Flatpaks the Nix way

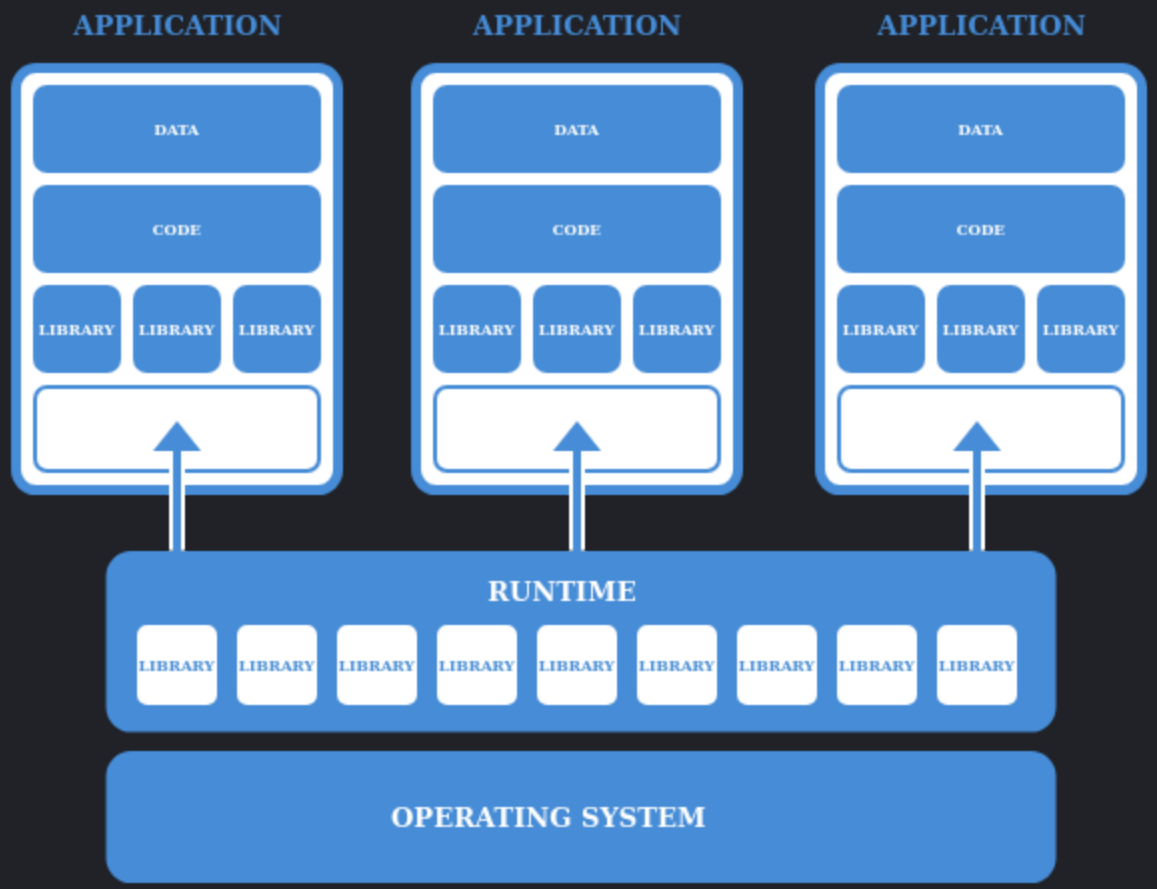
NixCon 2025

@gmodena

Origin story

- Linux Desktop (1998-2007):
  - Stable system vs bleeding.
- macOS:
  - stable Unix core.
  - GUI apps are orthogonal.
  - nix-darwin (2022).
- NixOS (2023).
- Blending NixOS with Flathub for friends and family, Martin Wimpress, NixCon 2023.

Flatpak



Source: https://docs.flatpak.org/en/latest/basic-concepts.html, 2025.

Bundles

- A *single-file* distribution format: "just give me a file I can install anywhere":

```
flatpak install com.example.App.flatpak
flatpak install flathub com.example.App
```

- **Application** vs **Runtime** bundles.
- Repositories for distribution (Flathub).
- `.flatpakref` for an "Install from web" workflow (like mobile app stores).
- Version pinning for reproducible deployments.
- Under the hood: OSTree storage, bubblewrap, metadata, permissions, versioning.

Cool stuff

- Security and sandboxing.
- Cross-architecture emulation: run aarch64 apps on x86_64 (and vice versa).
- Overrides.
- Runtime sharing: multiple apps share common runtimes.
- Desktop Integration.

Flatpak on NixOS

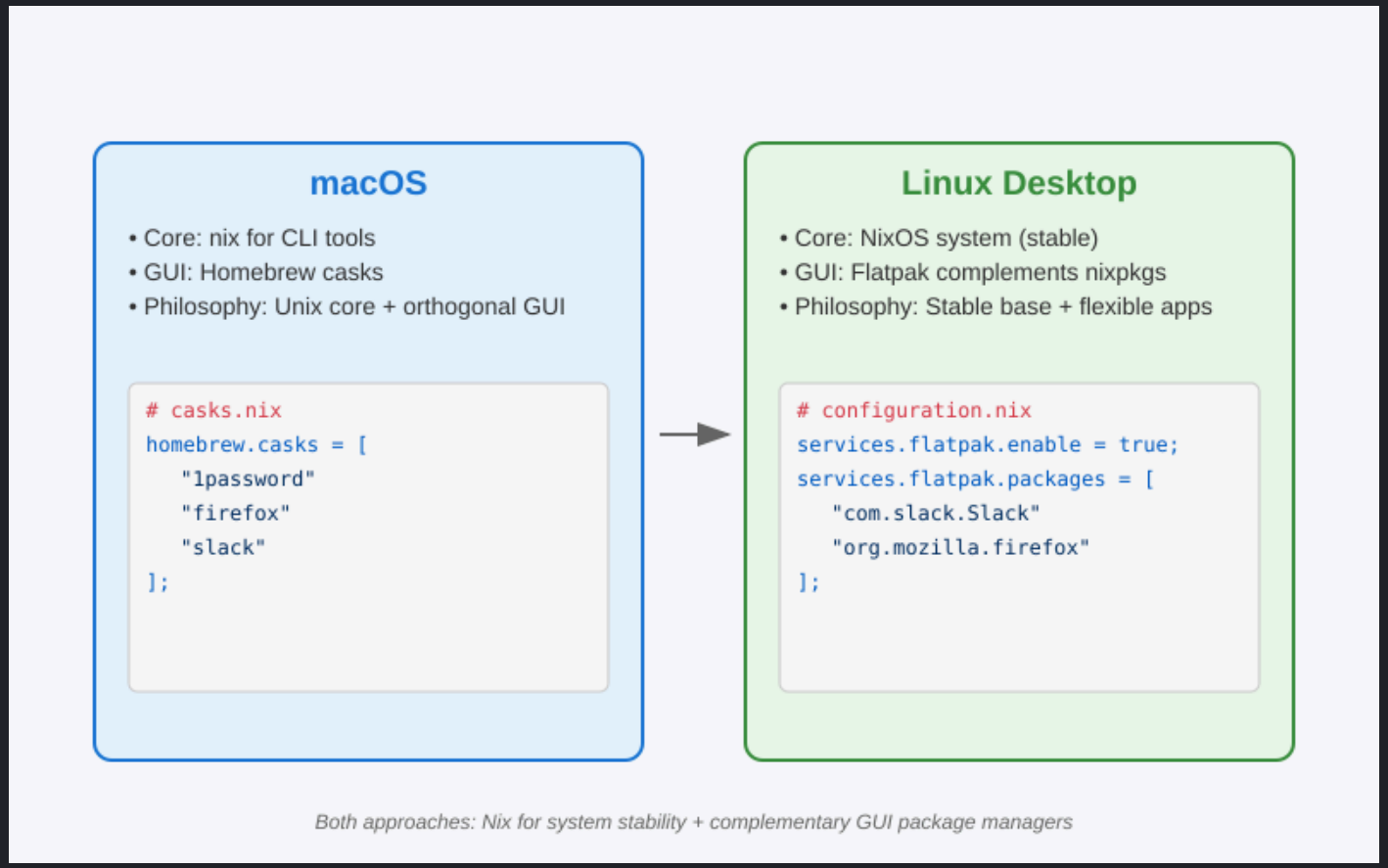
- https://flatpak.org/setup/NixOS
- Toggle config:

```
services.flatpak.enable = true;
```

- Enable Flathub:

```
flatpak remote-add --if-not-exists flathub https://dl.flathub.org/repo/flathub.flatpakrepo
```

Desktop Applications Strategy



Why is it hard to make flatpak reproducible?

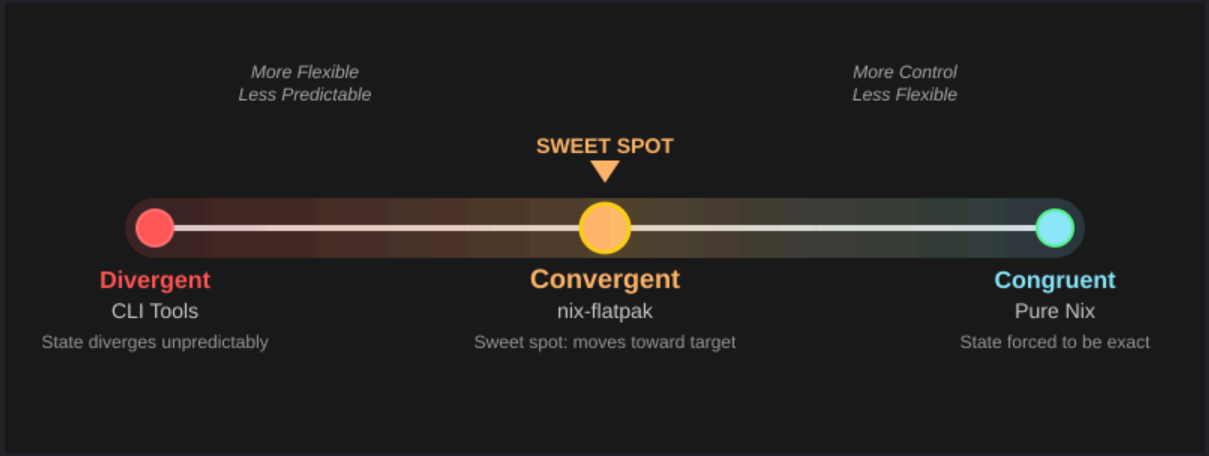
- Declarative Gaps.
- Version Pinning Limitations.
- Runtime Drift.
- Limited Tooling Integration.

Can we make flatpak state declarative?

- Remember: I care about a stable core, but I can deal with less stable desktop apps.

Yes, with tradeoffs

- Why Order Matters: Turing Equivalence in Automated Systems Administration, Steve Traugott, Lance Brown, 2002.



nix-flatpak

- https://github.com/gmodena/nix-flatpak.
- Convergent package management.
- Orthogonal to nix store.
- Extends `services.flatpak` with ad-hoc options.
- Responsible for managing lifecycle + overrides.
- Version pinning at commit id (allows precise rollbacks).
- Nix module, distributed as a flake (flakehub and flakestry).

Example

```
{ lib, ... }: {
  # Add a new repository. Keep the default one (flathub)
  services.flatpak.remotes = lib.mkOptionDefault [{
    name = "flathub-beta";
    location = "https://flathub.org/beta-repo/flathub-beta.flatpakrepo";
  }];

  # Declare apps to install
  services.flatpak.packages = [
    "com.logseq.Logseq"
    { appId = "com.visualstudio.code";
      origin = "flathub-beta";
      hash = "d01cce4b518f4017e19becd2793e777cebc44f3cc0506972606fd8c0c75b297b";
    }
    {
        bundle="file:///home/gmodena/config/bundles/io.github.softfever.OrcaSlicer.flatpak";
        appId="io.github.softfever.OrcaSlicer";
        sha256="1p5wx6v3y6yyl43aqfh7dak9pln6461civv2z4p02srcvd6j63l0";
    }
    {
      flatpakref = "https://sober.vinegarhq.org/sober.flatpakref";
      sha256 = "sha256:1pj8y1xhiwgbnhrr3yr3ybpfis9slrl73i0b1lc9q89vhip6ym2l";
    }
  ];
}
```

- Configuration changes are mapped to state changes.
- State (current) is kept in a `flatpak-state.json` file.

State evolution

- On activation `flatpak-state.json` (current state) is compared with `configuration.nix` (target state) and the system's runtime (actual state).
- Current and target state (nix config) are merged to generate a congruent state.
- If runtime diverges, we can (optionally) enforce the desired target state.
- To keep derivations pure, state lives outside of Nix store.
- Hack: comparison between the current/future state happens in the installer script.
- A **ton** of Json manipulation done via `jq`.

Lifecycle

```
# Enforce target state. Managed and unmanaged apps an can co-exist.
services.flatpak.uninstallUnmanaged = false;
# Update on activation.
services.flatpak.update.onActivation = false;
# Update flatpaks weekly.
services.flatpak.update.auto = {
  enable = true;
  onCalendar = "weekly";
};
# Cleanup unused dependencies.
services.flatpak.uninstallUnused = true;
```

Under the hood

- Nix module (`modules/flatpak/install.nix`) generates an installer bash script.
- Config is mapped to `flatpak` commands.
- `install` / `uninstall` / `update` based on state changes.

```
/nix/store/0yaxcb46sryf6ahh7m30lnf7df1bg70i-flatpak-1.16.1/bin/flatpak --user --noninteractive ins
tall --or-update flathub com.logseq.Logseq
```

- Installer script is orchestrated via systemd:

```
$ systemctl status --user flatpak-managed-install
flatpak-managed-install.service
  Loaded: loaded (.../flatpak-managed-install.service)
  Active: inactive (dead) since Fri 2025-08-29
  Process: 2062 ExecStart=.../flatpak-managed-install
  Main PID: 2062 (code=exited, status=0/SUCCESS)
  Mem peak: 7.3G, CPU: 1min 44.675s
```

Testing

- People started using it, things started breaking.
- Using `nixpkgs`' test suite.
- Testing nix functions not enough.
- Snapshot testing automatically captures the output of a component or function and compares it against a saved reference copy to detect unexpected changes.
- State manipulation logic -> testing `jq` scripts.
- QEMU VM for integration tests and demo (`testing-base`).

Whats next

- Learning project -> stable releases.
  - 1.0 by EoY.
  - Clean up APIs.
  - Feature complete for my needs.
- Improve overrides config (PR162: WIP).
- `jq` -> ???.
- Community engagement: more visibility on prios, less attrition to contribution, help with testing non-nixos targets.
- Question: Will nix-flatpak ever be upstreamed to nixpkgs?.
  - I don't know. Governance?
  - **Hot take**: I onboarded with flakes, and I *really* like flakes.