Frequently Asked Questions

General

Are there any downsides?

Yes. Here are some of the most common issues that people run in today:

- **Poor user experience:** While the *concepts* behind Nix are great and could make system management a lot easier, the current generation of Nix tools can still be rather awkward to use. A lot of user experience kinks haven't been quite worked out yet, although this is a problem that's being worked on, and solvable in the long term.
- **Poor documentation:** The documentation for Nix and associated tools isn't great yet. This wiki is a (hopefully successful!) attempt at improving that, but you'll most likely still run into quite a few 'documentation deserts' while using Nix, especially when just getting started. Please do contact me about documentation you're missing - it'll help me improve the wiki!
- **Incompatibility with other Linux systems:** This is a less solvable problem, mostly specific to NixOS. To be able to provide the guarantees that it does, Nix makes some unusual decisions in how it structures your system. This means that, for example, `/usr` doesn't exist - and neither do many of the other 'well-known locations' that tools and users alike expect.
  This isn't a problem for packaged software, but it means that random 'static Linux binaries' downloaded from the internet won't work on NixOS out of the box. This includes AppImages, and many proprietary games. Some workarounds exist (in the form of `steam-run` and `appimage-run`), but it's still not the click-and-run experience that you get on other distributions.
  Similarly, configuration files aren't in a big bucket that you can edit at will - it's not possible to edit configuration files from outside of your Nix configuration, and various other such limitations exist. This isn't *necessarily* a problem (since you can still modify configuration files through your Nix configuration, for example), but it means you will have to adjust your habits and essentially re-learn a lot of Linux things.
  While work is ongoing on reducing the impact of these differences to end users, NixOS will never work *exactly* like the distributions you're used to - the model that those distributions use is fundamentally incompatible with the guarantees that Nix provides.

Should I use Nix?

That depends. Given how different Nix is from 'traditional' package management systems and Linux distributions, it will take quite some time to learn how all of it works.

If you decide to use Nix on another distribution, then you can do this while you continue using your other package manager, so you won't get 'blocked' on not understanding something about Nix yet. However, the benefits you get from Nix will also be limited to *just* package management.

If you decide to use NixOS, then you can use the full range of system management niceties available - but there won't be an 'escape hatch' when you're not sure how to do something, so this is a somewhat steeper learning curve. You should expect to be figuring things out for at least a week, when taking this route.

It really all comes down to this: you'll have to spend effort upfront to learn about how Nix works, and that can be quite a bit of work. But in the long term you'll get a much more reliable system out of it, and it'll make your life easier. Whether you're willing to make that tradeoff, is up to you.

The Nix language (nixlang)

Why a custom language? Why not use something that already exists?

Most languages are one of roughly two types:

- **Declarative:** These are languages like JSON, YAML, XML, TOML, and so on. They're typically very simple, only supporting statically specified data or maybe really basic references, without any logic. They're especially suited for serialization, and for cases where you need to extensively inspect the information written in it, like in configuration files.
- **Imperative:** These are what you might think of when you hear 'programming language'; Python, C++, JavaScript, Rust, Lua, and so on. They typically specify a sequence of instructions that modify state and might produce output. They're especially suited towards implementing arbitrary logic, like in software, but it's hard to 'inspect' the state of the application.

The problem is that Nix needs a little bit of both. It's mostly data-oriented - a system configuration is data, package metadata is data - but it's also about data that's complex enough and sometimes repetitive enough that you want to be able to use logic to *produce* it.

nixlang is precisely that - a simple and mostly-declarative language with enough abstraction that you can programmatically generate data, but not so much that it becomes difficult to inspect.

One particularly unique aspect of it is that it's lazily-evaluated; that is, instead of executing some logic and storing the result in a variable, you store the *logic itself* in a variable, and the first time that variable is accessed, the logic is executed and the result remembered for future access. This is what makes it possible to have tens of thousands of packages in a Nix package set, without needing to execute *all* logic every time you need a single package.

Revision #1
Created 2024-12-11 19:06:00 UTC by joepie91
Updated 2024-12-11 21:27:58 UTC by joepie91