



Nixology

Yifei Sun

July 1, 2024

Problem

「\ (ツ) /」

IT WORKS
on my machine

Solution

Functions:

```
{ inputs = { ... }; }
```

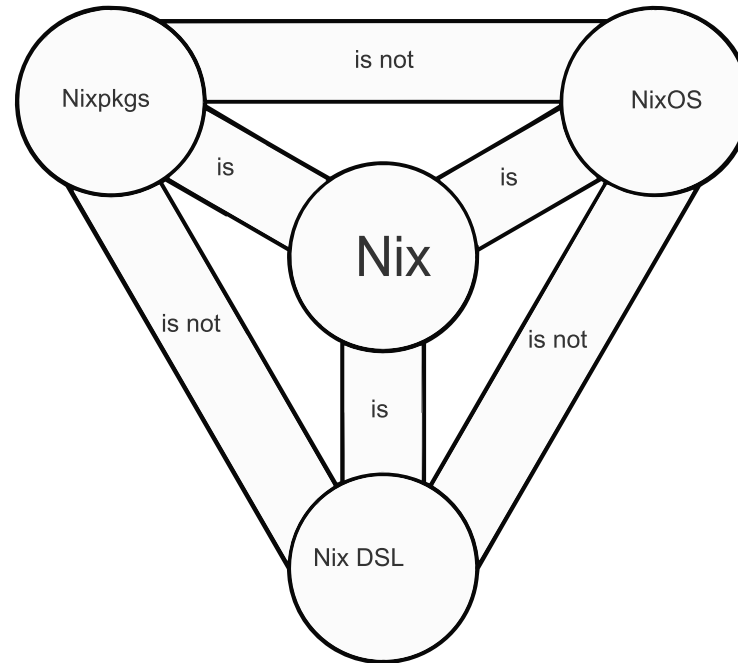
- Dependencies are inputs
- Usually tarballs or git repos
- Pinned and hashed

```
{ outputs = inputs: { ... }; }
```

- Outputs are functions of inputs
- Can be anything
- Lazily evaluated

Trinity

- Nix - the package manager
- Nix - the DSL
- Nixpkgs - the package collection
- NixOS - the operating system



Language Basics

Integers:

```
> x = 1 + 1  
> x  
2
```

Floats:

```
> y = 1.0 + 1.0  
> y  
2.0
```

Strings:

```
> z = "world"  
> "hello ${z}"  
"hello world"
```

Attribute sets:

```
> s = { a = { b = 1; }; }  
> s.a.b  
1
```

Language Basics

Lists:

```
> [ 1 "2" (_: 3) ]  
[ 1 "2" <thunk> ]
```

Recursive attrsets:

```
> rec { x = 1; y = x; }  
{ x = 1; y = 1; }
```

Bindings:

```
> let x = 1; in x + 1  
2
```

Inherits:

```
> let x = 1; y = x; in  
    { inherit x y; }  
{ x = 1; y = 1; }
```

Language Basics

Functions 1:

```
> f = x: x + 1
```

```
> f 2
```

```
3
```

```
> g = g': x: g' x + 1
```

```
> g f 2
```

```
4
```

Functions 2:

```
> h = { x ? 1 }: x + 1
```

```
> h
```

```
<function>
```

```
> h { }
```

```
2
```

```
> h { x = 2; }
```

```
3
```

Derivation

A derivation

- is plan / blueprint
- it's used for producing
 - lib: library outputs
 - bin: binary outputs
 - dev: header files, etc.
 - man: man page entries
 - ...

```
derivation ::  
  { system      : String  
    , name      : String  
    , builder    : Path | Drv  
    , ? args     : [String]  
    , ? outputs  : [String]  
  } -> Drv
```


Derivation

Example:

```
derivation ::  
  { system      : String  
  , name        : String  
  , builder     : Path | Drv  
  , ? args      : [String]  
  , ? outputs   : [String]  
  } -> Drv
```

```
derivation {  
  system = "aarch64-darwin";  
  name   = "hi";  
  builder = "/bin/sh";  
  args   = ["-c" "echo hi >$out"];  
  outputs = ["out"];  
}
```

Derivation

Special variables:

```
derivation {  
    system = "aarch64-darwin";  
    name = "hi";  
    builder = "/bin/sh";  
    args = ["-c" "echo hi >$out"];  
    outputs = ["out"];  
}
```

- \$src: build source
- \$out: build output (default)
- custom outputs

Nix Store

```

/nix/store/l2h1lyz50rz6z2c8jbni9daxjs39wmn3-hi
|-----|-----|
store      hash                                     name
prefix

```

- Store prefix can be either local or remote (binary cache)
- Hash either derived from input (default) or output (CA derivation)
- The hash ensures two realised derivations with the same name have different paths if the inputs differ at all

Packaging

The process of: Nix expressions \Rightarrow derivation(s)

- `builtins.derivation`
- `stdenv.mkDerivation` (from `nixpkgs`)
- `pkgs.buildGoApplication` (from `nixpkgs`)
- ...

Packaging¹

```
{
  inputs = { ... };

  outputs = { self, nixpkgs, flake-utils }:
    flake-utils.lib.eachDefaultSystem (system:
      let
        pkgs = nixpkgs.legacyPackages.${system};
      in
      {
        packages.default = pkgs.writeShellApplication {
          name = "moo";
          runtimeInputs = [ pkgs.cowsay ];
          text = "cowsay moo";
        };
      });
}
```

```
< moo >
-----
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
              ||----w |
              ||     ||
```

¹Example 1

Development²

Shell:

- nix develop
- direnv

```
devShells.default = pkgs.mkShell {  
  packages = with pkgs; [  
    cargo  
    rustc  
    rustfmt  
  ];  
};
```

Formatter:

- nix fmt
- a single package, or ↓

```
formatter = pkgs.writeShellScriptBin "formatter" ''  
  set -eoux pipefail  
  shopt -s globstar  
  ${pkgs.nixpkgs-fmt}/bin/nixpkgs-fmt .  
  ${pkgs.rustfmt}/bin/rustfmt **/*.rs  
'';
```

²Example 2

Pinning

w/ builtin versions:

```
nix-repl> pkgs.coq_8_  
pkgs.coq_8_10  pkgs.coq_8_12  
pkgs.coq_8_14  pkgs.coq_8_16  
pkgs.coq_8_18  pkgs.coq_8_5  
pkgs.coq_8_7   pkgs.coq_8_9  
...
```

w/ nix shell:

```
nix shell nixpkgs/<hash>#{pkg1,...}
```

or DIY!

w/ flakes:

```
inputs = {  
  nixpkgsForA.url = "github:nixos/nixpkgs/<branch or hash>";  
  nixpkgsForB.url = "github:nixos/nixpkgs/<branch or hash>";  
  ...  
};  
  
outputs = { self, ... }: {  
  ...  
  pkgsA.<some pkg>;  
  pkgsB.<some pkg>;  
  ...  
};
```

Build System³

Example: eJS

- Combine multiple build tools (ant, make, ...)
- Build multiple targets (binary target, jar, ...)
- Wrap programs
- ...

³Example 3

System Configuration⁴

i.e. NixOS

```
outputs = { nixpkgs, ... }: {  
  nixosConfigurations.test = nixpkgs.lib.nixosSystem {  
    modules = [ /* a list of modules goes here */ ];  
  };};
```

System Closure:

```
nix build .#nixosConfigurations.test.config.system.build.toplevel
```

Rebuild:

```
nixos-rebuild <switch|boot|...> --flake .#test
```

⁴Ugawa lab infra

Resources

- Installer: <https://github.com/determinatesystems/nix-installer>
- REPL is your friend: `nix repl`
- Intro: <https://zero-to-nix.com>
- Manual: <https://nixos.org/manual/nix/unstable/>
- Forum: <https://discourse.nixos.org>
- Options: <https://mynixos.com>
- Source code search:
 - <https://github.com/features/code-search>
 - <https://sourcegraph.com>