Universal Packages

Sumner Evans and Robby Zampino

February 1, 2018

Mines Linux Users Group

Introduction

What are packages?

A **package** is an archive containing a collection of executable files or source code, along with metadata, which represent a computer program.

What is a package format?

A **package format** is an organizational structure for delivering packages to users.

Why do we need package formats?

- They provide a common way to bundle executables, libraries, assets, etc. for deployment on user machines.
- They provide metadata about programs for use in package managers.
- It would suck if we had to go find the source code for every single program we want to use and compile from source.[1]

[1] Actually, some package formats do require compilation from source (for example some AUR packages) but at least it helps automate this process.

A bit of history

1994 `dpkg` — the package format behind `apt` and `apt-get`. Used by Debian-based systems.

1997 `RPM` — the package format behind `yum` and `dnf`. Used by RHEL-like systems.

2002 `pacman` — the package manager for Arch Linux. It just uses `tar` files.

2004 `klik`/`PortableLinuxApps` (2011)/`AppImage` (2013) — a package format built to be Linux-distro agnostic.

2006 `nix` — a purely functional package format. Primarily used by NixOS.

June 2016 `snapd` — the Canonical-backed universal package format is ported to a wide range of Linux distros.

June 2016 `Flatpak` — the Red Hat-backed universal package format becomes generally available.

Universal Package Formats

Common objectives

- Linux distro agnosticism
- Solve the “dependency hell”
- Create a “single” deployment target for all of Linux

AppImage

Why is AppImage cool?

- **AppImage does not require installation.** The AppImage file is just its compressed image that is mounted with FUSE when it runs.
- **AppImage does not require root permission.** The application is run as the user and the base system is left untouched.
- **The AppImage itself is executable.** Just `chmod +x` the `.AppImage` file and run.
- **Linus says so**

  "This is just very cool."

  ~ Linus Torvalds

How does AppImage work?

Application developers use the `appimagetool` converts an AppDir into a self-mounting filesystem image.

AppImages can be integrated with the system via menu entries, icons, MIME types, etc. The `appimaged` daemon handles this registering and unregistering process.

How to create an AppImage

Create an AppDir with the following files (totally copied from their documentation):

- The files of the original application.
- A `.desktop` file that tells `appimagetool` about the name of the application, and the icon it should use.
- A PNG, SVG or an XPM icon with the name given in the `.desktop` file with the Icon entry.
- An AppRun file, which is used to start up the application inside the filesystem. Once the AppImage ELF has mounted the filesystem, it invokes this file. In the AppRun file, you can run some initialization procedures (such as setting environment variables), and then start up the real application.
- Optionally, you should also add AppStream metadata in `usr/share/metainfo`.

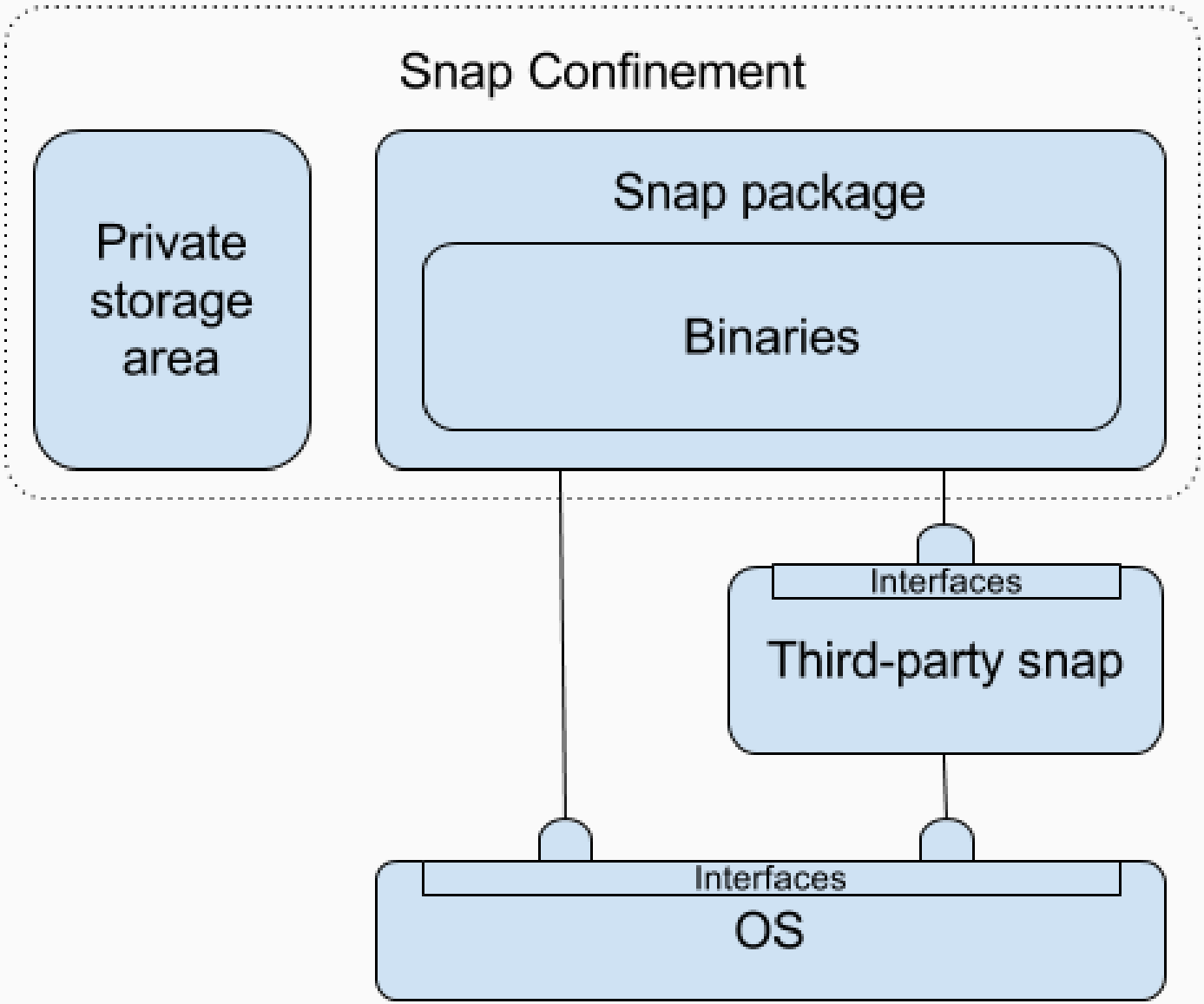
Live Demo: Running an AppImage

Snaps & `snapd`

Why are Snaps cool?

- **Snaps are `squashFS` filesystems.** They contain your app code and a `snap.yaml` file with metadata.
- **Snaps are self-contained.** The necessary libraries and runtimes are bundled in the snap. This allows you to have different library versions in your application than exist on your base system.
- **Snaps can have different levels confinement.**
  - `strict` is the default policy. The snap has read and/or write rights only in its own install space and selected areas.
  - `devmode` is for development of Snaps.
  - `classic` confinement behaves as a traditionally packaged application, with full access to the system.
- **Snaps can communicate with one another via *interfaces*.**

How do Snaps work?



How to create a Snap

- Make your application.
- Make a `snapcraft.yaml` with a bunch of stuff.
  - name
  - version
  - summary
  - description
  - grade
  - confinement
  - ...
- Run `snapcraft`.

Live Demo: Running a Snap

flatpak

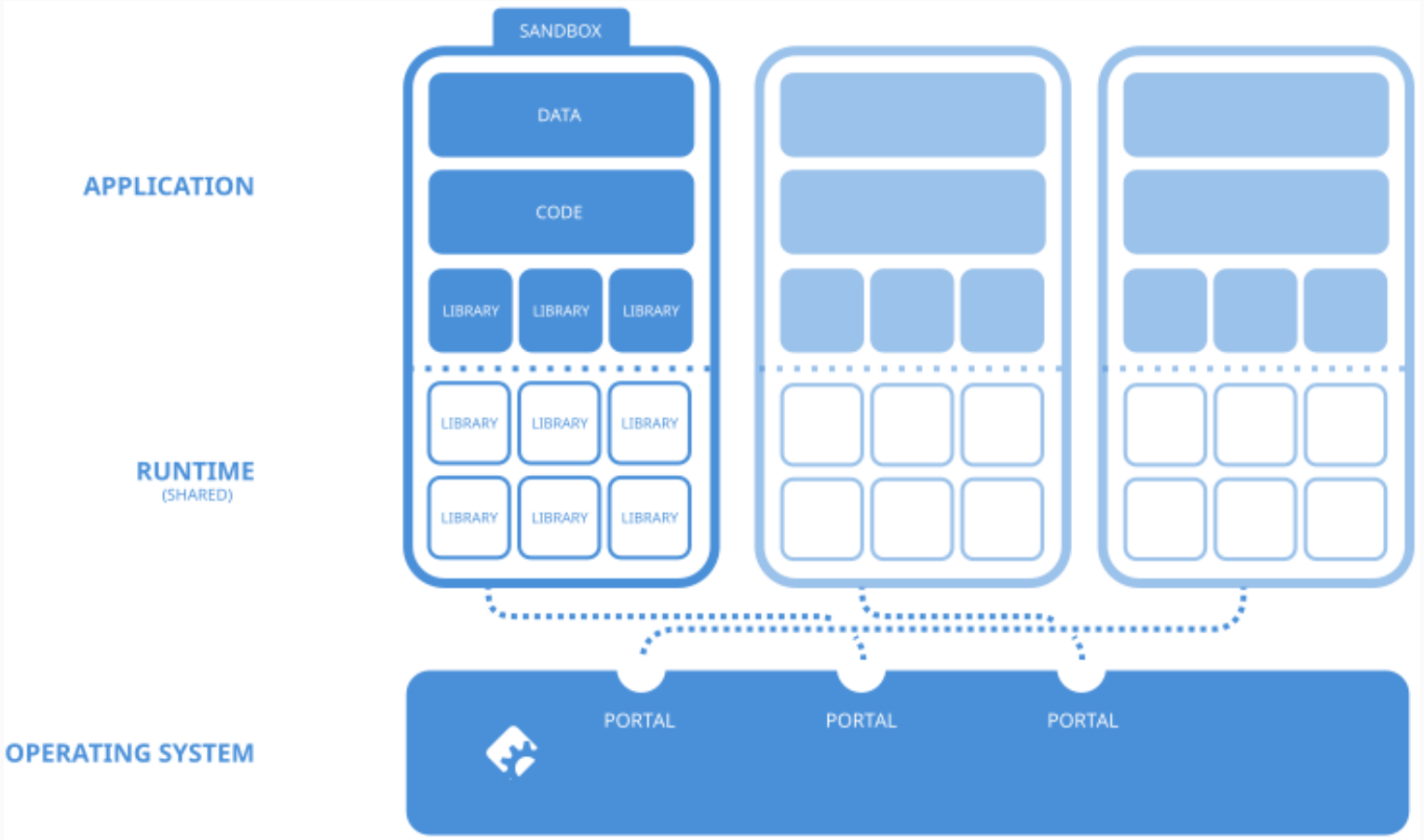
flatpak

- Flatpak is a system for building, distributing and running sandboxed desktop applications on Linux. (https://github.com/flatpak/flatpak)

Why is flatpak cool?

- Flatpak includes a system of runtimes that allow developers to build their application against a stable base.
- Runtimes allow dedeuplication of dependencies between packages
- Flatpak makes uses of bubblewrap for sandboxing
- Flatpak supports a system of Appstream metadata to allow packages to show up nicely in various package managers

flatpak Overview



Runtimes

- org.freedesktop.Platform
  - D-Bus
  - GLib
  - PulseAudio
  - X11
  - Wayland
- org.gnome.Platform (based on freedesktop)
  - GStreamer
  - PyGObject
  - Vala
  - GVFS
  - other stuff to make Gnome work...

Runtimes

- org.kde.Platform
  - Qt Frameworks
  - KDE Frameworks

Sandboxing

- All processes run as the user with no capabilities
- All processes run in a transient `systemd` user scope with the name `flatpak-$appid-$pid`
- `/` is a private tmpfs not visible anywhere else. This is `pivot_root:ed` into so it is the new and all other mounts from the host are unmounted from the namespace.
- Enviroment variables set:
  - PATH=/app/bin:/usr/bin
  - LD_LIBRARY_PATH=/app/lib
  - XDG_CONFIG_DIRS=/app/etc/xdg:/etc/xdg
  - XDG_DATA_DIRS=/app/share:/usr/share
  - XDG_RUNTIME_DIR=/run/user/$pid

How to build a flatpak package

flatpak-builder

- Install the flatpak-builder package
- See https://flatpak.org/getting.html for instructions

Runtimes

- Add the respository hosting your runtime
- `$ flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo`
- Install the runtime and corresponding SDK
- `$ flatpak install flathub org.freedesktop.Platform//1.6 org.freedesktop.Sdk//1.6`

Manifest

```
{
    "app-id": "org.flatpak.Hello",
    "runtime": "org.freedesktop.Platform",
    "runtime-version": "1.6",
    "sdk": "org.freedesktop.Sdk",
    "command": "hello.sh",
    "modules": [
        {
            "name": "hello",
            "buildsystem": "simple",
            "build-commands": [
                "install -D hello.sh /app/bin/hello.sh"
            ],
            "sources": [
                {
                    "type": "file",
                    "path": "hello.sh"
                }
            ]
        }
    ]
}
```

Packaging

- Package application
- `$ flatpak-builder app-dir org.flatpak.Hello.json`
- Test application
- `$ flatpak-builder --run app-dir org.flatpak.Hello.json hello.sh`
- Upload to repository
- `$ flatpak-builder --run app-dir org.flatpak.Hello.json hello.sh`

Comparison

Advantages of each of these universal package formats

- AppImage is great for portable, self-contained applications.
- Snaps are good for deploying single applications.
- Flatpak is good for distributing a set of applications. For example Gnome development builds are in a Flatpak repository.
- Nix is a cool functional package manager. Everytime you build the same version of the same package you should get the same output

Love to Hate Them

Proprietary enterprise applications are coming to Linux

Currently, when enterprises want to make a cross-platform application, they see this:

macOS	`.dmg`
Windows	`.exe`
Linux	`.deb` and `.rpm` and `PKGBUILD` and …, then deal with the dependency hell[2]

However, when companies like Canonical come in and say "just target snaps", all of a sudden, it may tip the scale at enterprises for them to start targeting Linux. If they create a snap, then they capture all of the Linux market, not just the subset that uses a particular format.

[2] Yes, you have to deal with dependency hell on other platforms too, but every platform has a different type of dependency hell. Coming to Linux is an expensive prospect for many enterprises.

Pros and cons

Pros

- More application availability.
- More abstraction! No more dealing with a bunch of different packaging formats.
- Easier troubleshooting: Developers can be certain (ideally) that users are using the same software configuration

Pros and cons

Cons

- The applications are going to be crap. Bloated, Electron, enterprise crap.
- More abstraction! Not much improvement on ease of deployment in comparison to deploying to `.deb`.
- Library version management is somewhat delegated to the application developers, Windows style. Heartbleed anybody. This hopefully can be somewhat alleviated by runtimes.

Questions?

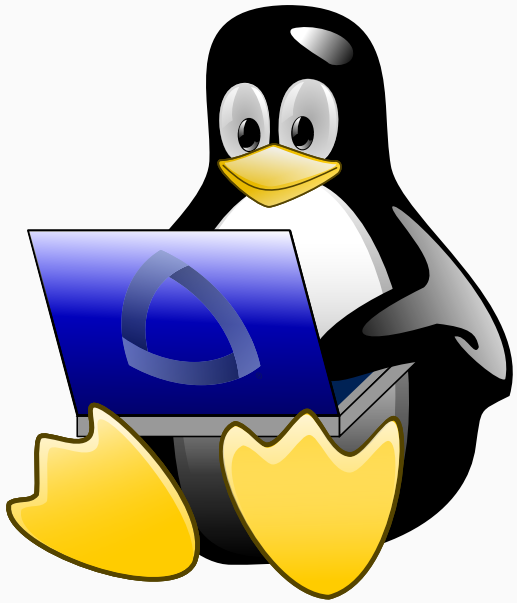
Resources

AppImage	`https://appimage.org/`
Snapcraft	`https://snapcraft.io/`
Flatpak	`https://flatpak.org/`
Nix	`https://nixos.org/nix/`

Copyright Notice

This presentation was from the **Mines Linux Users Group**. A mostly-complete archive of our presentations can be found online at `https://lug.mines.edu`.

Individual authors may have certain copyright or licensing restrictions on their presentations. Please be certain to contact the original author to obtain permission to reuse or distribute these slides.



Colorado School of Mines
Linux Users Group