NixOS: A Purely Functional Linux Distribution
ICFP 2008, Victoria, Canada

Eelco Dolstra\textsuperscript{1}  
Andres Löh\textsuperscript{2}

\textsuperscript{1}Delft University of Technology, EWI, 
Department of Software Technology

\textsuperscript{2}Utrecht University, 
Department of Information and Computing Sciences

September 24, 2008
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*
Your operating system and applications are managed in an **imperative way**.
Your operating system and applications are managed in an **imperative way**.

- **Examples**
  - Package managers (e.g. RPM) perform *destructive updates* to the filesystem while doing upgrades.
  - Packages have post-install scripts that scribble all over `/etc`.
  - Sysadmins make manual changes to config files in `/etc`. 
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 multiple versions
  - 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 multiple versions
  - 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 multiple versions
  - 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!

Solution: borrow from purely functional languages.

We’re going to...

- Build packages from pure functions
- Store packages as immutable values in the filesystem
- Extend this to the other static bits of the system
Solution: borrow from purely functional languages.

We’re going to...

- Build packages from pure functions
- Store packages as immutable values in the filesystem
- Extend this to the other static bits of the system

Old stuff: Nix
There is a better way!

Solution: **borrow from purely functional languages.**

We’re going to...

- Build packages from pure functions
- Store packages as immutable values in the filesystem
- Extend this to the other static bits of the system

Old stuff: Nix

New stuff: NixOS
Nix: Purely functional package management

Nix is a purely functional package manager.

- Lazy, higher-order, dynamically typed, purely functional language to describe how to build and compose packages.
- Build results only depend on declared inputs.
- Packages are stored as a kind of purely functional data structure:
  - They never change after they have been built.
  - So in particular they’re never overwritten by newer versions.
  - Unused packages are garbage collected automatically.
Nix: Purely functional package management

Nix is a purely functional package manager.

- Lazy, higher-order, dynamically typed, purely functional language to describe how to build and compose packages.
- Build results only depend on declared inputs.
- Packages are stored as a kind of purely functional data structure:
  - They never change after they have been built.
  - So in particular they’re never overwritten by newer versions.
  - Unused packages are garbage collected automatically.
Nix is a purely functional package manager.

- Lazy, higher-order, dynamically typed, purely functional language to describe how to build and compose packages.
- Build results only depend on declared inputs.
- Packages are stored as a kind of purely functional data structure:
  - They never change after they have been built.
  - So in particular they’re never overwritten by newer versions.
  - Unused packages are garbage collected automatically.
Central idea: store all packages in isolation from each other:

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

Paths contain a 160-bit \textbf{cryptographic hash} of all inputs used to build the package:

- Sources
- Libraries
- Compilers
- Build scripts
- ...
Nix expressions describe how to build packages.

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

\texttt{stdenv.mkDerivation \{}

\texttt{name = "openssh-4.6p1";}
\texttt{src = fetchurl \{}\texttt{
\texttt{url = http://.../openssh-4.6p1.tar.gz;}
\texttt{sha256 = "0fpjlrl3bfnd0y94bk442x2p...";}
\texttt{}}
\texttt{buildCommand = '}
\texttt{tar xjf $src}
\texttt{./configure \--prefix=$out \--with-openssl=$\{openssl\}}
\texttt{make; make install}
\texttt{'\};
\texttt{}}

}
Nix expressions describe how to build packages.

{stdenv, fetchurl, openssl, zlib}:

```nix
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'';
}
```

Evaluating the result of this function will produce an OpenSSH package in the Nix store.

```
/nix/store
  19w6773m1msy...-openssh-4.6p1
    ┌── bin
    │   └── ssh
    │        └── sshd
    │
    └── sbin
        └── sshd
...
```
NixOS

Taking it all the way

- Since we can build packages...
- ...why not build all the other stuff that goes into a system configuration?
  - i.e. configuration files, system startup scripts, Linux’s initial ramdisk, ...
- As long as it’s pure, we can build it!
- Result: NixOS, a Linux distribution that uses Nix to build all static parts of the system.
Consequences

- All static parts are stored under /nix/store; no /lib, /usr, ...
- Upgrades are non-destructive; can roll back.
- Upgrades are atomic.
- Stateless: upgrading equivalent to reinstalling from scratch.
- Deterministic: can easily reproduce a configuration on another machine.
Example

Nix expression for ssh_config

```nix
{config, pkgs}:

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

Nix expression for ssh_config

{config, pkgs}:

pkgs.writeText "ssh_config" ''
  SendEnv LANG LC_ALL ...
  ${if config.services.sshd.forwardX11 then ''
    ForwardX11 yes
    XAuthLocation ${pkgs.xorg.xauth}/bin/xauth
    ''
  else ''
    ForwardX11 ''
  }
''
Example

Nix expression for ssh_config

```nix
{config, pkgs}:

pkgs.writeText "ssh_config" "

SendEnv LANG LC_ALL ...
${if config.services.sshd.forwardX11 then 
  ForwardX11 yes
  XAuthLocation 
  
    "" else 
    ""
  ForwardX11 no
  ""}

"

Nix store

```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Example

Nix expression for `ssh_config`

```nix
{config, pkgs}:

pkgs.writeText "ssh_config" ''
  SendEnv LANG LC_ALL ...
  ${if config.services.sshd.forwardX11 then ''
    ForwardX11 yes
    XAuthLocation ${if config.services.sshd.forwardX11 then ''
      /nix/store/kyv6n69a40q6...-xauth-1.0.2/bin/xauth
    '' else ''
  '' else ''

Generated file: 33lcnh62yll3...-sshd_config

SendEnv LANG LC_ALL ...
ForwardX11 yes
XAuthLocation /nix/store/kyv6n69a40q6...-xauth-1.0.2/bin/xauth
```
Nix expressions to build each part of the system: system packages, applications, their dependencies, kernel modules, initrd, configuration files, Upstart jobs, boot scripts, ...
Nix expressions to build each part of the system: system packages, applications, their dependencies, kernel modules, initrd, configuration files, Upstart jobs, boot scripts, ...
The system configuration file

```nix
/etc/nixos/configuration.nix

```
The system configuration file

/etc/nixos/configuration.nix

```nix
{  
    boot = {  grubDevice = "/dev/sda";  };  
    fileSystems = [  
        {  mountPoint = "/";  
            device = "/dev/sda1";  
        }  
    ];  
    swapDevices = [  
    ];  
    services = {  
        sshd = {  
            enable = true;  
            forwardX11 = 
        };  
    };  
}
```

End-user perspective

- Edit `configuration.nix`.
- Run `nixos-rebuild`.
- This builds `system.nix` and runs its activation script.
- Non-destructive; various rollback mechanisms.
GNU GRUB version 0.97 (639K lower / 129984K upper memory)

NixOS — Configuration 128 (2008-09-17 14:36:01 - 2.6.26.5-default)
NixOS — Configuration 127 (2008-09-17 14:35:01 - 2.6.26.5-default)
NixOS — Configuration 126 (2008-09-17 09:54:34 - 2.6.26.5-default)
NixOS — Configuration 125 (2008-09-16 23:26:05 - 2.6.25.16-default)
NixOS — Configuration 124 (2008-09-02 14:41:24 - 2.6.25.16-default)
NixOS — Configuration 122 (2008-09-02 11:40:04 - 2.6.26.3-default)
NixOS — Configuration 121 (2008-08-29 14:08:21 - 2.6.26.3-default)
NixOS — Configuration 120 (2008-08-29 11:09:07 - 2.6.26.3-default)

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.
Evaluation: How pure are we?

- No /bin, /sbin, /lib, /usr.
- Only exception: /bin/sh
- Only minor changes needed to a few packages.
Evaluation: How pure are we?

Configuration data

- 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/nsswitch.conf, /etc/services), so we symlink them in /etc.
Mutable state

- E.g. `/var/log`.
- Handled normally; not part of the purely functional model.
- Nasty: hybrid configuration + state: `/etc/passwd`; handled by the imperative activation script.
Evaluation: How pure are we?

Build actions

- Can't *guarantee* that build actions are pure.
- E.g. output could use
  - System time
  - The network
  - Files outside the Nix store
- Experiment: built 485 packages on two machines.
- Result: out of 165 927 files, 5059 differed.
- Almost all (possibly all) are due to timestamps or the hostname.
Conclusion

- Applied lazy, purely functional paradigm to a very different domain — system configuration management.
  - Purity and laziness are incredibly useful here.
- NixOS shows that a purely functional system configuration model is feasible and practical.
- Advantages: reproducibility, predictable upgrading, rollbacks, multi-user package management, ...

More information / download

- http://nixos.org/
- ISO images for x86, x86_64 are available.