RPMs para Devs

martin.uy
Open by default.

Agenda

- Objetivos
- ¿Cómo funcionan los proyectos open source?
- Introducción a los RPMs
- Buildeando RPMs
- Un ejemplo rápido: buildeando, desplegando y debuggeando el kernel de Linux

Descargos

- No soy un experto en RPM o en empaquetamiento, solo un desarrollador
- Esta no es una charla exhaustiva sobre RPM pero más una receta paso a paso
- Esta charla está basada en Fedora Linux
  - Puede ser diferente en otras distribuciones y con otros formatos de paquetes, pero con suerte se podrán encontrar cosas en común

Objetivos

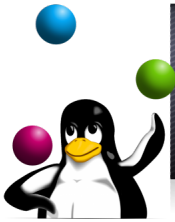
- Instalar un ambiente de desarrollo y debugging de forma rápida para casi cualquier proyecto open source (Linux kernel, NSS, Java, etc.)
  - Builds incrementales
  - Debuggear con símbolos y código fuente
  - Soporte para buildear múltiples proyectos en un mismo ambiente
- Una sola receta para gobernarlos a todos: ocultar las especificidades de cada proyecto al buildear e instalar
- No manchar el ambiente actual instalando dependencias para buildear
  - Y no mezclar paquetes de diferentes versiones!

Proyectos open source

- Hay una comunidad (upstream)
  - Generalmente patrocinada por una compañía o una fundación
  - Gobernada por sus autoridades, estructuras y reglas
  - Hay usualmente un repositorio de código, sistemas de seguimiento de bugs, listas de correo e IRC
- El código fuente está siempre disponible pero no todas las comunidades brindan builds binarios (para todas las arquitecturas y sistemas operativos)
- Las comunidades son abiertas pero algunas requieren firmar acuerdos para aceptar contribuciones mayores

Proyectos open source

THE LINUX FOUNDATION



KVM

redhat.

WildFly

OpenJDK

ORACLE®

Proyectos open source

- Hay distribuciones de Linux (downstream)
  - Patrocinan ingenieros para contribuir a las comunidades de upstream
  - Toman el código fuente de los repositorios de la comunidades de upstream (código fuente “vanilla”) y hacen algunos pocos cambios para buildear, empacar y distribuir
  - ¿Por qué “algunos pocos cambios”?
    - Integrar a su ambiente (ej.: esquema de archivos o directorios, ubicación de los certificados criptográficos, configuración y herramientas para administración, etc.)
    - Aplicar mejoras que no están en upstream aún (o que upstream rechazó por alguna razón)
    - Remover código por problemas de licencias o específico de una plataforma
    - Cambiar parámetros de buildeo
    - Otros

Proyectos open source

- Red Hat tiene una política de “upstream primero”, con algunas excepciones:
  - Acuerdos de Nivel de Servicios (SLAs) pueden requerir entregar más rápido que lo que le lleva a upstream aceptar una contribución
- Aún estas excepciones son open source: puedes obtenerlo desde el repositorio de código fuente de los RPMs, que está disponible públicamente
- Las excepciones tienden a ser mínimas (costo de mantenimiento) y generalmente no modifican APIs (no es un fork)

Proyectos open source

- Para contribuir a un proyecto open source, necesitas generar parches contra el branch master
- Buildear es diferente en cada proyecto:
  - dependencias?
  - ./configure? parámetros?
  - make? cmake?
    - all?
    - build?
    - install?
  - README?
- Necesitas realmente documentación para desarrolladores (o preguntar) e instalar un ambiente adecuado

Introducción a RPMs

- Formato binario para empaquetamiento de código fuente y binarios ejecutables
- Creado por Erik Troan y Marc Ewing (Red Hat), en 1997
- Usado en varias distribuciones de Linux (y unas pocas no-Linux)
- Para múltiples arquitecturas
- Evaluación de dependencias (para buildear o instalar)
- Delta RPMs (para acelerar actualizaciones)
- Firmas para chequeos de integridad

Introducción a RPMs

- Las distribuciones de Linux generalmente brindan paquetes RPM para descargar (desde un gestor de paquetes o HTTP)
  - Paquetes de código fuente (.src.rpm)
  - Paquetes binarios (para cada arquitectura soportada)
  - Paquetes de desarrollo (headers)
  - Paquetes con información de debug (símbolos de debug que son removidos de los binarios construidos, y código fuente)

    La disponibilidad de esta clase de paquetes depende en cada caso. Ej.: un paquete de “desarrollo” no tiene sentido para código interpretado.
- Adicionalmente, Fedora brinda acceso público de lectura a sus repositorios GIT de RPMs
  - En este repositorio es donde ocurren los cambios a los RPMs (mantenedores de paquetes)

Introducción a RPMs

- Descarga de RPMs
  - http://mirror.globo.com/fedora/linux/development/rawhide/Everything/
- Git
  - https://src.fedoraproject.org/rpms/<nombre-del-paquete>.git
- fedpkg (Fedora) es una herramienta útil para trabajar con RPMs y sus repositorios

Introducción a RPMs

- ¿Qué contiene un paquete SRPM?
  - Tarballs de código fuente (vanilla) y un archivo “source” para verificar la integridad
  - Parches RPM
  - Archivo SPEC
    - Receta (parecida a un makefile) para buildear un RPM a partir de un SRPM desempacado
  - Scripts auxiliares dependiendo de cada paquete
    - Ej.: script para buildear el tarball de código fuente desde un repositorio de upstream

Introducción a RPMs

- Archivo SPEC
  - Información del paquete (múltiples paquetes RPM pueden ser generados)
    - Nombre
    - Descripción
    - Licencia
    - Arquitecturas
    - Versión
  - Paquetes de dependencia
  - Parches RPM (diffs de código fuente)

Introducción a RPMs

- Archivo SPEC
  - Etapas
    - %prep → extraer tarballs de código fuente y aplicar parches RPM
    - %build → buildear el código parcheado
    - %install → desplegar a un BUILDROOT (esquema final de directorios y archivos)
    - %clean → hacer limpieza
    - %post → hacer post-procesamiento
    - %check → correr test de humo en binarios buildeados
  - Log de cambios

Buildeando RPMs

- Herramienta fedpkg
  - `fedpkg clone -a <nombre-del-paquete>`
  - `fedpkg switch-branch <branch>`
  - `fedpkg sources`
  - `fedpkg srpm`
- Elegir un branch igual al objetivo de despliegue (ej. f25 para Fedora 25). Esto va a simplificar las dependencias.
- En este punto, el código fuente del paquete RPM ha sido obtenido y un SRPM (con código fuente vanilla adentro) ha sido construido a partir de él.

Buildeando RPMs

- Mock
  - Herramienta para buildear paquetes en un ambiente *chroot*
  - Maneja las dependencias de forma segura y automática
  - Internamente usa “dnf/yum” para obtener las dependencias y la herramienta “rpmbuild” para trabajar con paquetes RPM.
  - Disponible en Fedora y CentOS. Puede ser construido para RHEL.
  - Buildear para múltiples distribuciones y arquitecturas. Ej.: ambiente de configuración para buildear para “Fedora 25 x86_64”.

Buildeando RPMs

- Inicializar un ambiente mock
  - `sudo /usr/sbin/usermod -a -G mock $(whoami)`
  - `mock -r fedora-25-x86_64 --rootdir=<ruta-al-chroot> --init`
- Instalar dependencias de buildeo en un ambiente mock
  - `mock -r fedora-25-x86_64 --rootdir=<ruta-al-chroot> --installdeps <ruta-al-srpm>`
- Mock puede ser utilizado para construir (`mock build`) pero lo vamos a hacer manualmente para tener mayor control.

Buildeando RPMs

- Preparación para buildear un paquete
  - `cd <ruta-al-chroot>/builddir`
  - `mkdir <nombre-del-paquete>_build`
  - `cd <nombre-del-paquete>_build`
  - `mkdir original`
  - `cp <ruta-al-srpm> original`
  - `mock -r fedora-25-x86_64 --rootdir=<ruta-al-chroot> --shell`

Buildeando RPMs

- Preparación para buildear un paquete
  - `mock -r fedora-25-x86_64 --rootdir=<ruta-al-chroot> --shell`
    - `export CURRENT_BUILD_PACKAGE=<nombre-del-paquete>`
    - `rpm --define "_topdir /builddir/${CURRENT_BUILD_PACKAGE}_build" -i /builddir/${CURRENT_BUILD_PACKAGE}_build/original/<archivo-srpm-del-paquete>`
- En este punto, el SRPM ha sido desempacado en el ambiente de buildeo.
- Guardar el archivo SPEC original
  - `cp <ruta-al-chroot>/builddir/<nombre-del-paquete>_build/SPECS/<nombre-del-paquete>.spec <ruta-al-chroot>/builddir/<nombre-del-paquete>_build/SPECS/<nombre-del-paquete>.spec.bak`

Buildeando RPMs

- Para incrementar la velocidad, cualquier directorio de buildeo (`/builddir/${CURRENT_BUILD_PACKAGE}_build/BUILDROOT o BUILD`) puede ser reemplazado por un directorio en *tmpfs* a través de links simbólicos.
  - En lugar de HDD I/O lentos, todo es escrito en memoria
  - Requiere una gran cantidad de espacio disponible en memoria
- Sin embargo, persistir los artefactos de buildeo en el directorio BUILD puede ser interesante para builds incrementales.

Buildeando RPMs

- Preparación para buildear un paquete
  - `mock -r fedora-25-x86_64 --rootdir=<ruta-al-chroot> --shell`
    - `export CURRENT_BUILD_PACKAGE=<nombre-del-paquete>`
    - `rpmbuild --define "_topdir /builddir/${CURRENT_BUILD_PACKAGE}_build" -bp --target=`uname -m` /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}.spec 2> /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}_build_err.log | tee /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}_build_out.log`

Buildeando RPMs

- Preparación para buildear un paquete
  - En este punto, la etapa de preparación (%prep) ha sido ejecutada. El fuente vanilla ha sido desempacado y los parches RPM aplicados sobre él. Este ese el código que va a ser buildeado.
  - Editar el archivo SPEC:
    - Agregar “exit 0” después de la línea “%prep”
    - Buscar cualquier instrucción que remueva o limpie archivos y comentarla. Ej.: “make -s mrproper” en kernel.spec

Buildeando RPMs

- Monitorear cambios en el código fuente (opcional)
  - `cd <ruta-al-chroot>/builddir/<nombre-del-paquete>_build/BUILD/<nombre-del-paquete>`
  - `rm -rf .git`
  - `git init`
  - `git add .`
  - `git commit -m 'dev_baseline_source'`
  - `git tag -a dev_baseline_source -m "dev_baseline_source"`

Buildeando RPMs

- Buildear
  - `mock -r fedora-25-x86_64 --rootdir=<ruta-al-chroot> --shell`
    - `export CURRENT_BUILD_PACKAGE=<nombre-del-paquete>`
    - `rpmbuild --define "_topdir /builddir/${CURRENT_BUILD_PACKAGE}_build" -bb --target=`uname -m` /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}.spec 2> /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}_build_err.log | tee /builddir/${CURRENT_BUILD_PACKAGE}_build/SPECS/${CURRENT_BUILD_PACKAGE}_build_out.log`
  - Los RPMs van a ser escritos en <ruta-al-chroot>/builddir/<nombre-del-paquete>_build/RPMS

Buildeando RPMs

- Builds incrementales
  - Modificar el código fuente y re-ejecutar el comando de buildeo
  - Los objetos que no son afectados por cambios de archivos, no son re-buildeados acelerando todo el proceso.

Ejemplo rápido: kernel

- Antes de ejecutar la etapa “prep”, modificar el archivo kernel.spec:
  - %define buildid .dev (no dejar espacio en blanco antes o después de “%”)
  - Deshabilitar firmado (para x86_64)
    - %global signkernel 0
    - %global signmodules 0
  - En %build:
    - Comentar “make -s mrproper” con un “#” al comienzo
- Agregar las siguientes opciones a los comandos rpmbuild para las etapas “prep” y “build”:
  - --without debuginfo --without debug --without perf --without cross_headers --without headers --without doc --without tools

Ejemplo rápido: kernel

- Antes de ejecutar el comando rpmbuild de la etapa “build”, modificar <ruta-al-chroot>/builddir/kernel_build/BUILD/<kernel>/<kernel-2>/configs/<kernel> (ej.: kernel-4.9.14-x86_64.config)
  - Cambiar:
    - CONFIG_RANDOMIZE_BASE=n
    - CONFIG_RANDOMIZE_MEMORY=n
    - CONFIG_MODULE_SIG=n
    - CONFIG_MODULE_SIG_ALL=n
    - CONFIG_MODULE_SIG_UEFI=n
    - CONFIG_MODULE_SIG_SHA256=n
    - CONFIG_KEXEC_BZIMAGE_VERIFY_SIG=n
    - CONFIG_KEXEC_VERIFY_SIG=n

Ejemplo rápido: kernel

- Extras
  - Eclipse es, en mi experiencia, un buen IDE para desarrollar y debuggear (como front-end de gdbserver) el kernel
    - Debuggear con fuentes es un poco tramposo, debido a las optimizaciones del compilador
  - QEMU es un buen hipervisor para debuggear el kernel. Tiene un stub de gdbserver. He tenido algunos problemas para debuggear las etapas de booteo.
    - Ejecutar la imagen de QEMU con el parámetro “-s” y attacharse con gdb al puerto 1234.

Referencias

- https://github.com/rpm-software-management/mock/wiki