Flakes at Work

How flakes can be utilized in a workplace environment

Luukas Pörtfors

2026-04-14

OtaNix ry 

Overview

Presentation goals

- The idea of this presentation is to showcase:
  - very practical things regarding flakes
  - parts of a flake-heavy workflow
  - useful in a company/large project where nix users are first-class citizens

Flake components

Inputs

- local directories
- private git inputs
- public git inputs
- non-flakes

Outputs

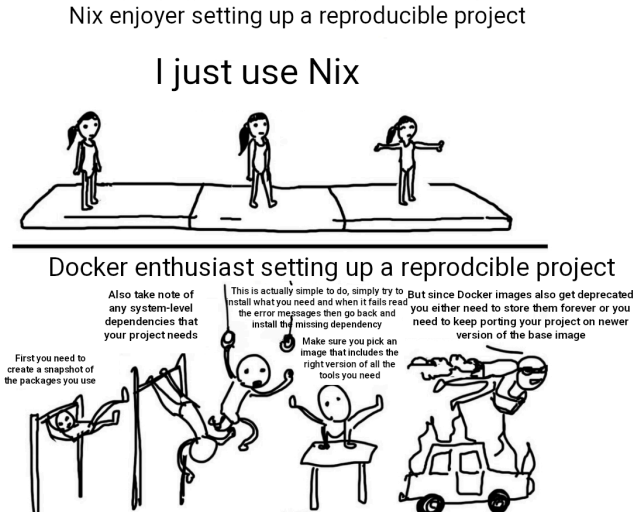
- packages
- apps
- formatters
- checks
- devShells
- overlays
- nixosModules
- nixosConfigurations
- templates

The obvious: project devShells

The most obvious, immediate benefit of nix users being first-class citizens

Setting up development environment

1. Open project
2. Run `nix develop`
3. Start working



The obvious: nix builds are reproducible

Perhaps even more obvious

Building the project

1. Install nix
2. `nix build`

Bonus: Cross-compilation is also ~~trivial~~ much easier to setup with Nix

Sharing shells with colleagues

> Colleague: hi, what tools do you use to do <thing>?
> me: `nix develop <thing-shell>`
> Colleague: this is the way.
> me: this is the way.

Running the CI on any machine

- Anti-pattern: Developing against the CI
- With nix, you can run the CI locally on your machine
- `nix build` locally will do the exact same thing as `nix build` on the CI machine
  - this has the additional benefit of making the CI script trivial (install nix, run nix build)
- `nix flake check` will run CI checks (fmt check, even NixOS integration tests 🤯)

Writing (internal) tools and scripts for everyone to use

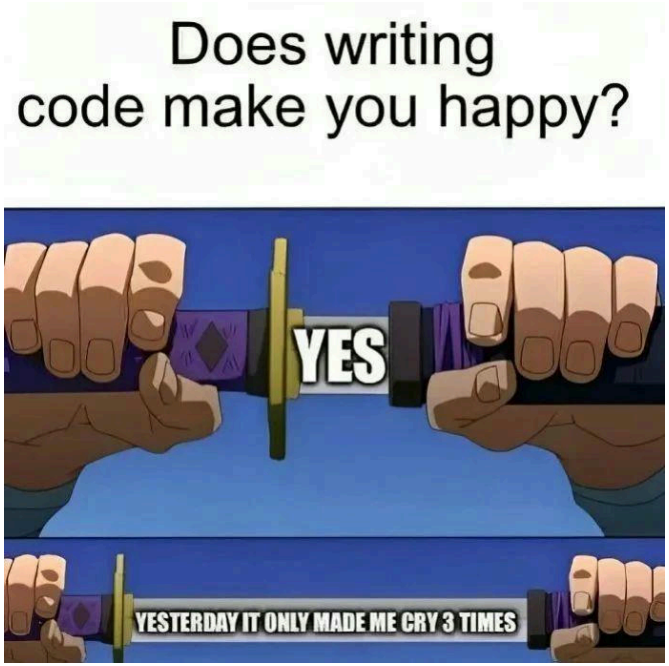
Scenario: Developer from team A wants to use an internal tool from team B

With Nix (Team B creates flake app)

1. `nix run <team-b/repo>#tool`

Without Nix

1. clone repo
2. figure out how to build the tool
3. figure out what runtime dependencies are used
4. iterate 2 & 3
5. cry



Enforcing company-wide standards with a nix library

- All company codebases can use unified components:
  - Formatters & Linters
  - Overlays
  - package generation functions
    - e.g. `buildCrate` that wraps Crane
  - could expose flake templates
- Can be maintained by the Nix gurus and used by everyone

Action plan

1. Create a devshell, (CI = `nix develop` + normal CI) ⟵ This is already great!
2. Start running builds with `nix build`
3. Move CI scripts to flake checks and flake apps

Example of a flake setup for Rust