

# Purely Functional System Configuration Management

HotOS XI, San Diego, CA

Eelco Dolstra<sup>1</sup>   Armijn Hemel<sup>2</sup>

<sup>1</sup>Universiteit Utrecht, Faculty of Science,  
Department of Information and Computing Sciences

<sup>2</sup>Loohuis Consulting

May 8, 2007



**Universiteit Utrecht**

# The state of the art in system configuration management

## Upgrading considered harmful

After an upgrade from SUSE 10.0 to 10.2:

```
$ /etc/init.d/squid start
```

```
Starting WWW-proxy squid
```

```
startproc: signal caught /usr/sbin/squid: Aborted
```

Cause: `/etc/squid/squid.conf` wasn't updated.

- ▶ Operating systems are installed and managed using tools that have an *imperative model*
- ▶ This causes lots of problems: upgrading is unreliable, rollbacks are hard, etc.
- ▶ This paper shows that it is possible to implement a system with a *purely functional model*
- ▶ Implemented in a Linux distribution called *NixOS*

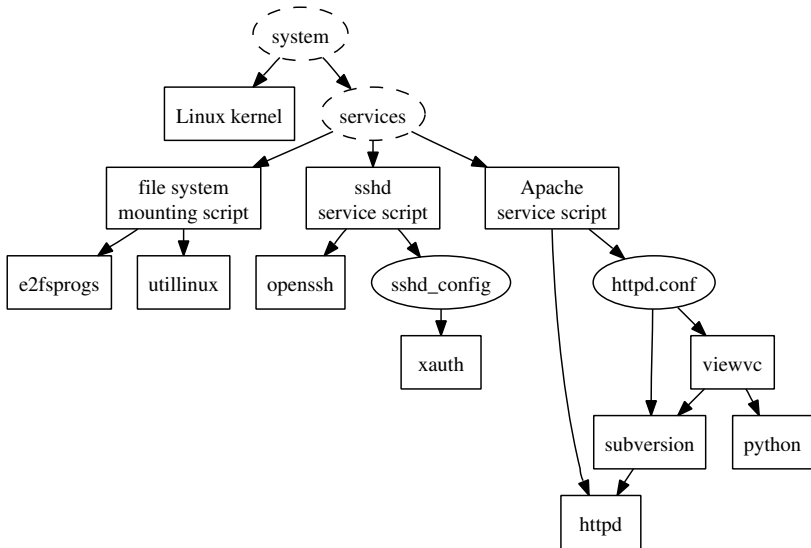
## What is a system configuration?

The static parts that make up a system, e.g.,

- ▶ Software packages
- ▶ Configuration files
- ▶ System scripts

Not included: mutable state.

# Example of a configuration



# Imperative model

System configuration tools have an *imperative model*: configuration actions are *stateful*.

## Examples

- ▶ Package management tools such as RPM, apt, Windows installers perform *destructive updates*: they overwrite existing files.
- ▶ Windows installers overwrite registry entries.

# Why is statefulness bad?

- ▶ No *predictability* (determinism)
  - ▶ If an action depends on an ill-defined initial state, then the result is probably ill-defined
  - ▶ This is why upgrading is riskier than a full re-install
- ▶ Configuration actions clobber the previous configuration
  - ▶ No atomicity
  - ▶ No rollbacks
  - ▶ Hard to safely test a configuration
- ▶ No *traceability*
  - ▶ Configuration is the result of a sequence of (sometimes manual) imperative actions over time
  - ▶ Hard to reproduce a configuration

# Why is statefulness bad?

- ▶ No *predictability* (determinism)
  - ▶ If an action depends on an ill-defined initial state, then the result is probably ill-defined
  - ▶ This is why upgrading is riskier than a full re-install
- ▶ Configuration actions clobber the previous configuration
  - ▶ No atomicity
  - ▶ No rollbacks
  - ▶ Hard to safely test a configuration
- ▶ No *traceability*
  - ▶ Configuration is the result of a sequence of (sometimes manual) imperative actions over time
  - ▶ Hard to reproduce a configuration



# Why is statefulness bad?

- ▶ No *predictability* (determinism)
  - ▶ If an action depends on an ill-defined initial state, then the result is probably ill-defined
  - ▶ This is why upgrading is riskier than a full re-install
- ▶ Configuration actions clobber the previous configuration
  - ▶ No atomicity
  - ▶ No rollbacks
  - ▶ Hard to safely test a configuration
- ▶ No *traceability*
  - ▶ Configuration is the result of a sequence of (sometimes manual) imperative actions over time
  - ▶ Hard to reproduce a configuration

# There is a better way!

- ▶ *Purely functional languages* like Haskell
  - ▶ No mutable variables, data structures
  - ▶ Function result only depends on function arguments:  
 $x = y \Rightarrow f(x) = f(y)$
  - ▶ *Referential transparency*
- ▶ No referential transparency in existing system CM tools

# There is a better way!

Goal: purely functional system configuration management

- ▶ The entire configuration should be rebuildable from a single declarative specification
- ▶ All static parts of the configuration should be *immutable*
  - ▶ No /bin, /usr, /etc...

# There is a better way!

## Goal: purely functional system configuration management

- ▶ The entire configuration should be rebuildable from a single declarative specification
- ▶ All static parts of the configuration should be *immutable*
  - ▶ No /bin, /usr, /etc...

# Nix: Purely functional package management

- ▶ Package manager developed at Utrecht University:  
<http://nix.cs.uu.nl/>
- ▶ *Purely functional* package management:
  - ▶ Purely functional language to describe how to build packages.
  - ▶ Build results only depend on declared inputs.
  - ▶ Packages never change after they have been built.

# Nix store

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

```
/nix/store/axrzx0rh0ivw...  
-firefox-2.0.0.3
```

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

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

```
/nix/store  
├── l9w6773m1msy...-openssh-4.6p1  
│   ├── bin  
│   │   └── ssh  
│   └── sbin  
│       └── sshd  
├── smkabrbibqv7...-openssl-0.9.8e  
│   └── lib  
│       └── libssl.so.0.9.8  
├── c6jbqm2mc0a7...-zlib-1.2.3  
│   └── lib  
│       └── libz.so.1.2.3  
└── im276akmsrhv...-glibc-2.5  
    ├── lib  
    └── libc.so.6
```

# Nix expressions

*Nix expressions* describe how to build packages.

```
{stdenv, fetchurl, openssl, zlib}:
```

```
stdenv.mkDerivation {  
  name = "openssh-4.6p1";  
  src = fetchurl {  
    url = http://.../openssh-4.6p1.tar.gz;  
    sha256 = "0fpjlr3bfind0y94bk442x2p...";  
  };  
  buildCommand = "  
    tar xjf $src  
    ./configure --prefix=$out --with-openssl=${openssl}  
    make; make install ";  
}
```

## Taking it all the way

- ▶ Since we can build packages...
- ▶ ...why not build all the other stuff that goes into a configuration?
- ▶ It's all the same, really. As long as it's pure, we can build it!



# Example

## Nix expression for sshd\_config

```
{writeText, forwardX11, xauth}:
```

```
writeText "sshd_config" "
```

```
  UsePAM yes
```

```
  ${if forwardX11 then "
```

```
    X11Forwarding yes
```

```
    XAuthLocation ${xauth}/bin/xauth
```

```
  " else "
```

```
    X11Forwarding no
```

```
  "}
```

```
"
```

# Example

/nix/store

└─ 068q49fhc600...-sshd

└─ etc/event.d

└─ sshd (*Upstart job*)

└─ lahm12vmh052...-sshd\_config

└─ r1k7gb1capq0...-xauth-1.0.2

└─ bin

└─ xauth

└─ im276akmsrhv...-glibc-2.5

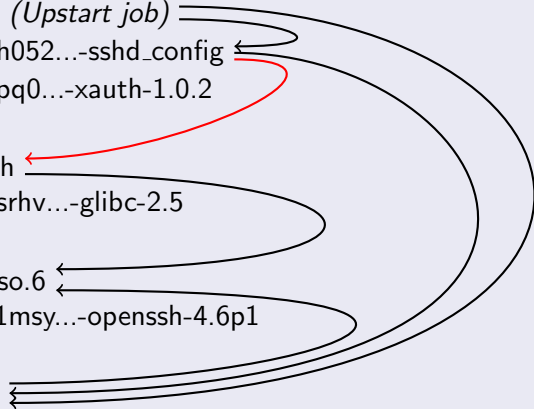
└─ lib

└─ libc.so.6

└─ l9w6773m1msy...-openssh-4.6p1

└─ sbin

└─ sshd



# Example

/nix/store

└─ 068q49fhc600...-sshd

└─ etc/event.d

└─ sshd (*Upstart job*)

└─ lahm12vmh052...-sshd\_config

└─ r1k7gb1capq0...-xauth-1.0.2

└─ bin

└─ xauth

Generated file: lahm12vmh052...-sshd\_config

UsePAM yes

X11Forwarding yes

XAuthLocation /nix/store/r1k7...-xauth-1.0.2/bin/xauth

...

└─ sshd

## Status

What we have:

- ▶ Hardware support: networking, sound, video
- ▶ RAID, LVM
- ▶ System daemons: SSH, Apache, CUPS, dhcpd, NTP, Cron, Mingetty, ...
- ▶ X11
- ▶ KDE, most of Gnome
- ▶ All the applications and tools in Nixpkgs (850 or so Unix packages)



# NixOS — Grub boot menu

GNU GRUB version 0.97 (639K lower / 130048K upper memory)

NixOS - Default

NixOS - Configuration 27 (2007-04-16 16:18:13)

NixOS - Configuration 26 (2007-04-16 16:17:40)

NixOS - Configuration 25 (2007-04-16 15:31:26)

NixOS - Configuration 24 (2007-04-04 19:09:46)

NixOS - Configuration 23 (2007-04-04 19:06:14)

NixOS - Configuration 22 (2007-04-04 11:53:53)

NixOS - Configuration 21 (2007-03-01 16:36:00)

NixOS - Configuration 20 (2007-02-28 14:20:35)

NixOS - Configuration 19 (2007-02-28 10:35:47)

NixOS - Configuration 18 (2007-02-28 02:30:04)

NixOS - Configuration 17 (2007-02-22 22:45:12)

↓

Use the ↑ and ↓ keys to select which entry is highlighted.  
Press enter to boot the selected OS, 'e' to edit the  
commands before booting, or 'c' for a command-line.

# The top-level system configuration

`/etc/nixos/nixos/system/system.nix`

- ▶ Top-level Nix expression.
- ▶ Calls other expressions to build the Upstart jobs, kernel, initrd, boot scripts...
- ▶ Takes as argument a *system configuration* — nested attribute set specifying system parameters.
- ▶ To build and activate a configuration:  
`$ nixos-rebuild switch`

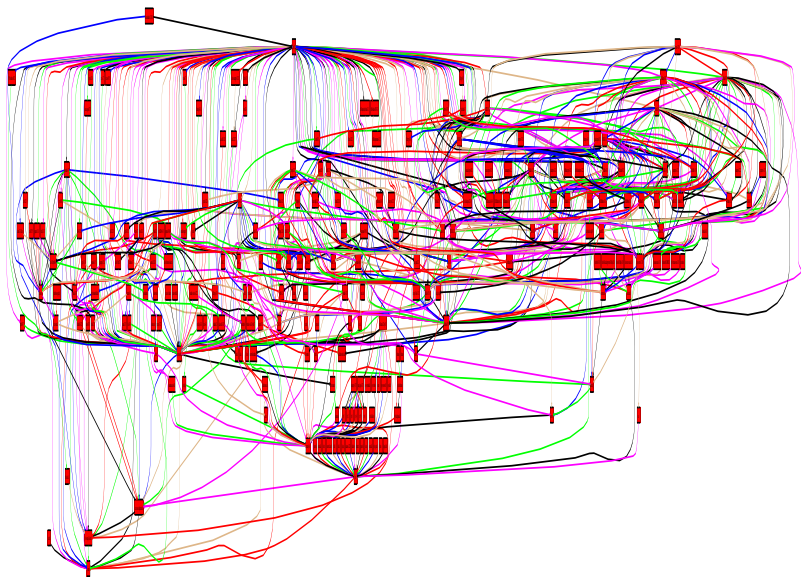
# The system configuration file

/etc/nixos/configuration.nix

```
{
  boot = { grubDevice = "/dev/hda"; };
  fileSystems = [
    { mountPoint = "/";
      device = "/dev/hda1";
    }
  ];
  swapDevices = [ { device = "/dev/hdb1"; } ];
  services = {
    sshd = {
      enable = true;
      forwardX11 = true;
    };
  };
}
```



# A complete system configuration



## How pure are we?

- ▶ No `/bin` (with 1 exception), `/sbin`, `/lib`, `/usr`.
  - ▶ Sole exception: `/bin/sh`
- ▶ Almost all of `/etc` resides in the Nix store
  - ▶ E.g. `sshd_config`, Upstart job specifies full store path
  - ▶ But some configuration files are cross-cutting (`/etc/resolv.conf`, `/etc/services`), so we symlink them in `/etc`
- ▶ Mutable state (`/var`): don't do anything special with it
  - ▶ Nasty: hybrid configuration+state: `/etc/passwd`

## How pure are we?

- ▶ No `/bin` (with 1 exception), `/sbin`, `/lib`, `/usr`.
  - ▶ Sole exception: `/bin/sh`
- ▶ Almost all of `/etc` resides in the Nix store
  - ▶ E.g. `sshd_config`, Upstart job specifies full store path
  - ▶ But some configuration files are cross-cutting (`/etc/resolv.conf`, `/etc/services`), so we symlink them in `/etc`
- ▶ Mutable state (`/var`): don't do anything special with it
  - ▶ Nasty: hybrid configuration+state: `/etc/passwd`

## How pure are we?

- ▶ No `/bin` (with 1 exception), `/sbin`, `/lib`, `/usr`.
  - ▶ Sole exception: `/bin/sh`
- ▶ Almost all of `/etc` resides in the Nix store
  - ▶ E.g. `sshd_config`, Upstart job specifies full store path
  - ▶ But some configuration files are cross-cutting (`/etc/resolv.conf`, `/etc/services`), so we symlink them in `/etc`
- ▶ Mutable state (`/var`): don't do anything special with it
  - ▶ Nasty: hybrid configuration+state: `/etc/passwd`

# The downside

- ▶ X11/KDE configuration takes up 656 MiB in 236 store paths.
- ▶ That's okay, but...
- ▶ ...if Glibc changes, than we'll need another 656 MiB.
- ▶ Analogous to purely functional data structures:
  - ▶ `x` : tail list is cheap
  - ▶ `init list ++ [x]` is expensive

# Conclusion

- ▶ NixOS shows that a purely functional system configuration model is feasible and practical.
- ▶ Advantages: reproducibility, determinism, predictable upgrading.
- ▶ Disadvantage: can take up to 2x as much disk space.

## More information / download

- ▶ <http://www.nixos.org/>
- ▶ ISO images for x86, x86\_64.