Nix on SHARCNET

Tyson Whitehead

May 14, 2015

Nix Overview

An enterprise approach to package management

- a package is a specific piece of code compiled in a specific way
- each package is entirely self contained and does not change
- each users select what packages they want and gets a custom enviornment

`https://nixos.org/nix`

Ships with several thousand packages already created

`https://nixos.org/nixos/packages.html`

SHARCNET

What this adds to SHARCNET

- each user can have their own custom environments
- environments should work everywhere (closed with no external dependencies)
- several thousand new and newer packages

Current issues (first is permanent, second will likely be resolved)

- newer glibc requires kernel 2.6.32 so no requin
- package can be used but not installed/removed on viz/vdi

`https://sourceware.org/ml/libc-alpha/2014-01/msg00511.html`

Enabling Nix

Nix is installed under /home/nixbld on SHARCNET. Enable for a single sessiong by running

```
source /home/nixbld/profile.d/nix-profile.sh
```

To always enable add this to the end of *~/.bash_profile*

```
echo source /home/nixbld/profile.d/nix-profile.sh \
  >> ~/.bash_profile
```

Reseting Nix

A basic reset is done by removing all .nix* files from your home directory

```
rm -fr ~/.nix*
```

A complete reset done by remove your Nix *per-user* directories

```
rm -fr /home/nixbld/var/nix/profile/per-user/$USER
rm -fr /home/nixbld/var/nix/gcroots/per-user/$USER
```

The *nix-profile.sh* script will re-create these with the defaults next time it runs.

Environment

The `nix-env` commands maintains your environments

- query packages (available and installed)
- create a new environment from current one by adding packages
- create a new environment from current one by removing packages
- switching between existing environments
- delete unused environements

Querying Packages

The `nix-env {--query | -q} ...` command queries package. Flags include

`{--available | -a}`	query available (instead of installed)
`{--attr-path | -P}`	display attribute path (unambiguous identifier)
`--description`	display description

Querying available packages is *very slow*. Store output in a file for reference

```
nix-env -qaP --description > ~/nix-packages.txt
```

Adding Packages

The `nix-env {--install | -i} ...` creates a new environment from the current one by adding additional packages. Flags include

`{--attr	-A}`	install by attribute path instead of name
`{--remove-all	-r}`	create new environment from empty one instead of current one

Adding packages by name (i.e., without `-A`) does an implicit query of available packages (*very slow*) to match the name. Use attribute paths instead

```
nix-env -iA nixpkgs.emacs nixpkgs.vim
nix-env -q
```

Removing Packages

The `nix-env {--uninstall | -e} ...` command creates a new environment from the current one by removing packages.

This must be done by name (for technical reasons), but it is not slow as the implicit query to match the name is on installed packages and not available packages.

```
nix-env -e vim
nix-env -q
```

Switching Environments

All the above commands create a new environment and then switch to it. They do not modify the current one. All previous (generations) of the environment remain and can be re-enabled at any time.

`nix-env --list-generations` list all environments

`nix-env {--switch-generation | -G} ...` switch to specified environment

`nix-env {--roll-back}` switch to previous environment

`nix-env {--delete-generations} ...` delete specified environment

```
nix-env --list-generations
nix-env --roll-back
nix-env -q
nix-env -G 2
nix-env --delete-generations 1
```

Packages and Environments

Packages are stored under `/home/nixbld/store`. Individual package directory name includes a hash of all dependencies (including entire build instructions) to keep everything separate

- `/home/nixbld/store/${HASH}-${NAME}`

An environment is a package containing bin, sbin, lib, etc. directories filled with symlinks to the components of the packages installed in that environment.

User Environments

Each user has a list of environments

- `/home/nixbld/var/nix/profiles/per-user/$USER/\`
  `profile-$GENERATION`
- links to associated environment package

Active environment is by special `profile` link

- `/home/nixbld/var/nix/profiles/per-user/$USER/\`
  `profile-$GENERATION`
- links to active environment

SHARCNET

System environment is augmented by by adding Nix environments first to system search paths like `$PATH` and `$MANPATH` (*nix-profile.sh*)

- includes directories under `~/.nix-profile`
- links to `/home/nixbld/var/nix/profiles/per-user/\ $USER/profile`

Includes a default SHARCNET environment too (just the nix commands so far) in system search paths

- includes directories under `/home/nixbld/var/nix/profiles/default`
- link to first generation `default-1`

Configuration (1/2)

Packages correspond to Nix expressions which tell the Nix builder how to compile the package. These expressions are quite readable and reveal many options

`https://nixos.org/nixos/packages.html`

The collection of packages available on SHARCNET are a newer snapshot than those referenced in the above link. The definitive reference is

~/.nix-defexpr/nixpkgs/pkgs/top-level/all-packages.nix

Configuration (2/2)

Use a search/grep to locate last bit of attribute path (`nix-env -qaP`) to find the associated file. For example, attribute path for emacs was `nixpkgs.emacs`, searching for `emacs` reveals

- `emacs = emacs24`
- `emacs24 = callPackage ../applications/editors/emacs-24 ...`

so the associated Nix expression is in

~/.nix-defexpr/nixpkgs/pkgs/applications/editors/emacs-24/default.nix

Nix Expression

Say we want to disable X11 support in emacs. The Nix expression is

```
{ stdenv, fetchurl, ncurses, x11, libXaw, libXpm, Xaw3d
, pkgconfig, gtk, libXft, dbus, libpng, libjpeg, libungif
, libtiff, librsvg, texinfo, gconf, libxml2, imagemagick, g
, alsaLib, cairo
, withX ? !stdenv.isDarwin
, withGTK3 ? false, gtk3 ? null
, withGTK2 ? true, gtk2
}:

assert (libXft != null) -> libpng != null;  # probably a bu
assert stdenv.isDarwin -> libXaw != null;   # fails to lin
assert withGTK2 -> withX || stdenv.isDarwin;
assert withGTK3 -> withX || stdenv.isDarwin;
assert withGTK2 -> !withGTK3 && gtk2 != null;
assert withGTK3 -> !withGTK2 && gtk3 != null;
```

Override

Packages are overriden in *~/.nixpkgs/config.nix*. From the Nix expression we can guess we want

- `withX = false`
- `withGTK2 = false`
- `withGTK3 = false`

This is easily done

```
{
  packageOverrides = pkgs: {
    emacs = pkgs.emacs.override {
      withX = false; withGTK2 = false; withGTK3 = false;
    };
  };
}
```