Declarative Distros

Quick Disclaimer

We are both GNU Guix Users

While we will try to talk about generalities of declarative operating system paradigms, Guix and Nix specific things, etc, we may get some Nix (and even Guix) things wrong.

What is a Declarative Distro?

What does it mean?

- A operating system defined by configuration
- Without state
- System is reproducible

Declarative v. Imperative

- Most distros are imperative
  - You give them commands to get a result
- Nix and Guix are declarative
  - You define a wanted end state and Guix/Nix gets you there its way

Reproducible

- Systems have a given configuration
- Configuration can produce identical systems
  - If you pin versions, you can make your system completely reproducible
- Everything is isolated

Reproducible

Isolation

- Packages are built in isolated packages
- No random spare files in file system
  - Anything not being used by a profile is eligible for garbage—collection
- All state is containted
  - Preserves state—less root
  - Can be further extended to your home directory

Stability

- Reproducibility and Isolation ensure the ability to rollback
- Always have some working system state
- Since any transaction is atomic, it either works or doesn't
  - Your system will never be left in a half updated state that is difficult to recover
  - When updating, the actual "commit" is changing a symlink around

NixOS

- Released Jun 2003
  - Eelco Dolstra
- Nix package manager (nix)
- Nix language
- Systemd
- Wrote like an entire research paper on the theory behind Nix

Guix

- Released 2012
  - Forked from Nix
- Guix package manager (guix)
- GNU Guile (scheme/lisp)
- GNU Shepherd
  - Not GNU/Herd

Declarative Package Management

`nix` and `guix` commands

Declarative Package Management

- Update system repositories
- Build and/or install packages
- Manage `the store`

Distro Store

- Guix `/gnu/store`
- Nix `/nix/store`
- Stores all package and system files
- Contents are based off of configuration

What is the store like?

- Lots of files that look like {hash}—package—version

Example

```
/gnu/store/spxcaq8gnmckhzz9a1wm3qc9dmz5bvsd-gcc-toolchain-12.3.0
```

What is the store like?

- Hash is derived from a build dependency graph
  - A program can use any version of a library it needs
  - A program build hash is unique, if any dependency changes the hash will as well.
  - This means new updates never overwrite old versions

Profile

- Guix `~/.guix-profile`
- Nix `~/.nix-profile`
- Symlink Read only system
  - Links store to current profile
  - Based off configuration

- Store holds system and package files
- Profile links to those files

Multidirection

```
$ which gcc
/home/tylerm/.guix-home/profile/bin/gcc # profile

$ readlink /home/tylerm/.guix-home/profile/bin/gcc
/gnu/store/spxcaq8gnmckhzz9a1wm3qc9dmz5bvsd-gcc-toolchain-12.3.0/bin/gcc # store
```

Multiple Profiles

- System Profile (All users)
  - Good to contain system needed pacakges
  - Graphics drivers
  - Zsh
- User Profile (Specific user)
  - Good to contain all user data
  - Browser
  - Applications
  - DE/WM

To Build The System

- Package manager takes in user and system configuration
- Downloads needed data into store
  - Prebuilt, signed binaries are called "substitutes"
- Build any needed packages
- Symlinks the user and system profiles

Temporary Packages

- Nix and Guix allow installing packages temporarily
- Uses contained environment
- `guix shell <package>`
- `nix shell -p <package>`

What is a package?

- Since Nix and Scheme are functional, packages are immutable pure functions
- They define everything from:
  - Inputs
  - Dependencies
  - Expected Outputs
  - Build environment
- Builds are completely isolated from one another
  - You cannot forget to define a dependency
  - You can guarantee that package will build for everyone once you get it working