Nix

The purely functional package manager

dram 2018-12-08

Nix*

- Nix
- Hydra
- Nixpkgs
- NixOS
- NixOps
- Disnix

Package management

- `pacman`
- `apt`
- `cabal`
- `cargo`
- `pip`
- …

Typical package management

- `pacman -S <foo>`
- Download `foo` from index
- Copy files
  - `/usr`: Distribution `$PREFIX`
  - `/usr/bin/python`, `/usr/lib/libncurses.so`

I want more

- Reproducible
- Isolated
- Declarative
- Customizable

Scenario

- Crash mid-upgrade
- *Systemd* was left unusable
- Good luck
- Needed: **Atomicity**

Scenario

- Dependency hell
- Can't install `libfoo-5.3` and `libfoo-5.4` together
  - Say, conflicting files `/usr/lib/libfoo.so`
- Bar requires `libfoo-5.3`, Baz requires `libfoo-5.4`
- Can't install both Bar and Baz.

Scenario

- Dependency hell
- “每次 Python 第二版本号升级我都要忙活一阵” (TUNA chat)
- If I just want to install two separate Python programs, I shouldn't care that they depend on different versions of Python.
- Needed: **Isolation** of packages, esp. dependencies
- Identical deps should still be sharable

Nix

Nix is a powerful package manager […] that makes package management reliable and reproducible.
(https://nixos.org/nix)

```
$ curl https://nixos.org/nix/install | sh
```

Rest of this talk

- Using Nix packages
- Creating packages with Nix
- Customizing Nix packages
- Development environment

Basic and imperative

nix-env ...	**pacman ...**
`-iA nixpkgs.bat`	`-S bat`
`-e bat`	`-Rs bat`(1)
`-u bat`	`-Su bat`
`nix-channel --update`(2)	`-Syy`

1. Dependencies don't automatically look installed to the user unless explicitly otherwise
2. `nix-channel` is orthogonal to `nix-env` and *optional*

What's in a Nix package

```
$ nix-env -iA nixpkgs.bat  # No need for root!
...
$ which bat
/home/dram/.nix-profile/bin/bat
$ readlink $(which bat)
/nix/store/a4him9701lkaaivdi0i3ffbpla829msp-bat-0.6.1/bin/bat
```

- **Hashed** and **immutable**
  - Stored as readonly from day zero!

```
$ ls -l /nix/store/a4him9701lkaaivdi0i3ffbpla829msp-bat-0.6.1/
-r-xr-xr-x 2 root root 5801848 Jan  1  1970 /nix/store/a4him97
```

What's in a Nix package

```
/nix/store/a4him9701lkaaivdi0i3ffbpla829msp-bat-0.6.1
├── bin
│   └── bat
└── share
    └── man
        └── man1
            └── bat.1.gz

4 directories, 2 files
```

- bat mini-prefix

```
--prefix=/nix/store/a4him9701lkaaivdi0i3ffbpla829msp-bat-0.6.1
```

Installing

Merging prefixes using symlinks

```
$ nix-env -iA nixpkgs.screenfetch
...
$ tree ~/.nix-profile
/home/dram/.nix-profile
├── bin
│   ├── bat -> ...-bat-0.6.1/bin/bat
│   └── screenfetch -> ...-screenFetch-3.8.0/bin/screenfetch
├── manifest.nix -> ...-env-manifest.nix
└── share
    ├── doc -> ...-screenFetch-3.8.0/share/doc
...
```

- Atomicity through copy-on-write

```
$ realpath ~/.nix-profile
/nix/store/4zhpp40awamgf9fisxgqmi1619i2j7gk-user-environment
```

Runtime dependencies

```
$ ldd $(which bat)
        linux-vdso.so.1 (0x00007fffbf6bf000)
        libz.so.1 => /nix/store/bv6...-zlib-1.2.11/lib/libz.so
        libc.so.6 => /nix/store/fg4...-glibc-2.27/lib/libc.so.
        ...
```

- Immutable dependencies
- Deps never break: `node_modules` but shared
- Isolated: Two versions of zlib don't interfere

Isolation

- Isolation by immutability
- Virtual machine: Isolation at CPU
- Container: Isolation at kernel
- Nix: Isolation at userland

(https://vimeo.com/223525975)

Nixpkgs

- Where do all those software come from?
- **Derivation** → **Output**
- The vast collection of software, utilities and more

Progress

- Using Nix packages
- Creating and Building packages
- Customizing Nix packages
- Development environments

Nix language

- Lazy functional
- Dynamic typed
- *Somewhat* purely-functional
- 'Sets'

```
{
  a = 1;
  b = {
    x = 0;
  };
}
```

Derivation

- Example: GNU Hello

```
{ stdenv, fetchurl }:

stdenv.mkDerivation rec {
  name = "hello-${version}";
  version = "2.10";

  src = fetchurl {
    url = "mirror://gnu/hello/${name}.tar.gz";
    sha256 = "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89n
  };
}
```

Derivation

- Example:
  bat

```
{ stdenv, fetchFromGitHub, rustPlatform   # Utilities
, cmake, pkgconfig, zlib, libiconv  # Dependent derivations
}:

rustPlatform.buildRustPackage rec {
  name     = "bat-${version}";
  # Build-time deps
  nativeBuildInputs = [ cmake pkgconfig zlib ];
  # Runtime deps to 'link' in
  buildInputs = [ libiconv ];
  # The two are separated to facilitate cross-compilation
  ...
}
```

Building

```
$ nix-build -E '(import <nixpkgs> {}).callPackage ./hello.nix
```

```
$ cat default.nix
(import <nixpkgs> {}).callPackage ./hello.nix {}
$ nix-build
```

Building

```
$ nix-build -E '(import <nixpkgs> {}).callPackage ./hello.nix
these derivations will be built:
  /nix/store/sc0y084gglb92gz79nsgjcp4q0i7sh70-hello-2.10.drv
building '/nix/store/sc0y084gglb92gz79nsgjcp4q0i7sh70-hello-2.
unpacking sources
...
checking whether build environment is sane... yes
...
/nix/store/rgbrg1zizqi9iq0zjrdx86jp2rq03syh-hello-2.10
$ readlink ./result # Output symlink
/nix/store/rgbrg1zizqi9iq0zjrdx86jp2rq03syh-hello-2.10
$ ls result
bin  share
```

Building

- `hello.nix`
- Evaluate → Derivation: `sc0...-hello-2.10.drv`
- Build → Output: `rgb...-hello-2.10`
- Output path is determined by derivation hash and not output hash.

Building

- Build time deps: only explicit deps are available
  - Tracked in `.drv`
- Result: **Reprodicibility**
- (Name of Nix: Dutch *niks* meaning 'nothing')
- (Runtime: Find store paths in file contents)

Binary distribution

- Builder side:
- Archive and upload:
  (Approximation)

```
$ mkdir binary-cache
$ nix copy --to file://binary-cache /nix/store/rgb...-hello-2.
$ nix copy --to s3://nix-cache /nix/store/rgb...-hello-2.10
```

Binary distribution

- User side:
- Same expression
- Same derivation: `sc0...-hello-2.10.drv`
- Download pre-built: `rgb...-hello-2.10`
- Download dependencies
- Result: **Source distro with binaries as optimization**

Progress

- Using Nix packages
- Creating packages with Nix
- Customizing Nix packages
- Development environment

Customization support in Nixpkgs

- Configuration options
- Dependency injection
- Flexibility of source distro with reduced cost

override

- Overriding configuration
- Example: Static Musl busybox

```
busybox.override {
  enableStatic = true;
  useMusl = true;
}
```

`override`

- Overriding derivation dependencies
- Example: `boost` with Python 3

```
boost.override {
  python = python3;
}
```

Overlays

- ~/.config/nixpkgs/overlays/foo.nix

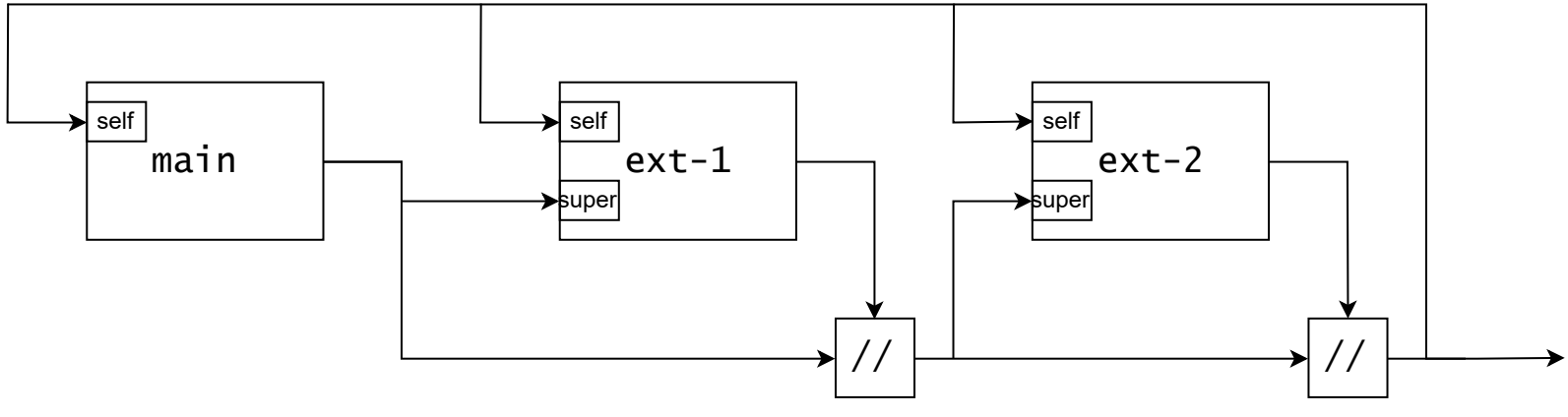
```
self: super:
{
  sarasa-gothic = self.callPackage ./sarasa-gothic.nix {};

  boost = super.boost.override {
    python = self.python3;
  };
}
```

Overlays

- Like class-based inheritance

```
class Mypkgs extends Nixpkgs {
  Derivation boost() {
    return super.boost().override(python=this.python3);
  }
}
```



Nix language

- Expressing Nixpkgs requires a higher-than-Bash level language
  - Involves higher-order functions
- Was a **new** language needed?
  - What do you think?
  - See also: Guix which uses Guile

Progress

- Using Nix packages
- Creating packages with Nix
- Customizing Nix packages
- Development environment

nix-shell

```
$ nix-shell -E '(import <nixpkgs> {}).callPackage ./hello.nix
[nix-shell]$ unpackPhase
...
[nix-shell]$ ls
hello-2.10
[nix-shell]$ cd $sourceRoot # sourceRoot=hello-2.10
[nix-shell]$ configurePhase
configure flags: --prefix=/nix/store/rgbrg1zizqi9iq0zjrdx86jp2
checking for a BSD-compatible install... /nix/store/wm8va53fh5
checking whether build environment is sane... yes
...
```

`nix-shell`

- Just replicates the build environment
- Isolated (like `~/.nix-profile`)
- Similar to
  - `apt build-depends`
  - RVM, Pipenv
  - Docker

Documentation

- https://nixos.org/nix/manual
- https://nixos.org/nixpkgs/manual

Thanks. Questions?