Introduction to Nix & NixOS

Nix is a declarative package manager that enables users to declare the desired system state in configuration files (declarative configuration), and it takes responsibility for achieving that state.

> In simple terms, "declarative configuration" means that users only need to declare the desired outcome. For instance, if you declare that you want to replace the i3 window manager with sway, Nix will assist you in achieving that goal. You don't have to worry about the underlying details, such as which packages sway requires for installation, which i3-related packages need to be uninstalled, or the necessary adjustments to system configuration and environment variables for sway. Nix automatically handles these details for the user (provided that the Nix packages related to sway and i3 are properly designed).

NixOS, a Linux distribution built on top of the Nix package manager, can be described as "OS as Code." It employs declarative Nix configuration files to describe the entire state of the operating system.

An operating system consists of various software packages, configuration files, and text/binary data, all of which represent the current state of the system. Declarative configuration can manage only the static portion of this state. Dynamic data, such as PostgreSQL, MySQL, or MongoDB data, cannot be effectively managed through declarative configuration (it is not feasible to delete all new PostgreSQL data that is not declared in the configuration during each deployment). Therefore, **NixOS primarily focuses on managing the static portion of the system state in a declarative manner**.

Dynamic data, along with the contents in the user's home directory, remain unaffected by NixOS when rolling back to a previous generation.

Although we cannot achieve complete system reproducibility, the `/home` directory, being an important user directory, contains many necessary configuration files - Dotfiles. A significant community project called home-manager is designed to manage user-level packages and configuration files within the user's home directory.

Due to Nix's features, such as being declarative and reproducible, Nix is not limited to managing desktop environments but is also extensively used for managing development environments, compilation environments, cloud virtual machines, and container image construction. NixOps (an official Nix project) and colmena (a community project) are both operational tools based on Nix.

Why NixOS?

I first learned about the Nix package manager several years ago. It utilizes the Nix language to describe system configuration. NixOS, the Linux distribution built on top of it, allows for rolling back the system to any previous state (although only the state declared in Nix configuration files can be rolled back). While it sounded impressive, I found it troublesome to learn a new language and write code to install packages, so I didn't pursue it at the time.

However, I recently encountered numerous environmental issues while using EndeavourOS, and resolving them consumed a significant amount of my energy, leaving me exhausted. Upon careful consideration, I realized that the lack of version control and rollback mechanisms in EndeavourOS prevented me from restoring the system when problems arose.

That's when I decided to switch to NixOS.

To my delight, NixOS has exceeded my expectations. The most astonishing aspect is that I can now restore my entire i3 environment and all my commonly used packages on a fresh NixOS host with just one command `sudo nixos-rebuild switch --flake .` . It's truly fantastic!

The rollback capability and reproducibility of NixOS has instilled a great deal of confidence in me—I no longer fear breaking the system. I've even ventured into experimenting with new things on NixOS, such as the hyprland compositor. Previously, on EndeavourOS, I wouldn't have dared to tinker with such novel compositors, as any system mishaps would have entailed significant manual troubleshooting using various workarounds.

As I get more and more involved with NixOS and Nix, I find it also very suitable for synchronously managing the configuration of multiple hosts. Currently my personal nix-config synchronously manages the configuration of many hosts:

- Desktop computers
  - 1 Macbook Pro 2020 (Intel amd64).
  - 1 Macbook Pro 2022 (M2 aarch64).
  - 1 NixOS desktop PC (amd64).
- Servers
  - 3 NixOS virtual machines (amd64).
  - Several development boards for aarch64 and riscv64.

The development environment of three desktop computers is managed by Home Manager, the main configuration is completely shared, and the configuration modified on any host can be seamlessly synchronized to other hosts through Git.

Nix almost completely shielded me from the differences between OS and architecture at the bottom of the three machines, and the experience was very smooth!

7 reactions

👍 7

0 comments – *powered by giscus*