividir la ejecución mediante ssh de modo que el primero sume de 1 a 400.000.000, el segundo, de 400.000.001 a 800.000.000, el tercero, de 800.000.001 a 1.200.000.000 y el cuarto, de 1.200.000.001 a 1.600.000.000. Los siguientes comandos muestran una posibilidad. Consideramos que el sistema tiene el directorio `/home` compartido por NFS y el usuario **adminp**, que tiene adecuadamente configurado las llaves para ejecutar el código sin *password* sobre los nodos, ejecutará:

```
    mkfifo out1     Crea una cola fifo en /home/adminp
    ./distr.sh & time cat out1 | awk '{total + = $1 } END {printf "%lf", total}'
```

Se ejecuta el comando `distr.sh`; se recolectan los resultados y se suman mientras se mide el tiempo de ejecución. El *shell script* `distr.sh` puede ser algo como:

```
ssh lucix1 /home/nteum/sumdis 1 400000000 > /home/nteum/out1 < /dev/null &
ssh lucix2 /home/nteum/sumdis 400000001 800000000 > /home/nteum/out1 < /dev/null &
ssh lucix3 /home/nteum/sumdis 800000001 1200000000 > /home/nteum/out1 < /dev/null &
ssh lucix4 /home/nteum/sumdis 1200000001 1600000000 > /home/nteum/out1 < /dev/null &
```

Podremos observar que el tiempo se reduce notablemente (aproximadamente en un valor cercano a 4) y no exactamente de forma lineal, pero muy próxima. Obviamente, este ejemplo es muy simple y solo válido para fines demostrativos. Los programadores utilizan bibliotecas que les permiten realizar el tiempo de ejecución, la creación y comunicación de procesos en un sistema distribuido (por ejemplo MPI u OpenMP).

1.3.1. ¿Cómo hay que programar para aprovechar la concurrencia?

Existen diversas maneras de expresar la concurrencia en un programa. Las tres más comunes son:

1) Utilizando hilos (o procesos) en el mismo procesador (multiprogramación con solapamiento del cómputo y la E/S).

2) Utilizando hilos (o procesos) en sistemas *multicore*.

3) Utilizando procesos en diferentes procesadores que se comunican por medio de mensajes (MPS, *Message Passing System*).

Estos métodos pueden ser implementados sobre diferentes configuraciones de hardware (memoria compartida o mensajes) y, si bien ambos métodos tienen sus ventajas y desventajas, los principales problemas de la memoria compartida son las limitaciones en la escalabilidad (ya que todos los *cores*/procesadores utilizan la misma memoria y el número de estos en el sistema está limitado por el ancho de banda de la memoria) y, en los los sistemas de paso de mensajes, la latencia y velocidad de los mensajes en la red. El programador deberá evaluar qué tipo de prestaciones necesita, las características de la aplicación subyacente y el problema que se desea solucionar. No obstante, con los avances de las tecnologías de *multicores* y de red, estos sistemas han crecido en popularidad (y en cantidad). Las API más comunes hoy en día son Posix Threads y OpenMP para memoria compartida y MPI (en sus versiones OpenMPI o Mpich) para paso de mensajes. Como hemos mencionado anteriormente, existe otra biblioteca muy difundida para pasos de mensajes, llamada PVM, pero que la versatilidad y prestaciones que se obtienen con MPI ha dejado relegada a aplicaciones pequeñas o para aprender a programar en sistemas distribuidos. Estas bibliotecas, además, no limitan la posibilidad de utilizar hilos (aunque a nivel local) y tener concurrencia entre procesamiento y entrada/salida.

Para realizar una aplicación paralela/distribuida, se puede partir de la versión serie o mirando la estructura física del problema y determinar qué partes pueden ser concurrentes (independientes). Las partes concurrentes serán candidatas a re-escribirse como código paralelo. Además, se debe considerar si es posible reemplazar las funciones algebraicas por sus versiones paralelizadas (por ejemplo, ScaLapack *Scalable Linear Algebra Package* (se puede probar en Debian

los diferentes programas de prueba que se encuentran en el paquete scalapack-mpi-test, por ejemplo y directamente, instalar las librerías libscalapack-mpi-dev). También es conveniente averiguar si hay alguna aplicación similar paralela que pueda orientarnos sobre el modo de construcción de la aplicación paralela*.

*http://www.mpich.org/

Paralelizar un programa no es una tarea fácil, ya que se debe tener en cuenta la **ley de Amdahl**, que afirma que el incremento de velocidad (*speedup*) está limitado por la fracción de código (f), que puede ser paralelizado de la siguiente manera:

$$\text{speedup} = \frac{1}{1-f}$$

Esta ley implica que con una aplicación secuencial $f = 0$ y el speedup = 1, mientras que con todo el código paralelo $f = 1$ y el speedup se hace infinito. Si consideramos valores posibles, un 90 % ($f = 0{,}9$) del código paralelo significa un speedup igual a 10, pero con $f = 0{,}99$ el speedup es igual a 100. Esta limitación se puede evitar con algoritmos escalables y diferentes modelos de programación de aplicación (paradigmas):

1) Maestro-trabajador: el maestro inicia a todos los trabajadores y coordina su trabajo y el de entrada/salida.

2) *Single Process Multiple Data* (SPMD): mismo programa que se ejecuta con diferentes conjuntos de datos.

3) Funcional: varios programas que realizan una función diferente en la aplicación.

En resumen, podemos concluir:

1) Proliferación de máquinas multitarea (multiusuario) conectadas por red con servicios distribuidos (NFS y NIS).

2) Son sistemas heterogéneos con sistemas operativos de tipo NOS (*Networked Operating System*), que ofrecen una serie de servicios distribuidos y remotos.

3) La programación de aplicaciones distribuidas se puede efectuar a diferentes niveles:

a) Utilizando un modelo cliente-servidor y programando a bajo nivel (*sockets*) o utilizando memoria compartida a bajo nivel (Posix Threads).

b) El mismo modelo, pero con API de "alto" nivel (OpenMP, MPI).

c) Utilizando otros modelos de programación como, por ejemplo, programación orientada a objetos distribuidos (RMI, CORBA, Agents, etc.).

1.4. Memoria compartida. Modelos de hilos (*threading*)

Normalmente, en una arquitectura cliente-servidor, los clientes solicitan a los servidores determinados servicios y esperan que estos les contesten con la mayor eficacia posible. Para sistema distribuidos con servidores con una carga muy alta (por ejemplo, sistemas de archivos de red, bases de datos centralizadas o distribuidas), el diseño del servidor se convierte en una cuestión crítica para determinar el rendimiento general del sistema distribuido. Un aspecto crucial en este sentido es encontrar la manera óptima de manejar la E/S, teniendo en cuenta el tipo de servicio que ofrece, el tiempo de respuesta esperado y la carga de clientes. No existe un diseño predeterminado para cada servicio, y escoger el correcto dependerá de los objetivos y restricciones del servicio y de las necesidades de los clientes.

Las preguntas que debemos contestar antes de elegir un determinado diseño son: ¿Cuánto tiempo se tarda en un proceso de solicitud del cliente? ¿Cuántas de esas solicitudes es probable que lleguen durante ese tiempo? ¿Cuánto tiempo puede espera el cliente? ¿Cuánto afecta esta carga del servidor a las prestaciones del sistema distribuido? Además, con el avance de la tecnología de procesadores nos encontramos con que disponemos de sistemas *multicore* (múltiples núcleos de ejecución) que pueden ejecutar secciones de código independientes. Si se diseñan los programas en forma de múltiples secuencias de ejecución y el sistema operativo lo soporta (y GNU/Linux es uno de ellos), la ejecución de los programas se reducirá notablemente y se incrementarán en forma (casi) lineal las prestaciones en función de los *cores* de la arquitectura.

1.4.1. Multihilos (*multithreading*)

Las últimas tecnologías en programación para este tipo aplicaciones (y así lo demuestra la experiencia) es que los diseños más adecuados son aquellos que utilizan modelos de multi-hilos (*multithreading models*), en los cuales el servidor tiene una organización interna de procesos paralelos o hilos cooperarantes y concurrentes.

Un hilo (*thread*) es una secuencia de ejecución (hilo de ejecución) de un programa, es decir, diferentes partes o rutinas de un programa que se ejecutan concurrentemente en un único procesador y accederán a los datos compartidos al mismo tiempo.

¿Qué ventajas aporta esto respecto a un programa secuencial? Consideremos que un programa tiene tres rutinas A, B y C. En un programa secuencial, la rutina C no se ejecutará hasta que se hayan ejecutado A y B. Si, en cambio, A, B y C son hilos, las tres rutinas se ejecutarán concurrentemente y, si en ellas hay E/S, tendremos concurrencia de ejecución con E/S del mismo programa (proceso), cosa que mejorará notablemente las prestaciones de dicho programa. Generalmente, los hilos están contenidos dentro de un proceso y diferentes hilos de un mismo proceso pueden compartir algunos recursos, mientras que diferentes procesos no. La ejecución de múltiples hilos en paralelo necesita el soporte del sistema operativo y en los procesadores modernos existen optimizaciones del procesador para soportar modelos multihilo (*multithreading*) además de las arquitectura multicores donde existe múltiples núcleo y donde cada uno de ellos puede ejecutar un thread.

Generalmente, existen cuatro modelos de diseño por hilos (en orden de complejidad creciente):

1) Un hilo y un cliente: en este caso el servidor entra en un bucle sin fin escuchando por un puerto y ante la petición de un cliente se ejecutan los servicios en el mismo hilo. Otros clientes deberán esperar a que termine el primero. Es fácil de implementar pero solo atiende a un cliente a la vez.

2) Un hilo y varios clientes con selección: en este caso el servidor utiliza un solo hilo, pero puede aceptar múltiples clientes y multiplexar el tiempo de CPU entre ellos. Se necesita una gestión más compleja de los puntos de comunicación (*sockets*), pero permite crear servicios más eficientes, aunque presenta problemas cuando los servicios necesitan una alta carga de CPU.

3) Un hilo por cliente: es, probablemente, el modelo más popular. El servidor espera por peticiones y crea un hilo de servicio para atender a cada nueva petición de los clientes. Esto genera simplicidad en el servicio y una alta disponibilidad, pero el sistema no escala con el número de clientes y puede saturar el sistema muy rápidamente, ya que el tiempo de CPU dedicado ante una gran carga de clientes se reduce notablemente y la gestión del sistema operativo puede ser muy compleja.

4) Servidor con hilos en granja (*worker threads*): este método es más complejo pero mejora la escalabilidad de los anteriores. Existe un número fijo de hilos trabajadores (*workers*) a los cuales el hilo principal distribuye el trabajo de los clientes. El problema de este método es la elección del número de trabajadores: con un número elevado, caerán las prestaciones del sistema por saturación; con un número demasiado bajo, el servicio será deficiente (los clientes deberán esperar). Normalmente, será necesario sintonizar la aplicación para trabajar con un determinado entorno distribuido.

Existen diferentes formas de expresar a nivel de programación con hilos: paralelismo a nivel de tareas o paralelismo a través de los datos. Elegir el modelo

adecuado minimiza el tiempo necesario para modificar, depurar y sintonizar el código. La solución a esta disyuntiva es describir la aplicación en términos de dos modelos basados en un trabajo en concreto:

- Tareas paralelas con hilos independientes que pueden atender tareas independientes de la aplicación. Estas tareas independientes serán encapsuladas en hilos que se ejecutarán asincrónicamente y se deberán utilizar bibliotecas como Posix Threads (Linux/Unix) o Win32 Thread API (Windows), que han sido diseñadas para soportar concurrencia a nivel de tarea.

- Modelo de datos paralelos para calcular lazos intensivos; es decir, la misma operación debe repetirse un número elevado de veces (por ejemplo comparar una palabra frente a las palabras de un diccionario). Para este caso es posible encargar la tarea al compilador de la aplicación o, si no es posible, que el programador describa el paralelismo utilizando el entorno OpenMP, que es una API que permite escribir aplicaciones eficientes bajo este tipo de modelos.

Una aplicación de información personal (*Personal Information Manager*) es un buen ejemplo de una aplicación que contiene concurrencia a nivel de tareas (por ejemplo, acceso a la base de datos, libreta de direcciones, calendario, etc.). Esto podría ser en pseudocódigo:

```
Function addressBook;
Function inBox;
Function calendar;
Program PIM     {
    CreateThread (addressBook);
    CreateThread (inBox);
    CreateThread (calendar);   }
```

Podemos observar que existen tres ejecuciones concurrentes sin relación entre ellas. Otro ejemplo de operaciones con paralelismo de datos podría ser un corrector de ortografía, que en pseudocódigo sería: `Function SpellCheck {loop (word = 1, words_in_file) compareToDictionary (word);}`

Se debe tener en cuenta que ambos modelos (hilos paralelos y datos paralelos) pueden existir en una misma aplicación. A continuación se mostrará el código de un productor de datos y un consumidor de datos basado en Posix Threads. Para compilar sobre Linux, por ejemplo, se debe utilizar `gcc -o pc pc.c -lptread`.

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define QUEUESIZE 10
#define LOOP 20

void *producer (void *args);
```

```
void *consumer (void *args);
typedef struct { /* Estructura del buffer compartido y descriptores de threads */
int buf[QUEUESIZE]; long head, tail; int full, empty;
pthread_mutex_t *mut; pthread_cond_t *notFull, *notEmpty;
    } queue;
queue *queueInit (void);   /* Prototipo de función: inicialización del buffer */
void queueDelete (queue *q); /* Prototipo de función: Borrado del buffer*/
void queueAdd (queue *q, int in); /* Prototipo de función: insertar elemento en el buffer */
void queueDel (queue *q, int *out); /* Prototipo de función: quitar elemento del buffer */

int main () {
queue *fifo; pthread_t pro, con; fifo = queueInit ();
if (fifo ==  NULL) { fprintf (stderr, " Error al crear buffer.\n");  exit (1);   }
pthread_create (&pro, NULL, producer, fifo);   /* Creación del thread productor */
pthread_create (&con, NULL, consumer, fifo); /* Creación del thread consumidor*/
pthread_join (pro, NULL);   /* main () espera hasta que terminen ambos threads */
pthread_join (con, NULL);
queueDelete (fifo);       /* Eliminación del buffer compartido */
return 0; }               /* Fin */

void *producer (void *q) { /*Función del productor */
queue *fifo; int i;
fifo = (queue *)q;
for (i = 0; i < LOOP; i++) {    /* Inserto en el buffer elementos=LOOP*/
pthread_mutex_lock (fifo->mut);   /* Semáforo para entrar a insertar */
while (fifo->full) {
printf ("Productor: queue FULL.\n");
pthread_cond_wait (fifo->notFull, fifo->mut);  }
/* Bloqueo del productor si el buffer está lleno, liberando el semáforo mut
    para que pueda entrar el consumidor. Continuará cuando el consumidor ejecute
    pthread_cond_signal (fifo->notFull);*/
         queueAdd (fifo, i); /* Inserto elemento en el buffer */
                   pthread_mutex_unlock (fifo->mut); /* Libero el semáforo */
         pthread_cond_signal (fifo->notEmpty);/*Desbloqueo consumidor si está bloqueado*/
         usleep (100000);  /* Duermo 100 mseg para permitir que el consumidor se active */
}
return (NULL); }

void *consumer (void *q) { /*Función del consumidor */
queue *fifo; int i, d;
fifo = (queue *)q;
for (i = 0; i < LOOP; i++) { /* Quito del buffer elementos=LOOP*/
pthread_mutex_lock (fifo->mut); /* Semáforo para entrar a quitar */
while (fifo->empty) {
printf (" Consumidor: queue EMPTY.\n");
pthread_cond_wait (fifo->notEmpty, fifo->mut); }
    /* Bloqueo del consumidor si el buffer está vacío, liberando el semáforo mut
    para que pueda entrar el productor. Continuará cuando el consumidor ejecute
    pthread_cond_signal (fifo->notEmpty);*/
queueDel (fifo, &d); /* Quito elemento del buffer */
pthread_mutex_unlock (fifo->mut); /* Libero el semáforo */
pthread_cond_signal (fifo->notFull); /*Desbloqueo productor si está bloqueado*/
printf (" Consumidor: Recibido %d.\n", d);
usleep(200000);/* Duermo 200 mseg para permitir que el productor se active */
}
return (NULL); }

queue *queueInit (void) {
queue *q;
q = (queue *)malloc (sizeof (queue)); /* Creación del buffer */
if (q == NULL) return (NULL);
q->empty = 1;  q->full = 0;  q->head = 0;  q->tail = 0;
q->mut = (pthread_mutex_t *) malloc (sizeof (pthread_mutex_t));
pthread_mutex_init (q->mut, NULL); /* Creación del semáforo */
q->notFull = (pthread_cond_t *) malloc (sizeof (pthread_cond_t));
pthread_cond_init (q->notFull, NULL); /* Creación de la variable condicional notFull*/
q->notEmpty = (pthread_cond_t *) malloc (sizeof (pthread_cond_t));
pthread_cond_init (q->notEmpty, NULL);   /* Creación de la variable condicional notEmpty*/
return (q); }

void queueDelete (queue *q) {
pthread_mutex_destroy (q->mut);  free (q->mut);
```

```
pthread_cond_destroy (q->notFull); free (q->notFull);
pthread_cond_destroy (q->notEmpty); free (q->notEmpty);
free (q); }

void queueAdd (queue *q, int in) {
q->buf[q->tail] = in; q->tail++;
if (q->tail == QUEUESIZE) q->tail = 0;
if (q->tail == q->head) q->full = 1;
q->empty = 0;
return; }

void queueDel (queue *q, int *out){
*out = q->buf[q->head]; q->head++;
if (q->head == QUEUESIZE) q->head = 0;
if (q->head == q->tail) q->empty = 1;
q->full = 0;
return; }
```

1.5. OpenMP

El OpenMP (*Open-Multi Processing*) es una interfaz de programación de aplicaciones (API) con soporte multiplataforma para la programación en C/C++ y Fortran de procesos con uso de memoria compartida sobre plataformas Linux/Unix (y también Windows). Esta infraestructura se compone de un conjunto de directivas del compilador, rutinas de la biblioteca y variables de entorno que permiten aprovechar recursos compartidos en memoria y en tiempo de ejecución. Definido conjuntamente por un grupo de los principales fabricantes de hardware y software, OpenMP permite utilizar un modelo escalable y portátil de programación, que proporciona a los usuarios un interfaz simple y flexible para el desarrollo, sobre plataformas paralelas, de aplicaciones de escritorio hasta aplicaciones de altas prestaciones sobre superordenadores. Una aplicación construida con el modelo híbrido de la programación paralela puede ejecutarse en un ordenador utilizando tanto OpenMP como Message Passing Interface (MPI) [5].

OpenMP es una implementación multihilo, mediante la cual un hilo maestro divide la tareas sobre un conjunto de hilos trabajadores. Estos hilos se ejecutan simultáneamente y el entorno de ejecución realiza la asignación de estos a los diferentes procesadores de la arquitectura. La sección del código que está diseñada para funcionar en paralelo está marcada con una directiva de preprocesamiento que creará los hilos antes que la sección se ejecute. Cada hilo tendrá un identificador (`id`) que se obtendrá a partir de una función (`omp_get_thread_num()` en C/C++) y, después de la ejecución paralela, los hilos se unirán de nuevo en su ejecución sobre el hilo maestro, que continuará con la ejecución del programa. Por defecto, cada hilo ejecutará una sección paralela de código independiente pero se pueden declarar secciones de "trabajo compartido" para dividir una tarea entre los hilos, de manera que cada hilo ejecute parte del código asignado. De esta forma, es posible tener en un programa OpenMP paralelismo de datos y paralelismo de tareas conviviendo conjuntamente.

Los principales elementos de OpenMP son las sentencias para la creación de hilos, la distribución de carga de trabajo, la gestión de datos de entorno, la sincronización de hilos y las rutinas a nivel de usuario. OpenMP utiliza en C/C++ las directivas de preprocesamiento conocidas como *pragma* (`#pragma omp <resto del pragma>`) para diferentes construcciones. Así por ejemplo, `omp parallel` se utiliza para crear hilos adicionales para ejecutar el trabajo indicado en la sentencia paralela donde el proceso original es el hilo maestro (`id=0`). El conocido programa que imprime "Hello, world" utilizando OpenMP y multihilos es*:

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){
   #pragma omp parallel
   printf("Hello, world.\n");
   return 0;}
```

***Compilad con** `gcc -fopenmp -o hello hello.c` **(en algunas distribuciones -debian está instalada por defecto- se debe tener instalada la biblioteca GCC OpenMP (GOMP)** `apt-get install libgomp1`

Donde ejecutará un thread por cada core disponible en la arquitectura. Para especificar *work-sharing constructs* se utiliza:

- **omp for o omp do**: reparte las iteraciones de un lazo en múltiples hilos.
- **sections**: asigna bloques de código independientes consecutivos a diferentes hilos.
- **single**: especifica que el bloque de código será ejecutado por un solo hilo con una sincronización (*barrier*) al final del mismo.
- **master**: similar a `single`, pero el código del bloque será ejecutado por el hilo maestro y no hay *barrier* implicado al final.

Por ejemplo, para inicializar los valores de un *array* en paralelo utilizando hilos para hacer una porción del trabajo (compilad, por ejemplo, con `gcc -fopenmp -o init2 init2.c`):

```
#include <stdio.h>
#include <omp.h>
#define N 1000000
int main(int argc, char *argv[]) {
   float a[N]; long i;
    #pragma omp parallel for
   for (i=0;i<N;i++)  a[i]= 2*i;
return 0;
}
```

Si ejecutamos con el *pragma* y después lo comentamos y calculamos el tiempo de ejecución (`time ./init2`), vemos que el tiempo de ejecución pasa de 0.003 s a 0.007 s, lo que muestra la utilización del *dualcore* del procesador.

Ya que OpenMP es un modelo de memoria compartida, muchas variables en el código son visibles para todos los hilos por defecto. Pero a veces es necesario tener variables privadas y pasar valores entre bloques secuenciales del código y bloques paralelos, por lo cual es necesario definir atributos a los datos (*data clauses*) que permitan diferentes situaciones:

- **shared**: los datos en la región paralela son compartidos y accesibles por todos los hilos simultáneamente.

- **private**: los datos en la región paralela son privados para cada hilo, y cada uno tiene una copia de ellos sobre una variable temporal.

- **default**: permite al programador definir cómo serán los datos dentro de la región paralela (`shared`, `private` o `none`).

Otro aspecto interesante de OpenMP son las directivas de sincronización:

- **critical section**: el código enmarcado será ejecutado por hilos, pero solo uno por vez (no habrá ejecución simultánea) y se mantiene la exclusión mutua.
- **atomic**: similar a `critical section`, pero avisa al compilador para que use instrucciones de hardware especiales de sincronización y así obtener mejores prestaciones.
- **ordered**: el bloque es ejecutado en el orden como si de un programa secuencial se tratara.
- **barrier**: cada hilo espera que los restantes hayan acabado su ejecución (implica sincronización de todos los hilos al final del código).
- **nowait**: especifica que los hilos que terminen el trabajo asignado pueden continuar.

Además, OpenMP provee de sentencias para la planificación (*scheduling*) del tipo `schedule(type, chunk)` (donde el tipo puede ser `static`, `dynamic` o `guided`) o proporciona control sobre la sentencias `if`, lo que permitirá definir si se paraleliza o no en función de si la expresión es verdadera o no. OpenMP también proporciona un conjunto de funciones de biblioteca, como por ejemplo:

- **omp_set_num_threads**: define el número de hilos a usar en la siguiente región paralela.
- **omp_get_num_threads**: obtiene el número de hilos que se están usando en una región paralela.
- **omp_get_max_threads**: obtiene la máxima cantidad posible de hilos.
- **omp_get_thread_num**: devuelve el número del hilo.
- **omp_get_num_procs**: devuelve el máximo número de procesadores que se pueden asignar al programa.
- **omp_in_parallel**: devuelve un valor distinto de cero si se ejecuta dentro de una región paralela.

Veremos a continuación algunos ejemplos simples (compilad con la instrucción `gcc -fopenmp -o out_file input_file.c`):

```
/* Programa simple multihreading con OpenMP */
#include <omp.h>
int main()  {
     int iam =0, np = 1;

     #pragma omp parallel private(iam, np)
     #if defined (_OPENMP)
         np = omp_get_num_threads();
         iam = omp_get_thread_num();
     #endif

     printf("Hello from thread %d out of %d \n", iam, np);
     }
 }

 /* Programa simple con Threads anidados con OpenMP */
 #include <omp.h>
 #include <stdio.h>
     main(){
         int x=0,nt,tid,ris;

         omp_set_nested(2);
         ris=omp_get_nested();
         if (ris) printf("Paralelismo anidado activo %d\n", ris);
         omp_set_num_threads(25);
         #pragma omp parallel private (nt,tid) {
             tid = omp_get_thread_num();
             printf("Thread %d\n",tid);
             nt = omp_get_num_threads();
             if (omp_get_thread_num()==1)
             printf("Número de Threads: %d\n",nt);
             }
}

 /* Programa simple de integración con OpenMP */
 #include <omp.h>
 #include <stdio.h>
 #define N 100
     main() {
    double local, pi=0.0, w; long i;
    w = 1.0 / N;
    #pragma omp parallel private(i, local)
        {
        #pragma omp single
        pi = 0.0;
        #pragma omp for reduction(+: pi)
        for (i = 0; i < N; i++) {
          local = (i + 0.5)*w;
          pi = pi + 4.0/(1.0 + local*local);
          printf ("Pi: %f\n",pi);
            }
        }
     }

 /* Programa simple de reducción con OpenMP */
 #include <omp.h>
 #include <stdio.h>
 #define NUM_THREADS 2
     main () {
    int i; double ZZ, res=0.0;
         omp_set_num_threads(NUM_THREADS);
     #pragma omp parallel for reduction(+:res) private(ZZ)
        for (i=0; i< 1000; i++) {
       ZZ = i*i;
       res = res + ZZ;
       printf("ZZ: %f, res: %f\n", ZZ, res);}
               }
         }
```

Nota

Se recomienda ver también los ejemplos que se pueden encontrar en la referencia [6].

1.6. MPI, *Message Passing Interface*

La definición de la API de MPI [9, 10] ha sido el trabajo resultante del MPI Forum (MPIF), que es un consorcio de más de 40 organizaciones. MPI tiene influencias de diferentes arquitecturas, lenguajes y trabajos en el mundo del paralelismo, como por ejemplo: WRC (Ibm), Intel NX/2, Express, nCUBE, Vertex, p4, Parmac y contribuciones de ZipCode, Chimp, PVM, Chamaleon, PICL.

El principal objetivo de MPIF fue diseñar una API, sin relación particular con ningún compilador ni biblioteca, y que permitiera la comunicación eficiente (*memory-to-memory copy*), cómputo y comunicación concurrente y descarga de comunicación, siempre y cuando exista un coprocesador de comunicaciones. Además, se pedía que soportara el desarrollo en ambientes heterogéneos, con interfaz C y F77 (incluyendo C++, F90), donde la comunicación fuera fiable y los fallos, resueltos por el sistema. La API también debía tener interfaz para diferentes entornos, disponer una implementación adaptable a diferentes plataformas con cambios insignificantes y que no interfiera con el sistema operativo (*thread-safety*). Esta API fue diseñada especialmente para programadores que utilizaran el *Message Passing Paradigm* (MPP) en C y F77, para aprovechar su característica más relevante: la portabilidad. El MPP se puede ejecutar sobre máquinas multiprocesador, redes de estaciones de trabajo e incluso sobre máquinas de memoria compartida. La primera versión del estándar fue MPI-1 (que si bien hay muchos desarrollos sobre esta versión, se considera en EOL), la versión MPI-2 incorporó un conjunto de mejoras, como creación de procesos dinámicos, *one-sided communication,* entrada/salida paralela entre otras, y finalmente la última version, MPI-3 (considerada como una revisión mayor), incluye *nonblocking collective operations, one-sided operations* y soporte para Fortran 2008.[7]

Muchos aspectos han sido diseñados para aprovechar las ventajas del hardware de comunicaciones sobre SPC (*scalable parallel computers*) y el estándar ha sido aceptado en forma mayoritaria por los fabricantes de hardware en paralelo y distribuido (SGI, SUN, Cray, HPConvex, IBM, etc.). Existen versiones libres (por ejemplo, Mpich, LAM/MPI y openMPI) que son totalmente compatibles con las implementaciones comerciales realizadas por los fabricantes de hardware e incluyen comunicaciones punto a punto, operaciones colectivas y grupos de procesos, contexto de comunicaciones y topología, soporte para F77 y C y un entorno de control, administración y *profiling*. [11, 12, 10]

Pero existen también algunos puntos que pueden presentar algunos problemas en determinadas arquitecturas, como son la memoria compartida, la ejecución remota, las herramientas de construcción de programas, la depuración, el control de hilos, la administración de tareas y las funciones de entrada/salida concurrentes (la mayor parte de estos problemas de falta de herramientas están resueltos a partir de la versión 2 de la API -MPI2-). Una de los

problemas de MPI1, al no tener creación dinámica de procesos, es que solo soporta modelos de programación MIMD (*Multiple Instruction Multiple Data*) y comunicándose vía llamadas MPI. A partir de MPI-2 y con la ventajas de la creación dinámica de procesos, ya se pueden implementar diferentes paradigmas de programación como *master-worker/farmer-tasks*, *divide & conquer*, paralelismo especulativo, etc. (o al menos sin tanta complejidad y mayor eficiencia en la utilización de los recursos).

Para la instalación de MPI se recomienda utilizar la distribución (en algunos casos la compilación puede ser compleja debido a las dependencias de otros paquetes que puede necesitar. Debian incluye la versión OpenMPI (sobre Debian 7.5 es version 2, pero se pueden bajar los fuentes y compilarlos) y Mpich2 (Mpich3 disponible en http://www.mpich.org/downloads/). La mejor elección será OpenMPI, ya que combina las tecnologías y los recursos de varios otros proyectos (FT-MPI, LA-MPI, LAM/MPI y PACX-MPI) y soporta totalmente el estándar MPI-2 (y desde la versión 1.75 soporta la versión MPI3). Entre otras características de OpenMPI tenemos: es conforme a MPI-2/3, *thread safety & concurrency*, creación dinámica de procesos, alto rendimiento y gestión de trabajos tolerantes a fallos, instrumentación en tiempo de ejecución, *job schedulers*, etc. Para ello se deben instalar los paquetes `openmpi-dev`, `openmpi-bin`, `openmpi-common` y `openmpi-doc`. Además, Debian 7.5 incluye otra implementación de MPI llamada LAM (paquetes `lam*`). Se debe considerar que si bien las implementaciones son equivalentes desde el punto de vista de MPI, tienen diferencias en cuanto a la gestión y procedimientos de compilación/ejecución/gestión.

1.6.1. Configuración de un conjunto de máquinas para hacer un clúster adaptado a OpenMPI

Para la configuración de un conjunto de máquinas para hacer un clúster adaptado a OpenMPI [14], se han de seguir los siguientes pasos:

1) Hay que tener las máquinas "visibles" (por ejemplo a través de un `ping`) a través de TCP/IP (IP pública/privada).

2) Es recomendable que todas las máquinas tengan la misma arquitectura de procesador, así es más fácil distribuir el código, con versiones similares de Linux (a ser posible con el mismo tipo de distribución).

3) Se recomienda tener NIS o si no se debe generar un mismo usuario (por ejemplo, mpiuser) en todas las máquinas y el mismo directorio `$HOME` montado por NFS.

4) Nosotros llamaremos a las máquinas como `slave1`, `slave2`, etc., (ya que luego resultará más fácil hacer las configuraciones) pero se puede llamar a las máquinas como cada uno prefiera.

5) Una de las máquinas será el maestro y las restantes, *slaveX*.

6) Se debe instalar en todos los nodos (supongamos que tenemos Debian): `openmpi-bin`, `openmpi-common`, `openmpi-dev`. Hay que verificar que en todas las distribuciones se trabaja con la misma versión de OpenMPI.

7) En Debian los ejecutables están en `/usr/bin` pero si están en un *path* diferente, deberá agregarse a mpiuser y también verificar que `LD_LIBRARY_PATH` apunta a `/usr/lib`.

8) En cada nodo esclavo debe instalarse el SSH server (instalad el paquete `openssh-server`) y sobre el maestro, el cliente (paquete `openssh-client`).

9) Se deben crear las claves públicas y privadas haciendo `ssh-keygen -t dsa` y copiar a cada nodo con `ssh-copy-id` para este usuario (solo se debe hacer en un nodo, ya que, como tendremos el directorio $HOME compartido por NFS para todos los nodos, con una copia basta).

10) Si no se comparte el directorio hay que asegurar que cada esclavo conoce que el usuario mpiuser se puede conectar sin passwd, por ejemplo haciendo: `ssh slave1`.

11) Se debe configurar la lista de las máquinas sobre las cuales se ejecutará el programa, por ejemplo `/home/mpiuser/.mpi_hostfile` y con el siguiente contenido:

```
# The Hostfile for Open MPI
# The master node, slots=2 is used because it is a dual-processor machine.
    localhost slots=2
# The following slave nodes are single processor machines:
    slave1
    slave2
    slave3
```

12) OpenMPI permite utilizar diferentes lenguajes, pero aquí utilizaremos C. Para ello hay que ejecutar sobre el maestro `mpicc testprogram.c`. Si se desea ver que incorpora mpicc, se puede hacer `mpicc -showme`.

13) Para ejecutar en local podríamos hacer `mpirun -np 2 ./myprogram` y para ejecutar sobre los nodos remotos (por ejemplo 5 procesos) `mpirun -np 2 -hostfile ./mpi_hostfile ./myprogram`.

Es importante notar que `np` es el número de procesos o procesadores en que se ejecutará el programa y se puede poner el número que se desee, ya que OpenMPI intentará distribuir los procesos de forma equilibrada entre todas las máquinas. Si hay más procesos que procesadores, OpenMPI/Mpich utilizará las características de intercambio de tareas de GNU/Linux para simular la ejecución paralela. A continuación se verán dos ejemplos: Srtest es un programa simple para establecer comunicaciones entre procesos punto a punto, y cpi calcula el valor del número π de forma distribuida (por integración).

```
/* Srtest Progam */
#include "mpi.h"
#include <stdio.h>
```

```
#include <string.h>
#define BUFLEN 512

int main(int argc, char *argv[]){
    int myid, numprocs, next, namelen;
    char buffer[BUFLEN], processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Status status;
    MPI_Init(&argc,&argv);  /* Debe ponerse antes de otras llamadas MPI, siempre */
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);    /*Integra el proceso en un grupo de comunicaciones*/
    MPI_Get_processor_name(processor_name,&namelen); /*Obtiene el nombre del procesador*/

    fprintf(stderr,"Proceso %d sobre %s\n", myid, processor_name);
    fprintf(stderr,"Proceso %d de %d\n", myid, numprocs);
    strcpy(buffer,"hello there");
    if (myid == numprocs-1) next = 0;
    else next = myid+1;

    if (myid == 0) { /*Si es el inicial, envía string de buffer*/
        printf("%d sending '%s' \n",myid,buffer);fflush(stdout);
    MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
        /*Blocking Send, 1:buffer, 2:size, 3:tipo, 4:destino, 5:tag, 6:contexto*/
    printf("%d receiving \n",myid);fflush(stdout);
    MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
    printf("%d received '%s' \n",myid,buffer);fflush(stdout);
       /* mpdprintf(001,"%d receiving \n",myid); */
        }
    else {
       printf("%d receiving  \n",myid);fflush(stdout);
   MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
       /* Blocking Recv, 1:buffer, 2:size, 3:tipo, 4:fuente, 5:tag, 6:contexto, 7:status*/
   printf("%d received '%s' \n",myid,buffer);fflush(stdout);
   /* mpdprintf(001,"%d receiving \n",myid); */
   MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
   printf("%d sent '%s' \n",myid,buffer);fflush(stdout);
       }
    MPI_Barrier(MPI_COMM_WORLD); /*Sincroniza todos los procesos*/
    MPI_Finalize(); /*Libera los recursos y termina*/
    return (0);
    }



/* CPI Program */
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double );
double f( double a) { return (4.0 / (1.0 + a*a)); }
int main( int argc, char *argv[] ) {
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264338327950288419716939937510;
    double mypi, pi, h, sum, x; double startwtime = 0.0, endwtime;
    int namelen; char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs); /*Indica el número de procesos en el grupo*/
    MPI_Comm_rank(MPI_COMM_WORLD,&myid); /*Id del proceso*/
    MPI_Get_processor_name(processor_name,&namelen); /*Nombre del proceso*/
    fprintf(stderr, "Proceso %d sobre %s\n", myid, processor_name);
    n = 0;
    while (!done) {
        if (myid ==0) { /*Si es el primero...*/
            if (n ==0) n = 100; else n = 0;
            startwtime = MPI_Wtime();} /* Time Clock */
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); /*Broadcast al resto*/
            /*Envía desde el 4 arg. a todos los procesos del grupo Los restantes que no son 0
            copiarán el buffer desde 4 o arg -proceso 0-*/
            /*1:buffer, 2:size, 3:tipo, 5:grupo */
        if (n == 0) done = 1;
        else {h = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs) {
```

```
                x = h * ((double)i - 0.5); sum += f(x); }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,MPI_COMM_WORLD);
                /* Combina los elementos del Send Buffer de cada proceso del grupo usando la
                operación MPI_SUM y retorna el resultado en el Recv Buffer. Debe ser llamada
                por todos los procesos del grupo usando los mismos argumentos*/
                /*1:sendbuffer, 2:recvbuffer, 3:size, 4:tipo, 5:oper, 6:root, 7:contexto*/
            if (myid == 0){ /*solo el P0 imprime el resultado*/
                printf("Pi es aproximadamente %.16f, el error es %.16f\n", pi,fabs(pi - PI25DT));
                endwtime = MPI_Wtime();
                printf("Tiempo de ejecución = %f\n", endwtime-startwtime); }
            }
        }
        MPI_Finalize(); /*Libera recursos y termina*/
        return 0;
    }
```

Para visualizar la ejecución de un código paralelo/distribuido en MPI existe una aplicación llamada XMPI (en Debian xmpi) que permite 'ver', durante la ejecución, el estado de la ejecución de las aplicaciones sobre MPI, pero está vinculada al paquete LAM/MPI. Para visualizar y analizar código sobre OpenMPI, se debería (y es recomendable) bajar y compilar el código de MPE (http://www.mcs.anl.gov/research/projects/perfvis/software/MPE/) o TAU (http://www.cs.uoregon.edu/research/tau/home.php) que son herramientas de *profiling* muy potentes y que no requieren gran trabajo ni dedicación para ponerlas en marcha.

1.7. Rocks Cluster

Rocks Cluster es una distribución de Linux para clústers de computadores de alto rendimiento. Las versiones actuales de Rocks Cluster están basadas en CentOS (CentOS 6.5 a julio 2014) y, como instalador, Anaconda con ciertas modificaciones, que simplifica la instalación 'en masa' en muchos ordenadores. Rocks Cluster incluye muchas herramientas (tales como MPI) que no forman parte de CentOS pero son los componentes que transforman un grupo de ordenadores en un clúster. Las instalaciones pueden personalizarse con paquetes de software adicionales llamados *rolls*. Los *rolls* extienden el sistema integrando automáticamente los mecanismos de gestión y empaquetamiento usados por el software básico, y simplifican ampliamente la instalación y configuración de un gran número de computadores. Se han creado una gran cantidad de *rolls*, como por ejemplo SGE *roll*, Cóndor *roll*, Xen *roll*, el Java *roll*, Ganglia *roll*, etc. (http://www.rocksclusters.org/wordpress/?page_id=4). Rocks Cluster es una distribución altamente empleada en el ámbito de clústers, por su facilidad de instalación e incorporación de nuevos nodos y por la gran cantidad de programas para el mantenimiento y monitorización del clúster.

Enlaces de interés

Para una lista detallada de las herramientas incluidas en Rocks Cluster, consultad: http://www.rocksclusters.org/roll-documentation/base/5.5/.

Las principales características de Rocks Cluster son:

1) Facilidad de instalación, ya que solo es necesario completar la instalación de un nodo llamado *nodo maestro* (*frontend*), el resto se instala con Avalache,

que es un programa P2P que lo hace de forma automática y evita tener que instalar los nodos uno a uno.

2) Disponibilidad de programas (conjunto muy amplio de programas que no es necesario compilar y transportar) y facilidad de mantenimiento (solo se debe mantener el nodo maestro).

3) Diseño modular y eficiente, pensado para minimizar el tráfico de red y utilizar el disco duro propio de cada nodo para solo compartir la información mínima e imprescindible.

La instalación se puede seguir paso a paso desde el sitio web de la distribución [15] y los autores garantizan que no se tarda más de una hora para una instalación básica. Rocks Cluster permite, en la etapa de instalación, diferentes módulos de software, los *rolls*, que contienen todo lo necesario para realizar la instalación y la configuración de sistema con estas nuevas 'adiciones' de forma automática y, además, decidir en el *frontend* cómo será la instalación en los nodos esclavos, qué *rolls* estarán activos y qué arquitectura se usará. Para el mantenimiento, incluye un sistema de copia de respaldo del estado, llamado *Roll Restore*. Este *roll* guarda los archivos de configuración y *scripts* (e incluso se pueden añadir los archivos que se desee).

1.7.1. Guía rápida de instalación

Este es un resumen breve de la instalación propuesta en [15] para la versión 6.1 y se parte de la base que el *frontend* tiene (mínimo) 30 GB de disco duro, 1 GB de RAM, arquitectura x86-64 y 2 interfaces de red (eth0 para la comunicación con internet y eth1 para la red interna); para los nodos 30 GB de disco duro, 512 MB de RAM y 1 interfaz de red (red interna). Después de obtener los discos *Kernel/Boot Roll, Base Roll, OS Roll CD1/2* (o en su defecto DVD equivalente), insertamos *kernel boot*, seguimos los siguientes pasos:

Enlaces de interés

Se puede descargar los discos desde: http://www.rocksclusters.org/wordpress/?page_id=80

1) Arrancamos el *frontend* y veremos una pantalla en la cual introducimos `build` y nos preguntará la configuración de la red (IPV4 o IPV6).

2) El siguiente paso es seleccionar los `rolls` (por ejemplo seleccionando los "CD/DVD-based Roll" y vamos introduciendo los siguientes *rolls*) y marcar en las sucesivas pantallas cuáles son los que deseamos.

3) La siguiente pantalla nos pedirá información sobre el clúster (es importante definir bien el nombre de la máquina, *Fully-Qualified Host Name*, ya que en caso contrario fallará la conexión con diferentes servicios) y también información para la red privada que conectará el *frontend* con los nodos y la red pública (por ejemplo, la que conectará el *frontend* con Internet) así como DNS y pasarelas.

4) A continuación se solicitará la contraseña para el root, la configuración del servicio de tiempo y el particionado del disco (se recomienda escoger "auto").

5) Después de formatear los discos, solicitará los CD de *rolls* indicados e instalará los paquetes y hará un *reboot* del *frontend*.

6) Para instalar los nodos se debe entrar como root en el *frontend* y ejecutar `insert-ethers`, que capturará las peticiones de DHCP de los nodos y los agregará a la base de datos del *frontend*, y seleccionar la opción *Compute* (por defecto, consultad la documentación para las otras opciones).

7) Encendemos el primer nodo y en el *boot order* de la BIOS generalmente se tendrá CD, PXE (Network Boot), Hard Disk, (si el ordenador no soporta PXE, entonces haced el *boot* del nodo con el *Kernel Roll CD*). En el *frontend* se verá la petición y el sistema lo agregará como `compute-0-0` y comenzará la descarga e instalación de los archivos. Si la instalación falla, se deberán reiniciar los servicios httpd, mysqld y autofs en el *frontend*.

8) A partir de este punto se puede monitorizar la instalación ejecutando la instrucción `rocks-console`, por ejemplo con `rocks-console compute-0-0`. Después de que se haya instalado todo, se puede salir de `insert-ethers` pulsando la tecla F8. Si se dispone de dos *racks*, después de instalado el primero se puede comenzar el segundo haciendo `insert-ethers -cabinet=1` los cuales recibirán los nombres `compute-1-0`, `compute-1-1`, etc.

9) A partir de la información generada en consola por `rocks list host`, generaremos la información para el archivo `machines.conf`, que tendrá un aspecto como:

```
nteum slot=2
compute-0-0 slots=2
compute-0-1 slots=2
compute-0-2 slots=2
compute-0-3 slots=2
```

y que luego deberemos ejecutar con

```
mpirun -np 10 -hostfile ./machines.conf ./mpi_program_to_execute.
```

1.8. FAI

FAI es una herramienta de instalación automatizada para desplegar Linux en un clúster o en un conjunto de máquinas en forma desatendida. Es equivalente a *kickstart* de RH, o Alice de SuSE. FAI puede instalar Debian, Ubuntu y RPMs de distribuciones Linux. Sus ventajas radican en que mediante esta herramienta se puede desplegar Linux sobre un nodo (físico o virtual) simplemente haciendo que arranque por PXE y quede preparado para trabajar y totalmente configurado (cuando se dice uno, se podría decir 100 con el mismo esfuerzo por parte del administrador) y sin ninguna interacción por medio. Por lo tanto, es un método escalable para la instalación y actualización de un clúster Beowulf o una red de estaciones de trabajo sin supervisión y con poco

esfuerzo. FAI utiliza la distribución Debian, una colección de scripts (su mayoría en Perl) y *cfengine* y *Preseeding d-i* para el proceso de instalación y cambios en los archivos de configuración.

Es una herramienta pensada para administradores de sistemas que deben instalar Debian en decenas o cientos de ordenadores y puede utilizarse además como herramienta de instalación de propósito general para instalar (o actualizar) un clúster de cómputo HPC, de servidores web, o un pool de bases de datos y configurar la gestión de la red y los recursos en forma desatendida. Esta herramienta permite tratar (a través de un concepto que incorpora llamado clases) con un hardware diferente y diferentes requisitos de instalación en función de la máquina y características que se disponga en su preconfiguración, permitiendo hacer despliegues masivos eficientes y sin intervención de administrador (una vez que la herramienta haya sido configurada adecuadamente). [29, 30, 31]

1.8.1. Guía rápida de instalación

En este apartado veremos una guía de la instalación básica sobre máquinas virtuales (en Virtualbox) para desplegar otras máquinas virtuales, pero se puede adaptar/ampliar a las necesidades del entorno con mínimas intervenciones y es una herramienta que no cuenta con *daemons*/DB y solo consiste en scripts (consultar las referencias indicadas).

Instalación del servidor. Si bien existe un paquete llamado `fai-quickstart` es mejor hacerlo por partes ya que muchos de los paquetes ya están instalados y sino se pueden instalar cuando es necesario (ver más info en http://fai-project.org/fai-guide/_anchor_id_inst_xreflabel_inst_installing_fai.html).

Descarga de los paquetes: en Debian Wheezy están todos los paquetes. Es mejor bajarse la última versión de http://fai-project.org/download/wheezy/ y aunque se deben descargar todos los paquetes `fai-alguna-cosa-.deb` solo utilizaremos `fai-server` y `fai-doc` en este ejemplo.

Instalación de la llave (*key*):

```
wget -O - http://fai-project.org/download/074BCDE4.asc | sudo
apt-key add -
```

Instalación de los paquetes: dentro del directorio donde se encuentren y como root ejecutar

```
dkpg -i fai-server_x.y_all.deb; dkpg -i fai-doc_x.y_all.deb
```

reemplazando la 'x.y' por la versión descargada.

Configurar el DHCP del servidor: el *boot* de los nodos y la transferencia del sistema operativo la realizaremos por una petición PXE y quien primero debe actuar es el `dhcp` para asignarle al nodo los parámetros de red y el archivo de boot, por lo que deberemos modificar */etc/dhcp/dhcpd.conf* agregando las primitivas **next-server** y **filename** en las zonas de nuestra red:

```
...
authoritative;
...
subnet 192.168.168.0 netmask 255.255.255.0 {
  ...
  next-server 192.168.168.254;
  filename "fai/pxelinux.0";
  ...
}
```

Configuración del servicio TFTP: la transferencia de los archivos del SO se realizará por el servicio TFTP (*Trivial File Transfer Protocol*) por lo que deberemos comprobar si el `tftp-hpa` está instalado y bien configurado con `netstat -anp|grep :69`. Deberá dar algo similar a :

```
udp 0 0 0.0.0.0:69 0.0.0.0:*
```

Si no, habrá que verificar si está instalado (`dpkg -l | grep tftp`) y en caso de que no esté instalado, ejecutar `apt-get install tftpd-hpa`. A continuación verificar que la configuración es la adecuada */etc/default/tftpd-hpa* (y recordar de reiniciar el servicio si se modifica):

```
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/srv/tftp"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"
```

Crear la configuración de FAI: ejecutando como root (Los archivos de configuración se encontrarán en */srv/fai/config* y hay reglas para modificarlos -ver la documentación-):

```
fai-setup
echo '/srv/fai/config 192.168.168.0/24(async,ro,no_subtree_check,no_root_squash)' >> /etc/exports
/etc/init.d/nfs-kernel-server restart

Con un exportfs veremos algo como:
    /home          192.168.168.0/24
    ...
    /srv/fai/nfsroot 192.168.168.0/24
    /srv/fai/config 192.168.168.0/24
```

Generar una configuración FAI-client: para todos los clientes (luego podremos hacer configuraciones diferentes):

```
fai-chboot -IBv -u nfs://192.168.168.254/srv/fai/config default
```

Deberemos modificar el archivo generado para que trabaje con NFSv3; con la versión del kernel utilizado en Wheezy (3.2.x) solo podemos utilizar V3 (si se desea utilizar NFSv4, deberemos cambiar a un kernel 3.12.x que están disponibles en repositorios como BackPorts http://backports.debian.org/). El archivo */srv/tftp/fai/pxelinux.cfg/default* debe quedar como (se ha modificado la variable root):

```
default fai-generated

label fai-generated
kernel vmlinuz-3.2.0-4-amd64
append initrd=initrd.img-3.2.0-4-amd64 ip=dhcp  root=192.168.168.254:/srv/fai/nfsroot:vers=3
aufs FAI_FLAGS=verbose,sshd,reboot FAI_CONFIG_SRC=nfs://192.168.168.254/srv/fai/config
FAI_ACTION=install
```

Copiar los mínimos ficheros de configuración: `cp -a /usr/share/doc/fai-doc/examples/simple/* /srv/fai/config/`

Configuración del nodo: nuestra prueba de concepto la haremos para un nodo en Virtualbox y primero es necesario instalar las expansiones de la versión que tenemos instalada. Por ejemplo, si tenemos VirtualBox 4.3.10, deberemos ir a *https://www.virtualbox.org/wiki/Downloads* e instalar *VirtualBox 4.3.10 Oracle VM VirtualBox Extension Pack All supported platforms* (esto nos permitirá tener un dispositivo que haga *boot* por PXE) y podremos verificar que están instaladas en el menu de VirtualBox – >*File* – >*Preferences* – >*Extensions*. Luego deberemos crear una nueva máquina virtual con un disco vacío y seleccionarla en *System* la opción de *boot order network* en primer lugar. Arrancar la máquina y ver que recibe IP (verificar los posibles errores en */var/log/messages|syslog*) y que comienza bajándose el **initrd-img** y luego el **kernel**.

Los detalles de la configuración se pueden consultar en http://fai-project.org/fai-guide/_anchor_id_config_xreflabel_config_installation_details.html

1.9. Logs

Linux mantiene un conjunto de registros llamados *system logs* o *logs* simplemente, que permiten analizar qué ha pasado y cuándo en el sistema, a través de los eventos que recoge del propio *kernel* o de las diferentes aplicaciones y casi todas las distribuciones se encuentran en */var/log*. Probablemente los dos archivos más importantes (y que en mayor o menor presencia están en todas las distribuciones) son */var/log/messages* y */var/log/syslog* los que tienen registros (Ascii) de eventos tales como errores del sistema, (re)inicios/apagados, errores de aplicaciones, advertencias, etc. Existe un comando, `dmesg`, que permite además ver los mensajes de inicio (en algunas distribuciones son los que aparecen en la consola o con la tecla Esc en el modo gráfico de arranque, o Crtl+F8 en otras distribuciones) para visualizar los pasos seguidos durante

el arranque (teniendo en cuenta que puede ser muy extenso, se recomienda ejecutar `dmesg | more`).

Por lo cual, el sistema nos permitirá analizar qué ha pasado y obtener las causas que han producido este registro y dónde/cuándo. El sistema incluido en Debian es `rsyslog` (http://www.rsyslog.com/) con un servicio (a través del *daemon* **rsyslogd** y que reemplaza al original *syslog*. Es un sistema muy potente y versátil y su configuración es través del archivo */etc/rsyslog.conf* o archivos en el directorio */etc/rsyslog.d*. Mirar la documentación sobre su configuración (`man rsyslog.conf` o en la página indicada) pero en resumen incluye una serie de directivas, filtros, templates y reglas que permiten cambiar el comportamiento de los logs del sistema. Por ejemplo, las reglas son de tipo **recurso.nivel acción** pero se puede combinar más de un recurso separado por ',', o diferentes niveles, o todos '*' o negarlo '!', y si ponemos '=' delante del nivel, indica solo ese nivel y no todos los superiores, que es lo que ocurre cuando solo se indica un nivel (los niveles pueden ser de menor a mayor importancia *debug, info, notice, warning, err, crit, alert, emerg*). Por ejemplo `*.warning /var/log/messages` indicará que todos los mensajes de *warning,err,crit,alert,emerg* de cualquier recurso vayan a parar a este archivo. Se puede incluir también un '-' delante del nombre del fichero, que indica que no se sincronice el fichero después de cada escritura para mejorar las prestaciones del sistema y reducir la carga.

Un aspecto interesante de los logs es que los podemos centralizar desde las diferentes máquinas de nuestra infraestructura en un determinado servidor para no tener que conectarnos si queremos 'ver' qué está ocurriendo a nivel de logs en esas máquinas (sobre todo si el número de máquinas es elevado). Para ello, debemos en cada cliente modificar el archivo */etc/rsyslog.conf* (donde la IP será la del servidor que recogerá los logs):

```
# Provides TCP forwarding.
*.*                             @@192.168.168.254:514
```

En el servidor deberemos quitar el comentario (#) de los módulos de recepción por TCP en */etc/rsyslog.conf*:

```
# Provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

Después de hacer las modificaciones de cada archivo no hay que olvidar hacer un `/etc/init.d/rsyslog restart`. A partir de ese momento, podemos mirar el */var/log/syslog* del servidor, por ejemplo, y veremos los registros marcados con el nombre (o IP) de la máquina donde se ha generado el log.

Existe un paquete llamado `syslog-ng` que presenta características avanzadas (por ejemplo, cifrado o filtros basados en contenidos) u otras equiva-

lentes con `rsyslog`. Más información en http://www.balabit.com/network-security/syslog-ng.

1.9.1. Octopussy

Octopussy es una herramienta gráfica que permite analizar los *logs* de diferentes máquinas y se integra con *syslog* o equivalentes reemplazándolos. Entre sus principales características soporta LDAP, alertas por mail, mensajes IM (Jabber), se integra con Nagios y Zabbix, permite generar reportes y enviarlos por mail, FTP y scp, hacer mapas de la arquitectura para mostrar estados e incorpora una gran cantidad de servicios predefinidos para registrar los *logs*.[32]

Su instalación (que puede resultar un poco complicada) comienza por añadir el repositorio *non-free /etc/apt/sources.list* para descargar paquetes adicionales (libmail-sender-perl) que necesitará este software:

```
deb http://ftp.es.debian.org/debian/ wheezy non-free
deb-src http://ftp.es.debian.org/debian/ wheezy non-free
```

Luego deberemos ejecutar `apt-get update`.

Después deberemos descargar el paquete debian desde la dirección web siguiente http://sourceforge.net/projects/syslog-analyzer/files/ (por ejemplo ctopussy_x.y.x_all.deb donde la x.y.z es la version -10.0.14 en julio de 2014-) e instalarlo.

```
dpkg -i octopussy_1.0.x_all.deb
apt-get -f install
```

Luego deberemos modificar el archivo *etc/rsyslog.conf*:

```
$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog   # provides kernel logging support (previously done by rklogd)
#$ModLoad immark  # provides --MARK-- message capability
# provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514
# provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

Y reinicia el rsyslog `/etc/init.d/rsyslog restart`.

Luego se deberán generar los certificados para *Octopussy Web Server* (también se puede utilizar la OpenCA para generar los certificados web propios tal y como se explicó en el capítulo de servidores):

```
openssl genrsa > /etc/octopussy/server.key
openssl req -new -x509 -nodes -sha1 -days 365 -key /etc/octopussy/server.key >
   /etc/octopussy/server.crt
```

En el último no introducir *passwd* y recordar en *Common Name* incluir el nombre.dominio de nuestra máquina.

Configurar el archivo de la configuración de Apache */etc/octopussy/apache2.conf* por el valor correcto:

```
SSLCertificateFile    /etc/octopussy/server.crt
SSLCertificateKeyFile /etc/octopussy/server.key
```

Reiniciar el servidor (propio) de web que luego se interconectará con nuestro Apache por SSL y puerto 8888.

```
/etc/init.d/octopussy web-stop
/etc/init.d/octopussy web-start
```

Y comprobar que funciona en la URL https://sysdw.nteum.org:8888/index.asp, aceptar el certificado e introducir como Usuario/Passwd admin/admin. A partir de la interfaz general se deberá configurar los *Devices, Service, Alerts*, etc. para definir el comportamiento que queremos que tenga la aplicación.

Si se desea deshabilitar y volver a `rsyslog` original (y dado que tiene servicios propios que se ponen en marcha durante el arranque) deberemos ejecutar la instrucción `update-rc.d octopussy remove` y renombrar los archivos de configuración en */etc/rsyslog.d* de la aplicación (por ejemplo,

```
mv /etc/rsyslog.d/xyz-ocotopussy.conf /etc/rsyslog.d/xyz-octopussy.conf.org
```

reemplazando xyz por lo que corresponda) y reiniciando *rsyslog* con

```
/etc/init.d/rsylog restart
```

1.9.2. Herramientas de monitorización adicionales

Además de las herramienta que se vieron en el capítulo de monitorización (como Ganglia, Nagios|Icinga, MRTG o Zabbix) existen otras herramientas que pueden ayudar en la gestión y administración de un clúster como son Cacti, XyMon, Collectd (también disponibles en Debian). En este apartado dedicaremos una pequeña referencia a Cacti y XYMon, ya que por su visión o capacidad de integración permiten tener información al instante sobre qué y cómo están ocurriendo las ejecuciones en el clúster.

Cacti Cacti [16] es una solución para la visualización de estadísticas de red y fue diseñada para aprovechar el poder de almacenamiento y la funcionalidad de generar gráficas que posee RRDtool (similar a Ganglia). Esta herramienta, desarrollada en PHP, provee diferentes formas de visualización, gráficos avanzados y dispone de una una interfaz de usuario fácil de usar, que la hacen interesante tanto para redes LAN como para redes complejas con cientos de dispositivos.

Su instalación es simple y requiere previamente tener instalado MySQL. Luego, haciendo `apt-get install cacti` se instalará el paquete y al final nos pedirá si queremos instalar la interfaz con la base de datos, solicitándonos el nombre de usuario y contraseña de la misma y la contraseña del usuario admin de la interfaz de Cacti (si no le hemos dado, contendrá la que define por defecto, que es usuario admin y *passwd* admin). Después instalará la configuración en el sitio de Apache2 (*/etc/apache2/conf.d/cacti.conf*) y re-arrancará los servidores y podremos conectarnos a la URL *http://sysdw.nteum.org/cacti/*. Las primeras tareas a realizar es hacer los ajustes pertinentes de la instalación (nos mostrará un *checklist*), cambiar la contraseña y nos mostrará a continuación una pantalla con las pestañas de gráficos y la de Consola, en la cual encontraremos las diferentes opciones para configurar y poner en marcha la monitorización más adecuada para nuestro sitio. Siguiendo el procedimiento indicado en *http://docs.cacti.net/manual:088:2_basics.1_first_graph*, será muy fácil incorporar los gráficos más adecuados para monitorizar nuestra instalación. Es interesante incorporar a Cacti un plugin llamado *weathermap* (http://www.network-weathermap.com/) que nos permitirá ver un diagrama de la infraestructura en tiempo real y los elementos monitorizados con datos dinámicos sobre su estado*.

***http://www.network-weathermap.com/manual/0.97b/pages/cacti-plugin.html**

Xymon Xymon [17], es un monitor de red/servicios (bajo licencia GPL) que se ejecuta sobre máquinas GNU/Linux. La aplicación fue inspirada en la version *open-source* de la herramienta Big Brother y se llamó Hobbit pero, como esta era una marca registrada, finalmente cambió a Xymon. Xymon ofrece una monitorización gráfica de las máquinas y servicios con una visualización adecuada y jerárquica óptima para cuando se necesita tener en una sola visualización el estado de la infraestructura (y sobre todo cuando se deben monitorizar centenas de nodos). También se puede entrar en detalles y mirar cuestiones específicas de los nodos, gráficos y estadísticas, pero entrando en niveles interiores de detalles. La monitorización de los nodos requiere de un cliente que será el encargado de enviar la información al *host* (equivalente a NRPE en Nagios). Su instalación en Debian es simple `apt-get install xymon`. Después podremos modificar el archivo */etc/hobbit/bb-host* para agregar una línea como group Server `158.109.65.67 sysdw.nteum.org ssh http://sysdw.nteum.org` y modificar */etc/apache2/conf.d/hobbit* para cambiar la línea `Allow from localhost` por `Allow from all`, reiniciar los dos servicios: `service hobbit restart` y `service apache2 restart` y lo tendremos disponible en la URL: *http://sysdw.nteum.org/hobbit/* (no descuidar la barra final). Para agregar nuevos *hosts* es muy fácil (se declaran en el archivo anterior), e instalar el cliente en los nodos tampoco tiene ninguna dificultad.

2. *Cloud*

Las infraestructuras *cloud* se pueden clasificar en 3+1 grandes grupos en función de los servicios que prestan y a quién:

1) Públicas: los servicios se encuentran en servidores externos y las aplicaciones de los clientes se mezclan en los servidores y con otras infraestructuras. La ventaja más clara es la capacidad de procesamiento y almacenamiento sin instalar máquinas localmente (no hay inversión inicial ni mantenimiento) y se paga por uso. Tiene un retorno de la inversión rápido y puede resultar difícil integrar estos servicios con otros propios.

2) Privadas: las plataformas se encuentran dentro de las instalaciones de la empresa/institución y son una buena opción para quienes necesitan alta protección de datos (estos continúan dentro de la propia empresa), es más fácil integrar estos servicios con otros propios, pero existe inversión inicial en infraestructura física, sistemas de virtualización, ancho de banda, seguridad y gasto de mantenimiento, lo cual supone un retorno más lento de la inversión. No obstante de tener infraestructura a tener infraestructura *cloud* y virtualizada, esta última es mejor por las ventajas de mayor eficiencia, mejor gestión y control, y mayor aislamiento entre proyectos y rapidez en la provisión.

3) Híbridas: combinan los modelos de nubes públicas y privadas y permite mantener el control de las aplicaciones principales al tiempo de aprovechar el *Cloud Computing* en los lugares donde tenga sentido con una inversión inicial moderada y a la vez contar los servicios que se necesiten bajo demanda.

4) Comunidad: Canalizan necesidades agrupadas y las sinergias de un conjunto o sector de empresas para ofrecer servicios de *cloud* a este grupos de usuarios.

Existen además diferentes capas bajo las cuales se pueden contratar estos servicios:

1) *Infraestructure as a Service* (IaaS): se contrata capacidad de proceso y almacenamiento que permiten desplegar aplicaciones propias que por motivos de inversión, infraestructura, coste o falta de conocimientos no se quiere instalar en la propia empresa (ejemplos de este tipo de EC2/S3 de Amazon y Azure de Microsoft).

2) *Platform as a Service* (PaaS): se proporciona además un servidor de aplicaciones (donde se ejecutarán nuestras aplicaciones) y una base de datos, donde

se podrán instalar las aplicaciones y ejecutarlas -las cuales se deberán desarrollar de acuerdo a unas indicaciones del proveedor- (por ejemplo Google App Engine).

3) *Software as a Service* (SaaS): comúnmente se identifica con *'cloud'*, donde el usuario final paga un alquiler por el uso de software sin adquirirlo en propiedad, instalarlo, configurarlo y mantenerlo (ejemplos Adobe Creative Cloud, Google Docs, o Office365).

4) *Business Process as a Service* (BPaaS): es la capa más nueva (y se sustenta encima de las otras 3), donde el modelo vertical (u horizontal) de un proceso de negocio puede ser ofrecido sobre una infraestructura *cloud*.

Si bien las ventajas son evidentes y muchos actores de esta tecnología la basan principalmente en el abaratamiento de los costes de servicio, comienzan a existir opiniones en contra (http://deepvalue.net/ec2-is-380-more-expensive-than-internal-cluster/) que consideran que un *cloud* público puede no ser la opción más adecuada para determinado tipo de servicios/infraestructura. Entre otros aspectos, los negativos pueden ser la fiabilidad en la prestación de servicio, seguridad y privacidad de los datos/información en los servidores externos, *lock-in* de datos en la infraestructura sin posibilidad de extraerlos, estabilidad del proveedor (como empresa/negocio) y posición de fuerza en el mercado no sujeto a la variabilidad del mercado, cuellos de botellas en la transferencias (empresa/proveedor), rendimiento no predecible, tiempo de respuesta a incidentes/accidentes, problemas derivados de la falta de madurez de la tecnología/infraestructura/gestión, acuerdos de servicios (SLA) pensados más para el proveedor que para el usuario, etc. No obstante es una tecnología que tiene sus ventajas y que con la adecuada planificación y toma de decisiones, valorando todos los factores que influyen en el negocio y escapando a conceptos superficiales (todos lo tienen, todos lo utilizan, bajo costo, expansión ilimitada, etc.), puede ser una elección adecuada para los objetivos de la empresa, su negocio y la prestación de servicios en IT que necesita o que forma parte de sus fines empresariales.[18]

Es muy amplia la lista de proveedores de servicios *cloud* en las diferentes capas, pero de acuerdo a la información de Synergy Research Group en 2014 (https://www.srgresearch.com/articles/amazon-salesforce-and-ibm-lead-cloud-infrastructure-service-segments), los líderes en el mercado de infraestructura *cloud* son:

1) *IasS*: Amazon con casi el 50 % e IBM y Rackspace con valores inferiores al 10 % cada uno y Google en valores inferiores.

2) *PaaS*: Salesforce (20 %) seguidas muy de cerca por Amazon, Google y Microsoft.

3) *Private/Hibrid*: IBM (15 %), Orange y Fujitsu con valores cercanos al 5 % cada uno.

4) *Business Process as a Service* (BPaaS): Existen varios operadores por ejemplo IBM, Citrix, VMware entre otros pero no hay datos definidos de cuota de mercado.

Esto significa un cambio apreciable con relación a los datos de 2012, donde se puede observar una variabilidad de la oferta muy alta*.

***https://www.srgresearch.com/articles/amazons-cloud-iaas-and-paas-investments-pay**

En cuanto a plataformas para desplegar *clouds* con licencias GPL, Apache, BSD (o similares), podemos contar entre las más referenciadas (y por orden alfabético):

1) AppScale: es una plataforma que permite a los usuarios desarrollar y ejecutar/almacenar sus propias aplicaciones basadas en Google AppEngine y puede funcionar como servicio o en local. Se puede ejecutar sobre AWS EC2, Rackspace, Google Compute Engine, Eucalyptus, Openstack, CloudStack, así como sobre KVM y VirtualBox y soporta Python, Java, Go, y plataformas PHP Google AppEngine. http://www.appscale.com/

2) Cloud Foundry: es plataforma *open source* PaaS desarrollada por VMware escrita básicamente en Ruby and Go. Puede funcionar como servicio y también en local. http://cloudfoundry.org/

3) Apache CloudStack: diseñada para gestionar grandes redes de máquinas virtuales como IaaS. Esta plataforma incluye todas las características necesarias para el despliegue de un IaaS: CO (*compute orchestration*), *Network-as-a-Service*, gestión de usuarios/cuentas, API nativa, *resource accounting*, y UI mejorada (*User Interface*). Soporta los *hypervisores* más comunes, gestión del *cloud* vía web o CLI y una API compatible con AWS EC2/S3 que permiten desarrollar *clouds* híbridos. http://cloudstack.apache.org/

4) Eucalyptus: plataforma que permite construir *clouds* privados compatibles con AWS. Este software permite aprovechar los recursos de cómputo, red y almacenamiento para ofrecer autoservicio y despliegue de recursos de *cloud* privado. Se caracteriza por la simplicidad en su instalación y estabilidad en el entorno con una alta eficiencia en la utilización de los recursos. https://www.eucalyptus.com/

5) Nimbus: es una plataforma que una vez instalada sobre un clúster, provee IasS para la construcción de *clouds* privados o de comunidad y puede ser configurada para soportar diferentes virtualizaciones, sistemas de colas, o Amazon EC2. http://www.nimbusproject.org/

6) OpenNebula: es una plataforma para gestionar todos los recursos de un centro de datos permitiendo la construcción de IasS privados, públicos e híbridos. Provee una gran cantidad de servicios, prestaciones y adaptaciones que han permitido que sea una de las plataformas más difundidas en la actualidad. http://opennebula.org/

7) OpenQRM: es una plataforma para el despliegue de *clouds* sobre un centro de datos heterogéneos. Permite la construcción de *clouds* privados, públicos e híbridos con IaaS. Combina la gestión del la CPU/almacenamiento/red para ofrecer servicios sobre máquinas virtualizadas y permite la integración con recursos remotos o otros *clouds*.
http://www.openqrm-enterprise.com/community.html

8) OpenShift: apuesta importante de la compañía RH y es una plataforma (version Origin) que permite prestar servicio *cloud* en modalidad PasS. OpenShift soporta la ejecución de binarios que son aplicaciones web tal y como se ejecutan en RHEL por lo cual permite un amplio número de lenguajes y *frameworks*. https://www.openshift.com/products/origin

9) OpenStack es una arquitectura software que permite el despliegue de *cloud* en la modalidad de IaaS. Se gestiona a través de una consola de control vía web que permite la provisión y control de todos los subsistemas y el aprovisionamiento de los recursos. El proyecto iniciado (2010) por Rackspace y NASA es actualmente gestionado por OpenStack Foundation y hay más de 200 compañías adheridas al proyecto, entre las cuales se encuentran las grandes proveedoras de servicios *clouds* públicos y desarrolladoras de SW/HW (ATT, AMD, Canonical, Cisco, Dell, EMC, Ericsson, HP, IBM, Intel, NEC, Oracle, RH, SUSE Linux, VMware, Yahoo entre otras). Es otra de las plataformas más referenciadas. http://www.openstack.org/

10) PetiteCloud: es una plataforma software que permite el despliegue de *clouds* privados (pequeños) y no orientado a datos. Esta plataforma puede ser utilizada sola o en unión a otras plataformas *clouds* y se caracteriza por su estabilidad/fiabilidad y facilidad de instalación. http://www.petitecloud.org

11) oVirt: si bien no puede considerarse una plataforma *cloud*, oVirt es una aplicación de gestión de entornos virtualizados. Esto significa que se puede utilizar para gestionar los nodos HW, el almacenamiento o la red y desplegar y monitorizar las máquinas virtuales que se están ejecutando en el centro de datos. Forma parte de RH Enterprise Virtualization y es desarrollado por esta compañía con licencia Apache. http://www.ovirt.org/

2.1. Opennebula

Considerando las opiniones en [19],[20], nuestra prueba de concepto la realizaremos sobre OpenNebula. Dado que la versión en Debian Wheezy es la 3.4 y la web de los desarrolladores es la 4.6.2 (http://opennebula.org/) hemos decidido instalar los paquetes Debian con KVM como *hypervisor* de la nueva versión*. Esta instalación es una prueba de concepto (funcional pero mínima), pero útil para después hacer un despliegue sobre una arquitectura real, donde por un lado ejecutaremos los servicios de OpenNebula y su interfaz gráfica (llamada Sunstone) y por otro, un hypervisor (*host*) que ejecutará las máquinas virtuales. OpenNebula asume dos roles separados: *Frontend* y Nodos. El ***Frontend*** es quien ejecuta los servicios/gestión web y los **Nodos** ejecutan la

***http://docs.opennebula.org/4.6/design_and_installation/quick_starts/qs_ubuntu_kvm.html**

máquina virtuales, y si bien en nuestra instalación de pruebas ejecutaremos el Frontend y Nodos en la misma máquina, se recomienda ejecutar las máquinas virtuales en otros *hosts* que tengan las extensiones de virtualización (en nuestro caso KVM). Para verificar si disponemos estas extensiones, HW podemos ejecutar `grep -E 'svm|vmx'/proc/cpuinfo` y si nos da salida es casi seguro que el sistema soporta estas extensiones. Tened en cuenta que esto lo deberemos hacer sobre un sistema operativo base (*Bare-Metal*) y no sobre uno ya virtualizado, es decir, no podemos instalar OpenNebula sobre una máquina virtualizada por VirtualBox por ejemplo (hay hypervisores que permiten esta instalación como VMWare ESXi pero Virtualbox no). Los paquetes que conforma la instalación son:

1) opennebula-common: archivos comunes.

2) libopennebula-ruby: librerías de ruby

3) opennebula-node: paquete que prepara un nodo donde estarán la VM.

4) opennebula-sunstone: OpenNebula Sunstone *Web Interface*

5) opennebula-tools: *Command Line interface*

6) opennebula-gate: permite la comunicación entre VMs y OpenNebula

7) opennebula-flow: gestiona los servicios y la "elasticidad"

8) opennebula: OpenNebula *Daemon*

Instalación del Frontend (como root): Instalar el repositorio: `wget -q -O- http://downloads.opennebula.org/repo/Debian/repo.key | apt-key add -` y luego los agregamos a la lista

```
echo "deb http://downloads.opennebula.org/repo/Debian/7 stable
opennebula"> /etc/apt/sources.list.d/opennebula.list
```

Instalar los paquetes: primero `apt-get update` y luego `apt-get install opennebula opennebula-sunstone` (si el nodo está en un *host* separado deberán compartir un directorio por NFS, por lo cual es necesario también instalar el `nfs-kernel-server`).

Servicios: deberemos tener dos servicios en marcha que son OpenNebula *daemon* (oned) y la interfaz gráfica (sunstone) la cual solo está configurada por seguridad para atender el *localhost* (si se desea cambiar, editar */etc/one/sunstone-server.conf* y cambiar **:host: 127.0.0.1** por **:host: 0.0.0.0** y reiniciar el servidor `/etc/init.d/opennebula-sunstone restart`.

Configurar el NFS (si estamos en un único servidor con *Frontend*+Nodo esta parte no es necesario): Agregar a */etc/exports* del *Frontend* `/var/lib/one/ *(rw,sync,no_subtree_check,root_squash)` y después reiniciar el servicio (`service nfs-kernel-server restart`).

Configurar la llave pública de SSH: OpenNebula necesita acceder por SSH a los nodos como el usuario **oneadmin** y sin *passwd* (desde cada nodo a otro, incluido el *frontend*) por lo cual, nos cambiamos como usuario oneadmin (`su - oneadmin`) y ejecutamos

```
cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
```

y agregamos el siguiente texto a *~/.ssh/config* (para que no pida confirmación de known_hosts):

```
cat << EOT > ~/.ssh/config
Host *
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
EOT
chmod 600 ~/.ssh/config
```

Instalación de los nodos: repetimos el paso de configurar el repositorio y ejecutamos la instrucción `apt-get install opennebula-node nfs-common bridge-utils` (en nuestro caso no es necesario, ya que es la misma máquina y solo debemos instalar `opennebula-node` y `bridge-utils`). Verificar que podemos acceder como oneadmin a cada nodo y copiamos las llaves de `ssh`.

Configuración de la red: (hacer un *backup* de los archivos que modificaremos previamente) generalmente tendremos eth0 y la conectaremos a un *bridge* (el nombre del *bridge* deberá ser el mismo en todos los nodos) y en */etc/network/interfaces* agregamos (poner las IP que correspondan):

```
auto lo
iface lo inet loopback
auto br0
iface br0 inet static
        address 192.168.0.10
        network 192.168.0.0
        netmask 255.255.255.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

Si tenemos servicio de DHCP reemplazar por:

```
auto lo
iface lo inet loopback
auto br0
iface br0 inet dhcp
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

Y reiniciamos la red `/etc/init.d/networking restart`

Configuramos el NFS sobre los nodos (no necesario en nuestro caso): agregamos en */etc/fstab*

```
192.168.1.1:/var/lib/one/ /var/lib/one/ nfs soft,intr,rsize=8192,wsize=8192,noauto
```

donde 192.168.1.1 es la IP del *frontend*; luego montamos el directorio `mount /var/lib/one/`.

Configuramos Qemu: el usuario **oneadmin** debe poder manejar libvirt como root por lo cual ejecutamos:

```
cat << EOT > /etc/libvirt/qemu.conf
user  = "oneadmin"
group = "oneadmin"
dynamic_ownership = 0
EOT
```

Reiniciamos libvirt con `service libvirt-bin restart`

Uso básico: En el Frontend podemos conectarnos a la interfaz web en la URL *http://frontend:9869*. El usuario es **oneadmin** y el *passwd* está su HOME en */.one/one_auth* que se genera en forma aleatoria (*/var/lib/one/.one/one_auth*). Para interactuar con OpenNebula se debe hacer desde el usuario **oneadmin** y desde el frontend (para conectarse hacer simplemente `su - oneadmin`).

Agregar un host: es lo primero que hay que hacer para luego ejecutar VM, se puede hacer desde la interfaz gráfica o desde CLI haciendo como oneadmin `onehost create localhost -i kvm -v kvm -n dummy` (poner el nombre del *host* correcto en lugar de *localhost*).

Si hay errores probablemente son de `ssh` (verificar que igual que en el caso de oneadmin se puede conectar a los otros *hosts* sin *passwd*) y los errores los veremos en */var/log/one/oned.log*.

El segundo paso es agregar la red, una imagen y un template antes de lanzar una VM:

Red: (se puede hacer desde la interfaz gráfica también), creo el archivo *mynetwork.one* con el siguiente contenido:

```
NAME = "private"
TYPE = FIXED
BRIDGE = br0
LEASES = [ IP=192.168.0.100 ]
LEASES = [ IP=192.168.0.101 ]
LEASES = [ IP=192.168.0.102 ]
```

Donde las IP deberán estar libres en la red que estamos haciendo el despliegue y ejecutamos `onevnet create mynetwork.one`

En el caso de la imagen, esta se puede hacer desde CLI pero es más simple hacerla desde la interfaz web. Seleccionamos **Marketplace** y luego **ttylinux-kvm** (es una imagen pequeña de prueba) y luego le decimos **Import**. Podremos verificar que la imagen está en el apartado correspondiente, pero el estado es LOCK hasta que se la haya bajado y cuando finalice, veremos que cambia a READY. Luego podremos crear un template seleccionando esta imagen, la red, y allí podremos poner la llave pública del *∼/.ssh/id_dsa.pub* en el apartado **Context**, y desde este apartado (**Templates**) podremos decirle que cree una instancia de este template (o en su defecto en **VirtualMachines** crear una utilizando este template). Veremos que la VM pasa de PENDING – > PROLOG – > RUNNING (si falla podremos ver en los *logs* la causa) y luego nos podremos conectar o bien por `ssh` a la IP de la VM o a través de **VNC** en la interfaz web.

Como se ha podido comprobar, la instalación está bastante automatizada pero se debe tener en cuenta que es una infraestructura compleja y que se debe dedicar tiempo y análisis a determinar las causas por las cuales se producen los errores y solucionarlos. Existe una gran cantidad de documentación y sitios en Internet pero recomiendo comenzar por las fuentes [21] y [22].

En este apartado se ha visto una prueba de concepto funcional pero OpenNebula es muy extenso y flexible, y permite gran cantidad de opciones/extensiones. A través de OpenNebula C12G Labs (http://c12g.com/) se podrán encontrar y descargar imágenes de máquinas virtuales creadas para OpenNebula* y listas para ponerlas en funcionamiento (son las que se ven desde el MarketPlace de la interfaz web), pero al descargarlas sobre nuestro servidor la podremos utilizar siempre cuando la necesitemos y no se deberá descargar cada vez (no se debe descomprimir ni hacer nada, utilizar tal y como está). Una de las extensiones interesantes es OpenNebula Zones (llamada **ozones**) que nos permite una administración centralizada de múltiples instancias de OpenNebula (zones), gestionando diferentes dominios administrativos. El módulo es gestionado por el *oZones administrator*, que es que el administra los permisos a las diferentes zonas de los usuarios individuales**. Su configuración puede encontrarse en http://docs.opennebula.org/4.4/advanced_administration/multiple_zone_and_virtual_data_centers/zonesmngt.html.

***http://marketplace.c12g.com/appliance**

****http://archives.opennebula.org/documentation:archives:rel3.0:ozones**

3. DevOps

Tal y como hemos comentado al inicio de este capítulo, Devops se puede considerar, en palabras de los expertos, un movimiento tanto en lo profesional como en lo cultural del mundo IT. Si bien no existen todas las respuestas todavía, un administrador se enfrentará a diferentes 'comunidades' (y él mismo formará parte de una de ellas), que tendrán nuevas necesidades de desarrollo y producción de servicios/productos dentro del mundo IT y con las premisas de 'más rápido', 'más eficiente', 'de mayor calidad' y totalmente adaptable a los 'diferentes entornos'. Es decir, el grupo de trabajo deberá romper las barreras existentes entre los departamentos de una empresa permitiendo que un producto pase rápidamente desde el departamento de investigación al de diseño, luego al de desarrollo + producción, luego al de test + calidad y por último, a ventas, y con las necesidades de herramientas que permitan desplegar todas estas actividades en cada una de las fases y llevar el control de todos por los responsables de cada uno de los ámbitos, incluidos los funcionales y los de la organización/directivos. Es por ello por lo que un administrador necesitará herramientas de gestión, control, despliegue, automatización y configuración. Una lista (corta) de las herramientas que podríamos considerar de acuerdo a nuestros objetivos de licencia GPL-BSD-Apache o similares (es interesante el sitio http://devs.info/, ya que permite acceder a la mayor parte de las herramientas/entornos/documentación para programadores y una lista más detallada -que incluye SW propietario- se puede encontrar en [27]):

1) **Linux**: Ubuntu|Debian^{v}, Fedora^{v}|CentOS|SL

2) **IasS**: Cloud Foundry, OpenNebula^{v}, OpenStack

3) **Virtualización**: KVM^{v}, Xen, VirtualBox^{v}, Vagrant^{v}

4) Contenedores: LXC^{v}, Docker^{v}

5) Instalación SO: Kickstart y Cobbler (rh), Preseed|Fai^{v} (deb), Rocks^{v}

6) Gestión de la configuración: Puppet^{v}, Chef^{v}, CFEngine, SaltStack, Juju, bcfg2, mcollective, fpm (effing)

7) Servidores Web y aceleradores: Apache^{v}, nginx^{v}, varnish, squid^{v}

8) BD: MySQL^{v}|MariaDB^{v}, PostgreSQL^{v}, OpenLDAP^{v}, MongoDB, Redis

9) Entornos: Lamp, Lamr, AppServ, Xampp, Mamp

10) Gestión de versiones: Git^{v}, Subversion^{v}, Mercurial^{v}

11) Monitorización/supervisión: Nagios^{v}, Icinga, Ganglia^{v}, Cacti^{v}, Monin^{v}, MRTG^{v}, XYmon^{v}

12) Misc: pdsh, ppsh, pssh, GNUparallel, nfsroot, Multihost SSH Wrapper, lldpd, Benchmarks^{v}, Librerías^{v}

13) Supervisión: Monit^{v}, runit, Supervisor, Godrb, BluePill-rb, Upstart, Systemd^{v}

14) Security: OpenVas^{v}, Tripwire^{v}, Snort^{v}

15) Desarrollo y Test: Jenkins, Maven, Ant, Gradle, CruiseControl, Hudson

16) Despliegue y Workflow: Capistrano

17) Servidores de aplicaciones: JBoss, Tomcat, Jetty, Glassfish,

18) Gestión de Logs: Rsyslog^{v}, Octopussy^{v}, Logstash

Excepto las que son muy orientadas a desarrollo de aplicaciones y servicios, en las dos asignaturas se han visto (o se verán dentro de este capítulo) gran parte de ellas (marcadas con ^{v}) y sin ninguna duda, con los conocimientos obtenidos, el alumno podrá rápidamente desplegar todas aquellas que necesite y que no se han tratado en estos cursos.

A continuación veremos algunas herramientas (muy útiles en entornos DevOps) más orientadas a generar automatizaciones en las instalaciones o generar entornos aislados de desarrollos/test/ejecución que permiten de forma simple y fácil (y sin pérdidas de prestaciones/rendimiento) disponer de herramientas y entornos adecuados a nuestras necesidades.

3.1. Linux Containers, LXC

LXC (LinuX Containers) es un método de virtualización a nivel del sistema operativo para ejecutar múltiples sistemas Linux aislados (llamados contenedores) sobre un único *host*. El *kernel* de Linux utiliza `cgroups` para poder aislar los recursos (CPU, memoria, E/S, network, etc.) lo cual no requiere iniciar ninguna máquina virtual. `cgroups` también provee aislamiento de los espacios de nombres para aislar por completo la aplicación del sistema operativo, incluido árbol de procesos, red, id de usuarios y sistemas de archivos montados. A través de una API muy potente y herramientas simples, permite crear y gestionar contenedores de sistema o aplicaciones. LXC utiliza diferentes módulos del *kernel* (ipc, uts, mount, pid, network, user) y de aplicaciones (Apparmor, SELinux profiles, Seccomp policies, Chroots -pivot_root- y Control groups -cgroups-) para crear y gestionar los contenedores. Se puede considerar que LXC está a medio camino de un 'potente' `chroot` y una máquina virtual, ofreciendo un entorno muy cercano a un Linux estándar pero sin necesidad de tener un kernel separado. Esto es más eficiente que utilizar virtualización con un `hypervisor` (KVM, Virtualbox) y más rápido de (re)iniciar, sobre todo si se están haciendo desarrollos y es necesario hacerlo frecuentemente, y su impacto en el rendimiento es muy bajo (el contenedor no ocupa recursos)

y todos se dedicarán a los procesos que se estén ejecutando. El único inconveniente de la utilización de LXC es que solo se pueden ejecutar sistemas que soportan el mismo *kernel* que su anfitrión, es decir, no podremos ejecutar un contenedor BSD en un sistema Debian.

La instalación se realiza a través de `apt-get install lxc lxctl` y se pueden instalar otros paquetes adicionales que son opcionales (`bridge-utils libvirt-bin debootstrap`), no obstante, si queremos que los contenedores tengan acceso a la red, con Ip propia es conveniente instalar el paquete `bridge-utils`: `apt-get install bridge-utils`. Para preparar el *host* primero debemos montar el directorio `cgroup` añadiendo a */etc/fstab* la siguiente línea `cgroup /sys/fs/cgroup cgroup defaults 0 0` y verificando que podemos hacer `mount -a` y el sistema *cgroups* aparecerá montado cuando ejecutemos el comando `mount`. Ver detalles y posibles soluciones a errores en [23, 24, 26]. A partir de este momento podemos verificar la instalación con `lxc-checkconfig` que dará una salida similar a:

```
Found kernel config file /boot/config-3.2.0-4-amd64
--- Namespaces ---
 Namespaces: enabled
 Utsname namespace: enabled
 Ipc namespace: enabled
 Pid namespace: enabled
 User namespace: enabled
 Network namespace: enabled
 Multiple /dev/pts instances: enabled

--- Control groups ---
 Cgroup: enabled
 Cgroup clone_children flag: enabled
 Cgroup device: enabled
 Cgroup sched: enabled
 Cgroup cpu account: enabled
 Cgroup memory controller: enabled
 Cgroup cpuset: enabled

--- Misc ---
 Veth pair device: enabled
 Macvlan: enabled
 Vlan: enabled
 File capabilities: enabled
```

Si surgen opciones deshabilitadas, se debe mirar la causa y solucionarlo (aunque algunas pueden estarlo). Si bien la distribución incluye un contenedor 'debian' (*/usr/share/lxc/templates*) es recomendable obtener uno desde la dirección https://github.com/simonvanderveldt/lxc-debian-wheezy-template que da solución a algunos problemas que tiene el original cuando se instala sobre Debian Wheezy. Para ello podemos hacer:

```
wget https://github.com/simonvanderveldt/lxc-debian-wheezy-template/raw/master/lxc-debian-wheezy-robvdhoeven
-O /usr/share/lxc/templates/lxc-debianW
host# chown root:root /usr/share/lxc/templates/lxc-debianW
host# chmod +x /usr/share/lxc/templates/lxc-debianW
```

A continuación se crear el primer contenedor como `lxc-create -n mycont -t debianW` (en este caso lo llamo **mycont** y utilizo el template de **debianW**

pero se pueden también debian que es el que incluye la distribución) Si deseamos configurar la red debemos primero configurar el *bridge* modificando (en el host) */etc/network/interfaces* con lo siguiente:

```
# The loopback network interface
auto lo br0
iface lo inet loopback

# The primary network interface
iface eth0 inet manual          # la ponemos en manual para evitar problemas
iface br0 inet static           # inicializamos el bridge
address 192.168.1.60
netmask 255.255.255.0
gateway 192.168.1.1
bridge_ports eth0
bridge_stp off       # disable Spanning Tree Protocol
    bridge_waitport 0    # no delay before a port becomes available
    bridge_fd 0          # no forwarding delay
```

Hacemos un `ifdown eth0` y luego un `ifup br0` y verificamos con `ifconfig` y por ejemplo `ping google.com` que tenemos conectividad. Luego modificamos el archivo de configuración */var/lib/lxc/mycont/config* para insertar el bridge:

```
lxc.utsname = myvm
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0          #debe existir en el host
lxc.network.ipv4 = 192.168.1.100/24     # Ip para el contenedor 0.0.0.0 indica dhcp
lxc.network.hwaddr = 00:1E:1E:1a:00:00
```

Los comandos más útiles para gestionar los contenedores son:

1) Para iniciar la ejecución como *daemon* -en *background*- (el login/*passwd* por defecto es root/root): `lxc-start -n mycont -d`

2) Para conectarnos al contenedor: `lxc-console -n mycont` "Ctrl+a q" para salir de la consola.

3) Para iniciar el contenedor anexado a la consola: `lxc-start -n mycont` (en *foreground*)

4) Para parar la ejecución del contenedor: `lxc-stop -n myvm`

5) Para iniciar los contenedores al *boot* en forma automática se deberá hacer un enlace del archivo de configuración en el directorio */etc/lxc/auto/*, por ejemplo `ln -s /var/lib/lxc/mycont/config /etc/lxc/auto/mycont`

6) Para montar sistemas de archivos externos dentro del contenedor agregar a */var/lib/lxc/mycont/config* la línea

```
lxc.mount.entry=/path/in/host/mount_point /var/lib/lxc/mycont/rootfs/mount_moint
none bind 0 0
```

y reiniciar el contenedor.

Se debe tener cuidado cuando se inicia el contenedor sin '-d', ya que no hay forma de salir (en la versión actual el 'Ctrl+a q' no funciona). Iniciar siempre los contenedores en *background* (con '-d') a no ser que necesite depurar porque el contenedor no arranca. Otra consideración a tener es que el comando `lxc-halt` ejecutará el `telinit` sobre el archivo */run/initctl* si existe y apagará el *host*, por lo cual hay que apagarlo con `lxc-stop` y tampoco hacer un `shutdown -h now` dentro del contenedor, ya que también apagará el *host*. También existe un panel gráfico para gestionar los contenedores vía web http://lxc-webpanel.github.io/install.html (en Debian hay problemas con el apartado de red; algunos de ellos los soluciona https://github.com/vaytess/LXC-Web-Panel/tree/lwp-backup. Para ello clonar el sitio

```
git clone https://github.com/vaytess/LXC-Web-Panel.git
```

y luego reemplazar el directorio obtenido por */srv/lwp* -renombrar este antes- y hacer un `/etc/init.d/lwp restart`). También es importante notar que la versión disponible en Debian es la 0.8 y en la web del desarrollador* es la 1.04, que tiene muchos de los errores corregidos y su compilación en muy simple (ver archivo INSTALL dentro de paquete) por lo cual, si se va a trabajar con ella, se recomienda esta versión.

***https://linuxcontainers.org /downloads/**

3.2. Docker

Docker es una plataforma abierta para el desarrollo, empaquetado y ejecución de aplicaciones de forma que se puedan poner en producción o compartir más rápido separando estas de la infraestructura, de forma que sea menos costoso en recursos (básicamente espacio de disco, CPU y puesta en marcha) y que esté todo preparado para el siguiente desarrollador con todo lo que Ud. puso pero nada de la parte de infraestructura. Esto significará menor tiempo para para probar y acelerará el despliegue acortando en forma significativa el ciclo entre que se escribe el código y pasa a producción. Docker emplea una plataforma (contenedor) de virtualización ligera con flujos de trabajo y herramientas que le ayudan a administrar e implementar las aplicaciones proporcionando una forma de ejecutar casi cualquier aplicación en forma segura un contenedor aislado. Este aislamiento y seguridad permiten ejecutar muchos contenedores de forma simultánea en el *host* y dada la naturaleza (ligera) de los contenedores todo ello se ejecuta sin la carga adicional de un *hypervisor* (que sería la otra forma de gestionar estas necesidades compartiendo la VM) lo cual significa que podemos obtener mejores prestaciones y utilización de los recursos. Es decir, con un VM cada aplicación virtualizada incluye no solo la aplicación, que pueden ser decenas de MBytes -binarios + librerías- sino también el sistema operativo 'guest', que pueden ser varios Gbytes; en cambio, en Docker lo que se denomina DE (Docker Engine) solo comprende la aplicación y sus dependencias, que se ejecuta como un proceso aislado en el espacio de usuario del SO 'host', compartiendo el *kernel* con otros contenedores. Por lo tanto,

tiene el beneficio del aislamiento y la asignación de recursos de las máquinas virtuales, pero es mucho más portátil y eficiente transformándose en el entorno perfecto para dar soporte al ciclo de vida del desarrollo de software, test de plataformas, entornos, etc.[33]

El entorno está formado por dos grandes componentes: **Docker** [34] (es la plataforma de virtualización -contenedor-) y **Docker Hub** [35] (una plataforma SasS que permite obtener y publicar/gestionar contenedores ya configurados). En su arquitectura, Docker utiliza una estructura cliente-servidor donde el cliente (CLI **docker**) interactúa con el *daemon* Docker, que hace el trabajo de la construcción, ejecución y distribución de los contenedores de Docker. Tanto el cliente como el *daemon* se pueden ejecutar en el mismo sistema, o se puede conectar un cliente a un *daemon* de Docker remoto. El cliente Docker y el servicio se comunican a través de sockets o una API RESTful.

Para la instalación no están disponibles los paquetes sobre Debian Wheezy (sí están sobre Jessie) y debemos actualizar el kernel, ya que Docker necesita una versión 3.8 (o superior). Para ello cargaremos el nuevo kernel 3.14 (julio 2014) desde Debian Backports ejecutando:

```
echo "deb http://ftp.debian.org/debian/ wheezy-backports main non-free contrib" >
/etc/apt/sources.list.d/backport.list
apt-get update
apt-get -t wheezy-backports install linux-image-amd64 linux-headers-amd64
reboot
```

Una vez que hemos seleccionado el nuevo *kernel* durante el arranque podemos hacer:

```
Instalamos unas dependencias:
    apt-get install apt-transport-https
Importamos la key de Ubuntu:
    apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
    36A1D7869245C8950F966E92D8576A8BA88D21E9
Agregamos el repositorio de Docker:
    echo "deb http://get.docker.io/ubuntu docker main" > /etc/apt/sources.list.d/docker.list"
Actualizamos e instalamos:
    apt-get update
    apt-get install lxc-docker
Ahora podemos verificar si la instalación funciona ejecutando una imagen de Ubuntu (local)
dentro de su contenedor:
    docker run -i -t ubuntu /bin/bash           (Crtl+D para salir del contenedor)
Desde dentro podremos ejecutar \texttt{more lsb-release} y nos indicará:
        DISTRIB_ID=Ubuntu
        DISTRIB_RELEASE=14.04
        DISTRIB_CODENAME=trusty
        DISTRIB_DESCRIPTION="Ubuntu 14.04 LTS"
Pero también podemos hacer \texttt{docker run -i -t centos /bin/bash} como que la imagen no la tiene en
local se la descargará del Docker Hub y la ejecutará.
```

Haciendo more */etc/os-release* nos indicará que es un CentOS Linux, 7 (Core) e información complementaria.

Antes de cargar el primer contenedor nos puede dar un error como *Cannot start container ... mkdir /sys/fs/devices: operation not permitted*. Esto es debido a una instalación previa de LXC anterior y solo debemos comentar en el archivo */etc/fstab* la línea `#cgroup /sys/fs/cgroup cgroup defaults 0 0`, reiniciar la máquina y ejecutar nuevamente el contenedor. Los comandos para inciarse (además de los que hemos visto) son:

1) docker: muestra todas las opciones.

2) docker images: lista las imágenes localmente.

3) docker search patrón: busca contenedores/imágenes que tengan ese patrón.

4) docker pull nombre: obtiene imágenes del Hub. nombre puede ser <username>/<repository>para la particulares.

5) docker run name cmd: ejecutará el contenedor y dentro el comando indicado por cmd

6) docker ps -l: permite obtener el ID e información de una imagen.

7) docker commit ID name: actualiza la imagen con lo que se ha instalado hasta ese momento (con 3-4 números del ID es suficiente).

8) docker inspect: muestra las imágenes corriendo -similar a docker ps-.

9) docker push: salva la imagen el el Hub (debemos registrarnos primero) y está disponible para otros usarios/host que se la quieran instalar con todo lo que hemos configurado dentro.

10) docker cp: copia archivos/folders desde el contenedor al host.

11) docker export/import: exporta/importa el contenedor en un archivo tar.

12) docker history: muestra la historia de una imagen.

13) docker info: muestra información general (imagenes, directorios, ...).

14) docker kill: para la ejecución de un contenedor.

15) docker restart: reinicia un contenedor.

16) docker rm: elimina un contendor.

17) docker rmi: elimina imagenes.

18) docker start/stop: inicia/para un contenedor.

3.3. Puppet

Puppet es una herramienta de gestión de configuración y automatización en sistemas IT (licencia Apache, -antes GPL-). Su funcionamiento es gestionado a través de archivos (llamados *manifiests*) de descripciones de los recursos del sistema y sus estados, utilizando un lenguaje declarativo propio de la herramienta. El lenguaje Puppet puede ser aplicado directamente al sistema, o compilado en un catálogo y distribuido al sistema de destino utilizando un modelo

cliente-servidor (mediante una API REST), donde un agente interpela a los proveedores específicos del sistema para aplicar el recurso especificado en los *manifiests*. Este lenguaje permite una gran abstracción que habilita a los administradores describir la configuración en términos de alto nivel, tales como usuarios, servicios y paquetes sin necesidad de especificar los comandos específicos del sistema operativo (apt, dpkg, etc.).[36, 37, 38] El entorno Puppet tiene dos versiones Puppet (*Open Source*) y Puppet Enterprise (producto comercial) cuyas diferencias se pueden ver en http://puppetlabs.com/puppet/enterprise-vs-open-source y además se integran con estos entornos otras herramientas como MCollective (*orchestration framework*), Puppet Dashboard (consola web pero abandonada en su desarrollo), PuppetDB *(datawarehouse* para Puppet), Hiera (herramienta de búsqueda de datos de configuración), Facter (herramienta para la creación de catálogos) y Geppetto (IDE -integrated development environment- para Puppet).

3.3.1. Instalación

Es importante comenzar con dos máquinas que tengan sus *hostname* y definición en */etc/hosts* correctamente y que sean accesibles a través de la red con los datos obtenidos del */etc/hosts* (es importante que el nombre de la máquina -sin dominio- de este archivo coincida con el nombre especificado en *hostname* ya que si no habrá problemas cuando se generen los certificados).

```
Obtenemos los repositorios e instalamos puppetmaster en el servidor:
    wget http://apt.puppetlabs.com/puppetlabs-release-wheezy.deb
    dpkg -i puppetlabs-release-wheezy.deb
    apt-get update
    apt-get install puppetmaster
Puppetlabs recomienda ejecutar el siguiente comando antes de correr puppetmaster:
    puppet resource service puppetmaster ensure=running enable=true
    service puppetmaster restart

Sobre el cliente, instalar los repositorios (tres primeras instrucciones) e instalar puppet
(si el cliente es Ubuntu, no es necesario instalar el repositorio):
    apt-get install puppet
Sobre el cliente modificar /etc/puppet/puppet.conf para agregar en la sección [main] el
servidor (sysdw.nteum.org en nuestro caso) y reiniciar:
    [main]
    server=sysdw.nteum.org
    /etc/init.d/puppet restart
Luego ejecutar sobre el cliente:
    puppet agent  --waitforcert 60 --test
Este comando enviará una petición de firma del certificado al servidor y esperará a que este se lo
firme, por lo cual, rápidamente sobre el servidor se debe hacer:
    puppetca --list
Nos responderá con algo como "pucli.nteum.org" (...) y ejecutamos:
    puppetca --sign pucli.nteum.org
Veremos información por parte del servidor y también por parte del cliente y la ejecución de los
catálogos que haya pendientes.
```

De esta forma tenemos instalado el cliente y el servidor y ahora deberemos hacer nuestro primer 'manifiesto'. Para ello organizaremos la estructura de acuerdo a las recomendaciones de PuppetLabs haciendo un árbol que tendrá la siguiente estructura a partir de */etc/puppet*:

```
+-- auth.conf
+-- autosign.conf
+-- environments
|   +-- common
|   +-- development
|   |   +-- modules
|   +-- production
|       +-- modules
|           +-- ntp
+-- etckeeper-commit-post
+-- etckeeper-commit-pre
+-- files
|   +-- shadow.sh
+-- fileserver.conf
+-- manifests
|   +-- nodes
|   |   +-- client1.pp
|   |   +-- server.pp
|   +-- site.pp
+-- modules
|   +-- accounts
|   |   +-- manifests
|   |       +-- init.pp
|   |       +-- system.pp
|   +-- elinks
|   |   +-- manifests
|   |       +-- init.pp
|   +-- nmap
|       +-- manifests
|           +-- init.pp
+-- node.rb
+-- puppet.conf
+-- rack
+-- templates
```

Comenzaremos por el archivo */etc/puppet/manifests/site.pp*, que como contenido tendrá `import 'nodes/*.pp'`. En el directorio */etc/puppet/manifests/nodes* tendremos dos archivos (server.pp y client1.pp) con el siguiente contenido:

```
Archivo server.pp:
    node 'sysdw.nteum.org' {
    }
Archivo client1.pp:
   node 'pucli.nteum.org' {
        include nmap
        include elinks
}
```

Donde definimos dos servicios a instalar en el cliente pucli.nteum.org (nmap y elinks). Después definimos los directorios */etc/puppet/modules/elinks/manifests* y */etc/puppet/modules/nmap/manifests* donde habrá un archivo *init.pp* con la tarea a realizar:

```
Archivo /etc/puppet/modules/elinks/manifests/init.pp:
    class elinks {
        case $operatingsystem {
            centos, redhat: {
                package { " elinks":
                ensure => installed,}}
            debian, ubuntu: {
                package { " elinks":
                ensure => installed,}}
```

```
            }
    }
Archivo /etc/puppet/modules/nmap/manifests/init.pp:
   class nmap {
    case \$operatingsystem {
        centos, redhat: {
            package { "nmap":
            ensure => installed,}}
        debian, ubuntu: {
            package { "nmap":
            ensure => installed,}}
        }
    }
```

En estos podemos ver la selección del SO y luego las indicaciones del paquete a instalar. Todo esto se podría haber puesto en el archivo inicial (site.pp), pero se recomienda hacerlo así para mejorar la visibilidad/estructura y que puedan aprovecharse los datos para diferentes instalaciones. Sobre el servidor deberíamos ejecutar `puppet apply -v /etc/puppet/manifests/site.pp` y sobre el cliente para actualizar el catálogo (e instalar las aplicaciones) `puppet agent -v -test` (en caso de no hacerlo, el cliente se actualizará a los 30 minutos por defecto). Se podrá verificar sobre el cliente que los paquetes se han instalado e incluso se puede desinstalar desde la línea de comandos y ejecutar el agente que los volverá a instalar.[37]

Como siguiente acción procuraremos crear un usuario y ponerle un *password*. Comenzamos creando un módulo */etc/puppet/modules/accounts/manifests* y dentro de él, dos archivos llamados *init.pp* y *system.pp* con el siguiente contenido:

```
Archivo /etc/puppet/modules/accounts/manifests/system.pp:
    define accounts::system ($comment,$password) {
    user { $title:
       ensure => 'present',
       shell => '/bin/bash',
       managehome => true,}
    }
Archivo /etc/puppet/modules/accounts/manifests/init.pp:
    class accounts {
    file { '/etc/puppet/templates/shadow.sh':
        ensure => file,
        recurse => true,
        mode => "0777",
        source => "puppet:///files/shadow.sh",}
    @accounts::system { 'demo':
       comment      => 'demo users',
       password     => '*',}
    exec { "demo":
          command => 'echo "demo:123456" | chpasswd',
          provider => 'shell',
          onlyif => " /etc/puppet/templates/shadow.sh demo",}
}
```

En el archivo system hemos definido el tipo de accounts::system para asegurar que cada usuario tiene directorio HOME y *shell* (en lugar de los valores predeterminados del comando `useradd`), contraseña y el campo Gecos. Luego en el *init.pp* le indicamos el nombre del usuario a crear y el campo Geco. Para crear este usuario sin *passwd* debemos modificar *client1.pp* para que quede:

```
Archivo client1.pp:
   node 'pucli.nteum.org' {
     #include nmap
     #include elinks
include accounts
    realize (Accounts::System['demo'])
}
```

Como se puede observar, hemos comentado lo que no quiero que se ejecute, ya que el nmap y el elinks se instalaron en la ejecución anterior. Aquí indicamos que se incluya el módulo *accounts* y que se realice la acción. Con esto solo crearía usuarios pero no les modificaría el *password*. Hay diferentes formas de inicializar el *password*, ya que solo se debe ejecutar una vez y cuando no esté inicializado y aquí seguiremos una de ellas, tratada en [37]. Para ello utilizaremos un *script* que deberemos enviar desde el servidor al cliente y por lo cual debemos:

```
Archivo /etc/puppet/fileserver.conf modificar:
    [files]
        path /etc/puppet/files
        allow *
Archivo /etc/puppet/auth.conf incluir:
    path /files
        auth *
Archivo /etc/puppet/puppet.conf en la sección [main] incluir:
        pluginsync = true
Rearrancar el servidor: /etc/init.d/puppetmaster restart
```

Ahora crearemos un script */etc/puppet/files/shadow.sh* que nos permitirá saber si debemos modificar el passwd del */etc/shadow* o no:

```
    #!/bin/bash
    rc=` /bin/grep $1 /etc/shadow | awk -F":" '($2 == " ! ")' | wc -l `
    if [ $rc -eq 0 ]
        then
            exit 1
        else
            exit 0
        fi
```

Ya tenemos las modificaciones en el archivo */etc/puppet/modules/accounts/init.pp* donde le decimos qué archivo hay que transferir y dónde (file) y luego el exec, que permite cambiar el *passwd* si el *script* nos devuelve un 0 o un 1. Luego deberemos ejecutar nuevamente sobre el servidor `puppet apply -v /etc/puppet/manifests/site.pp` y sobre el cliente para actualizar el catálogo `puppet agent -v -test` y verificar si el usuario se ha creado y podemos acceder. Como complemento a Puppet existe el Puppet Dashboard, pero este software está sin mantenimiento, por lo cual no se recomienda instalarlo (a no ser que sea estrictamente necesario -indicaciones en Wiki de Debian*-). En su lugar se puede instalar **Foreman** [39, 40], que es una aplicación que se integra muy bien con Puppet, permite ver en una consola toda la información y seguimiento, así como hacer el despliegue y automatización en forma gráfica de una instalación. La instlación es muy simple [40, 41]:

***https://wiki.debian.org/PuppetDashboard**

```
Inicializamos el repositorio e instalamos el paquete foreman-installer:
        echo "deb http://deb.theforeman.org/ wheezy stable" > \
            /etc/apt/sources.list.d/foreman.list
        wget -q http://deb.theforeman.org/foreman.asc -O- | apt-key add -
        apt-get update
        apt-get install -y foreman-installer
Ejecutamos el instalador con -i (interactive) para verificar los setting:
        foreman-installer -i

Contestamos (y) a la pregunta y seleccionamos las opciones
1. Configure foreman, 2. Configure foreman_proxy, 3. Configure puppet.
Veremos que el proceso puede acabar con unos errores (apache), pero no debemos preocuparnos
y tenemos que reiniciar el servicio con service apache2 restart.

Nos conectamos a la URL https://sysdw.nteum.org/, aceptamos el certificado y entramos
con usuario admin y passwd changeme.
Sobre Infrastructure > Smart Proxy hacemos clic en New Proxy y seleccionamos el nombre
y la URL: https://sysdw.nteum.org:8443.

Lo siguiente es hacer que se ejecute el puppet sobre el servidor (si no está puesto
en marcha se debe poner en marcha service puppet start y puede ser necesario modificar
el archivo /etc/default/puppet y ejecutamos puppet agent -t.

Sobre Foreman veremos que el DashBoard cambia y actualiza los valores para la gestión
integrada con puppet.
```

3.4. Chef

Chef es otra de la grandes herramientas de configuración con funcionalidades similares a Puppet (existe un buen artículo donde realiza una comparación sobre el dilema de Puppet o Chef [42]). Esta herramienta utiliza un lenguaje (basado en Ruby) DSL (*domain-specific language*) para escribir las 'recetas' de configuración que serán utilizadas para configurar y administrar los servidores de la compañía/institución y que además puede integrarse con diferentes plataformas (como Rackspace, Amazon EC2, Google y Microsoft Azure) para, automáticamente, gestionar la provisión de recursos de nuevas máquinas. El administrador comienza escribiendo las 'recetas' que describen cómo maneja Chef las aplicaciones del servidor (tales como Apache, MySQL, or Hadoop) y cómo serán configuradas indicando qué paquetes deberán ser instalados, los servicios que deberán ejecutarse y los archivos que deberán modificarse informando además de todo ello en el servidor para que el administrador tenga el control del despliegue. Chef puede ejecutarse en modo cliente/servidor o en modo *standalone* (llamado 'chef-solo'). En el modo C/S, el cliente envía diversos atributos al servidor y el servidor utiliza la plataforma `Solr` para indexar estos y proveer la API correspondiente, y utiliza estos atributos para configurar el nodo. 'chef-solo' permite utilizar las recetas en nodos que no tengan acceso al servidor y solo necesita 'su' receta (y sus dependencias) que deben estar en el disco físico del nodo (chef-solo es una versión con funcionalidad limitada de chef-client). Chef junto con Puppet, CFEngine, Bcfg2 es una de las herramientas más utilizadas para este tipo de funcionalidad y que ya forma parte de las media-grandes instalaciones actuales de GNU/Linux (es interesante ver los detalles del despliegue de la infraestructura Wikipedia que, excepto *passwords* y certificados, todo está documentado en http://blog.wikimedia.org/2011/09/19/ever-wondered-how-the-wikimedia-servers-are-configured/).

La instalación básica de Chef-server y Chef-client puede ser un poco más complicada, pero comenzamos igualmente con máquinas que tengan el */etc/hosts* y *hostname* bien configuradas. En este caso sugerimos trabajar con máquinas virtuales Ubuntu (14.04 a ser posible), ya que existen dependencias de la librería Libc (necesita la versión 2.15) y ruby (1.9.1) que en Debian Wheezy generan problemas (incluso utilizando el repositorio experimental). El primer paso es bajarnos los dos paquetes de *http://www.getchef.com/chef/install/* y ejecutar en cada uno de ellos `dpkg -i paquete-correspondiente.deb` (es decir, el servidor en la máquina servidora y el cliente en la máquina cliente). Después de cierto tiempo en el servidor podemos ejecutar `chef-server-ctl reconfigure`, que reconfigurará toda la herramienta y creará los certificados. Cuando termine podremos conectarnos a la URL *https://sysdw.nteum.org/* con usuario y *password* admin/pssw0rd1.

Sobre la interfaz gráfica, ir a *Client* > *Create* y crear un cliente con el nombre deseado (pucli en nuestro caso) y marcando en la casilla Admin y hacer *Create Client*. Sobre la siguiente pantalla se generarán las llaves privadas y pública para este cliente y deberemos copiar y salvar la llave privada en un archivo (por ejemplo */root/pucli.pem*). Sobre el cliente deberemos crear un directorio */root/.chef* y desde el servidor le copiamos la llave privada `scp /root/pucli.pem pucli:/root/.chef/pucli.pem`. Sobre el cliente hacemos `knife configure` y los datos deberán quedar como:

```
log_level                :info
log_location             STDOUT
node_name                'pucli'
client_key               '/root/.chef/pucli.pem'
validation_client_name   'chef-validator'
validation_key           '/etc/chef/validation.pem'
chef_server_url          'https://sysdw.nteum.org'
cache_type               'BasicFile'
cache_options( :path => '/root/.chef/checksums' )
cookbook_path ['/root/chef-repo/cookbooks' ]
```

Luego podremos ejecutar `knife client list` y nos deberá mostrar los clientes, algo como:

```
chef-validator
chef-webui
pucli
```

A partir de este punto estamos en condiciones de crear nuestra primera receta (*cookbook*) pero dada la complejidad de trabajar con este paquete y los conocimientos necesarios, recomendamos comenzar trabajando en el cliente como 'chef-solo' para automatizar la ejecución de tareas y pasar luego a describir las tareas del servidor y ejecutarlas remotamente. Para ello recomendamos seguir la guía en http://gettingstartedwithchef.com/.[44, 43, 45], que si bien se deben hacer algunos cambios en cuanto a los *cookbook* que se deben instalar, la secuencia es correcta y permite aprender cómo trabajar y repetir los

resultados en tantos servidores como sea necesario (se recomienda hacer el procedimiento en uno y luego sobre otra máquina sin ningún paquete, instalar el chef-solo, copiar el repositorio del primero y ejecutar el chef-solo para tener otro servidor instalado exactamente igual que el primero.

3.5. Vagrant

Esta herramienta es útil en entornos DevOps y juega un papel diferente al de Docker pero orientado hacia los mismos objetivos: proporcionar entornos fáciles de configurar, reproducibles y portátiles con un único flujo de trabajo que ayudará a maximizar la productividad y la flexibilidad en el desarrollo de aplicaciones/servicios. Puede aprovisionar máquinas de diferentes proveedores (VirtualBox, VMware, AWS, u otros) utilizando *scripts*, Chef o Puppet para instalar y configurar automáticamente el software de la VM. Para los desarrolladores Vagrant aislará las dependencias y configuraciones dentro de un único entorno disponible, consistente, sin sacrificar ninguna de las herramientas que el desarrollador utiliza habitualmente y teniendo en cuenta un archivo, llamado *Vagrantfile*, el resto de desarrolladores tendrá el mismo entorno aun cuando trabajen desde otros SO o entornos, logrando que todos los miembros de un equipo estén ejecutando código en el mismo entorno y con las mismas dependencias. Además, en el ámbito IT permite tener entornos desechables con un flujo de trabajo coherente para desarrollar y probar scripts de administración de la infraestructura ya que rápidamente se pueden hacer pruebas de scripts, 'cookbooks' de Chef, módulos de Puppets y otros que utilizan la virtualización local, tal como VirtualBox o VMware. Luego, con la misma configuración, puede probar estos *scripts* en el *cloud* (p. ej., AWS o Rackspace) con el mismo flujo de trabajo.

En primer lugar es necesario indicar que deberemos trabajar sobre un sistema Gnu/Linux base (lo que se define como *Bare-Metal*, es decir, sin virtualizar, ya que necesitaremos las extensiones de HW visibles y si hemos virtualizado con Virtualbox, por ejemplo, esto no es posible). Es recomendable instalar la versión de Virtualbox (puede ser la del repositorio Debian), verificar que todo funciona, y luego descargar la última versión de Vagrant desde *http://www.vagrantup.com/downloads.html* e instalarla con la instrucción `dpkg -i vagrant_x.y.z_x86_64.deb` (donde x.y.z será la versión que hemos descargado).

A partir de este punto es muy simple ejecutando [46] la instrucción `vagrant init hashicorp/precise32`, que inicializará/generará un archivo llamado **Vagrantfile** con las definiciones de la VM y cuando ejecutemos **vagrant up** descargará del repositorio *cloud* de Vagrant una imagen de Ubuntu 12.04-32b y la pondrá en marcha. Para acceder a ella simplemente debemos hacer `vagrant ssh` y si queremos desecharla, `vagrant destroy`. En el *cloud* de Vagrant [47] podemos acceder a diferentes imágenes preconfiguradas* que po-

***https://vagrantcloud.com /discover/featured**

dremos cargar simplemente haciendo `vagrant box add nombre`, por ejemplo `vagrant box add chef/centos-6.5`. Cuando se ejecute el comando 'up' veremos que nos da una serie de mensajes, entre los cuales nos indicará el puerto para conectarse a esa máquina virtual directamente (p.ej., `ssh vagrant@localhost -p 2222` y con *passwd* **vagrant**). Para utilizar una VM como base, podemos modificar el archivo Vagrantfile y cambiar el contenido:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
end
```

Si deseamos trabajar con dos máquinas en forma simultánea deberemos modificar el archivo Vagrantfile con lo siguiente:

```
Vagrant.configure("2") do |config|
  config.vm.define "centos" do |centos|
    centos.vm.box = "chef/centos-6.5"
  end
  config.vm.define "ubu" do |ubu|
    ubu.vm.box = "hashicorp/precise32"
  end
end
```

Podremos ver que les asigna un puerto SSH a cada una para evitar colisiones cuando hacemos el `vagrant up`. Desde la VM podríamos instalar el software como se hace habitualmente, pero para evitar que cada persona haga lo mismo existe una forma de aprovisionamiento que se ejecutará cuando se haga el 'up' de la VM. Para ello, escribimos un *script init.sh* con el siguiente contenido:

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
rm -rf /var/www
ln -fs /tmp /var/www
```

En este script (para no tener problemas de permisos para esta prueba) hemos apuntando el directorio */var/www* a */tmp*. A continuación modificamos el Vagrantfile (solo hemos dejado una VM para acelerar los procesos de carga):

```
Vagrant.configure("2") do |config|
  config.vm.define " ubu" do |ubu|
    ubu.vm.box = "hashicorp/precise32"
    ubu.vm.provision :shell, path: " init.sh"
  end
end
```

Luego deberemos hacer `vagrant reload -provision`. Veremos cómo se aprovisiona y carga Apache y luego, si en entramos en la máquina y creamos un */tmp/index.html* y hacemos `wget 127.0.0.1` (o la ip interna de la

máquina),veremos que accedemos al archivo y lo bajamos (igualmente si hacemos `ps -edaf | grep apache2` veremos que está funcionando). Por último y para verla desde el *host* debemos redireccionar el puerto 80 por ejemplo al 4444. Para ello debemos agregar en el mismo archivo (debajo de donde pone ubu.vm.provision) la línea: `config.vm.network :forwarded_port, host: 4444, guest: 80`. Volvemos a hacer `vagrant reload -provision` y desde el host nos conectamos a la url: *127.0.0.1:4444* y veremos el contenido del index.html.

En estas pruebas de concepto hemos mostrado algunos aspectos interesantes pero es una herramienta muy potente que se debe analizar con cuidado para configurar las opciones necesarias para nuestro entorno, como por ejemplo, aspectos del *Vagrant Share* o cuestiones avanzadas del *Provisioning* que aquí solo hemos tratado superficialmente.[46]

Actividades

1. Instalad y configurad OpenMPI sobre un nodo; compilad y ejecutad el programa `cpi.c` y observad su comportamiento.

2. Instalad y configurad OpenMP; compilad y ejecutad el programa de multiplicación de matrices (https://computing.llnl.gov/tutorials/openMP/exercise.html) en 2 *cores* y obtened pruebas de la mejora en la ejecución.

3. Utilizando dos nodos (puede ser con VirtualBox), Cacti y XYmon y monitorizad su uso.

4. Utilizando Rocks y VirtualBox, instalad dos máquinas para simular un clúster.

5. Instalad Docker y cread 4 entornos diferentes.

6. Instalad Puppet y configurar un máquina cliente.

7. Ídem punto anterior con Chef.

8. Con Vagrant cread dos VM, una con Centos y otra con Ubuntu y aprovisionar la primera con Apache y la segunda con Mysql. Las máquinas se deben poder acceder desde el host y comunicar entre ellas.

Bibliografía

[1] *Beowulf cluster*.
<http://en.wikipedia.org/wiki/Beowulf_cluster>

[2] **S. Pereira** *Building a simple Beowulf cluster with Ubuntu*.
<http://byobu.info/article/Building_a_simple_Beowulf_cluster_with_Ubuntu/>

[3] **Radajewski, J.; Eadline, D.** *Beowulf: Installation and Administration*. TLDP.
<http://www2.ic.uff.br/ vefr/research/clcomp/Beowulf-Installation-and-Administration-HOWTO.html>

[4] **Swendson, K**. *Beowulf HOWTO* (tlpd).
<http://www.tldp.org/HOWTO/Beowulf-HOWTO/>

[5] **Barney, B.** *OpenMP*. Lawrence Livermore National Laboratory.
<https://computing.llnl.gov/tutorials/openMP/>

[6] *OpenMP Exercise*.
<https://computing.llnl.gov/tutorials/openMP/exercise.html>

[7] *MPI 3*.
<http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>

[8] *MPI Examples*.
<http://www.mcs.anl.gov/research/projects/mpi/usingmpi/examples/main.htm>(Download Section)

[9] *Mpich1 Project*.
<http://www.mcs.anl.gov/research/projects/mpi/>

[10] *MPICH, High-Performance Portable MPI* .
<http://www.mpich.org/>

[11] *LAM/MPI*.
<http://www.lam-mpi.org/ >

[12] *OpenMPI*.
<http://www.open-mpi.org/>

[13] *FAQ OpenMPI*.
<http://www.open-mpi.org/faq/>

[14] **Woodman, L.** *Setting up a Beowulf cluster Using Open MPI on Linux*.
<http://techtinkering.com/articles/?id=32>

[15] *Rocks Cluster. Base Users Guide*.
<http://central6.rocksclusters.org/roll-documentation/base/6.1.1/>

[16] *Cacti*.
<http://www.cacti.net/>

[17] *Xymon*.
<http://xymon.sourceforge.net/>

[18] *Cloud Computing. Retos y Oportunidades*.
<http://www.ontsi.red.es/ontsi/sites/default/files/2-_resumen_ejecutivo_cloud_computing_vf.pdf>

[19] *Eucalyptus, CloudStack, OpenStack and OpenNebula: A Tale of Two Cloud Models*.
<http://opennebula.org/eucalyptus-cloudstack-openstack-and-opennebula-a-tale-of-two-cloud-models/>

[20] *OpenNebula vs. OpenStack: User Needs vs. Vendor Driven*.
<http://opennebula.org/opennebula-vs-openstack-user-needs-vs-vendor-driven/>

[21] *OpenNebula Documentation*.
<http://opennebula.org/documentation/>

[22] *Quickstart: OpenNebula on Ubuntu 14.04 and KVM*.
<http://docs.opennebula.org/4.6/design_and_installation/quick_starts/qs_ubuntu_kvm.html>

[23] *LXC*.
<https://wiki.debian.org/LXC>

[24] *LXC en Ubuntu*.
<https://help.ubuntu.com/lts/serverguide/lxc.html>

[25] **S. Graber***LXC 1.0*.
<https://wiki.debian.org/BridgeNetworkConnections>

[26] *LXC: Step-by-Step Guide*.
<https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series/>

[27] *A Short List of DevOps Tools*.
<http://newrelic.com/devops/toolset>

[28] *Preseeding d-i en Debian*.
<https://wiki.debian.org/DebianInstaller/Preseed>

[29] *FAI (Fully Automatic Installation) for Debian GNU/Linux*.
<https://wiki.debian.org/FAI>

[30] *FAI (Fully Automatic Installation) project and documentation*.
<http://fai-project.org/>

[31] *El libro del administrador de Debian. Instalación automatizada*.
<http://debian-handbook.info/browse/es-ES/stable/sect.automated-installation.html>

[32] *Octopussy. Open Source Log Management Solution*.
<http://8pussy.org/>

[33] *Understanding Docker*.
<https://docs.docker.com/introduction/understanding-docker/>

[34] *Docker Docs*.
<https://docs.docker.com/>

[35] *Docker HUB*.
<https://registry.hub.docker.com/>

[36] *Puppet Labs Documentation*.
<http://docs.puppetlabs.com/>

[37] *Puppet - Configuration Management Tool*.
<http://puppet-cmt.blogspot.com.es/>

[38] *Learning Puppet*.
<http://docs.puppetlabs.com/learning/ral.html>

[39] *Foreman - A complete lifecycle management tool*.
<http://theforeman.org/>

[40] *Foreman - Quick Start Guide*.
<http://theforeman.org/manuals/1.5/quickstart_guide.html>

[41] *How to Install The Foreman and a Puppet Master on Debian Wheezy*.
<http://midactstech.blogspot.com.es/2014/02/PuppetMasterForeman-Wheezy.html>

[42] *Puppet or Chef: The configuration management dilemma*.
<http://www.infoworld.com/d/data-center/puppet-or-chef-the-configuration-management-dilemma-215279?page=0,0>

[43] *Download Chef: Server and Client*.
<http://www.getchef.com/chef/install/>

[44] **A. Gale***Getting started with Chef*.
<http://gettingstartedwithchef.com/>

[45] *Chef Cookbooks*.
<https://supermarket.getchef.com/cookbooks-directory>

[46] *Vagrant: Getting Started*.
<http://docs.vagrantup.com/v2/getting-started/index.html >

[47] *Vagrant Cloud*.
<https://vagrantcloud.com/>

[48] *KVM*.
<https://wiki.debian.org/es/KVM>

[49] *VirtualBox*.
<https://wiki.debian.org/VirtualBox>

[50] *Guía rápida de instalación de Xen*.
<http://wiki.debian.org/Xen>

[51] *The Xen hypervisor*.
<http://www.xen.org/>

[52] *Qemu*.
<http://wiki.qemu.org/Main_Page>

[53] *QEMU. Guía rápida de instalación e integración con KVM y KQemu*.
<http://wiki.debian.org/QEMU>

View publication stats