NixOS: The Purely Functional Linux Distribution

Eelco Dolstra
LogicBlox

21 November 2013
So what was wrong with the previous 298 distributions?
So what was wrong with the previous 298 distributions?

- Upgrading is risky business
  - No way to roll back
  - Not atomic / transactional
So what was wrong with the previous 298 distributions?

- Upgrading is risky business
  - No way to roll back
  - Not atomic / transactional

- Not declarative
  - Configuration is result of automated/manual changes over time
  - Therefore hard to reproduce
So what was wrong with the previous 298 distributions?

- Upgrading is risky business
  - No way to roll back
  - Not atomic / transactional
- Not declarative
  - Configuration is result of automated/manual changes over time
  - Therefore hard to reproduce
- Package management limitations
  - Hard to have multiple versions of a package
  - Need to be root to install packages
  - Upgrading one package may break others ("DLL hell")
Conventional Linux distributions have an **imperative** (stateful) approach to configuration / package management.
Conventional Linux distributions have an **imperative** (stateful) approach to configuration / package management.

```
/  
  | bin
  |   | sh
  |   | ls
  | lib
  |   | libc.so.6
  | etc
  |   | resolv.conf
  |   | passwd
  |   | crontab
  | usr
  |   | bin
  |   | firefox
  |   | ssh
  |   | ooffice
  | ...
```

**Examples**

- Package managers like 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`. 
Enter NixOS
NixOS has a purely functional model

(Based on Nix, the purely functional package manager.)
This means:

- Packages, config files and other static parts of the system are built by *pure functions*.
- They are *immutable* after they have been built.
Consequences

- All static parts are stored under `/nix/store`; (almost) no `/bin`, `/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.
How it works: the Nix store

Nix stores all packages in isolation from each other:

```
/nix/store/rpdqxn00c00g...-firefox-25.0
```

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

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

```
/nix/store
|-- 19w6773m1msy...-openssh-4.6
|   |-- bin
|   |   |-- ssh
|   |   |   |-- sshd
|   |   |-- smkabrbibqv7...-openssl-0.9.
|   |   |   |-- 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
```
NixOS is declarative

/etc/nixos/configuration.nix

```plaintext
{
    boot.loader.grub.bootDevice = "/dev/sda";
    fileSystems."/".device = "/dev/sda1";
    swapDevices = [ { device = "/dev/sdb1"; } ];
    services.sshd.enable = true;
    services.sshd.forwardX11 = true;
}
```
NixOS is declarative

To enable Apache, add this to `configuration.nix`:

```nix
services.httpd.enable = true;
services.httpd.documentRoot = "/bla";
```

Or to enable X11/KDE:

```nix
services.xserver.enable = true;
services.xserver.displayManager.kde4.enable = true;
```

Or to make Firefox available to users:

```nix
environment.systemPackages = [ pkgs.firefox ];
```

Then run:

```
$ nixos-rebuild switch
```
NixOS is declarative

To enable Apache, add this to configuration.nix:

```nix
services.httpd.enable = true;
services.httpd.documentRoot = "/bla";
```

Or to enable X11/KDE:

```nix
services.xserver.enable = true;
services.xserver.displayManager.kde4.enable = true;
```
NixOS is declarative

To enable Apache, add this to configuration.nix:

```nix
services.httpd.enable = true;
services.httpd.documentRoot = "/bla";
```

Or to enable X11/KDE:

```nix
services.xserver.enable = true;
services.xserver.displayManager.kde4.enable = true;
```

Or to make Firefox available to users:

```nix
environment.systemPackages = [ pkgs.firefox ];
```
NixOS is declarative

To enable Apache, add this to `configuration.nix`:

```nix
services.httpd.enable = true;
services.httpd.documentRoot = "/bla";
```

Or to enable X11/KDE:

```nix
services.xserver.enable = true;
services.xserver.displayManager.kde4.enable = true;
```

Or to make Firefox available to users:

```nix
environment.systemPackages = [ pkgs.firefox ];
```

Then run:

```
$ nixos-rebuild switch
```
Reproducibility

`nixos-rebuild switch` invokes Nix to build or download the entire system configuration: packages, boot scripts, config files, ...
Reproducibility

\texttt{nixos-rebuild switch} invokes Nix to build or download the entire system configuration: packages, boot scripts, config files, ...
Reproducibility

`nixos-rebuild switch` invokes Nix to build or download the entire system configuration: packages, boot scripts, config files, ...

Since this is purely functional, configurations can always be reproduced.
Rollbacks are easy

To undo the last upgrade:

$ nixos-rebuild switch --rollback
Rollbacks are easy

To undo the last upgrade:

```
$ nixos-rebuild switch --rollback
```

Alternatively, revert `configuration.nix`, e.g. remove

```
services.httpd.enable = true;
services.httpd.documentRoot = "/bla";
```

and run

```
$ nixos-rebuild switch
```
Rollbacks are really easy

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. ESC to return.
Testing a configuration is painless

To test a new configuration without making it the boot default:

```
$ nixos-rebuild test
```
Testing a configuration is painless

To test a new configuration without making it the boot default:

```bash
$ nixos-rebuild test
```

To build a VM containing the new configuration:

```bash
$ nixos-rebuild build-vm
```
Done. The virtual machine can be started by running ./result/bin/run-hagbard-vm.

[uelco@hagbard:~/Dev/nixos]

[-uelco@hagbard:~/Dev/nixos]$ ./result/bin/run-hagbard-vm

```
starting device mapper and LVM...

Reading all physical volumes. This may take a while...
No volume groups found
No volume groups found

[ 0.944976] Switching to clocksource hpet

Failed to resume...

mounting /dev/uda on /.

fsck from util-linux-ng 2.27.2

```

```
running activation script...
setting up /etc...

<<<< NixOS Stage 2 >>>
```
Done. The virtual machine can be started by running ./result/bin/run-hagbard-vm.

[eelec@hagbard:~/Dev/nixos]$ ./result/bin/run-hagbard-vm
building the system configuration...
trace: Obsolete option `services.sshd.enable` is defined instead of `services.openssh.enable`.

Done. The virtual machine can be started by running ./result/bin/run-hagbard-vm.

./result/bin/run-hagbard-vm
Multi-user package management

Non-root users can install software:

```
alice$ nix-env -i firefox-25.0.1

alice$ firefox --version
Mozilla Firefox 25.0.1
```
Multi-user package management

Non-root users can install software:

```bash
alice$ nix-env -i firefox-25.0.1

alice$ firefox --version
Mozilla Firefox 25.0.1
```

Without interfering with each other:

```bash
bob$ firefox
firefox: No such file or directory

bob$ nix-env -i firefox-3.6.27

bob$ firefox --version
Mozilla Firefox 3.6.27
```
NixOps: Declarative Provisioning and Deployment
NixOps extends the NixOS approach to networks of machines

```json
{
  database =
  {
    deployment.targetEnv = "virtualbox";
    services.postgresql.enable = true;
  };

  webserver =
  {
    deployment.targetEnv = "virtualbox";
    services.httpd.enable = true;
    services.httpd.documentRoot = ...;
  };
}
```
Deploying to EC2

```json
{
  database =
    { deployment.targetEnv = "ec2";
      deployment.ec2.region = "us-east-1";
      services.postgresql.enable = true;
    }
  
  webserver =
    { deployment.targetEnv = "ec2";
      deployment.ec2.region = "eu-west-1";
      services.httpd.enable = true;
      services.httpd.documentRoot = ...;
    }
}
```
NixOps — Deploying

$ nixops create -n foo ./network.nix
$ nixops deploy -d foo

This will:

- Create all machines
- Build/download all dependencies
- Upload them to the machines
- Activate any necessary services

To redeploy: just run nixops deploy again.
Conclusion

Why NixOS is awesome
- Completely declarative
- Atomic upgrades and rollbacks
- Reproducibility
- Multi-user package management
- NixOps extends this to networks and adds provisioning

More information
- http://nixos.org/