A brief introduction to Nix & NixOS

May 23, 2017

Contents

Self Intro

- digital IC design engineer(aka *frontend*), day job
- GNU/Linux user
  - Debian(~0.5y)
  - Gentoo(>10yr)
  - NixOS(~1yr)
- maintainer(one of) of SHLUG

What is Nix & NixOS

```
The Purely Functional Package Manager
                        --- http://www.nixos.org/nix

The Purely Functional Linux Distribution
                        --- http://www.nixos.org
```

Nix

Nix is a powerful package manager for Linux and other Unix systems that makes package management reliable and reproducible. It provides atomic upgrades and rollbacks, side-by-side installation of multiple versions of a package, multi-user package management and easy setup of build environments. Read more...

NixOS

NixOS is a Linux distribution with a unique approach to package and configuration management. Built on top of the Nix package manager, it is completely declarative, makes upgrading systems reliable, and has many other advantages.

Nix Expression Language

The Nix expression language is a pure, lazy, functional language.

values

```
1234

"single line string"

''
  multiline
  string
''

true

false

[1 2 3 4]

{ a = 2; b = "zz"; x = { y = 1; z = 2; }; }
```

simple values

- boolean
  - true
  - false
- integer
  - 1234
  - -2345
  - 0

simple values

- string
  - between double quotation

    ```
    "this is string"
    "a two
       line string"
    ```

  - *indented string*, between two single quotes

    ```
    ''
      1. first line
      2. second line
    ''
    ==> "1. first line\n2.second line"
    ```

  - URI, `https://example.com/`

simple values

- path
  - /dev/null
  - ../tmp
  - ~/Desktop
- null

lists

- enclose with '[' & ']', whitespace separated
- lazy in values, strict in length(not actually the lazy linked lists)

sets

- enclose with '{' & '}', always terminal with ';' for each "assignment"
- is list of key/value pair(*attribute*)
- core of Nix language
  - we use Nix to describe *derivation*
  - *derivation* is just a set of attributes to decribe the build process

antiquotation

- let a = "abc"; in "a = ${a}"
- let b = { c = 123; }; in "b = ${toString b.c}"
- let x = "a"; y = { a = 1; }; in y.${x}

language constructs

recursive set

- rec { a = b; b = 3; }
- rec { x = y; y = [1] ++ x; } (ERROR!)
- [rec { x = y; y = [1] ++ x; }] (works)

function

- a: a + 1
- a: b: a + b
- a: {x = a + 1; y = a - 1;}
- (a: a 2) (a: 4 / a)

and more ...

- let
- with
- inheriting attribute
- conditional
- assertion

[1] DEFINITION NOT FOUND.

operators

- *e.*/attrpath/ [or /def/]
  - {a = 1;}.a
  - {"a b" = 2;}."a b"
  - {c = 3;}.d or 4
- *e1 e2*
- *e ? attrpath*
  - {a = 1;} ? a
  - {a = 1;} ? b

operators

- *e1 ++ e2*
  - [1] ++ [2]
- *e1 + e2*
  - 1 + 2
  - "a" + "b"
  - /home + /a/b
- //
  - {a = 2; b = 3;} // {b = 4;}
- and more
  - !, `=, !`, &&, ||, ->

built-in function

- builtins.head
- builtins.div
- builtins.toJSON, builtins.fromJSON

[2] DEFINITION NOT FOUND.

- builtins.readFile, builtins.toFile
  - builtins.toFile "test.txt" "file content"
- builtins.fetchurl
  - builtins.fetchurl "`http://www.baidu.com`" (changed every fetch)
  - builtins.fetchurl "`http://example.com`" (keep unchanged)

built-in function

- import
  - import ./a.nix
  - import ./hello
  - import <nixpkgs> (*search `$NIX_PATH`*)
- derivation
- and more...

derivation

A derivation is a build action, which return by built-in function *derivation*. The function take a set to describe the build process.

non-optional attributes of *derivation* function input

- *system* (eg. ="$x86_{64}$-linux"=)
- *name*, string
- *builder*, derivation or source(local reference, like *./build.sh*)
  - derivation => if output is an executable

attribute translation

All attributes are pass to builder as environment variable

- string & integer just passed verbatim
- *path* will copy into store, and return the location
- *derivation* will be built before presentat derivation, return output path

- lists of above type is allowed, and will simply concatenated, with space
- *true* is pass as string 1, *false* and *null* pass as empty string

optional attributes

- args
- outputs

mkDerivation

- set *system* to current system
- always use bash as builder

example

```
{ stdenv, fetchurl, perl }:

stdenv.mkDerivation {
  name = "hello-2.10";

  builder = builtins.toFile "builder.sh" "
    source $stdenv/setup

    PATH=$perl/bin:$PATH

    tar xvfz $src
    cd hello-*
    ./configure --prefix=$out
    make
    make install
  ";

  src = fetchurl {
    url = http://mirrors.163.com/gentoo/distfiles/hello-2.8.tar.gz;
    sha256 = "0wqd8sjmxfskrflaxywc7gqw7sfawrfvdxd9skxawzfgyy0pzdz6";
  };
  inherit perl;
}
```

manual build example

```
pkgs = import <nixpkgs> {}
hello = import ./hello.nix {
  stdenv = pkgs.stdenv;
  fetchurl = pkgs.fetchurl;
  perl = pkgs.perl;
}
```

```
# shell
nix-store --realise \
  /nix/store/c6950gxq0ig84q1n00ykg0jaaydx3q81-hello-2.10.drv
```

```
with import <nixpkgs> {};
callPackage ./hello.nix {}
```

How Nix Works?

cli tools

- nix-env
  - –install(-i), –uninstall(-e),
  - –list-generation, –rollback
  - -q, -qa
- nix-instantiate
- nix-store
  - –realise
  - -qR
  - –gc
- nix-build, nix-collect-garbage, nix-shell ...

nix store

- *derivation* -> package
- all output of *derivation* in a subdirectory in *nix store*
  - usually "/nix/store"

- path of output is determinate and is only depend on input
  - same input -> same output
- *nix store* is read only -> immutable

build-time dependency

- build environment is clean & determinate -> reproducible
  - external file(like ./build.sh) is copied to *nix store*
  - package dependency is reference via *derivation*
- all build time dependency is contained in *derivation*

runtime dependency

- scan all files for hash part of *nix store path*
  - nix-store -qR /nix/store/*hello/bin/hello (strings /nix/store/*hello/bin/hello)
  - nix-store -qR /nix/store/*-unit-script/bin/pdnsd-post-start
- binary distribution

user profile

How to access program install by Nix?

- user profile at ~/.nix-profile
  - ~/.nix-profile/bin in $PATH
- -> /nix/var/nix/profiles/per-user/ycy/profile
- -> /nix/var/nix/profiles/per-user/ycy/profile-XXX-link
- link will update when install or uninstall program with nix-env

garbage collecting

nix-* never delete any files, so we need GC

- /nix/var/nix/profile
- /nix/var/nix/gcroots
  - booted-system (current running system)
  - current-system (current NixOS)

From Nix to NixOS

package manager(Nix) + ??? = OS(NixOS)

- kernel
- bootloader
- service
- config files in /etc

cont.

- package
  - kernel
  - bootloader(grub)
- config
  - static files in /etc
  - service(systemd unit file)
- state
  - /etc/passwd

cli tools

- nixos-rebuild
  - -switch
  - -build

profile

- /nix/var/nix/profiles/system
- /run/current-system

activation script

- setup /etc
- setup bootloader
- other package specific setup

References & Resources

manual, guide & tutorial

- nix manual: `http://nixos.org/nix/manual`
- nixos manual: `http://nixos.org/nixos/manual/`
- nix pills: `http://lethalman.blogspot.com/search/label/nixpills`
- a gentle introduction to the nix family: `http://ebzzry.io/en/nix/`
- source code: `https://github.com/NixOS/nixpkgs`
- nix cook book: `http://funops.co/nix-cookbook/`
- man configuration.nix

cont.

Papers, see `https://nixos.org/docs/papers.html`

- Eelco Dolstra. The Purely Functional Software Deployment Model. PhD thesis, Faculty of Science, Utrecht, The Netherlands. January 2006. ISBN 90-393-4130-3.
- Armijn Hemel. NixOS: the Nix based operating system. Master's thesis, INF/SCR-2005-091, Institute of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands. August 2006.