The Future of Nix

Eelco Dolstra

2018-02-02

History of the Nix project

- **2003:** First commit
- **2004:** First papers on Nix
- **2006:** Nix PhD thesis
- **2006:** NixOS (Armijn's MSc thesis)
- **2007:** First Ludovic commit
- **2008:** NixOS module system
- **2011:** NixOps
- **2012:** Nix 1.0
- **2015:** First NixCon
- **2017:** Second NixCon
- **2018:** Nix 2.0 (promise!)

Is the purely functional approach still relevant in the age of Docker?

Is the purely functional approach still relevant in the age of Docker?

No

- In a Docker image, you don't care about version conflicts etc.
- Building an image in an "ad hoc" way using a shell script that calls cargo, npm, ... is more convenient than writing a Nix expression where you have to figure out how those things map to Nix.

Is the purely functional approach still relevant in the age of Docker?

Yes

- Shell scripts that download random stuff from the Internet are not very reproducible.
- Proper configuration management for the entire dependency graph of your system is still useful. E.g. rebuilding your entire system with the retpoline patch is easy with Nix.
- For development, nix-shell more convenient than Docker.
- Containers are actually mostly a crutch to work around package management limitations. For stuff like network/pid/filesystem/... isolation we don't need Docker, namespaces are enough.
- De-dup still useful. In Nix you get that for free, no need for wacky filesystems.

Turing completeness considered harmful?

- Nix expression language is Turing complete, so...
- Have to evaluate a possibly non-terminating program to get any useful info.
- No bounds on CPU, memory usage.
- Should we use something like Dhall instead?

Abstraction considered harmful?

- Everybody invents their own abstractions.
- Makes it harder for people to understand / modify a package.

NixOS Module system

- Not very functional: you'd expect that you can "call" the PostgreSQL module twice to get two PostgreSQL instances, but you can't.
- No POLA: Any module can modify anything else.
- We suddenly forgot that Nix is a DSL.

DSL deficiencies

- Surprisingly for a package manager, the Nix language has no concept of a “package”.
- ... or of “package options”, “plugins”, “configurations”, ...
- Leads to suboptimal UI/UX: e.g. Nixpkgs has some package configuration mechanisms, but `nix-env` doesn’t expose them in a useful way.

Our nemesis: Language-specific package managers

- Modern languages all have their own build tools that fetch dependencies automatically from repositories on the Internet.
- This collides with the package manager.
- Solution: write language-specific tools that translate those package repositories to Nix expressions.
- Downsides:
  - Leads to Nixpkgs repo explosion.
  - Won't be up to date.
  - Extra steps needed to build things in Nix.

Reproducibility

- The holy grail: binary reproducibility.
- This will enable content-addressable Nix stores.
- Also: eval-level reproducibility, where a single Git commit hash uniquely identifies a NixOS configuration (“nix –pure-eval”).

Nix 2.0

- New CLI
  - Cleaned up, consistent interface
  - Progress indicator
  - Work in progress
- Store abstraction: replaces substituters, `nix-copy-closure`, `nix-push`, ...
- Chroot/namespace support
- Paths have signatures
- `builtins.fetchGit`
- Structured derivation attributes
- ...

Roadmap

- Binary reproducibility
- Content-addressable Nix store
- Eval-level reproducibility (“nix –pure-eval”)
- Declarative package management
- nixos-shell
- Nixpkgs scalability
- Recursive Nix
- Security / private paths
- Distributed Hydra
- Better DSL