Reflections on the Nix DSL

Symposium on Build Systems

Eelco Dolstra

2019-11-18

tweag.io

Nix

- Nix is a purely functional build system / package manager
  - Reproducible
  - Declarative
  - Transactional
- Started in 2003 at Utrecht University
- NixOS: Linux distribution based on Nix
  - NixOS 19.09: 960 contributors, 22,000 commits

Nix store

Main idea: store all packages in isolation from each other:

`/nix/store/rpdqxnilb0cg...-firefox-70.0`

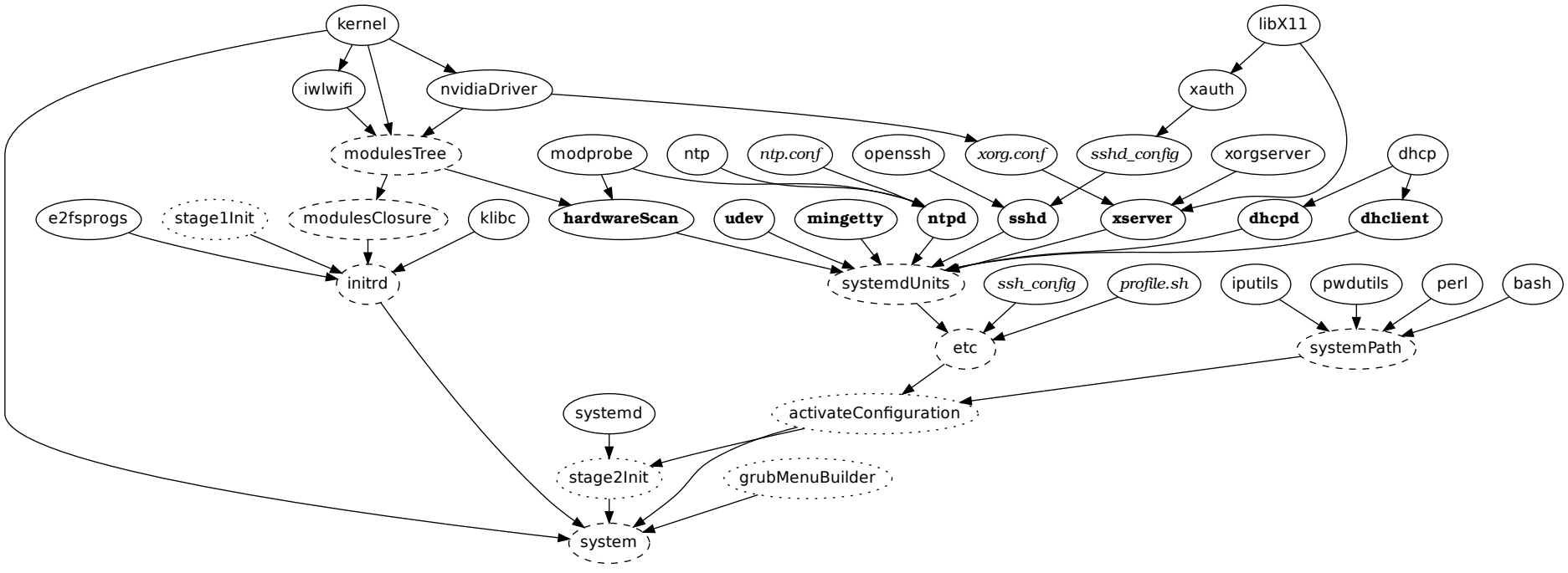
Paths contain a 160-bit **cryptographic hash** of **all** inputs used to build the package:

- Sources
- Libraries
- Compilers
- Build scripts
- ...

```
/nix/store
├─ l9w6773m1msy...-openssh-7.0
│  ├─ bin
│  │  └─ ssh
│  └─ sbin
│     └─ sshd
├─ smkabrbibqv7...-openssl-1.0.
│  └─ lib
│     └─ libssl.so.1.0.0
├─ c6jbqm2mc0a7...-zlib-1.2.8
│  └─ lib
│     └─ libz.so.1.2.8
└─ im276akmsrhv...-glibc-2.21
   └─ lib
      └─ libc.so.6
```

The Nix DSL

The goal of the Nix DSL is to specify a DAG of build actions:



To DSL or not to DSL?

Your options:

- No language: JSON, YAML, TOML, XML, ATerm, ...
- A simple DSL
- A rich DSL
- A general-purpose language

Example

```
openssh = stdenv.mkDerivation {
  name = "openssh-7.0p1";
  src = fetchurl {
    url = http://.../openssh-7.0p1.tar.gz;
    sha256 = "0fpjlr3bfind0y94bk442x2p...";
  };
  buildCommand = ''
    tar xjf $src
    ./configure --prefix=$out --with-openssl=${openssl}
    make; make install
  '';
}

openssl = stdenv.mkDerivation { ... };
```

Nix DSL features

- Pure, lazy
- Dynamically typed
- A few datatypes: lists, hashmaps
- Functions (lambdas): `buildRustPackage = args: ...`
- Higher-order functions: `map`, ...
- No module system: files are just expressions

Functions

docker/default.nix:

```
{ buildPythonPackage, fetchPypi, six, requests }:

buildPythonPackage rec {
  version = "4.0.2";
  pname = "docker";

  src = fetchPypi {
    inherit pname version;
    sha256 = "0r1i46h8...";
  };

  buildInputs = [ six requests ... ];

  meta = {
    description = "An API client for docker written in Python";
    homepage = https://github.com/docker/docker-py;
  };
}
```

The good

Laziness, antiquotations, string contexts

```
writeText "ssh_config" ''
  SendEnv LANG LC_ALL ...
  ${if config.services.sshd.forwardX11 then ''
    ForwardX11 yes
    XAuthLocation ${pkgs.xorg.xauth}/bin/xauth
  '' else ''
    ForwardX11 no
  ''}
''
```

The bad

- Lack of useful domain abstractions: packages, configurations, modules, overrides, ...
- This causes
  - Inconsistency
  - Inconvenient syntax
  - Inconvenient semantics
  - Inefficiency

Example: package overrides

Nixpkgs functions return a `.override` attribute in their output that allows the function to be called again with different arguments:

```
buildInputs =
  [ (openssl.override {
      stdenv = clangStdenv;
    })
  ];
```

Makes garbage collection ineffective.

Example: NixOS modules

NixOS modules return a nested hashmap of attributes that when composed together define a Linux system.

```
{ config, ... }:

{
  users.users.eelco = {
    description = "Eelco Dolstra";
    extraGroups = [ "wheel" ];
    openssh.authorizedKeys.keys = [ "ssh-ed25519 AAAAC3Nz
  };
  services.sshd.enable = true;
  services.sshd.forwardX11 = true;
}
```

Example: NixOS modules

```
{ config, pkgs, ... }:

{
  config = mkIf config.services.sshd.enable {
    networking.firewall.allowedTCPPorts = [ 22 ];
    systemd.services.openssh = {
      description = "SSH Daemon";
      serviceConfig.ExecStart =
        "${pkgs.openssh}/bin/sshd -f ${writeText ...}'';
    };
  };
```

Example: Nixpkgs overlays

```
final: prev: {
  firefox = prev.firefox.override {
    stdenv = clangStdenv;
  };
}
```

Abstraction considered harmful?

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

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.

What are the alternatives?

- Less expressiveness: e.g. YAML, CUE
- Use a general-purpose language: guix

Conclusion

- Nix DSL has served us well, but...
- Lisp Curse: too general-purpose
- while at the same time not having the convenience of a proper general-purpose language