Overview (Nix From First Principles: Flake Edition #1)

Oct 05
2022

#Nix

This is part 1 of the Nix from First Principles: Flake Edition series.

One technology that I'd been hearing more and more over the last two years has been Nix, in its various forms. The big draws I've heard about include declarative, portable, cross lang dependency specification, and the ability to have a config file declare the entire state of a system without storing huge docker container artifacts for every permutation.

Actually learning Nix though felt much harder than it needed to be. The biggest problem is that the documentation you need is often fragmented among the different subprojects, and also the transition that Nix is in at the moment from the current standard methods of `default.nix`, `configuration.nix` and `shell.nix` to a new world of `flakes`.

As an outsider looking in, the flake world felt kind of in limbo at the moment where anyone I talked to indicated that flakes fix so many of the problems with current nix, but also they're still officially experimental so the official documentation declines to mention flakes.

After spending a fair amount of time understanding Nix for myself, I've decided to write my own guide which explains it from scratch in a world using only the new Nix features, with the `nix` CLI and flakes for the use cases I want from Nix. Credit to ianthehenry for his guide on Nix focused on the "old world" approach, and Xe Iaso for explaining flakes. The information included in this guide is largely from synthesising their blogs and the official documentation.

This first post will describe the major parts of the Nix ecosystem.

So what is Nix, anyway?

At its core Nix is a technology for dealing with package management, primarily software packages. A software package can be a program in its own right, or some library that allows other programs or libraries to work. Like most package managers, Nix will also manage the dependencies of a program. For example if package X requires lib Y, obtaining package X will also obtain lib Y.

One of the common problems with understanding Nix is the many projects that are under the umbrella and how they interact. Depending on your use case you may use a subset of these. A brief overview of the moving pieces are.

- *Nix (Package Manager)* - The Nix package manger is the one piece of the Nix stack that you're going to using regardless of which use case you're hoping to tackle. This is the software application that actually turns your configuration or command line arguments into built software applications.
- *Nix (language)* - The Nix programming language, sometimes called Nixlang to distinguish from the package manager is a functional programming language which is the normal way of configuring Nix packages and NixOS systems. If you have some

familiarity with functional language concepts like currying or let-bindings then it will be familiar to you, but if you're more used to mainstream languages (including ones like Python or Java which have taken some functional concepts but not all), there will be a steeper learning curve.

- *Nixpkgs* - Nixpkgs is effectively the standard library for Nix. It's packaged seperately to the package manager and library because a given nix language program can choose to import different versions of Nixpkgs, which makes it a bit more like the Golang standard library than others where your version of the library is tied to the language. It contains both the official definitions of a wide variety of software, along with nix language functions useful in writing your own Nix code.
- *NixOS* - NixOS is a Linux distribution which is built upon the Nix package manager. This means that the usual way of configuring it is writing a program in Nix (the language) to be consumed by Nix (the package manager). You could also use the procedural commands of the Nix package manager and have an experience closer to a standard Linux distribution.

There are a number of adjacent projects like NixOps for deploying to remote NixOS systems or nixos-generators for producing images from Nix config files which I'll touch on later, but these four are the official core.

Next time, I'll discuss installing Nix.