Functional Programming: Real World Performance, Nix and Warp Server

Wong Ding Feng

July 30, 2019

Contents

1 Who am I? Introduction to myself

- Follow me on github! `https://github.com/TomatoCream`
- Linux user for 5 years now
  - Ubuntu
  - Proxmox
  - ArchLinux
  - Centos (server management)

1.1 My interests

- AI, ML
- Functional programming and abstraction (what the hell is so good about this?)

1.2 For whom is this talk for?

- Linux users! Sorry windows users
  - But not really (departs away from a unix way of doing things)
- Show you what functional programming can do?
  - purity?
  - referential transparency?
- State management
- DevOps
- Images, Docker, VM, Clusters
- I will give you a feel of `Nix` not the nitty gritty details

2 The big problem

- Has anyone ever used some sort of package management system?

2.1 Some modern day package management systems

Package manager	Distributions
apt, apt-get	Debian, Ubuntu
rpm, yum	Redhat, Centos
pacman	ArchLinux
brew	MacOS

2.2 What about sub ecosystems?

Package manager	???
pip, virtualenv, pipenv	Python2,3(???)
npm, yarn	Nodejs
cabal, stack, hackage	Haskell :)
go?	go?
brew	MacOS
use-package, vim, fish, zsh	...

2.3 How to make a package manager?

- What are the basic parts that we need?

2.4 How to make a package manager?

build dependencies	What do I need to build the program?
runtime dependencies	What `.so` shared objects do I need?
configurations	What in `/etc/...` config files

- essentially think of it as a graph, whenever we upgrade or install a package, we are mutating a node on this graph to point to something else.

2.5 Problems with modern package management

`https://wiki.debian.org/DontBreakDebian#Don.27t_make_a_FrankenDebian`

Contents

2.6 TODO Why imperative is bad? What is so imperative about installing packages?

referential transparency

2.7 Are you familiar with `DEPENDENCY HELL`?

- `https://www.reddit.com/r/ProgrammerHumor/comments/75txp4/nodejs_dependency_hell_visualized_for_the_first/?utm_source=share&`

`utm_medium=web2x`

- `https://github.com/vector-im/riot-web/network/dependencies`

2.8 All types of "DEPENDENCY HELL"

`https://miro.medium.com/max/984/0*7ezJOtYUkI5zyqWU.png`

- { DLL, dependency, npm, cabal } hell, different names for the same demon
- conflicting dependency
  - shared components like library links `cuda.7.so` vs `cuda.6.so`
- multiple version side by side and roll backs
- possible solutions
  - set of stable packages like `Debian` or `haskell stack snapshots`

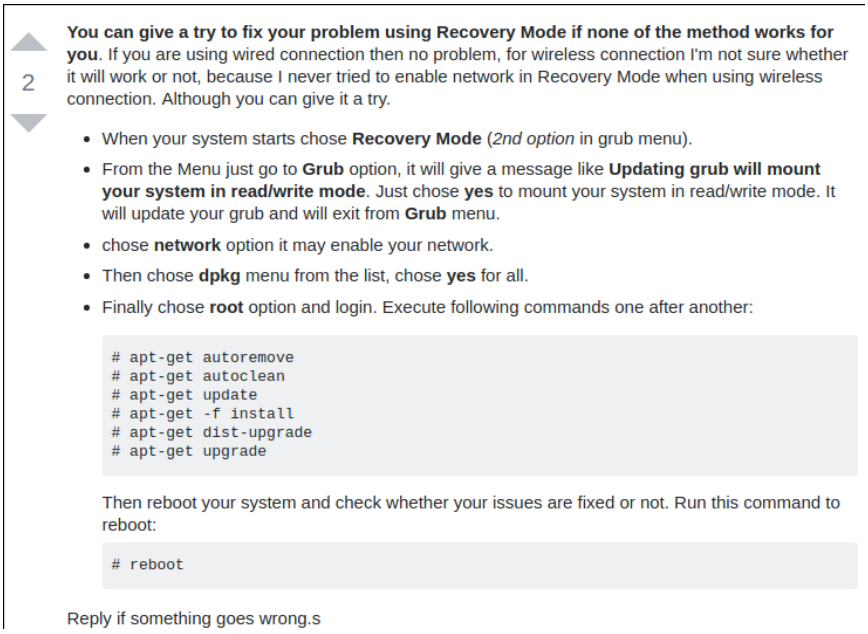
2.9 Not Atomic 01

- kill upgrades half way
  - packages left in a semi updated state
  - sometimes need to manually remove lock files

```
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
dpkg    29329 root    3uW  REG    8,7        0 262367 /var/lib/dpkg/lock
```

2.10 Not Atomic 02

- can be fixed but kinda troublesome.



2

You can give a try to fix your problem using Recovery Mode if none of the method works for you. If you are using wired connection then no problem, for wireless connection I'm not sure whether it will work or not, because I never tried to enable network in Recovery Mode when using wireless connection. Although you can give it a try.

- When your system starts chose **Recovery Mode** (*2nd option* in grub menu).
- From the Menu just go to **Grub** option, it will give a message like **Updating grub will mount your system in read/write mode**. Just chose **yes** to mount your system in read/write mode. It will update your grub and will exit from **Grub** menu.
- chose **network** option it may enable your network.
- Then chose **dpkg** menu from the list, chose **yes** for all.
- Finally chose **root** option and login. Execute following commands one after another:

```
# apt-get autoremove
# apt-get autoclean
# apt-get update
# apt-get -f install
# apt-get dist-upgrade
# apt-get upgrade
```

Then reboot your system and check whether your issues are fixed or not. Run this command to reboot:

```
# reboot
```

Reply if something goes wrong.s

2.11 Whats bad about imperative summary?

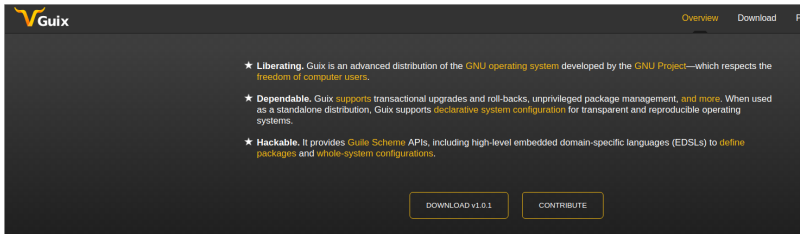
- No referential transparency
  - cannot point to older versions of the same thing
- Dependency hell
  - conflicting dependencies
- Not atomic upgrades
  - unknown state if break half way

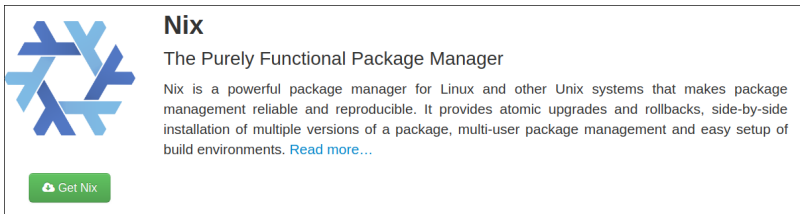
These problems are really similar to the problems with imperative languages! like `JAVA` and people have already made solutions for them like how `Haskell` does. We could learn a thing or two from them.

3 What it should/could/would have been?

- Imagine now that we implemented all the things of a functional programming language to create a functional package management system?
- What can we do with this?

3.1 GUIX vs Nix

- 

- 

3.2 Introducing Nix Package Management

- solves all of the problems above
  - No referential transparency
    - cannot point to older versions of the same thing
  - Dependency hell
  - Not atomic upgrades
    - unknown state if break half way

3.3 Main mechanism

- referential transparency
  - install everything in path `/nix/store/{hash}-name`
  - via `symlinking`

3.4 What you get for free with this mechanism?

- no `sudo`
- easy revert and roll back
- select specific version
- 2 different version can run at the same time

- same **development** environment as the **runtime** environment!
  - nix-shell

3.4.1 no `sudo`, where is my `sudo`?

- linux was developed as a `time sharing` system
- many users were expected to share a single computer.
- thus to manage conflicts, a `super user`, `root` was required to install and manage packages

```
nix-env -iA nixos.figlet
```

3.4.2 easy revert, rollback

```
figlet "I am here!"

nix-env --rollback

figlet "are you still here?"
```

3.4.3 Select specific version

```
cd ~/projects/nix-config/
git checkout ??
nix-env -f ~/projects/nix-config/ -iA screenfetch
```

screenfetch 2016 vs current

3.4.4 Installing and running 2 version of same software

```
stack --version
su
stack --version
```

3.4.5 Same development environment and runtime environment

- I am not an electrical engineer or something but I program my own keyboard. So I need some sort of firmware flasher. like `dfuprogrammer` I dont need it on my system.

```
cd ~/projects/qmk_firmware/
make
dfuprogrammer
nix-shell
make
dfuprogrammer
```

3.5 Going all the way, NixOS

- whole system management via Nix and thus NixOS
  - Version controlled operating system
  - show OS reboot
  - I wanted to show my generations so had been delaying removing my older generations

```
df -h /
nix-collect-garbage --delete-older-than 10 --dry-run
```

3.5.1 NixOS

- show `file:///home/df/nix-config/configuration.nix`
- python package management `file:///home/df/nix-config/configuration.nix`
- gnupg agent `file:///home/df/nix-config/configuration.nix`
- ports `file:///home/df/nix-config/configuration.nix`
  - I think it helps me get a state of all the ports in one place
- users and security all in one place `file:///home/df/nix-config/configuration.nix`
  - authorisedkeys
- postgresql can be packaged in `shell.nix` `file:///home/df/nix-config/configuration.nix`
  - separate project called `nixos-shell` `https://github.com/chrisfarms/nixos-shell`
- filesystems `file:///etc/nixos/hardware-configuration.nix`

3.5.2 docker

```
https://nixos.wiki/wiki/Docker

virtualisation.docker.enable = true;
users.users.<myuser>.extraGroups = [ "docker" ];

nix-build '<nixpkgs>' -A dockerTools.examples.redis
docker load < result

https://github.com/NixOS/nixpkgs/blob/master/pkgs/build-support/
docker/examples.nix
```

3.5.3 easy cd/dvd

```
cd ~/projects/nixpkgs
nix-build -A config.system.build.isoImage -I nixos-config=modules/installer/cd-dvd/inst
```

3.5.4 easy vm

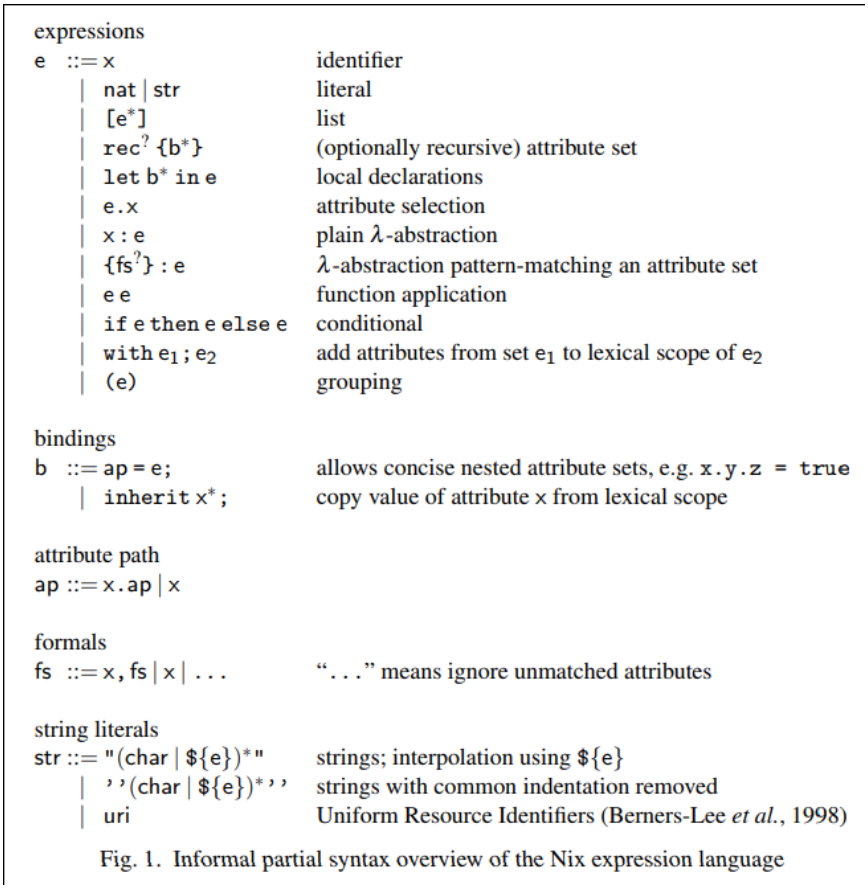
```
cd ./nixops
nixops create -d simple02 network.nix
nixops deploy -d simple02

deployment.targetEnv = "ec2";
deployment.region = "eu-west-1";
```

4 How does nix actually work?

4.1 Nix expressions

- functional expressions, not general purpose please do not program things with it
- comes with its own BNF grammar



expressions

e	::= x	identifier
	| nat | str	literal
	| [e*]	list
	| rec? {b*}	(optionally recursive) attribute set
	| let b* in e	local declarations
	| e.x	attribute selection
	| x : e	plain λ-abstraction
	| {fs?} : e	λ-abstraction pattern-matching an attribute set
	| e e	function application
	| if e then e else e	conditional
	| with e_1 ; e_2	add attributes from set e_1 to lexical scope of e_2
	| (e)	grouping

bindings

b	::= ap = e;	allows concise nested attribute sets, e.g. `x.y.z = true`
	| inherit x*;	copy value of attribute x from lexical scope

attribute path

ap	::= x.ap | x

formals

fs	::= x, fs | x | ...	"..." means ignore unmatched attributes

string literals

str	::= "(char | ${e})*"	strings; interpolation using ${e}
	| ''(char | ${e})*''	strings with common indentation removed
	| uri	Uniform Resource Identifiers (Berners-Lee *et al.*, 1998)

Fig. 1. Informal partial syntax overview of the Nix expression language

4.2 Language features

- Nix expressions
  - dynamically typed
  - lazy
  - pure

4.3 The main point

- Nix expressions are here to describe a graph of build actions called `derivations`
  - build script
  - set of environment variables

– set of dependencies

4.4 Example: Xmonad

```
{ stdenv, fetchurl, ghc, X11, xmessage }: 1

stdenv.mkDerivation 2 (rec {
  name = "xmonad-0.5";

  src = fetchurl {
    url = "http://hackage.haskell.org/.../${name}.tar.gz";
    sha256 = "1i74az7w7nbirw6n6lcm44vf05hjq1yyhnsssc779yh0n00lbk6g";
  };

  buildInputs = [ ghc X11 ]; 3

  configurePhase = '' 4
    substituteInPlace XMonad/Core.hs --replace \
      '"xmessage"' '"${xmessage}/bin/xmessage"' 5
    ghc --make Setup.lhs
    ./Setup configure --prefix="$out" 6
  '';

  buildPhase = ''
    ./Setup build
  '';

  installPhase = ''
    ./Setup copy
    ./Setup register --gen-script
  '';

  meta = { 7
    description = "A tiling window manager for X";
  };
})
```

Fig. 2. xmonad.nix: Nix expression for xmonad

4.5 Example: Xmonad

```
rec {
  xmonad = import .../xmonad.nix { 8
    inherit stdenv fetchurl ghc X11 xmessage;
  };

  xmessage = import .../xmessage.nix { ... };

  ghc = ghc68;

  ghc68 = import ../development/compilers/ghc-6.8 {
    inherit fetchurl stdenv readline perl gmp ncurses m4;
    ghc = ghcboot;
  };

  ghcboot = ...;
  stdenv = ...;
  ...
}
```

Fig. 4. all-packages.nix: Function calls to instantiate packages

4.6 Main mechanism

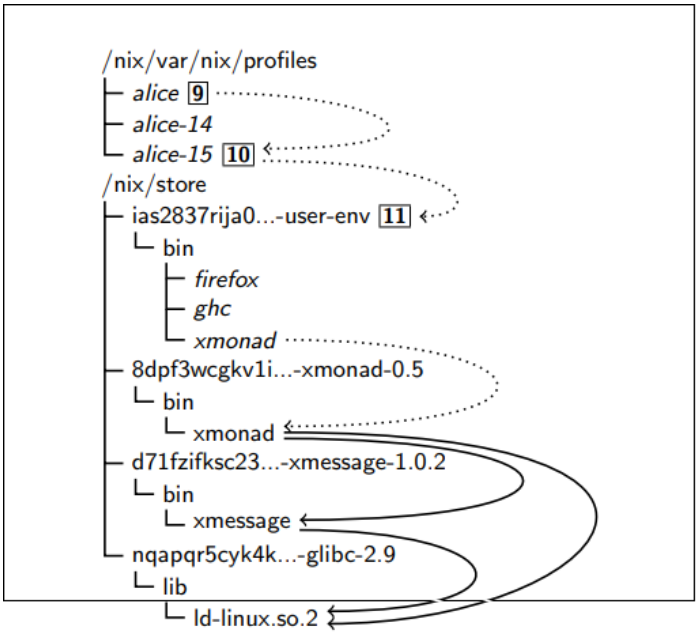


Fig. 5. The Nix store, containing xmonad and some of its dependencies, and profiles

4.7 Main mechanism

1. asd
2. asd

5 Warp server