Sandstone

Fast incremental Haskell builds using dynamic derivations

Planet Nix 2025

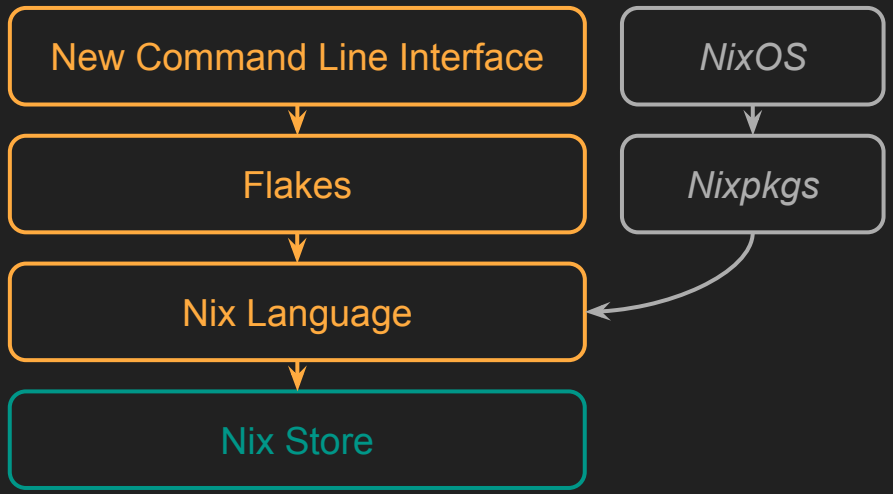
JOHN.ERICSON@OBSIDIAN.SYSTEMS

First things first — Thank you, Luigy!

- Luigy Leon wrote sandstone with me
- Couldn’t make it today

Recap

Nix's Layers — NixCon 2022

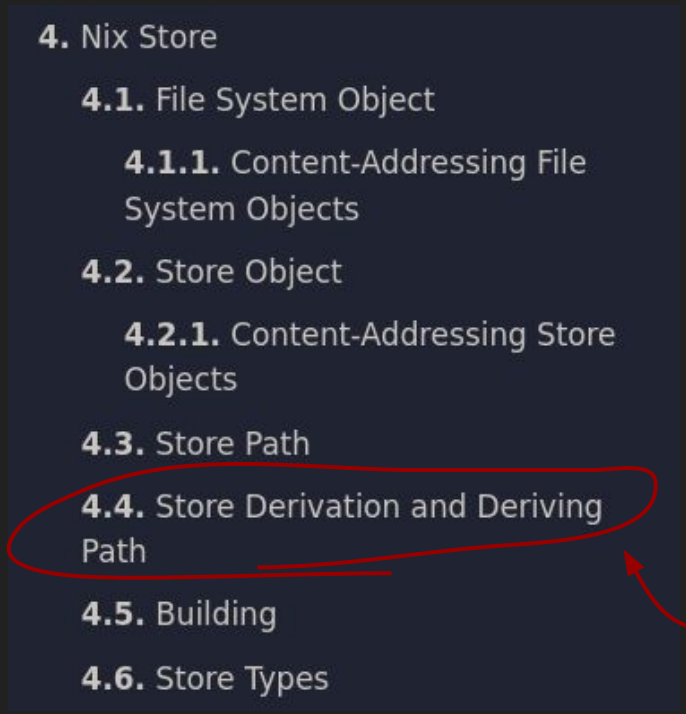


- Video: https://www.youtube.com/watch?v=utQf4VH7QMw
- Slides: https://docs.google.com/presentation/d/e/2PACX-1vSpc52s5KmWWXMvIpNXEfDVadrC7wT4SkcBaqKVcMxBVgKDQnzwZ-5FixbUpJIxWLHTjsSZix2pFtgI/pub

Now officially documented!

- Newish chapter in the manual on `master` https://nix.dev/manual/nix/development/
- Even more content coming soon!

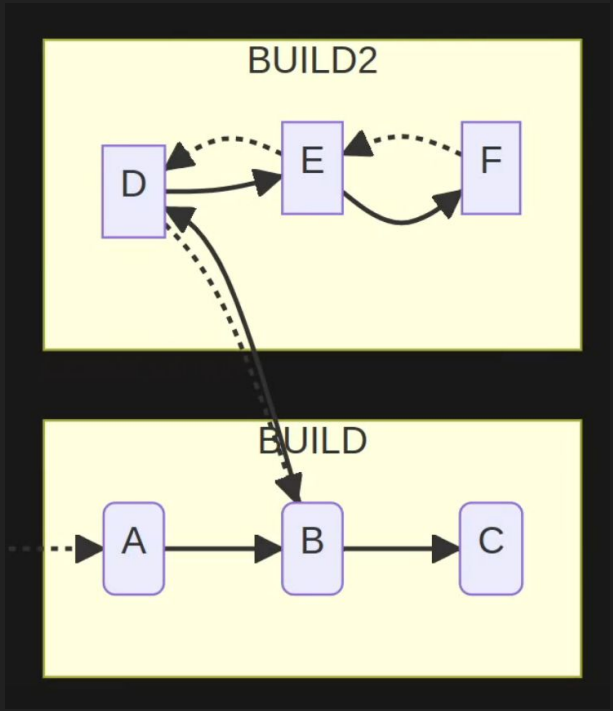
Now officially documented!



- Newish chapter in the manual on `master` https://nix.dev/manual/nix/development/
- Even more content coming soon!

dynamic derivations

Dynamic derivations — 2023 NixCon



- Thanks Tom Bereknyei!
- Video: https://media.ccc.de/v/nixcon-2023-36394-dynamic-derivations-what-and-why

Today’s talk

- I’ve been working on these things “breadth-first” for a while
- Time to show something end-to-end

Why

Why dynamic derivations

- Nix is a build system
- The Nix community doesn’t yet think of Nix that way
- Nix needs to to earn it

Core plank of my steering committee candidacy
(https://github.com/NixOS/SC-election-2024/blob/main/candidates/Ericson2314.md)

Why Sandstone

- Need to demonstrate how dynamic derivations are used
- Then *you all* write 10s more “sandstones” for other languages!
- Haskell is a fine place to begin
  - It is dear to me :)
  - Things about GHC make for good example

Let’s dive in

Quick GHC Background 1

- Source files: `*.hs`
- Compiler generates interface files: `*.hi`. Similar to:
  - C/C++ "precompiled headers"
  - The non-machine-code parts of Rust's `*.rlib`s
- Compiler also generates `*.o` files
  - Just regular machine code and data, not Haskell specific
- `.o` files archived/linked at the end
  - Also normal

Quick GHC Background 2

- If `import A` in module `B`
- Need `A.hi` to compile `B.{hi,o}`
- Link `A.o` and `B.o` together

Quick GHC Background 3

- GHC has `-M` flag to generate Makefile
- Makefile encodes `import` graph
- `.hi` of imported modules are makefile deps

Sandstone architecture

Simple!

1. Parse `ghc -M` Makefile
2. Translate graph to derivations

Done

Sandstone *how* 1

Use Nix CLI

- `nix store add` to add source file to store
- `nix derivation add` to add derivation to store
  - Uses JSON
  - No Nix language!
  - No ATerm!

Sandstone *how* 2

Could send raw JSON to CLI but…

- Serialization logic clutters core
- Types are nice!

Sandstone *how* 3

Leverage `hnix-store`!

- github.com/haskell-nix/hnix-store
- Fairly complete set of
  - Data types for core Nix Store interface
  - Parsing/printing logic
- Bonus: Also a daemon protocol implementation
  - Future work: no Nix CLI shell-out for perf!

Let's see the code!

Wait, what about dynamic derivations?

Didn't actually use yet!

So what is dynamic derivations?

- *Not* the ability to add derivations to the store
  - Just using "recursive Nix" for that
  - (In the future, want extra-restricted recursive Nix)
- Yes *depending* on derivations that are themselves outputs

Rolling it out

Priorities

I've been thinking lately about *finishing* experiments rather than *starting* them

First comes CA…

- One more change: "shallow" build traces
  - Otherwise cache sketches me out
  - Currently in progress
- Roadmap:
  - Shallow build traces for Nix
  - Shallow build traces for Hydra
  - (I hope) provisional try on `hydra.nixos.org`
  - (I hope) provisional try in Nixpkgs
  - Iterate, stabilize (RFC), and ship!

…Then comes dynamic derivations

- Roadmap
  - Hydra support
  - (I hope) `hydra.nixos.org` try out
  - (I hope) provisional try in Nixpkgs
  - Iterate, stabilize (RFC), and ship!

Ecosystem support

- Build systems gain Nix backend?
  - Cabal use Sandstone?
  - Cargo use equivalent?
  - Meson
  - CMake
- Focus on big builds in Nixpkgs
  - Chromium
  - Linux
- Focus on orgs with developers using Nix in CI
  - That's many of you!

Thank you!