Software Bundles and Linux

Neal Gompa (Conan Kudo [ニール・ゴンパ])

Who am I?

- Professional technologist
- Contributor and package maintainer in the Fedora Project
- Contributor and package maintainer in Mageia Linux
- Contributor to RPM, DNF, and various related projects
- Diligent follower of the telecommunications industry
- Production Engineer at Datto, Inc.

Contact Points:

- Twitter: @Det_Conan_Kudo
- Google+: +NealGompa

Recap on software management in Linux...

Software management in Linux vs other platforms...

Unlike most operating system platforms, Linux is made of several software components made by many different authors that are developed completely independently.

Consequently, Linux distributions came into existence to collect these components and put together a coherent system that can be used by people. Package managers and dependency resolvers were developed to solve this problem.

Packaging Systems on Linux

Most Linux distributions are descended from one of three Linux distribution families:

- Debian: The Debian family uses DPKG as its package system, and uses APT as the dependency resolver.
- Red Hat/Fedora: The Red Hat/Fedora family uses RPM as its package system, and primarily uses either Yum or DNF as the dependency resolver.
- SUSE: The SUSE family started out as a German translation of Slackware, but adopted RPM in the mid-90s to offer superior software management capabilities to attract a wider audience. It primarily uses Zypper as its dependency resolver.

Thus, most Linux distributions use either DPKG or RPM. There are a few other notable systems, but these two are the most common ones.

Overview of RPM and DPKG

RPM:

The RPM Package Manager was created by Red Hat in 1995 to replace their earlier iterations in the first versions of Red Hat Linux.

It uses the RPM format for binaries (and has a variant for storing source packages) and has a framework to support a wide variety of platforms and flexible dependency management.

It is used by Red Hat Enterprise Linux/CentOS, Fedora, SUSE/openSUSE, Mageia, and many others.

Notable front-ends for it are:

- DNF
- Yum
- Zypper
- Smart

DPKG:

The Debian package manager was created by Debian in 1994 for the Debian Linux distribution.

It uses the DEB format for binaries, and is designed as a framework that integrates tightly with various aspects of a Debian system (installation, configuration, etc.)

Debian and Ubuntu (and derivatives) are the primary users of this system.

Notable front-ends for it are:

- APT
- Smart

So these systems work great, right?

In practice, they do work well most of the time. However, there are a few problems:

- Packaging systems, as a general rule, are not interoperable.
  - RPMs on DPKG managed systems and DEBs on RPM managed systems don't generally work, requiring publishing in multiple formats
    - This is not *completely* true, as there have been two independent efforts to provide compatibility: Alien (converts packages between formats) and RPM5 (supports managing multiple packaging systems within a unified architecture). The former is known to be brittle and the latter is not a popular solution, due in part to the personalities in the project and internal incompatibilities with RPM on an API basis despite being a fork of it.
- It's impossible to track and safely handle every possible permutation of package sets that is possible.
  - There are simply too many possible mixes to make safe with a reasonable amount of effort.
- Scripts are evil.
  - Scripts in packages often modify the state of the system in ways that are not easy to identify from the package system's point of view, making it difficult to recover when bad scripts run.
- Distributions simply may not be compatible on a binary interface (ABI) level, breaking the expectation of a common Linux software package that Just Works.

So how do we solve these problems?

Possible solutions for packaging issues

- Collaborate to unify the binary interfaces across Linux distributions
  - To some extent, this is already happening. As it gets harder to keep up with more rapid development of software, the ability for distributions to make themselves “special” continues to be tempered by the need to be able to pull in newer software constantly.
- Collaborate to unify the mechanism for packaging software for Linux distributions
  - This is unlikely to happen. While RPM is the standard format that must be supported in some way (as the Linux Standards Base indicates), there continues to be arguments about details about all systems used for delivering software.
- Reduce the number of Linux distributions
  - This is *never* going to happen, at least not naturally.
- Provide a distribution-agnostic mechanism to install add-on software
  - This has been attempted many times in the past, though now Linux distributions are more keen on allowing it with the maturation of security technologies (SELinux, CGroups, containers, namespacing, etc.)

So what does a distro-agnostic mechanism look like?

A distribution agnostic mechanism to install add-on software has the key burden to try to preserve the security of the system at all costs, as the software is considered untrusted (unlike system packages).

A modern system would do the following:

- Automatically encapsulate the software in a sandbox to confine its access to resources
- Provide a way for it to export capabilities to the system to leverage
- Allow the user to be aware of what the software needs and how it is using it

Two big solutions for offering software bundles (again!)

Flatpak:

- Created by Alexander Larsson from Red Hat
- Supported by all major Linux distributions officially except Ubuntu
- Flatpak creation is distribution-agnostic and can be done from any Linux distribution
- Designed around runtimes
- Confinement uses kernel namespaces, CGroups, seccomp, and containers, which works on all major Linux distributions

Snap:

- Created by Canonical
- Supported officially by Ubuntu, though unofficial support is available for many distributions
- Snap creation via official tools only works through Ubuntu currently
- Originally developed to support Ubuntu Touch phone apps, and designed around fully bundled applications
- Confinement depends on AppArmor patches in the Ubuntu kernel

Technical similarities of the two systems

Flatpak:

- Flatpak bridging is done through mounting the environments of different Flatpaks into one virtual filesystem and containing them together
- Flatpaks support “portals” to bridge the Flatpak to the system for specific capabilities (such as accessing files through system file picker).
- Flatpak Builder uses a single JSON file to declare how to construct them

Snap:

- Snap bridging is done through “plugs” and “slots” that connect different snaps together along a defined interface
- Snaps also support merged filesystems for bridging snaps
- Snaps support “interfaces” to bridge the snap to the system for specific capabilities (such as manipulating network configuration)
- Snapcraft uses a single YAML file to declare how to construct them

Links to resources

- RPM: http://rpm.org/
- RPM5: http://rpm5.org/
- Alien: http://joeyh.name/code/alien/
- Flatpak: http://flatpak.org/
- Snap: http://snapcraft.io/