An Introduction to Nix

Reproducible builds and deployments

Yvan Sraka

~yvan

I am currently working for Numtide (a consulting company), contributing to Rust, Nix, and Haskell upstream tooling.

2pm: Rust Hands-On for Beginner Workshop.

You can find me on Mastodon: **https://functional.cafe/@yvan**

What's on the menu?

1. **Reproducible:** no more *"but it worked on my machine"* ;
2. **Declarative:** *"one configuration that rules all"*, set up a new machine is just one `Ctrl+C`/`Ctrl+V` (or rather `git clone && nixos-rebuild switch`) away ;
3. **Immutable:** an evil program couldn't any more change your system without you even notice... Also, your system doesn't *"get old"* because of partially uninstalled programs, so you need to reinstall it...
4. **Atomic transitions:** no more *"my system is broken because power got cut while the update"*! And, if your system is broken, you can always *"restore a previous version"* in one command!

0. What's the Nix?

- The Nix **language**
- The Nix **package manager**
- The NixOS **linux distribution**
- The `nixpkgs`[1] (Nix packages) **monorepo**

[1] nixpkgs.lib is the standard library of Nix language.

Interlude: **The Nix language**

I don't have the time to teach the Nix language here... but it could be learned in 1-hour of your time. It's really a small language, you can think about it as *"JSON + functions"*. Here's an example:

```
with (import <nixpkgs> {});
mkShell {
  buildInputs = [
    pandoc
    texlive.combined.scheme-full
  ];
}
```

Advice: Learn[2] the Nix syntax and semantics before starting to copy-paste others NixOS configurations.

[2] Read https://code.tvl.fyi/about/nix/nix-1p ;)

1. **Reproducible:** *Beyond the bash script...*

Consider this script:

```
#! /usr/bin/env bash
apt update
apt install python3
python3 ./myserver.py
```

What could go wrong?

- Having not the exact same `python3` version...
- Having a different `PATH` (with a custom `python3` binary)...
- Having `python3` dynamically linked against a different `glibc`...

Interlude: But Docker is already fine! *right?*

With Docker, **what you distribute is the image** (what `docker push/pull` does) not the `Dockerfile`, and it's a big binary. **Building the image is not reproducible**[3].

[3] A workaround would be to use Alpine Linux and to pin the version of the system used.

2. **Declarative:** *I can build whatever I want with Nix!*

Nix let's you describe **derivations** *(recipes)*, that can be anything:

- A binary[4], a script ;
- A developer environment ;
- A Docker or KVM image ;
- The content of `/etc/`[5] or `$XDG_CONFIG_HOME` on my machine, e.g.:

```
programs.git = {
  enable = true;
  userName = "Yvan Sraka";
  userEmail = "yvan@sraka.xyz";
  signing.key = "370B823A2A0C7478";
  signing.signByDefault = true;
}
```

---

[4]Where to download it or its sources and how to build it.

[5]That's what NixOS options does!

Interlude: `nixpkgs` (Nix packages)

The real value of Nix lives in `nixpkgs` (more than 100 000 package[6], 10 000 contributors and 500 000 commits):

- `nixpkgs` contains package definitions, but also NixOS options, that are indexed here https://search.nixos.org ;
- `home-manager` isn't part of `nixpkgs` but let you populate `$XDG_CONFIG_HOME`[7].

The idea here is to write all your configurations in one same language (Nix).

[6] NixOS is actually the Linux distribution with the most packages availables.
[7] Which is set to $HOME/.config on most systems.

Interlude: extract of my /etc/nixos/configuration.nix:

```
  programs.zsh = {
    enable = true;
    enableCompletion = true;
    promptInit = ''
      # Best defaults are https://grml.org/zsh/
      source ${pkgs.grml-zsh-config}/etc/zsh/zshrc
    '';
  };

  programs.tmux = {
    enable = true;
    extraConfig = builtins.fetchurl {
      url = "https://git.grml.org/f/grml-etc-core/etc/tmux.co
      sha256 = "1ysb9jzhhpz160kwcf4iafw7qngs90k3rgblp04qhz5f8
    };
  };
```

3. **Immutable:** *`/nix/store/everything`*

- Everything is symbolic linking that point to the `/nix/store/` (Nix store) ;
- Only the `nix-daemon` can write the `/nix/store/` ;
- A `/nix/store/` object is called a **derivation** ;
- Derivations are garbage-collected when no symbolic link points to them ;
- What about dynamic linking? `patchelf`[8] to the rescue!

[8] A small utility to modify the dynamic linker and RPATH of ELF executables https://github.com/NixOS/patchelf

Interlude: **How does that even work?**

In previous example, `${pkgs.grml-zsh-config}` is `/nix/store/bn226m74zgwdpdnp12lq01z89wh0mzn0-grml-zsh-config`

NixOS expression can only reference `/nix/store/` objects or external resources locked with a checksum hash (e.g. with `builtins.fetchurl`). That means that we control the **complete dependencies closure of our derivation** *(recipe)*, so we control also all systems dependency it rely on[9].

Finally, **derivations are realized in a sandbox**, or in other words, recipes are built in an environment where there is no access to network, or to environment variables or to your home directory. This ensure that those builds are **fully-reproducible**.

[9] What, e.g., package-lock.json does not.

4. **Atomic transitions:** *It's magic!*

- Upgrade system, install a program is just a change in symbolic linking ;
- Grub offers you to choose at boot the generation of your system you want to use and NixOS just has to change a few symbolic links ;
- That means you can have several versions of `node` or `glibc` on your system without any conflict ;
- That allows throwable developer environment (or per-directory, e.g. with `direnv` automation)!

Drawbacks

- The macOS[10] support isn't that great.
- Flakes experimental feature, that constrains the hermeticity[11], is controversial and fragments the ecosystem.
- Complexity... there are a lot more Nix features I don't mentioned yet, like **remote builders** or **binary caches**, and a lot of concepts to grasp to build a right mental model on Nix, but arguably that's also because it's full of great features!

[10] e.g. there is no sandbox on macOS...

[11] Nix expressions could no longer read ENV variables.

Q/A