Revista Cubana de Ciencias Informáticas (RCCI)
Vol. 6, No. 2, abril-junio, 2012
ISSN: 1994-1536 | e-ISSN: 2227-1899
RNPS: 0547 | e-RNPS: 2301
http://rcci.uci.cu

Tipo de artículo: Artículo de revisión
Temática: Software libre
Recibido: 7/06/2012 | Aceptado : 4/09/2012

Estado actual de los sistemas de construcción de paquetes en diferentes distribuciones de GNU/Linux

Current Status of the Build Systems of Packages in Different Distributions of GNU/Linux

Dariem Pérez Herrera[1*], Sonia Guerrero Lambert[2], Yusleydi Fernández del Monte[2], Miguel Albalat Águila[2], Jorge Luis Machín[2], Héctor Pérez Baranda[3], Ricardo Quevedo Mejías[3]

[1] Centro de Software Libre (CESOL). Departamento de Sistemas Operativos. Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, km 2 1⁄2, Torrens, Boyeros, La Habana, Cuba. CP: 19370. Correo-e: dariemp@uci.cu

[2] Centro de Software Libre (CESOL). Departamento de Sistemas Operativos. Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, km 2 1⁄2, Torrens, Boyeros, La Habana, Cuba. CP: 19370. Correo-e: {sguerrero,ydelmonte,malbalat,jlmachin}@uci.cu

[3] Correo-e: {hbaranda12,rquevedo12}@graduados.uci.cu

[*] Autor para correspondencia.

Resumen: En este trabajo se analizan los varios sistemas y herramientas de construcción de repositorios de paquetes usados por las distintas distribuciones de GNU/Linux con el objetivo de estudiar características que puedan ser útiles dada la necesidad de contar con un sistema propio. Para ello se dividen estos en dos grupos: las herramientas para la construcción de paquetes individuales y los sistemas de construcción de repositorios de paquetes. El primer grupo se estudia porque forman parte de los componentes del segundo grupo. Las herramientas y sistemas que se analizan son los utilizados por las distribuciones Fedora, openSUSE, Debian y Ubuntu por contar estas con ciclos de desarrollo estables y estar entre las más difundidas en el ámbito de los sistemas operativos libres. Las herramientas estudiadas en el primer grupo son: Mock, sbuild y pbuilder. Los sistemas de construcción de paquetes estudiados y agrupados en el segundo grupo son: Koji, Open Build Service, buildd y Soyuz. De cada sistema se estudia su arquitectura, componentes, lenguajes de programación empleados, seguridad y características distintivas. Se concluye haciendo una selección de las mejores características presentes en estos sistemas que pudieran imitarse, proponiendo

componentes para su reutilización, así como algunos aspectos que no son implementados en los sistemas estudiados pero que son de gran interés para el sistema propio que se desea desarrollar.
Palabras Claves: Linux, compilación, paquete, bootstrap, sistema distribuido

Abstract: In this paper, the various systems and tools for building package repositories used by the different distributions of GNU/Linux are analyzed, with the aim of studying features that may be useful given the need for an own system. This will divide these into two groups: the tools for the construction of individual packages and the build systems of package repositories. The first group was studied because they are part of the components of the second group. The tools and systems that are discussed are those used by the distributions Fedora, openSUSE, Debian and Ubuntu have these stable development cycles and be among the most widespread in the field of free operating systems. The tools studied in the first group are: Mock, sbuild and pbuilder. The build systems of packages studied and grouped in the second group are: Koji, Open Build Service, buildd and Soyuz. From each system are studied the architecture, components, programming languages used, security and distinctive features. Closure is met by selecting the best characteristics these systems present that can be imitated, proposing components to be re-used and some other aspects not implemented in studied systems but which are of great interest for the new own system to be developed.
Keywords: Linux, build, package, bootstrap, distributed system.

1. Introducción

El proceso de desarrollo de una distribución de GNU/Linux está intrínsecamente ligado a los objetivos que se persiguen con esta a corto, mediano y largo plazo. Se plantea el desarrollo de una distribución en la cual la soberanía tecnológica es uno de ellos, un concepto muy amplio y con varias aristas que deben ser cubiertas progresivamente. Dentro de las aristas a cubrir se encuentran algunos aspectos fundamentales como son la capacidad de adaptar las aplicaciones que se obtienen de la comunidad de software libre internacional a las necesidades nacionales y el fortalecimiento progresivo de un elemento muy importante: la seguridad informática.

Para satisfacer las necesidades de socio-adaptabilidad, es necesario poder modificar el código fuente y compilarlo para generar versiones modificadas de las aplicaciones que se ajusten a la realidad nacional. El aspecto de la seguridad informática se puede abordar desde diferentes direcciones. Una de estas es la utilización de herramientas automáticas de detección de vulnerabilidades que revisen el código fuente disponible de todos los componentes y aplicaciones, y que reporten cualquier problema que encuentren. Esto no es muy provechoso si no se puede certificar

que los archivos binarios que se publican para la distribución, fueron generados efectivamente a partir de ese código fuente. La única forma de asegurar que los archivos binarios que integran la distribución son aceptablemente confiables es compilando todo el código fuente a partir del cual se generan.

Por otro lado, poder compilar todo el código fuente que integra a una distribución tiene otra ventaja muy importante: la posibilidad de optimizar los elementos que la componen, tanto a nivel del código binarios de ejecutables y bibliotecas como al nivel de compresión con que se empaqueten. Teniendo en cuenta que los componentes y aplicaciones de una distribución de GNU/Linux por lo general se publican en la red, esto contribuye al uso óptimo del espacio de almacenamiento y del ancho de banda, algo que también influye en la socio-adaptabilidad. Construir todo el repositorio de paquetes a partir de código fuente no es una tarea que se puede realizar manualmente, por lo que se requiere de un sistema de construcción de paquetes que satisfaga las características particulares del proceso de desarrollo. Se desea tener control total sobre dicho sistema para dirigir a voluntad su evolución futura, por lo que se plantea el desarrollo de un sistema propio. Antes de emprender el desarrollo del nuevo sistema de construcción de paquetes, se hace necesario conocer las características de los ya existentes. El objetivo de este trabajo es realizar un estudio de las distintas herramientas y sistemas de construcción de paquete con la intención de identificar características útiles a tener en cuenta para la implementación de un sistema propio.

2. Desarrollo

Conceptos Fundamentales

Antes de describir las características fundamentales de los distintos sistemas de construcción de paquetes, es necesario esclarecer algunos conceptos que se manejan en el ámbito de las distribuciones de GNU/Linux:

Paquete: Archivo comprimido utilizando algún algoritmo de compresión conocido, que puede contener los elementos que integran una aplicación u otros componentes de una distribución de GNU/Linux (Fernandez-Sanguino, 2011a).

Paquete binario: Es un *paquete* en el cuál los componentes que contiene están listos para instalarse en el sistema operativo. En el caso de las aplicaciones desarrolladas en lenguajes que requieren ser compilados, el paquete contiene los archivos en algún formato binario compatible con el cargador de GNU/Linux, generalmente en formato ELF, además de otros archivos auxiliares como pudieran ser archivos de imágenes o de configuración. En un paquete binario los archivos a instalarse están organizados siguiendo la misma estructura con que serán instalados en el sistema de archivos del sistema operativo. En el caso de las distribuciones basadas en la distribución Debian

GNU/Linux – como la que da origen a este trabajo –, se refiere a archivos que tienen como extensión *.deb* o *.udeb*, y que cumplen con el estándar definido por los desarrolladores del mencionado proyecto (Fernandez-Sanguino, 2011a). Otras distribuciones usan paquetes con extensión *.rpm* .

Paquete fuente: Se conoce como *paquete fuente* o *paquete de código fuente* a un *paquete* que contiene los elementos necesarios para construir a un paquete binario. En el caso de los paquetes fuentes que contienen aplicaciones de código abierto o libres, estos contienen el código fuente de las mismas, así como el resto de los recursos que estas puedan requerir para su compilación, incluyendo imágenes, archivos de configuración, scripts para la creación de bases de datos.... En el caso de las distribuciones basadas en Debian GNU/Linux, los paquetes fuentes conforman un grupo de archivos: un archivo es el que trae empaquetado el código fuente original desarrollado por el autor original del programa o aplicación; otro contiene las modificaciones o parches[1] del mantenedor o desarrollador de Debian. Este segundo paquete contiene los datos y scripts con las reglas requeridas por las herramientas de empaquetado, construcción e instalación de paquetes. Además se provee un archivos de descripción que contiene información sobre el conjunto de paquetes fuentes, así como firmas de verificación de integridad de los mismos, datos del mantenedor, para qué versión de la distribución de GNU/Linux están empaquetados y demás. Las extensiones que por lo general tienen estos paquetes son las siguientes: el paquete con el código fuente original tiene como extensión *.orig.tar.gz* o *.orig.tar.bz2*; el archivo con las modificaciones del desarrollador de la distribución de GNU/Linux se provee con las extensiones *.diff.gz* o *.diff.bz2* para las versiones antiguas del estándar de empaquetado, y *.debian.tar.gz* o *.debian.tar.bz2* para la versión moderna. El archivo de descripción del *paquete fuente* lleva como extensión *.dsc* (Fernandez-Sanguino, 2011a).

Dependencias de paquetes: Las dependencias de un paquete, son aquellos *paquetes binarios* de los que este depende para poder funcionar en determinada plataforma de hardware y software. Existen dos tipos de dependencias, las de tiempo de construcción (build dependency) y las de tiempo de ejecución (runtime dependency). Las dependencias de construcción son aquellos paquetes binarios que se requiere que estén instalados para poder llevar a cabo el proceso de construcción de un paquete binario a partir de un paquete fuente. Por ejemplo, el paquete que contiene al compilador un lenguaje con que esté desarrollada determinada aplicación sería una dependencias de construcción. Lo mismo pasaría con los paquetes que contienen, por ejemplo, a los archivos de encabezado (.h) del kernel Linux. Por otro lado, las dependencias de tiempo de ejecución son aquellos paquetes que se requieren que estén instalados en el

[1]Parche (del inglés *patch*): Archivo de textos que contiene las diferencias entre uno archivo original y uno modificado. Se obtiene con la herramienta *diff* y se aplica con la herramienta *patch*. En el mundo del software libre se utilizan para distribuir modificaciones a los programas sin "contaminar" el código fuente original.

sistema operativo para que determinada aplicación, una vez instalada, pueda ejecutarse correctamente. Por ejemplo, los paquetes que instalan bibliotecas dinámicas – objetos compartidos o *shared objects* (.so) – son un ejemplo de los mismos, pero también pudiera ser una paquete que instale un tema de iconos que necesite determinada aplicación para cargar en su interfaz de usuario (Fernandez-Sanguino, 2011a). Un sistema de construcción de aplicaciones o componentes de software sólo trabaja con dependencias de construcción o compilación.

Repositorio de paquetes: Es una forma de organizar un conjunto de paquetes a través de una estructura de archivos que faciliten su localización y recuperación. Por lo general un repositorio está organizado en una estructura de índice alfabético, donde los paquetes cuyos nombres comienzan por una misma letra se encuentran en un mismo directorio, a la vez que existen archivos de índices que especifican en qué directorio se encuentra cada paquete así como los datos de los mismos. Los archivos fundamentales que identifican a un repositorio por lo general se encuentran firmados usando algoritmos de cifrado como PGP. Un repositorio puede estar alojado de forma local o remota, en este último caso por lo general se puede acceder a través de los protocolos HTTP o FTP (Fernandez-Sanguino, 2011b).

Construcción de un paquete: Proceso mediante el cual un paquete fuente es procesado por un conjunto de herramientas que, siguiendo un conjunto de algoritmos definidos por el conjunto de reglas y datos establecidos por el desarrollador que empaquetó el código fuente, generan un paquete binario (Fernandez-Sanguino, 2011a). La construcción de un paquete no siempre lleva implícito un proceso de compilación, pues a veces un paquete fuente sólo contiene datos (imágenes, tipos de letras, archivos de configuración, scripts...), sin embargo, es muy común usar los términos construcción y compilación indistintamente al referirse a este proceso.

Bootstrap o Bootstrapping: Esta expresión del inglés, en este contexto, carece de una traducción literal al idioma español. En el campo de la computación se refiere al proceso de generar un sistema o software básico – como un compilador o un sistema operativo – desde cero utilizando herramientas previamente existentes. Una analogía pudiera ser el popular problema del huevo y la gallina (¿quién vino primero al mundo?). Un enunciado similar para el caso de los sistemas operativos sería: "Se necesita construir un sistema operativo, pero este trabajo depende de contar con ese sistema operativo". Para el caso de los compiladores ocurre algo similar: "Si los compiladores se programan en C, ¿cómo se programa un compilador de C o cómo se programó el primer compilador de C?". La respuesta a este tipo de problemas son los procesos de bootstrapping. Esto no es más que una forma de nombrar o designar a cualquier proceso para crear un software básico en el seno o con la ayuda de otro software básico preexistente. Aunque en el caso de los sistemas operativos bootstrap se puede refererir simplemente al proceso de cargarlos en memoria y arrancarlos (conocido generalmente como "boot"), también se usa para designar al proceso de armar o construir al

sistema operativo en sí. En el caso de una distribución de GNU/Linux, construirla desde cero requiere iniciar sesión en un sistema operativo previamente existente, descomprimir los paquetes más básicos pertenecientes a la nueva distribución para tener un sistema operativo mínimo funcional en el directorio que desempeñará la función de la raíz del sistema de archivos del futuro sistema operativo, y una vez configurado correctamente, arrancar (hacerle "boot") a este nuevo sistema operativo (Farlex, 2012) (Wookey, 2012).

Construcción avanzada de paquetes individuales

Para entender sistemas más complejos de construcción de paquetes es preciso realizar un acercamiento a las herramientas existentes para la construcción de paquetes individuales pues esta tarea constituye el caso base del proceso en el que se centra este trabajo. Existen varias herramientas que posibilitan la construcción de paquetes individuales, pero las más avanzadas son aquellas que instalan las dependencias de construcción de forma automática y lo hacen sin "contaminar" el sistema operativo del usuario. Para lograrlo utilizan un chroot – una raíz alternativa del sistema de archivos – en la cual se realiza un bootstrap para crear un sistema operativo base a partir del repositorio, cambiando posteriormente la raíz del sistema de archivos del proceso que está realizando la construcción del paquete con la llamada al sistema chroot[2] antes de mandar a construir el paquete (instalando previamente sus dependencias). Este trabajo se centrará en herramientas que cumplen con esta buena práctica.

Mock

Es la herramienta que utilizan en Fedora. Está escrito en Python y utiliza a Yum[3] para poblar al chroot con un sistema operativo base mínimo. Mock no corre como root, pero usa un script con permisos setuid root que le permite elevar privilegios cuando requiere hacer llamadas al sistema privilegiadas como por ejemplo, chroot. Para que un usuario pueda usar esta herramienta, necesita pertenecer al grupo mock, creado durante la instalación de la herramienta con este fin. Es bien sabido que cualquier usuario perteneciente al grupo mock, puede escapar del chroot con privilegios de superusuario, los cuales adquiere a través de la elevación de privilegios que le confiere el permiso setuid root del script mencionado anteriormente. Por eso se recomienda sólo darle permisos para usar Mock a usuarios confiables. Mock brinda la opción de actualizar el chroot con que esté trabajando de una versión de la distribución anterior hacia una nueva a través del argumento "--update" que ejecuta a la función "update" de Yum (Red Hat, 2012b).

[2]En este caso, **chroot** se refiere tanto a la llamada al sistema que cambia la raíz del sistema de archivos de un proceso, como a la misma nueva raíz, la cual sería un directorio dentro del sistema de archivos original.
[3]Gestor de paquetes usado por Fedora y otras distribuciones basadas en paquetería .rpm .

Sbuild

Esta alternativa es la usada para la construcción automática de paquetes en Debian, fundamentalmente a través de una sistema distribuido de construcción de paquetes, aunque también se puede usar por un desarrollador para construir paquetes individuales (Regalli, 2012); sbuild usa a schroot para mantener distintos chroots con sus respectivas configuraciones donde el usuario puede mandar a construir un paquete para distintas variantes de Debian (Hesling, 2012). Con schroot se pueden ejecutar comandos en los distintos chroots que mantiene dada cada una de esas configuraciones, definidas en */etc/schroot/schroot.conf*.

Normalmente la llamada al sistema chroot requiere de privilegios administrativos, pero schroot permite definir grupos de usuarios que pueden tener acceso a determinado chroot. Esto lo logra a través de PAM (Pluggable Authentication Modules), el sistema de autenticación y autorización de GNU/Linux, que provee una interfaz que las aplicaciones pueden utilizar para asignar privilegios a través de módulos que los administradores pueden configurar de manera muy flexible (Morgan, 2004). Gracias a PAM, schroot logra aumentar los privilegios de los usuarios seleccionados para que estos logren hacer la llamada chroot del sistema. De esta forma, es posible construir en un chroot "sin tener privilegios de superusuario" por lo que un sistema automatizado de construcción de paquetes basado en sbuild no necesitaría dichos privilegios de forma explícita, sin embargo, el hecho de que internamente schroot requiera elevar los privilegios del proceso para cambiar la raíz del sistema operativo, implica que de forma implícita sí se necesitan estos privilegios. Esta escalada de privilegios interna representa un potencial problema de seguridad dado que la misma documentación de la llamada al sistema chroot plantea que no se debe usar como sandbox ya que es muy fácil escapar de él, lo que significa que un atacante con permisos para usar schroot podría escapar de la jaula que este representa, accediendo a la raíz original del sistema de archivos con los privilegios de superusuario adquiridos a través de schroot.

Preparar un chroot con sbuild (y schroot) es un proceso que requiere intervención humana, pues la herramienta no está pensada para ser usada cotidianamente por usuarios de poca experiencia en la construcción de repositorios de paquetes, sino para estar integrada a un sistema distribuido de construcción de paquetes (Leigh, 2008). Esta

herramienta cuenta con la posibilidad de ejecutar "apt-get dist-upgrade"[4] sobre el chroot que se utiliza como sistema base de construcción antes de realizar la construcción de cualquier paquete, permitiendo que el mismo se pueda actualizar partiendo de una versión anterior de la distribución hacia una nueva.

Pbuilder

Personal Package Builder o pbuilder es una herramienta usada por los desarrolladores de Debian y de distribuciones derivadas para la construcción de paquetes individuales en un entorno aislado mínimo dentro de un chroot. Es muy útil para determinar si un paquete se construye correctamente para una determinada distribución de GNU/Linux basada en paquetería .deb, pues al construirse sobre un sistema mínimo generado con debootstrap[5], permite detectar si este se empaquetó correctamente especificando todas las dependencias requeridas, algo que es muy difícil de determinar si se hace sobre un sistema operativo en producción, que puede tener dependencias instaladas previamente producto de la instalación o construcción de otros paquetes y por tanto pasar desapercibido el error provocado por no haber especificado una dependencias en los datos del paquete que se requiere probar. A diferencia de sbuild, pbuilder automatiza todo el proceso de crear el sistema mínimo a través de debootstrap, lo cual facilita mucho su uso por los desarrolladores de distribuciones, pero también podría ser de gran utilidad para la automatización de procesos de construcción de repositorios de paquetes (Uekawa, 2007).

Dado que la operación de chroot requiere privilegios de administración, pbuilder generalmente requiere que el usuario posea alguna forma de ganar privilegios de superusuario, por ejemplo, a través del comando sudo[6], lo cual se le puede especificar en sus archivos de configuración. Existen otras variantes para ejecutar chroot, y por consiguiente debootstrap (que hace uso de chroot) sin privilegios de superusuario, utilizando herramientas como fakeroot y fakechroot que utilizan el mecanismo LD_PRELOAD del cargador de GNU/Linux para insertar bibliotecas en un ejecutable antes de que se cargue cualquier otra y así lograr sobrescribir funciones de la biblioteca libc del sistema operativo que realiza llamadas privilegiadas por otras que no requieren privilegios en un intento por simular la ejecución de estas funciones normalmente privilegiadas en espacio de usuario (Dassen, 2004) (Roszatycki, 2008). A través de las opciones que se le pueden pasar a pbuilder tanto por línea de comandos como en su archivo de configuración, es posible especificar que use fakeroot y fakechroot, pero experimentos llevados a cabo y corroborados

[4]Comando del gestor de paquetes **apt** de Debian y derivadas, que actualiza al sistema operativo a través de un algoritmo inteligente de resolución de dependencias, de una versión anterior a una nueva.
[5]Herramienta que construye una distro basada en paquetería .deb a partir del repositorio de paquetes.
[6]Herramienta por línea de comandos que permite ejecutar a otras con privilegios de superusuario para usuarios específicos dada la configuración que se le haya definido a través del archivo /etc/sudoers.

en consultas a sitios de la comunidad Debian, demuestran que muchos paquetes requieren para su construcción estar en un entorno que provea ciertas condiciones privilegiadas que fakeroot y fakechroot no pueden simular correctamente por lo que estas herramientas son insuficientes para lograr correr pbuilder en modo usuario. Afortunadamente, existe una variante de pbuilder que utiliza las herramientas que provee el proyecto User Mode Linux (UML). Este proyecto provee un sistema operativo GNU/Linux corriendo en modo usuario sobre, valga la redundancia, GNU/Linux. Lo que se provee con User Mode Linux es una especie de sandbox o máquina virtual donde las aplicaciones pueden ejecutarse con todos los privilegios sin peligros de afectar el sistema operativo hospedero (Dike, 2004). La variante pbuilder-user-mode-linux posibilita que el proceso de construcción de un paquete pueda ejecutarse con privilegios de superusuario tal y como si estuviera corriendo directamente en un sistema operativo real, aunque en realidad se esté ejecutando dentro de una aplicación en modo usuario aislada del sistema operativo de la máquina (Uekawa, 2007). Esta variante de pbuilder brinda la posibilidad de hacer "apt-get dist-upgrade" al chroot desde una versión más vieja de la distribución hacia una más nueva.

Sistemas de construcción de paquetes

Los sistemas de construcción de paquetes son herramientas semiautomáticas o completamente automáticas de integración continua que agilizan el desarrollo de software al facilitar el proceso de compilación/construcción de aplicaciones y componentes de software, así como el empaquetado de las mismas. En muchos casos, este tipo de sistemas también es el encargado de ir conformando el repositorio de paquetes de la distribución de GNU/Linux. Estos sistemas por lo general se implementan como sistemas distribuidos, con el objetivo de aprovechar el poder de cómputo de varios equipos que se encuentren integrados a una red.

Koji

Es el sistema de construcción de paquetes en formato *.rpm* de la distribución Fedora, una herramienta distribuida compuesta por varios nodos que se encargan de la construcción de paquetes, y un nodo principal que controla todo el proceso (Red Hat, 2012a). A este nodo principal se conectan los desarrolladores utilizando un cliente para solicitar la construcción de un paquete y obtener información sobre la compilación/construcción. El nodo principal del sistema se encarga de dirigir la construcción de los paquetes para arquitecturas específicas, enviándolos a los nodos correspondientes que soporten dicha arquitectura. La principal vista para interactuar con la herramienta es una aplicación web la cual, en lo fundamental, es de solo lectura. La aplicación web utiliza autenticación SSL y Kerberos, necesitándose en el navegador un certificado SSL en el cual confiar. Koji esta formado por varios componentes:

• *Koji-Hub*: es el centro de todas las operaciones de Koji. Se trata de un servidor XML-RPC, se ejecuta en mod_python en Apache; es pasivo, ya que sólo recibe llamadas XML-RPC y se basa en los demonios de construcción y otros componentes para iniciar la comunicación; es el único componente que tiene acceso directo a la base de datos y es uno de los dos componentes que tienen acceso de escritura al sistema de archivos.

• *Kojid*: es el demonio de construcción que se ejecuta en cada una de las máquinas de compilación. Su responsabilidad primaria es la de procesar las solicitudes entrantes de construcción de forma adecuada. Esencialmente Kojid pide a Koji-Hub por trabajo. Koji también tiene soporte para otras tareas de construcción. La creación de imágenes de instalación es también responsabilidad de Kojid, también utiliza a Mock para construir y crea un buildroot (chroot) nuevo para cada generación, está escrito en Python y se comunica con koji-hub a través de XML-RPC.

• *Koji-Web*: es un grupo de script que corren en mod_python y utiliza el motor de plantillas Cheetah para proporcionar una interfaz web a Koji. Actúa como un cliente de Koji-Hub, ofreciendo una interfaz visual para realizar una cantidad limitada de administración. También expone una gran cantidad de información y proporciona medios para realizar determinadas operaciones, como por ejemplo, cancelar las construcciones de paquetes.

• *Koji-Client*: es un herramienta con interfaz CLI[7] escrito en Python, que permite al usuario consultar la mayor parte de los datos, así como realizar acciones como agregar usuarios e iniciar los pedidos de construcción de paquetes.

• *Kojira*: es un demonio que se encarga de mantener la raíces de construcción (chroots, buildroots) actualizados, siendo responsable por la eliminación de elementos redundantes y por limpiar después de cada operación de construcción.

Open Build Service (OBS)

Es un sistema genérico para construir y distribuir paquetes binarios a partir de código fuente, de una forma automática, consistente y reproducible. Fue desarrollado por el proyecto SUSE, inicialmente para dar servicio a los desarrolladores de las distribuciones openSUSE y SUSE Enterprise. Se pueden liberar paquetes al igual que actualizaciones, extensiones, productos, aplicaciones, así como distribuciones completas, para un amplio rango de sistemas operativos y arquitecturas de hardware (Novell, 2011b). Presenta diversas características para diferentes

[7]Command Line Interface: Interfaz por línea de comandos.

usuarios finales, por ejemplo, las liberaciones de los paquetes de software de openSUSE para la variante Factory de la distribución – lo último que se esté desarrollando – , así como se puede siempre obtener lo último para otras distribuciones, garantizando la disponibilidad a través de los diferentes repositorios espejos existentes alrededor del mundo. Para usuarios empaquetadores de software es de mucha ayuda ya que resuelve las dependencias de los paquetes de forma automática: si un paquete depende de otro, el primero pasará a reconstruirse si su dependencia cambia. Permite el enlace con otros proyectos de forma tal que los parches pueden ser probados sobre paquetes actuales de otros proyecto. Ofrece una interfaz abierta que posibilita a varios clientes y servicios externos, por ejemplo SourceForge, entre otros, interactuar con Open Build Service y usar sus recursos.

Open Build Service está bajo la licencia GPL y puede ser instalado en cualquier máquina que tenga 2GB de espacio en memoria. Por último, presenta una aplicación que brinda la posibilidad de correr una instancia de la plataforma o instalarla en un servidor, aunque la instalación manual también es posible pero es un poco más complicada. OBS está formado por varios componentes (Novell, 2011a):

- *obs-build*: Es el script usado por el servidor y el cliente para hacer el proceso de construcción. Puede construir en un chroot o en un ambiente seguro como XEN o KVM[8].

- *osc*: OBS Service Client es un cliente por línea de comandos usado por los empaquetadores avanzados para la solución de conflictos de integración y construcciones locales.

- *obs-service-source-validator*: Una extensión de OSC para encontrar errores comunes antes de subir un código fuente. Este chequeo es obligatorio para los paquetes oficiales de openSUSE.

- *open-build-service*: Código del servidor que instala una instancia del OBS.

- *obs-sign*: Es un demonio cuya función es firmar los paquetes en el servidor.

A OBS también se le pueden adicionar algunos componentes opcionales:

- *hermes*: es un componente extra opcional para OBS. Es una solución para distribuir eventos de OBS como notificaciones a los usuarios, con varias alternativas disponibles como son correo electrónico, RSS, o twitter.

- *software-o-o*: es un componente extra opcional de OBS para los usuarios finales, que aún no está como un

[8]Soluciones para la virtualización.

módulo para uso de propósito general. No es más que el código del sitio http://software.opensuse.org, que se utiliza para descargar distribuciones de openSUSE y para buscar paquetes en OBS.

Buildd

Es el sistema de construcción que gestiona lo que se conoce como Debian Autobuilder Network, donde se desarrolla el proceso de construcción de los paquetes que integran el repositorio de la distribución Debian GNU/Linux (Hodek, 2011). Incrementa la velocidad de las construcciones de paquetes para todas las arquitecturas soportadas por la distro[9].

Está compuesto por una serie de scripts, escritos en Python y Perl, que han evolucionado con el tiempo, ayudando así en varias tareas a los desarrolladores. Finalmente logró que Debian se convirtiera en una distribución que se mantiene al día con las versiones de sus paquetes en todas las arquitecturas soportadas. Los componentes que integran a esta solución son (Hodek, 2011):

- *wanna-build*: una herramienta que ayuda a coordinar la (re)construcción de paquetes a través de una base de datos que mantiene una lista de paquetes, sus versiones, sus estados y alguna otra información. Hay una base de datos central para cada arquitectura de hardware con esta información, que es alimentada con información extraída de los archivos de datos de los repositorios de Debian.

- *buildd*: un demonio que se ejecuta en los nodos de compilación, el cual periódicamente comprueba la base de datos mantenida por wanna-build y llama a sbuild para construir los paquetes. Después que el archivo de bitácora de construcción es detectado por buildd – lo que quiere decir que ya la construcción del paquete finalizó – este sube el paquete al repositorio apropiado (si no falló la construcción del mismo).

- *sbuild*: es el responsable directo de la compilación de los paquetes en chroots aislados. Se asegura de que todas las dependencias de construcción están instaladas dentro del chroot y luego llama a herramientas estándares de construcción de paquetes de Debian para inicar la construcción. Los archivos de bitácora de compilación son enviados a una base de datos que los almacena.

Actualmente los desarrolladores solo adicionan el paquete a la base de datos para todas las arquitecturas (con el estado Needs-Build). Las máquinas encargadas de construir buscarán los paquetes con este estado y lo tomarán de la lista, la cual está priorizada por: estado de compilaciones previos (out-of-date o uncompiled), y por prioridad del

[9]Forma coloquial de referirse a una distribución de GNU/Linux.

paquete, sección y nombre de paquete, datos que se incluyen en la información interna de cada paquete aportada por el mantenedor del mismo. Además, para prevenir que un paquete se estanque al final de la cola, la prioridad es ajustada dinámicamente según el tiempo de espera en la cola. Si la construcción fue satisfactoria para cada arquitectura, el sistema subirá los paquetes a su repositorio correspondiente, en caso contrario el paquete pasará a un estado especial (Build-Attempted), y el sistema avisa a los responsables.

Soyuz

Soyuz es un componente del sistema de gestión de Launchpad de la distribución de GNU/Linux Ubuntu, que abarca el sistema de construcción de paquetes, y la gestión de paquetes y repositorios. Permite a los usuarios subir los paquetes, compilarlos para las distintas arquitecturas, y luego publicarlos para que otros puedan descargarlos. La construcción es delegada a uno de varios nodos de construcción separados físicamente en distintas máquinas que llevarán a cabo la construcción en un chroot, en el procesador con la arquitectura de hardware solicitada (Edwards, 2011b).

Soyuz usa a Twisted para todo lo que tiene que ver con comunicación por la red. Este es un framework o marco de trabajo para la programación de redes orientado a eventos escrito en Python. Twisted soporta varios protocolos y capas de transporte que las aplicaciones pueden usar de forma relativamente sencilla. En el nodo principal de Soyuz se tiene una base de datos central donde se guarda toda la información relacionada con los procesos que maneja este sistema. Los componentes que integran la parte que se encarga del proceso de construcción de paquetes son (Edwards, 2011a):

- *buildd-manager*: es el demonio que corre en el nivel o capa superior, el cual se responsabiliza de elegir el siguiente elemento de construcción que está en cola en la tabla BuildQueue, y envía todos los archivos necesarios hacia los nodos de compilación, dando inicio al proceso de construcción. También monitorea a los nodos para obtener las últimas líneas del archivo bitácora de la compilación que se mostrará en la página web del servicio. Cuando el proceso de construcción se termina, descarga los archivos resultantes del nodo de construcción involucrado y el paquete es arrojado a un área temporal, para luego ser subido al repo[10].

- *buildd*: no confundir con el buildd de Debian; este es una implementación totalmente hecha por Canonical,

[10]Forma coloquial de referirse a un repositorio de paquetes de una distribución de GNU/Linux.

Ltd.[11] para Soyuz. Este componente realiza la misma función que su homónimo y sbuild juntos: construir paquetes en un chroot en cada uno de los nodos de construcción.

- *proccess-upload*: Es un script ejecutado como una tarea de cron[12] en la máquina máster después que buildd-manager colecta los archivos del proceso de construcción de un paquete desde un nodo de construcción y los pone en un área para paquetes entrantes en dicha máquina. Este script, auxiliándose de varios otros, se encarga de comprobar que el resultado de la construcción del paquete fue totalmente satisfactoria al revisar distintos parámetros establecidos por una política de recepción de paquetes binarios que se deben cumplir correctamente antes de considerar incluir al paquete en un repo. Durante el proceso, varias tablas de la base de datos central son actualizados con los datos extraídos de los archivos que componen el resultado de la compilación.

Soyuz utiliza dos tipos de nodos de construcción, virtuales y no virtuales. Los no virtuales actualmente sólo es utilizado por Ubuntu y el PPA[13] del equipo de seguridad de Ubuntu. Todas las otras construcciones se realizan en las máquinas virtuales. Cuando un paquete fuente se acepta y se crea una compilación para el mismo, varios requisitos se deben cumplir:

- La arquitectura solicitada por el fuente debe de machear con las arquitecturas soportadas por la tabla de la base de datos DistroArchSeries.
- Debe de haber un nodo de construcción disponible para la arquitectura solicitada.
- Debe de haber un fichero chroot (ver más adelante sobre estos ficheros) disponible para la entrada en DistroArchSeries soportada.

Los ficheros chroot mencionados anteriormente se refieren a las imágenes comprimidas de un sistema operativo base que se despliega en los chroots de las máquinas de compilación. Estos deben ser construidos previamente al inicio del proceso de construcción de paquetes destinado a crear una nueva versión de la distribución. Dicho trabajo lo realizan personas a las que se les asigna el rol de Administradores de Buildd. La tarea consiste en hacer un bootstrap de un sistema operativo mínimo y comprimirlo. Cuando comienza un proceso de construcción, los nodos descompactan dicho archivo enviado por buildd-manager y ejecutan la operación chroot hacia dentro del mismo. Cada DistroArchSeries tiene su propio fichero chroot, que es periódicamente actualizado para que los nodos de construcción logren un menor tiempo de arranque, ya que "apt-get dist-upgrade" tiene menos trabajo que hacer al

[11]Empresa que dirige el desarrollo de la distribución Ubuntu.
[12]En sistemas operativos basados en el estándar Posix (Unix-like), es el servicio que se encarga de las tareas programadas.
[13]Personal Package Archive: repositorio personal que puede crear cualquier usuario en Launchpad a donde puede subir sus propios paquetes fuentes para que sean construidos por Soyuz.

inicio de cada construcción.

3. Conclusiones

Del este estudio se puede concluir que los proyectos que desarrollan a las distribuciones de GNU/Linux más importantes utilizan sistemas distribuidos de construcción de paquetes en los que un nodo máster, ajustándose a las características del proceso de desarrollo del proyecto, dirige a varios nodos esclavos que se encargan de la construcción de los paquetes, lo cual posibilita la construcción en paralelo y por tanto la agilidad del proceso. Esto es muy importante dado que un repositorio de paquetes fuentes puede estar conformado por una cantidad enorme de ellos (más de 16 000 en el caso de la distro que da origen a este trabajo). El lenguaje más utilizado es Python, lo cual no es extraño dada la versatilidad, legibilidad y demás características positivas de este lenguaje.

Se pudo comprobar que ninguno de los sistemas estudiados realiza un bootstrapping automático partiendo de una versión anterior de la distribución, sino que personal humano tiene que intervenir en la creación de los chroots iniciales, algo que se pretende automatizar en el sistema propio que se quiere desarrollar. Además se comprobó que existen herramientas que pueden ser reutilizadas en la elaboración del nuevo sistema, siguiendo los ejemplos de las combinaciones Koji/Mock y wanna-build/sbuild. Dado que este trabajo se centra en la paquetería .deb, las herramientas candidatas más apropiadas para integrar los nodos esclavos serían sbuild y pbuilder, siendo pbuilder la que exhibe mejores características pues permite la construcción de paquetes sin requerir privilegios de administración (con la variante pbuilder-user-mode-linux), construye automáticamente los chroots, y brinda la posibilidad de hacer “apt-get dist-upgrade” en estos para actualizar a una versión más reciente de la distribución, una funcionalidad que viene como anillo al dedo para realizar un proceso de bootstraping cuando todavía no existen los binarios de la versión objetivo y se parte de un chroot de una versión anterior de la distribución. Por otro lado, vale la pena estudiar a Twisted, el marco de trabajo que utiliza Soyuz para la comunicación en red, como una posible alternativa para implementar los protocolos de comunicación del futuro sistema distribuido.

Referencias

DASSEN, J.H.M., 2004. fakeroot(1). [en línea]. 6 de agosto de 2004. [Consultado el: 5 de julio de 2012]. Disponible en: http://manpages.debian.net/cgi-bin/man.cgi?sektion=1&query=fakeroot&apropos=0&manpath=sid&locale=en.

DIKE, Jeff, 2004. The User-mode Linux Kernel Home Page. [en línea]. 2004. [Consultado el: 4 de julio de 2012].

Disponible en: http://user-mode-linux.sourceforge.net/.

EDWARDS, Julian, 2011a. Soyuz/TechnicalDetails/Building - Launchpad Development. [en línea]. 12 de septiembre de 2011. [Consultado el: 5 de mayo de 2012]. Disponible en: https://dev.launchpad.net/Soyuz/TechnicalDetails/Building.

EDWARDS, Julian, 2011b. Soyuz/TechnicalDetails - Launchpad Development. [en línea]. 13 de septiembre de 2011. [Consultado el: 5 de mayo de 2012]. Disponible en: https://dev.launchpad.net/Soyuz/TechnicalDetails.

FARLEX, 2012. bootstrap - definition of bootstrap by the Free Online Dictionary, Thesaurus and Encyclopedia. [en línea]. 2012. [Consultado el: 5 de julio de 2012]. Disponible en: http://www.thefreedictionary.com/bootstrap.

FERNANDEZ-SANGUINO, Javier, 2011a. The Debian GNU/Linux FAQ - Basics of the Debian package management system. [en línea]. 27 de agosto de 2011. [Consultado el: 5 de julio de 2012]. Disponible en: http://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html.

FERNANDEZ-SANGUINO, Javier, 2011b. The Debian GNU/Linux FAQ - The Debian FTP archives. [en línea]. 27 de agosto de 2011. [Consultado el: 5 de julio de 2012]. Disponible en: http://www.debian.org/doc/manuals/debian-faq/ch-ftparchives.en.html#s-dirtree.

HESLING, Craig, 2012. Schroot - Debian Wiki. [en línea]. 2 de abril de 2012. [Consultado el: 24 de junio de 2012]. Disponible en: http://wiki.debian.org/Schroot.

HODEK, Roman, 2011. Debian -- Autobuilder network. [en línea]. 2 de octubre de 2011. [Consultado el: 29 de junio de 2012]. Disponible en: http://www.debian.org/devel/buildd/index.en.html.

LEIGH, Roger, 2008. Ubuntu Manpage: sbuild-setup - sbuild setup procedure. [en línea]. 2008. [Consultado el: 4 de julio de 2012]. Disponible en: http://manpages.ubuntu.com/manpages/lucid/man7/sbuild-setup.7.html.

MORGAN, Andrew, 2004. Linux-PAM FAQ. [en línea]. 7 de junio de 2004. [Consultado el: 5 de julio de 2012]. Disponible en: http://www.kernel.org/pub/linux/libs/pam/FAQ.

NOVELL, 2011a. openSUSE:Build Service Tools - openSUSE. [en línea]. 2011. [Consultado el: 28 de junio de 2012]. Disponible en: http://en.opensuse.org/openSUSE:Build_Service_Tools.

NOVELL, 2011b. Portal:Build Service - openSUSE. [en línea]. 2011. [Consultado el: 28 de junio de 2012]. Disponible en: http://en.opensuse.org/openSUSE:Build_Service.

RED HAT, 2012a. Koji - FedoraProject. [en línea]. 11 de abril de 2012. [Consultado el: 6 de mayo de 2012]. Disponible en: http://fedoraproject.org/wiki/Koji.

RED HAT, 2012b. Projects/Mock - FedoraProject. [en línea]. 2012. [Consultado el: 24 de junio de 2012]. Disponible en: http://fedoraproject.org/wiki/Projects/Mock.

REGALLI, Fabrizio, 2012. sbuild - Debian Wiki. [en línea]. 1 de junio de 2012. [Consultado el: 4 de julio de 2012]. Disponible en: http://wiki.debian.org/sbuild.

ROSZATYCKI, Piotr, 2008. fakechroot(1): gives fake chroot environment - Linux man page. [en línea]. 2008.

[Consultado el: 5 de julio de 2012]. Disponible en: http://linux.die.net/man/1/fakechroot.

UEKAWA, Junichi, 2007. pbuilder User's Manual. [en línea]. 27 de mayo de 2007. [Consultado el: 4 de julio de 2012]. Disponible en: http://pbuilder.alioth.debian.org/.

WOOKEY, 2012. DebianBootstrap - Debian Wiki. [en línea]. 15 de mayo de 2012. [Consultado el: 5 de julio de 2012]. Disponible en: http://wiki.debian.org/DebianBootstrap.