GNURadio ecosystem management with Nix

Tom Bereknyei
Intro

- Defense Digital Service: "SWAT team of nerds."
- Me: Technical lead for several projects using GNU Radio in DoD.
- Assumptions: Intermediate-level user of GNU Radio and *nix systems.
- Why: Because this vastly improved confidence in our software and improved our time to delivery
Ask questions!

- If something isn't clear. Interrupt me
- No really... Interrupt me
Build problems - current state of affairs

- It compiles on my machine
- I installed SOMETHING into /usr/{bin,lib}, but now interferes with stuff in /usr/local/{bin,lib}
- I have both versions X and Y, but I can't seem to get things to link to version Y
- My package manager has version X, but I need version Y or patch it
- My component uses Boost version X, but another part of my application uses version Y
- GNU Radio Companion can't find ...
- Pip, virtualenv, setup.py, SWIG, PYTHONPATH, etc.
- Now cross-compile everything above
- Now do all of the above with RFNoC
- Insert story from the audience here
Solutions

- Use a VM and snapshots.
- Docker scripts
  - "Just use this install script on a fresh OS installation."
- Custom solutions: once it works, don't touch it.
Challenge

- Take a random commit from 5 years ago along with all the changes in libraries, compilers, operating systems, etc.
- Can you get the commit to build from scratch?
What do we want?

- Reliable builds
- If it builds on my machine, it should build on any machine, always
- If I build it today, I should be able to build it in 10 years
- Isolation
  - Multiple versions of the same software should be able to run next to each other
- Atomic updates
  - You either install something completely, or you do not install it at all
- Experimentation without fear
Idea

**Let's make package managers work like git!**


Idea

Lets make package managers work like git!

```
PREFIX= sha256(sha256(deps(package)) + sha256(src(package)) + sha256(options(package))

$PREFIX/bin , $PREFIX/lib $PREFIX/share
instead of:
/usr/bin, /usr/lib/, /usr/share
```

- Dependencies change? => Installed in different prefix
- Source code change? => Installed in different prefix
- Build options change? => Installed in different prefix
Nix

- Package manager
- Declarative language to describe package builds
- Isolated build environments
- Over 10000 packages and counting
- Mac OS X / GNU/Linux / BSD and Soon Windows Subsystem for Linux*
- Source-based package manager (Like Gentoo)
- But don't worry; also has a build cache
DEMO 0: Basic install of hello

Two styles

- Imperative, similar to apt, brew, dpkg, etc.
  - `nix-env -i hello` or `nix-env -iA nixpkgs.hello`
  - `nix-env -e hello`

- Declarative, similar to Dockerfile, package.json, etc.
  - `default.nix` or `shell.nix`
DEMO 1: Basic install of gnuradio

- To install a package, we build it from source, given a package description
- Nixpkgs is a set of expressions curated by the community.
- Observation: It was instant? It didn't build anything from source?
- Not very user-friendly to type in the large /nix/store/bLAHBLAH/ when I want to run a program
Important takeaways

- Each package is installed in its own unique path (think git commit hash)
- Software is installed into profiles, which are symlinks to packages (think HEAD)
- You can rollback to previous profiles, by changing a symlink (think git checkout)
- This allows for atomic updates, because symlink changes are atomic
- As an end user, not very different from homebrew or apt, except for rollbacks
The Nix Language in 1 minute

- Language of Nixfiles, which describes how to build packages
- Think Dockerfile or debinfo file
- Actually a proper programming language
- JSON-like with templating, functions and variables
- Side-effects only allowed *but* only if we know the *output* beforehand

```
"hello"
1 + 3
./a/path
[ "i" 3 5 ]
{ x = "Hello"; y=42;}
```

```
a = 3
b = 4
thing = { x = a;, y = b;}
add_struct = {x, y}:  x + y
add_struct thing  # Results in 7
```
Graphical representation of our Derivation

LoginService.java
coreutils.nix
jzmq.nix
openjdk.nix
bash.nix

login-service.nix
Evaluated derivation

```
/nix/store/aua3281du2-LoginService.java
/nix/store/asdqihe13729832-coreutils
/nix/store/8udol838313ye71-jzmq
/nix/store/813912831ys-openjdk
/nix/store/97iadlis342473-bash
```

login-service.nix
Evaluated derivation
If I update the source code

/nix/store/sjdaiewu23-LoginService.java

/nix/store/asdqhe13729832-coreutils

/nix/store/8udoi838313ye71-jzmq

/nix/store/813912831ys-openjdk

/nix/store/97aidils342473-bash
If I update the source code

/nix/store/sjdaiewu23-LoginService.java
/nix/store/asdqhe13729832-coreutils
/nix/store/8udol836313ye71-socket
/nix/store/813912831ys-openjdk
/nix/store/97idlis342473-bash

/nix/store/zudu323483-login-service
If I update one of the dependencies ...

/nix/store/sjdaiewu23-LoginService.java

/nix/store/asdqhe13729832-coreutilis

/nix/store/8udoi838313ye71-jzmq

/nix/store/kdausdu239-openjdk

/nix/store/97adlis342473-bash

/nix/store/zudu323493-login-service
If I update one of the dependencies ...
DEMO2: Reliable builds

- I am confident, that if I check out the Nix file of gnuradio from five years ago, it will build
- It will build all old versions of dependencies from source, and then build gnuradio from source
- Takes a long time. But it **will work**
commit 993dadd2136ffca9a6f81d7e4d6acd5116da83a0 (HEAD)
Author: Franz Pletz <fpletz@fnordicwalking.de>
Date:   Fri May 13 02:31:33 2016 +0200

gnuradio: 3.7.9.1 -> 3.7.9.2
How is a derivation built

- Source code is downloaded, fetched, obtained.
- Code is **checked against hash, otherwise abort**
- A chroot (container) is setup, containing *just* the build dependencies and source
- No network access
- All environment variables are *Cleared*
- No access to $HOME. No access to anything on disk..
- Time is set to 1970
- **The package build can only depend directly on the dependencies specified, and NOTHING else**
- The builder argument is executed, and its output copied to the Nix store
How is a derivation built

- Now, if a colleague forgets to write down what libraries *exactly* you need to install
- ... Or if uses library that is available by default on Ubuntu but not On Redhat
- ... The build is guaranteed to fail
- We explicitly state the hash of sources we download from the internet
- If the internet changes, then the build fails. No implicit changes!
- **Reliable builds**
Build Cache

- Remember, our build instructions uniquely determine where we install the package

```nix-repl
"${gnuradio}
"/nix/store/sqxwvn33x39sjfr47spib74gi3cqffv-gnuradio-3.7.11"
```

- We know *beforehand* where our build is going to be put!
- Simply *ask* if someone else already built it, and download it from there!
- Trust?
Build Cache

- Can also be used privately, for internal packages

- `nix build --store https://cache.nixos.org` (Default)
- `nix build --store s3://my-company-bucket`
- `nix build --store ssh://colleague-machine`
- `nix build --store file:///nfs/company-fileshare/`

- BuildCache As A Service: [https://cachix.org/](https://cachix.org/)

- If a colleague already built some project
- ... and you checkout the same git commit
- Then you don't have to rebuild everything! You just download it from the cache!
DEMO 3:

GNURadio OOT

- Reproducibly build a module
- Create a wrapped version of GNURadio
DEMO 4:

GNURadio OOT with integration

Same as demo3, but

- remove GTK errors
- allow for inspectability in nix-shell
DEMO 5: RFNoC

- Use provided toolchain
- Build UHD
- Build GNURadio
- Build gr-ettus
- All dependencies down to USRP images and glibc are pinned.
- Cross-compile libraries and produce a bundle ready for installation, testing, and deployment
Continuous integration script

- Typical Nix CI script

```
# .travis.yml
language: nix
script:
  - nix build . --store s3://company-bucket
after_success:
  - nix copy . --to s3://company-bucket
```
DEMO: Hydra
Solution? Docker

- Docker is an ubiquitous distribution format.
- Once it builds, send it to the registry.
- Solves the "runs on my machine" problem.
- Does not solve the "builds on my machine" problem.

OH: "Works on my Docker"

4:02 AM - 28 Jun 2018

4 Retweets 15 Likes
Best of both worlds

- Nix has support for building docker containers
- Copies your package + all its dependencies in a docker image
- **Bare image**, no FROM blah
- Super small (not quite as small as Alpine)
- You can easily integrate Nix in existing docker-compose or Kubernetes projects!

```nix
let
  pkgs = import ../nixpkgs.nix;
  some-service = import ./some-service.nix;
in
  pkgs.dockerTools.buildImage {
    name = "some-service";
    config = {
      Cmd = [ "$login-service/bin/login-service" ];
      Expose = [ 8080 ];
    };
  }
```
Possibilities

- Reproducible builds on local machines.
- Reproducible builds on CI with testing.
- Cross-compile to other architectures.
- Distributed builds. Build natively on other architectures.
- Testing.
End state

- Each push to testing/staging/master compiles all dependencies (cached).
- Manages Python 2 and 3 applications, C, C++, GNURadio OOT, Javascript (npm/yarn).
- Flowgraph tested in QEMU VM on recorded data.
- Distributed ARM builders connected via VPN.
- Single fat binary bundled for OS agnostic deployment to GNU/Linux.
- `.deb` packages created bundling all dependencies, systemd service configurations, udev rules, nginx, raster tiles for maps etc.
- `.deb` installation tested on fresh Ubuntu VM
- Docker containers built, tested, cached.
- Nix binary package cached on build server.
- Flowgraph tested on specific builders with SDR and other hardware attached.
Other thoughts:

- NixOS - OS based on similar mechanisms to also track OS, configs, services, etc.
- Everything including kernel, kernel patches, DTBs, GRUB/U-Boot, etc all managed.
- NixOps - infrastructure management tool
- Deploy NixOS to cloud/local virtualization/environment declaratively. VPC, routes, security rules, key distribution, etc. (Terraform-ish)
- The build system, pipeline, and testing from previous slide are declaratively defined.
- Check out commit from 5 years ago, get production environment from 5 years ago
- Experimental
- Disnix - Nix for services, but can be on Windows, non-NixOS, etc.
  - Dydisnix - Distributed service deployment
  - Dysnomia - Automated deployment of mutable components
### Downsides

- Steep learning curve. Thinking functionally is something to get used to

- You *can not* do dirty hacks. You can't go monkeypatch some python package in `/usr/lib/python`, or update `/etc/hosts` manually

- Unforgiving, enforces discipline

- Closure can get large, all dependencies included, nothing is used from host system other than POSIX sh.

- Documentation is ... not always great. "Read the source code" is a common philosophy among Nix'ers

- Hardcoded paths and functionality in GRC, UHD, etc requires some manipulation and patching. *Potential for improvement, PRs on the way.*
Recap

- Nix is a package manager, and build system coordinator.
- Build OOT modules in isolation with reliable dependency tracking.
- Test flowgraphs on a predictable system.
- Easily share build environments with colleagues.
- Is not docker, but works well with docker!
- Can be set up on 100% internal infrastructure.
Thanks! Questions?

- [https://github.com/tomberek/nixtalk](https://github.com/tomberek/nixtalk)
- [https://github.com/tomberek/gnuradio-demo](https://github.com/tomberek/gnuradio-demo)
- [https://nixos.org/nixos/nix-pills](https://nixos.org/nixos/nix-pills) - Tutorial to get up to speed quickly with how nix works
- [https://nixos.org/nix](https://nixos.org/nix)

- Credits: presentation adopted from Arian van Putten

- Before Ben asks, yes we have upstreamed bug-fixes, generic libraries not tied to the original mission, and this entire presentation plus demos are available on GitHub.
Bonus: Hey did gnuradio compile?