Flatpak Is Not the Future

Original date: 2021-11-18
Last updated: 2021-11-23

Deploying apps for the Linux desktop is hard. A major problem has historically been library compatibility. Different Linux distributions, and even different versions of the same distribution, have had incompatible libraries. Unfortunately, there hasn't always been a culture of backwards compatibility on the Linux desktop.

This is finally changing. The stability of the Linux desktop has dramatically improved in recent years. Core library developers are finally seeing the benefits of maintaining compatibility. Despite this, many developers are not interested in depending on a stable base of libraries for binary software. Instead, they have decided to ignore and override almost all libraries pre-installed on the user's system.

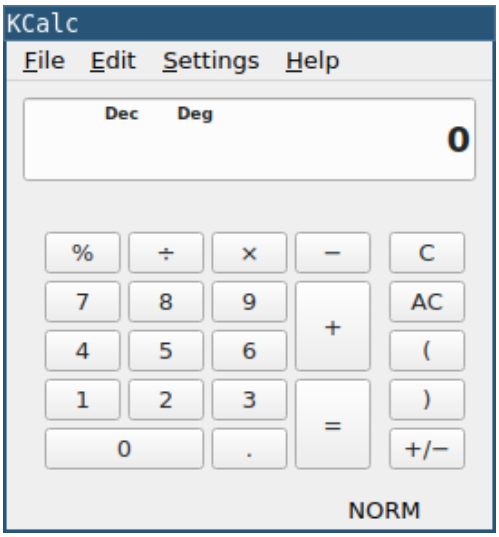
The current solutions involve packaging entire alternate runtimes in containerized environments. Flatpak, Snap, AppImage, Docker, and Steam: these all provide an app packaging mechanism that replaces most or all of the system's runtime libraries, and they now all use containerization to accomplish this.

Flatpak calls itself "the future of application distribution". I am not a fan. I'm going to outline here some of the technical, security and usability problems with Flatpak and others. I'll try to avoid addressing "fixable" problems (like theming) and instead focus on fundamental problems inherent in their design. I aim to convince you that these are not the future of desktop Linux apps.

Size

Suppose you want to make a simple calculator app. How big should the download be?

At the time of this writing, the latest KCalc AppImage (if you can even figure out how to download it) is 152 MB. For a *calculator*.



A screenshot of KCalc, a calculator app.

This is uncompetitive with Windows on its face. If I ship an app for Windows I don't have to include the entire Win32 or .NET runtimes with my app. I just use what's already on the user's system.

Other solutions like Flatpak or Steam download the runtime separately. Your app metadata specifies what runtime it wants to use and a service downloads it and runs your app against it.

So how big are these runtimes? On a fresh machine, install KCalc from Flathub. You're looking at a nearly 900 MB download to get your first runtime. *For a calculator*.

```
        ID                                      Branch    Op    Remote    Download
1.      org.freedesktop.Platform.GL.default     20.08     i     flathub   < 106.4 MB
2.      org.freedesktop.Platform.VAAPI.Intel    20.08     i     flathub    < 11.6 MB
3.      org.freedesktop.Platform.openh264       2.0       i     flathub     < 1.5 MB
4.      org.kde.KStyle.Adwaita                  5.15      i     flathub     < 6.6 MB
5.      org.kde.Platform.Locale                 5.15      i     flathub   < 341.4 MB (partial)
6.      org.kde.Platform                        5.15      i     flathub   < 370.1 MB
7.      org.kde.kcalc.Locale                    stable    i     flathub   < 423.1 kB (partial)
8.      org.kde.kcalc                           stable    i     flathub     < 4.4 MB
```

Note that the app package itself is only 4.4 MB. The rest is all redundant libraries that are already on my system. I just ran the `kcalc` binary straight out of its Flatpak install path unsandboxed and let it use my native libraries. It ran just fine, because all of the libraries it uses are backwards compatible.

Flatpak wants to download 3D drivers, patented video codecs, themes, locales, Qt 5, KDE 5, GTK 3, ICU, LLVM, ffmpeg, Python, and everything else in `org.kde.Platform`, all to run a calculator. Because unlike AppImage, the runtime isn't stripped down to just what the app needs. It's got every dependency for any app. It's an entire general-purpose OS on top of your existing OS.

Sharing Runtimes?

Flatpak says this is so that apps can share runtimes. But the whole point of their runtime system is to let apps use different runtimes. Each branch of a runtime (representing, say, a different base version of Ubuntu) is a completely independent runtime image.

They claim that they deduplicate runtimes. I question how much can really be shared between different branches when everything is recompiled. How much has `/usr` changed between releases of Ubuntu? I would guess just about all of it.

Steam at least fares somewhat better. The Soldier runtime is a 610 MB download, still large. But Steam only publishes a couple of official runtimes, so most games do actually share runtimes.

Flatpak allows anyone to define their own runtimes. freedesktop.org publishes some Flatpak runtimes for common use, but these aren't necessarily the ones apps are using. For example Fedora publishes apps with its own runtimes, and these are the ones available by default in its Software store.

If you install GIMP in Fedora 34's Software store, it defaults to Fedora's Flatpak of GIMP. This pulls in Fedora 35's 650 MB runtime, not any freedesktop.org runtime. Nothing will be shared with our freedesktop runtime KCalc we installed from Flathub earlier. On my machine `/var/lib/flatpak` is using over 3 GB of disk space for just these two apps.

This is apparently working as intended. They want runtimes to be a free-for-all, filling your hard drive with gigabytes of custom junk for every app. I can't imagine what system updates will be like in the future when you have a few dozen apps storing tens of gigabytes of runtimes that all want to be kept up to date.

"Disk space is cheap!"

They say disk space is cheap. This is not true, not for the root devices of modern computers. Built-in storage has in fact been shrinking.

Software has gotten so much slower and more bloated that operating systems no longer run acceptably on spinning rust. Laptop manufacturers are switching to smaller flash drives to improve performance while preserving margins. Budget laptops circa 2015 shipped with 256 GB or larger mechanical drives. Now in 2021 they ship with 120 GB flash. NVMe drives are around $100/TB and laptop manufacturers inflate these prices 500% or more so upgrading a new laptop can be pricey.

Chromebooks are even smaller as they push everything onto cloud storage. Smartphones are starting to run full-fledged Linux distributions. The Raspberry Pi 4 and 400 use an SD card as root device and have such fantastic performance that we're on the verge of a revolution in low-cost computing. Surely Flatpak should be usable on these systems! There is no reason why a 16 GB root device shouldn't fit every possible piece of non-game software we could want. Flatpak isn't part of the revolution; it's holding it back.

Why shouldn't storage shrink anyway? Software should be getting more efficient, not less. Even if bandwidth were free and hard drives grew on trees, it would not excuse such drastic bloat. The waste involved in these app packaging solutions is downright offensive. Apollo landed on the moon with 4 kB of RAM and 72 kB of ROM, yet we can't run a calculator with less than 150 MB. Any engineer worth his salt should be optimizing for efficiency, not dismissing it as irrelevant, especially when it so clearly impacts the user experience.

Memory Usage, Startup Time

The penalties of huge alternate runtimes aren't just in storage and bandwidth.

Each app with a new runtime adds another hundred megs or more of RAM usage. This adds up fast. Most computers don't have enough RAM to run all their apps with alternate runtimes. The Raspberry Pi 400 has only 4 GB of RAM. Low-end Chromebooks have only 2 GB. Budget laptops tend to have 8 GB, mostly thanks to the bloat of Windows 10, but these app packaging solutions are catching up to it.

A bigger problem is that these apps can actually take several seconds to start up. They have to load all their own libraries from disk instead of using what's already on the system, already in memory.

Snap is the slowest of all, largely because it stores all its data in squashfs images. Snap mounts all registered snaps at startup instead of just extracting the metadata they need beforehand, possibly in an effort to mitigate this slowness. They're just moving part of the slow startup time to the boot time of your computer. All sorts of snap crap now shows up in `mount` and `fdisk -l`. The more snaps you have installed, the slower your computer will start, even if you don't use them.

Today, on my machine, the KCalc Snap takes a full seven seconds to start up. Not just the first time after boot; every time, without fail. Seven seconds to start a calculator.

Canonical started converting their basic desktop apps like the GNOME Calculator to Snap in Ubuntu 18.04. The user experience was so terrible that they quietly converted them back to normal apps in Ubuntu 20.04. If these technologies are not even good enough for their own apps, those as basic as a calculator, why would they be good enough for yours?

Drivers

A major problem with alternate runtimes is drivers. New graphics hardware needs new graphics libraries which have a ton of dependencies. Mesa depends on LLVM for compiling shaders. The NVidia driver depends on a kernel module whose version must exactly match that of the library. All of these libraries have their own transitive dependencies like libdrm, libstdc++ and glibc. If you want new hardware to work, you need to be using new versions of all of these libraries.

Linux distributions, especially those with rolling releases or hardware enablement packages, do a great job of keeping these libraries up-to-date for new hardware. Bundled runtimes do not.

Take a look at the pain Steam has to go through to get drivers working in the Steam Runtime. They use various heuristics to determine whether to override each library in the runtime with that of the host system, so each app is running on a Frankenstein hodge-podge of libraries. This is not the way to build stable software.

Flatpak recognizes that mixing libraries with the host system is a mess. Instead, they want to bundle their own graphics drivers in the runtimes, keeping them regularly updated for new hardware. But they can't control the most important part, the kernel, so they can't directly package drivers that depend on specific kernel versions or proprietary kernel modules. So the runtime is broken into extensions, and extensions can go the Steam route of pulling in the native driver or even download specific NVidia drivers on-the-fly to match the kernel module.

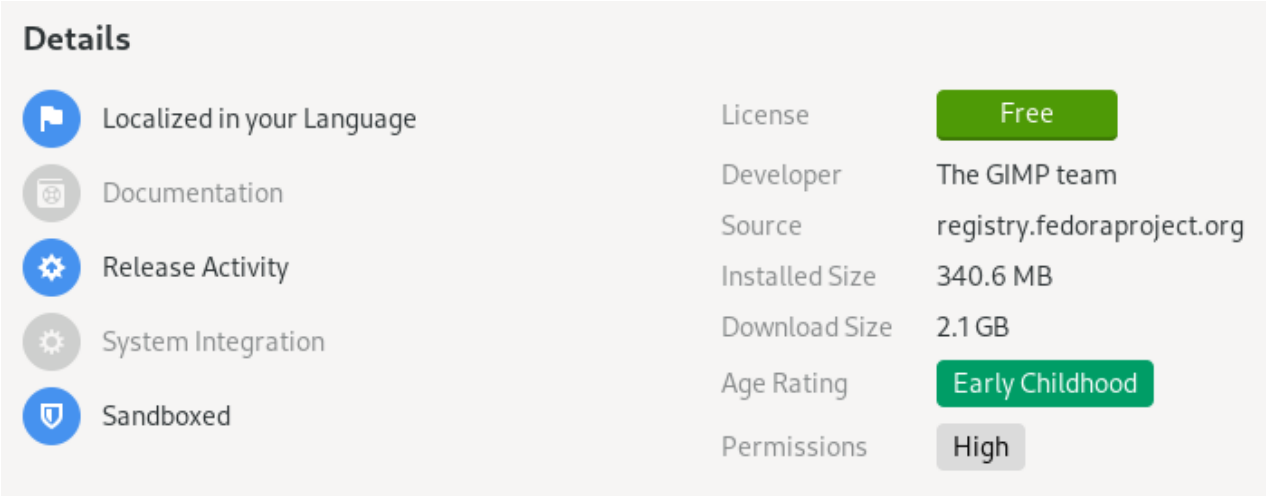
AppImage mixes host libraries with the runtime as well, but not dynamically; they just decide what to exclude at packaging time so it's even less reliable than Steam. Docker has an NVidia toolkit that must be installed on both the host system and the containerized runtime to get drivers working. Snap is doing who knows what. None of these are proper solutions.

Drivers are supposed to be the responsibility of the operating system. This is what it's good at. Why are we working so hard to get around what the operating system offers by default for native apps?

This isn't an issue specific to video games by the way. Lots of modern apps are now directly using hardware accelerated graphics, even terminals and text editors. Many apps need the GPU for background computation. Productivity apps like video editors, apps that use machine learning, scientific tools like space/astronomy visualization, anything that does 3D like Google Earth… All of these need access to modern video hardware. This will only grow as apps get faster and provide more features and richer user interfaces.

Security

Flatpak allows apps to declare that they need full access to your filesystem or your home folder, yet graphical software stores still claim such apps are sandboxed. This has been discussed before. Here's what happens when I search GIMP in the Software app on a fresh install of Fedora 34:



Fedora claims GIMP is sandboxed. If you click "High" next to "Permissions", you see a little exclamation mark saying it has "File system" permissions.

Such an app can drop a malware executable anywhere in your home folder and add a line to your `~/.profile` or a desktop entry to `~/.config/autostart/` to have it auto-started on your next login. Not only will it run outside of any container, it will even persist after the app is uninstalled.

This is exactly what a nefarious app would do to break out of its sandbox. This is also what a virus would do that exploited a security vulnerability in a sandboxed app.

Suppose libjpeg has a zero-day remote code execution vulnerability. You open a seemingly innocuous JPEG in Flatpak GIMP and it drops its payload set to autostart in your home folder. How did the sandbox help here? The behaviour of this virus is identical regardless of whether GIMP is sandboxed. The only differences are that a) the libjpeg in the alternate runtime is likely to stay out-of-date longer than the one from your Linux distribution; and b) the virus is more likely to work against the sandboxed app because all installations will be using the exact same binary of libjpeg.

The Snap Store app, to its credit, displays a warning: "This application is unconfined. It can access all personal files and system resources." Which apps display this warning, you ask? All of them. KCalc has it. All of the Editor's Picks have it. In my testing for this blog post I could not find an app that does not display the warning.

"It's better than nothing!"

Flatpak and Snap apologists claim that some security is better than nothing. This is not true. From a purely technical perspective, for apps with filesystem access the security is exactly equal to nothing. In reality it's actually worse than nothing because it leads people to place more trust than they should in random apps they find on the internet.

Take a look at this writer complaining about the security of AppImages while praising Flatpak and Snap:

> *I install applications via source, apt, Snap and Flatpak–I don't discriminate. As long as an application will install and run as expected, I'll install it, regardless of the package format.*
>
> *With one exception: AppImages.*
>
> *[…]*
>
> *On the rare occasion I'm willing to use an AppImage, I will only do so when I absolutely trust the developer. Why? Remember, an AppImage is an application you simply download and run. Anyone can build an AppImage, proclaim it a must-have piece of software, roll something nefarious into it, and make it available for download.*

He expresses deep skepticism of AppImages, but not of Flatpaks or Snaps, because he believes their security measures keep him safe. This is the danger of lying about sandboxing.

Permissions and Portals

Flatpak is working on a fine-grained permission system to improve the security of its sandbox. Permissions are things like whether the app is allowed to access the microphone or the printer. Portals are things like a file open dialog that runs outside the sandbox, so the app in the sandbox gets only the file the user chose.

Flatpak documents these portals and provides libportal, a client library to access them. However this isn't really intended for individual apps. It's all meant to be integrated in the toolkits. From the documentation:

> *Interface toolkits like GTK3 and Qt5 implement transparent support for portals, meaning that applications don't need to do any additional work to use them (it is worth checking which portals each toolkit supports).*

Apparently, developing client APIs for apps themselves is antithetical to Flatpak's mission. They want the apps running on Flatpak to be unaware of Flatpak. They would rather modify the core libraries like GTK to integrate with Flatpak. So for example if you want to open a file, you don't call a Flatpak API function to get a file or request permissions. Instead, you call for an ordinary GTK file open dialog and your Flatpak runtime's GTK internally does the portal interaction with the Flatpak service (using all sorts of hacks to let you access the file "normally" and pretend you're not sandboxed.)

This is the most complicated and brittle way to implement this. It's also not at all how other sandboxed platforms work. If I want file access permissions on Android, I don't just try to open a file with the Java File API and expect it to magically prompt the user. I have to call Android-specific APIs to request permissions first. iOS is the same. So why shouldn't I be able to just call `flatpak_request_permission(PERMISSION)` and get a callback when the user approves or declines?

This is why. Fedora is auto-converting all of their rpm apps to Flatpak. In order for this to work, they need the Flatpak permission system and Flatpak in general to require no app changes whatsoever.

Why on Earth would they do a mass automatic conversion of apps? Your guess is as good as mine. The video claims that Fedora's apps are higher quality than upstream, and Fedora makes their Flatpaks available on older distributions. I think it's more likely they just want huge numbers of auto-converted apps to make it look like Flatpak is useful. Whatever the reason, it's clear that this requirement has influenced many of their poor design decisions.

Identifier Clashes

So Fedora auto-converts all its apps to Flatpak. Does it at least namespace them to something specific to Fedora?

No, it doesn't. Fedora publishes its Flatpak of GIMP as `org.gimp.GIMP`. This conflicts with the official `org.gimp.GIMP` published by the GIMP developers on Flathub. On a fresh install of Fedora 34, if you add the Flathub repository and type `flatpak install org.gimp.GIMP`, you get prompted for which one to install:

```
[nick@fedora active]$ flatpak install org.gimp.GIMP
Looking for matches…
Remotes found with refs similar to 'org.gimp.GIMP':

   1) 'fedora' (system)
   2) 'flathub' (system)

Which do you want to use (0 to abort)? [0-2]:
```

If you choose option 1, you get a build of GIMP with Fedora's patches that uses the 650 MB Fedora 35 runtime. If you choose option 2, you get a different build of GIMP that uses the 1.8 GB freedesktop.org GNOME runtime.

Isn't the whole point of reverse DNS that the `org.gimp` prefix is reserved for people who actually own the `gimp.org` domain? How can Fedora justify publishing apps while masquerading as the upstream developers? If major Linux distributions won't even respect DNS, who will?

Flathub doesn't enforce DNS ownership either by the way. They let anyone publish anything; they merely prefer that apps are controlled by their authors. A sane security strategy would be to require a DNS challenge to allow a publisher to use a specific app prefix.

Complexity

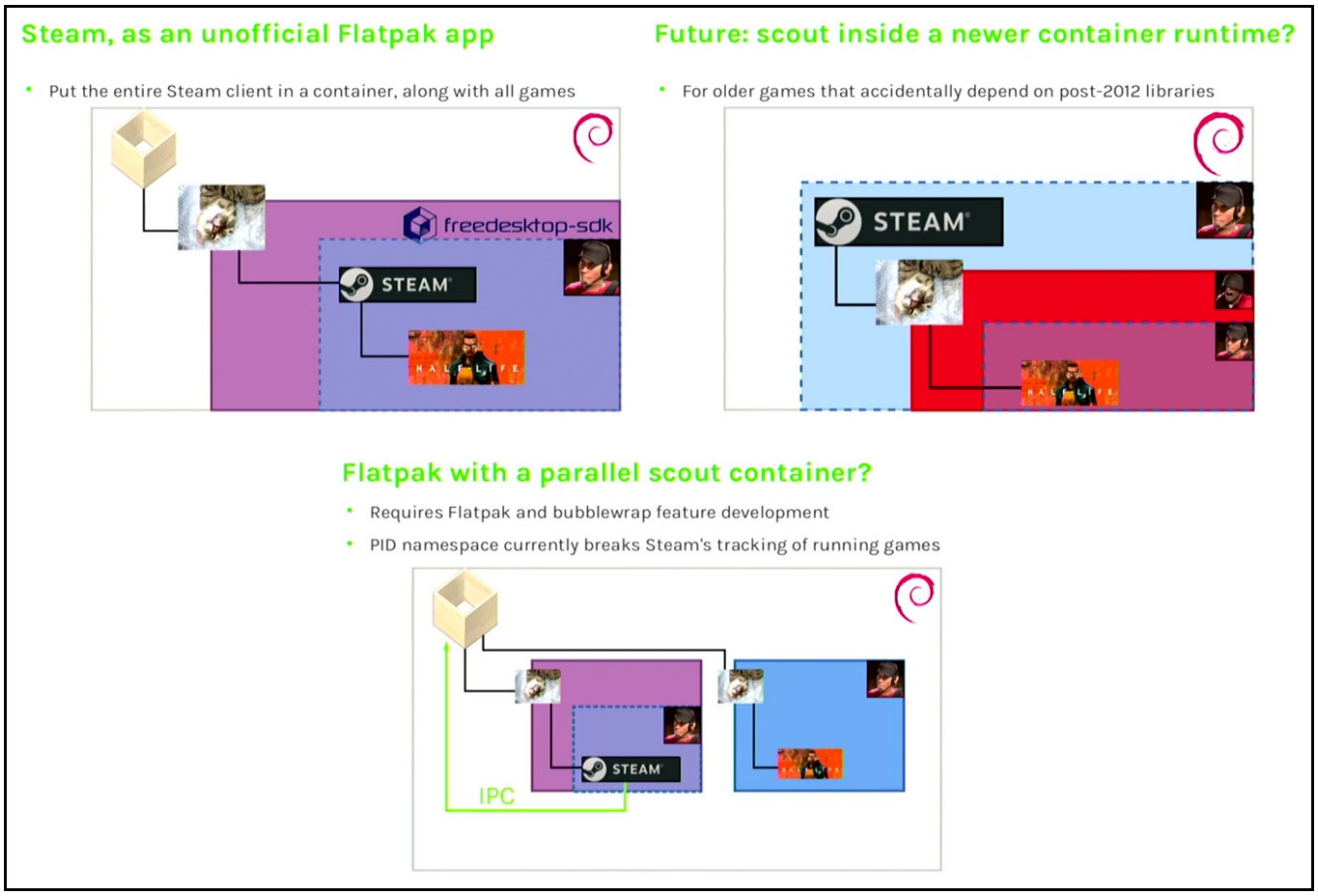
Endless articles have been written about the importance of simplicity in software design. Worse is Better is perhaps the earliest example. A modern refrain is "software complexity is killing us". The state of software development is downright miserable in 2021.

You would think that these packaging mechanisms would embrace simplicity if they want to attract software developers. In fact they are doing the opposite. The driver, sandboxing, and permission/portal issues we've

discussed barely scratch the surface of the complexity involved in making this all work. In “worse is better” terms, they’re not even “the right thing” anymore. At this point they’re just pushing complexity for its own sake.

Proponents of containerization believe it is the solution to every problem, the hammer for every nail. We now have multiple Linux distributions based on containerizing everything (e.g. Fedora CoreOS, MocaccinoOS.) We have a major movement to adopt Docker and Snap on embedded (e.g. balenaOS, Ubuntu Core.) Docker may soon be running on your fridge and in your car.

Keep watching the video linked earlier to see the descent into madness: Steam nested in a Flatpak; the Scout runtime nested in the Soldier runtime pulling in drivers from the Freedesktop runtime; Steam making IPC calls to its Flatpak container to make parallel containers with yet more runtimes.



Various combinations of Steam and Flatpak containers and runtimes. Not pictured: Steam and the game running directly on the user’s native environment, like they do on every other OS.

The complexity gets far worse than the above video even shows. Flatpak uses libostree to store all its data in a content-addressible deduplicated layer above the filesystem. Steam now uses libcapsule which uses `dlmopen()` namespaces so that game executables and driver libraries can load different versions of libstdc++ into the same process. We haven’t even talked about Wine/Proton which is a whole other layer of crazyness.

This is a vivid example of software complexity taking us to the brink of collapse. This is the most complicated way imaginable to distribute software. What will happen when large numbers of apps have been shipped that

depend on this scaffolding? How will this ever be maintained? How many people on Earth will truly understand how this all works?

Services

All of these app packaging systems require that the user have some service installed on their PC before any packages can be installed.

AppImage, to its credit, technically does not require a service to run apps, but it doesn't integrate with the desktop without it. I needed to use an AppImage app for a while and my solution was to just leave it in my `~/Downloads` folder and double click it from my file manager to run it. This is a terrible user experience.

All of the desktop integration (launcher entries, mimetypes, icons, updates) is provided by either `appimaged` or AppImageLauncher, one of which must be installed by the user for any of this to work. So in practice, AppImage is no different than any of our other solutions: it requires a service to be usable.

If a user doesn't have the service, well that's another hoop they have to jump through to install an app. A few distributions, notably Fedora, pre-install Flatpak. A few distributions, notably Ubuntu, pre-install Snap. SteamOS pre-installs Steam. Other than these special cases, users have to install the correct service before installing an app. Developers have to teach them to do so and lose all of the users who can't figure it out or can't be bothered.

App Stores

A major goal of most of these technologies is to support an "app store" experience: Docker Hub, Flathub, the Steam Store, Snapcraft, and AppImageHub (but not AppImageHub?) These technologies are all designed around this model because the owners want a cut of sales revenue or fees for enterprise distribution. (Flathub only says they don't process payments *at present*. It's coming.)

This is the real reason Ubuntu wants everyone to use Snap instead of helping us build apps that run natively on Ubuntu. This is also why they keep the Snap server source code closed. They want it to be "open" the way Android is "open", where there is only one official store and sideloading apps is as annoying as possible.

This is very far from the traditional Windows experience of just downloading an installer, clicking Next a few times, and having your app installed with complete desktop integration. This is true freedom. There are no requirements, no other steps, no hoops to jump through to install an app. This is why the Windows Store and to some extent even the macOS App Store are failing. They can't compete with the freedom their own platforms provide.

I know free software and privacy people don't want to allow proprietary apps this level of freedom on their PCs. But the vast majority of users and independent software vendors don't care. What users want is easy (and

efficient!) installation. What the biggest software vendors want is the freedom to ship software directly to customers, without the intervention of a service or hub.

This is indeed possible on the Linux desktop. Let's talk about that in the next section.

The Current State of Backwards Compatibility

There seems to be a collective trauma in the Linux community in depending on libraries to remain binary compatible across distributions and version upgrades. These alternate runtime tools are clearly built by people who refuse to consider using *anything* on the user's system (in the case of Flatpak, not even driver libraries.) It's true that in the past, open source library ABIs were flaky. I'm here to tell you that you no longer have to be afraid.

The backwards compatibility situation has gotten much better in recent years. The Linux kernel has been slowly exporting its legendary culture of backwards compatibility up the stack. Core libraries like glibc (since 2.1) and libstdc++ (since GCC 5) are intending to remain backwards compatible indefinitely. freedesktop.org has made huge progress standardizing runtime environments across distributions. Debian and others have mostly stopped making ABI-breaking customizations to their packages. Websites like ABI Laboratory track ABI changes to provide early warnings of breaks. The Linux Standard Base (LSB), though now seemingly defunct, made significant progress in standardizing capabilities and preventing ABI breakage.

Just in the past week, I've used several Linux apps distributed as plain binary tarballs that run on my native environment. The generic Linux binary releases of DevilutionX and OpenTTD use my native SDL for graphics, sound and input. Master PDF Editor uses a long list of native libraries including Qt 5. The QNX Software Center was compiled for the LSB and uses my native GTK 3. These run perfectly, no alternate runtime needed. Why? Because most libraries today are actually decently backwards compatible.

GOG.com's strategy for Linux support is just about the opposite of Steam's, and works just like traditional installers for Windows. Each game is built for a particular Ubuntu LTS release: 14.04, 16.04, 18.04. As long as your libraries are at least that new, the game will run. They ship self-extracting install wizards built with mojosetup, which is a bit like the Windows-based WISE or InstallShield installers of old (except they don't require root access.) You download and run the installer as your normal user, hit Next->Next->Next and your game is installed. Full desktop integration, no giant alternate runtimes, no service required. Any ISV can ship software this way: just a self-installing executable shipped directly to users.

It's not perfect. In particular mojosetup doesn't install missing dependencies (yet) and GOG doesn't totally follow the XDG specs for install paths (yet). These are minor, fixable problems, nothing on the scale of Flatpak. This is *simple*, this is true freedom, and the contrast in user experience is striking. I would take a mojosetup installer over an AppImage or Flatpak or Steam any day of the week.

For developers, yes, this may be harder than making an AppImage. There will still be occasional bugs that crop up between library versions. There will still be occasional differences between distributions. These are things you can work around. It may be painful at first but it will be worth it: you will provide a much superior user experience by working around the issues with your users' libraries rather than attempting to replace them.

Forcing Distributions to Maintain Compatibility

The backwards compatibility situation will continue to get better with time. The more proprietary software is shipped for the native Linux desktop, the less likely major distributions will accept upstream breakage. Imagine if millions of office workers used Excel on Ubuntu. How loudly do you think businesses would complain if a distribution upgrade broke it?

In 2014, Linus Torvalds suggested that Valve could save the Linux desktop. He hoped they would publish a large volume of proprietary games that depended on the Linux desktop core libraries, finally forcing distributions to preserve their ABI. Unfortunately, as he predicted, that didn't happen. Instead of relying on the native runtime libraries Steam replaced most of them with its own.

Steam needed to replace some libraries to make games work back in 2012. Today this is no longer necessary. So why are they still pushing their runtimes? How much progress could we make if Steam deprecated their runtimes, abandoned containerization for new games, and let all new games just use the native system libraries? How loudly do you think gamers would complain if a distribution upgrade broke their favourite game?

A few years ago Canonical decided to drop 32-bit libraries from Ubuntu 19.10. After much outrage, including Wine and Steam threatening to drop support for Ubuntu, they reversed their decision. Imagine how stable Ubuntu could be if Steam games depended on even more of the system libraries.

To any library developers reading this, it's hard to over-emphasize how important backwards compatibility is to the popularity of a platform. Backwards compatibility is a major part of why Windows is still the dominant desktop OS. Microsoft has managed to keep the Win32 API stable for over 25 years. GUI apps built for Windows 95 still work out of the box on Windows 10. Businesses actually care about this! They use ancient proprietary software that is critical to their business whose source code has long been lost to the sands of time. A platform that breaks their software is no platform at all.

The GTK problem

There is one glaring exception to my claims regarding Linux library stability. That exception is GTK. GTK famously breaks their libraries with reckless abandon. This has historically been one of the biggest problems with shipping a binary app for Linux.

I believe this is partly due to a militant position on free software. Some advocates believe so strongly that users should be able to recompile their software that they force them to do so. They break libraries seemingly on

purpose just to say, “Recompile! Oh you can’t? That’ll teach you to use binary software!” Of course users don’t want to recompile their software, but what users actually want is usually lost on GNOME developers.

With the recent release of GTK 4, there is little reason to believe the situation will improve. At one point they proposed bumping the soname on all major *and minor* releases. Thankfully they have reversed course, but their latest versioning scheme still indicates their desire to accelerate the pace of major releases, allowing them to more frequently make breaking changes and remove deprecated functions. They do at least intend to remain ABI stable within a major release series, but given their track record, by the time anyone trusts GTK 4 to remain stable it will be obsolete.

This sounds like bad news, but there is a silver lining here: while they’re mucking about with GTK 4, they *aren’t* breaking GTK 3.

The ongoing transition to GTK 4 gives us an opportunity here. The GTK 3 ABI is finally a stable target, and it’s still pre-installed in most distributions because lots of built-in apps haven’t upgraded yet. If we can get a sufficient number of binary apps depending on it, distributions will be forced to maintain it and even pre-install it “forever”, the same way Microsoft maintains the Win32 API “forever”.

I strongly recommend that all app developers (open source or otherwise) delay upgrading to GTK 4 as long as possible. The longer we can delay the switch, the more stable GTK 3 will be. If the GTK team want us to use GTK 4, they will need to prove its stability over many years, not simply replace it with GTK 5.

I’m not saying GTK 3 is good. All I’m saying is it exists. No other toolkit can claim to be practical for desktop apps *and* relatively stable *and* pre-installed on the vast majority of Linux desktops. We all hope something better will come along one day but in the meantime GTK 3 is what we’ve got.

There has never been a better time to ship a binary app that targets the Linux desktop. And I don’t mean targets its own bundled runtime; I mean truly targets the user’s runtime environment. The time is now.

Is Flatpak Fixable?

Here’s the thing. I actually think Bubblewrap, the sandboxing tool used by Flatpak (and now Steam), is pretty good. It’s the key technology to make app sandboxing good enough to compete with Android or iOS. If it was used to hide sensitive stuff like `/home` but otherwise just mounted the native `/usr` straight into the sandbox, it could be great. Unfortunately, Flatpak does the exact opposite.

If the Flatpak developers truly want an app distribution system that rivals Android and iOS, the sandbox, permissions and portal system should be the only focus.

They should:

- Abandon everything related to runtimes, and instead mount the native `/usr` (or a restricted set of core libraries from `/usr`) read-only into each container;
- Add metadata to Flatpaks to declare library dependencies that the Flatpak service will install with the host distribution's package manager (perhaps with generated meta-packages that can be cleanly removed);
- Build a fine-grained user-interactive runtime permission system that requires the app to make Flatpak-specific API calls to activate permission dialogs; and
- Deprecate install-time permissions (especially filesystem access) and remove all apps from Flathub that use them.

Under this system, apps would be encouraged to statically link many of their dependencies, but use the system GTK/Qt/SDL, OpenGL/Vulkan, OpenSSL/curl, and other large or security-critical libraries. The community could maintain guidelines and wrappers to make apps that dynamically link against the system libraries cross-version and cross-distribution. Apps would be expected to make changes to run sandboxed and request permissions directly through a Flatpak client API.

This is much more akin to how iOS and Android work. But it means abandoning like 95% of Flatpak. It would be such a drastic change that they might as well start over under a different name. I don't see it happening.

An app store based on a sandboxing technology for native apps is a reasonable idea. I certainly don't think it should be used for everything though. I don't think it's at all necessary for software like Excel or Photoshop. Users do not care about sandboxing those sorts of apps and their vendors will refuse to be sandboxed anyway.

But for small apps and games from independent developers, a proper sandboxed app marketplace can in theory increase their reach. It can eliminate much of the trust that would otherwise be necessary to run them on your computer and can bring native apps closer to the ease of use of web apps.

It's certainly not easier for developers, but good native apps still provide such a vastly better user experience than web apps that it's possible a proper sandboxed app store can create a resurgence in native apps. Microsoft obviously failed with their attempt, but macOS kind of didn't. Maybe there is a way Linux can succeed as well.

Conclusion

Snap and Flatpak in their current incarnations have been around for at least five years. AppImage, Steam and Docker have been around even longer. None of the above is new. The problems with alternate runtimes were known from the very beginning, yet little progress has been made in fixing them. I don't believe these are growing pains of a new technology. These are fundamental problems that are mostly not fixable.

All of these technologies are essentially building an entire OS on top of another OS just to avoid the challenges of backwards compatibility. In doing so, they create far more problems than they solve. Problems of compatibility are best solved by the OS, the real one, not some containerized bastardization on top. We need to make apps that

run natively, that use the system libraries as much as possible. We need to drastically simplify everything if we have any hope of attracting proprietary software to Linux.

If you are a Linux distribution maintainer, please understand what all of these solutions are trying to accomplish. All your hard work in building your software repository, maintaining your libraries, testing countless system configurations, designing a consistent user experience… they are trying to throw all of that away. Every single one of these runtime packaging mechanisms is trying to subvert the operating system, replacing as much as they can with their own. Why would you support this?

I implore you, do not use these packaging tools. Don't add their services to your Linux distributions, don't use apps packaged this way, and don't ship apps that use them. Mass containerization and alternate runtimes cannot possibly be the future of desktop apps on Linux. If this is really the direction it's going, the future will be so shitty that we'll all end up back on macOS or Windows.

Personally, I'm much more interested in how to get Excel and Photoshop on Linux rather than untrustworthy drive-by apps and games, so I don't really care about sandboxing, permissions, portals, app stores, alternate runtimes or really any of the stuff Flatpak does. Those are all counter-productive to convincing Microsoft and Adobe to port their software suites to Linux. Attracting these vendors will only happen by empowering them with a stable platform, not locking them in a box.

In future posts I'll be going over some of the problems with targeting desktop Linux directly, presenting some of the stuff I'm working on, and hopefully coming up with real solutions that don't depend on alternate runtimes.

Updates: A link to libportal was added, a reference to AppImageLauncher was added, and the description of GTK 4's versioning scheme was corrected.

← ludocode.com