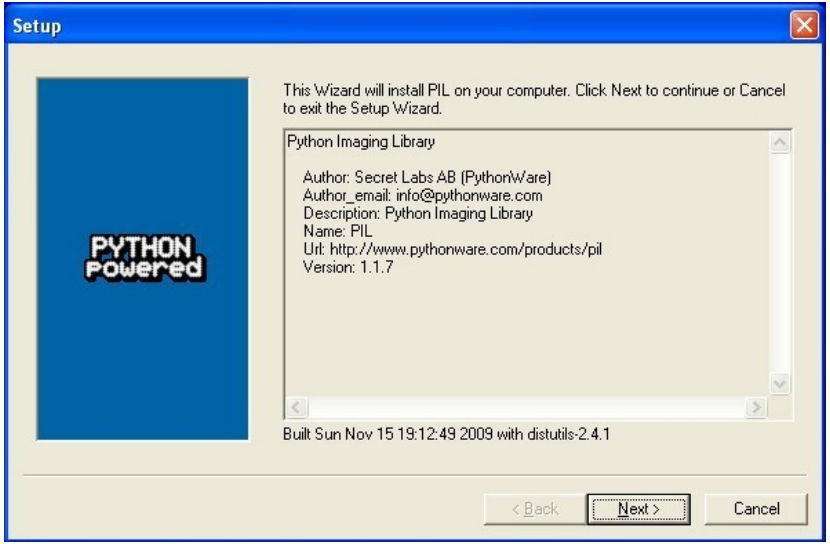
A Sack Full of Angry Snakes

Taming your Python dependencies with Nix

Thomas Woolford
twoolie
PyConline AU 04-08-2020

Packaging

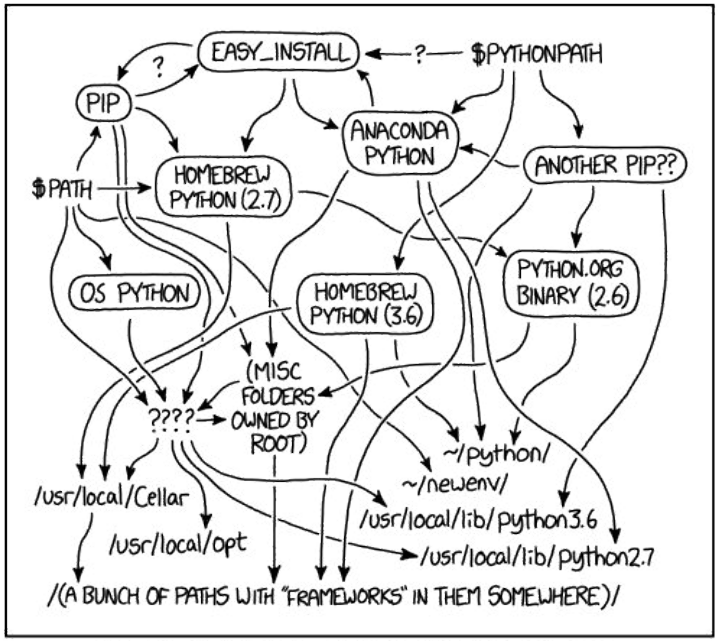
My first brush with packaging



How python packages get installed

- python site_packages
  - Managed by package manager (if lucky)
- setup.py install
  - Need root to install globally
- easy_install
  - Made it easier to get from pypi
- pip install
  - Introduced requirements files
- Virtualenv, python3 venv
  - Allowed separation
- scipy, anaconda, beeware
  - Curated package sets that integrate tightly

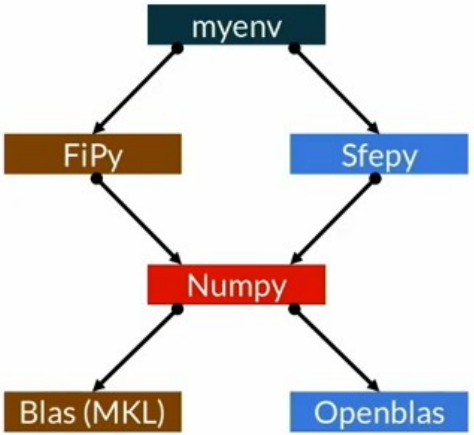
https://xkcd.com/1987/



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Issues still not fully solved

- Incremental installations
  - Install in different order, get different versions
  - Diamond dependency graphs
- Native dependencies
  - Install requires native libs e.g. lxml, pyqt, pycrypto
  - Wheels can contain shared libraries but...
  - How do you patch wheels across many venvs?
- System Dependencies
  - How to install services provided by host system
  - e.g. tests require ephemeral_pg and Redis



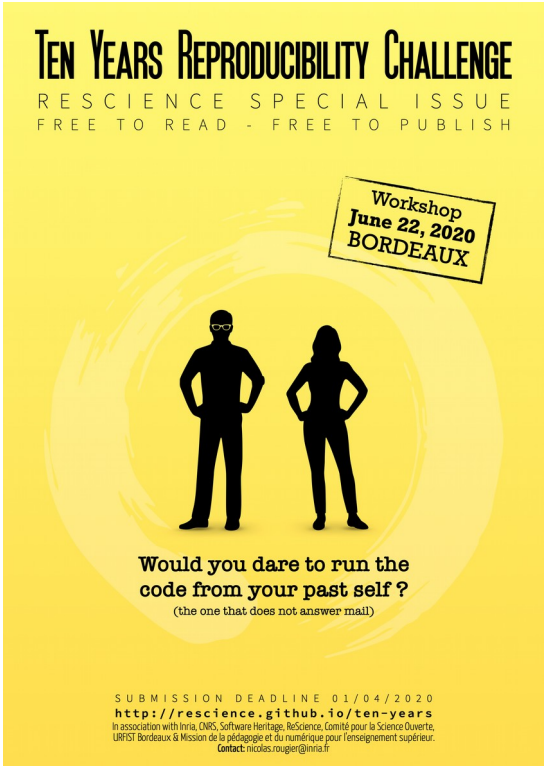
DIAMOND DEPENDENCY

Using Nix for Repeatable Python Environments
SciPy 2019 – Daniel Wheeler

Reproducability

Reproducibility / Repeatability

https://www.nature.com/articles/d41586-020-02462-7



Reproducibility / Repeatability

Not about proving software provenance. **Is** about operational concerns:

- How can I onboard a new developer in under a week?
  - Reproducible software requires reproducible build environments
  - Reproducible build environments are also **development** environments
- How do I containerise this thing?
  - Unless shipping static binaries, you should package apps within a container
  - Dockerfiles are a good start, but can have surprising semantics.
- Why can't I replicate that production bug in dev?
  - Someone patched the system libraries and I didn't notice (true story)
- How do I release my code along with my academic paper?
  - Will my peers be able to reproduce my results in 10 years time?

Introducing Nix

> **Nix** is a powerful package manager for Linux and other Unix systems that makes package management **reliable** and **reproducible**.

What is “Nix”?

- **Nix** – pure functional, lazy, configuration language
- *Nix* – package manager that builds and manages environments
  - Over 100 Contributors since 2003
  - A swiss-army knife of tools: nix-env, nix-shell, nix-channel
- **Nixpkgs** – community curated collection of Nix(lang) package definitions
  - Nixpkgs has over 40 000 packages, over 3000 Python packages
  - Multiple Stable releases per year, fast-moving unstable channel
- ***NixOS*** – GNU/Linux distribution configured declaratively with Nix(lang)
  - Module system makes it simple to specify services and configuration within Nix.

Nix Vocabulary

- **Expression** – A **pure function** that describes your desired package. Dependencies become parameters to this function.
- **Derivation** – The result of an expression, **instantiated** into a .drv file, that concretely defines the build inputs and build process for a package.
- **Result** – The **build output** from following the rules in a Derivation. Contains only absolute paths other derivations in it's **closure**.
- **Closure** – The **graph of outputs** referenced by a **Result.**
- **Store** – The **graph database** that **outputs** are cached into

Installing Nix

Single User installation - for development

- `$ sh <(curl https://nixos.org/nix/install) --no-daemon`

Multi-User installation – for deployment

- `$ sh <(curl https://nixos.org/nix/install) --daemon`

Enable non-free packages (e.g. chrome webdriver)

- `$ echo "{allowUnfree=true;}" > ~/.config/nixpkgs/config.nix`

Uninstall

- Remove path hook from ~/.profile
- `$ rm -rf /nix`

https://nixos.org/manual/nix/stable/#chap-quick-start

Nix-env ad-hoc package management

Find a Package

- `$ nix-env -query --available python3`

Install a package

- `$ nix-env --install python37`
  `installing 'python3-3.7.8'`

Upgrade a package

- `$ nix-env --upgrade python3`
  `upgrading 'python3-3.7.8' to 'python3-3.9.0b5' ...`

Remove a package

- `$ nix-env --uninstall python3`
  `uninstalling 'python3-3.9.0b5' ...`

Nix-store

```
twoolie@thinclient : ~
[0] % nix-store --query --requisites $(which python) | cat
/nix/store/y7ya11l8qy5z2zyhc8db7km7kykkl1sw-libunistring-0.9.10
/nix/store/zzp30ib8rgm42d80gw3jl5cy5vgvaq9y-libidn2-2.3.0
/nix/store/2pi6zgkwnr3zdslvlv16nixpzvbyjx1n-glibc-2.31
/nix/store/1miiswm7gb1jr6k161llxvbxfzza00w2-gdbm-1.18.1
/nix/store/4ix6qybncqc0n65q9gqqkw4l1rq7xyj4-xz-5.2.5
/nix/store/5h540ccjpnjwjpwns35c9a8d89rhli46-expat-2.2.8
/nix/store/6c8sdzvzq0145pjdx7h10y1414x9hljj-libffi-3.3
/nix/store/9mjf7b7rc8g25wh7kbvh3l41s109spn7-bzip2-1.0.6.0.1
/nix/store/synxrw09w934ld6mnx0d9cndf51ak99b-zlib-1.2.11
/nix/store/m6zv2a7kl7f6w4m3x8fgaanbd9ipijvy-sqlite-3.32.3
/nix/store/npfsrhkjww5q7sax7p7ijcrj3wlbrxn7-bash-4.4-p23
/nix/store/zcg8cyz14p8q1d51w6f8ybc4fvl28f6i-ncurses-6.2
/nix/store/xhxply22pim9sx5b8fi2xsdxy67c9g6k-readline-6.3p08
/nix/store/y6rqpym24xy2jiq79hhkdy7316s7cdij-openssl-1.1.1g
/nix/store/n7kkqxl7ilnjfiir4rqja5f9mdig6ban-python3-3.7.8
```

Maintenance

Update package definitions (active channels are 19.09, 20.03, unstable)

- `$ nix-channel --update`

Rollback updates

- `$ nix-env -rollback`
  `switching from generation 3 to 2`

Cleanup unreferenced derivations and generations

- `$ nix-garbage-collect --delete-older-than 30d`
- `$ nix-store --gc`

Hardlink duplicate files in store to save space

- `$ nix-store --optimize`

Taming the Snakes

Nix-build declarative package management

```
$ nix-build env.nix -o ./env
```

- Saves a symlink to the build environment, prevents nix gc from removing referenced derivations

```
$ nix-shell env.nix
```

- Starts a transient sub-shell with the listed packages available on the path

env.nix

```
{ pkgs ? import <nixpkgs> {} }:

pkgs.buildEnv {
    name = "project-env";
    paths = [
        pkgs.chromedriver
        (pkgs.python38.withPackages (ps: [
            ps.selenium
            ps.flask
        ]))
    ];
}
```

Anatomy of a Nix Derivation

- This file contains a function that takes dependencies as arguments and returns a Derivation for a python package
- Package will be downloaded from pypi, but must match the hash to be valid
  - Source is cached in nix store
- Built in a sandbox, can't do networking.
- Customary to provide metadata for tooling. Powers search.nixos.org

```
{ buildPythonPackage, fetchPypi, lib }:

buildPythonPackage rec {
  pname = "dnspython";
  version = "1.16.0";

  src = fetchPypi {
    inherit pname version;
    extension = "zip";
    sha256 = "36c5e8e38d4369a08b6780b7f27d790a292b2b08eea01607865bf0936c558e01";
  };

  # needs networking for some tests
  doCheck = false;

  meta = {
    description = "A DNS toolkit for Python 3.x";
    homepage = "http://www.dnspython.org";
    # BSD-like, check http://www.dnspython.org/LICENSE for details
    license = lib.licenses.free;
  };
}
```

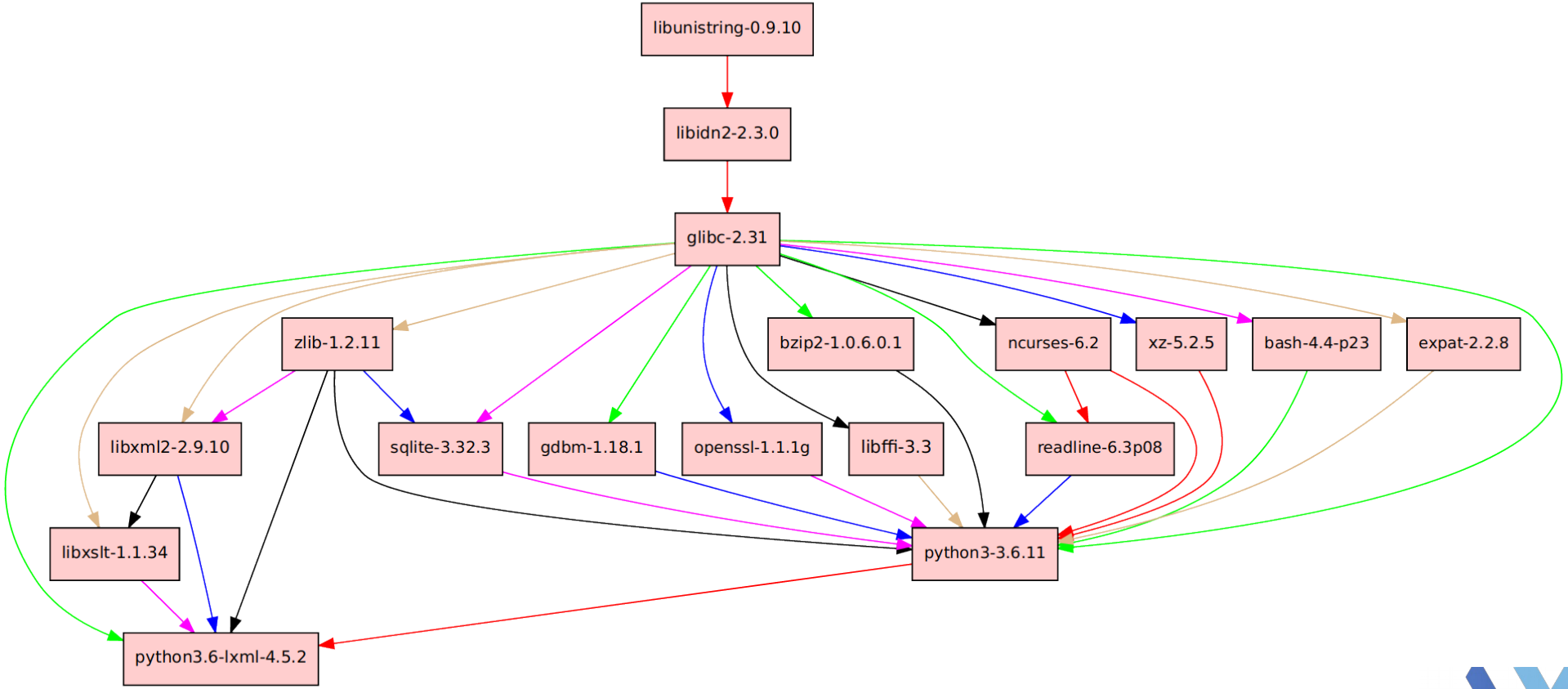
https://github.com/NixOS/nixpkgs/blob/master/pkgs/development/python-modules/dnspython/default.nix

Integrating C Libs

- Tarball pulled from Github by tag name.
- Native Build Inputs are from host system
  - “dev” package contains headers
  - LXML links against libxml2 & libxslt
- Build Inputs are from target system
  - Cross-compilation compatible
- Don’t run test-suite, but check that resulting modules import cleanly.

```
{ lib, buildPythonPackage, fetchFromGitHub
, cython, libxml2 , libxslt, zlib }:

buildPythonPackage rec {
  pname = "lxml";
  version = "4.5.2";

  src = fetchFromGitHub {
    owner = pname;
    repo = pname;
    rev = "${pname}-${version}";
    sha256 = "1d0cpwdjxfzwjzmnz066ibzicyj2vhx15qxmm775l8hxqi65xps4";
  };

  # setuptoolsBuildPhase needs dependencies to be passed through nativeBuildInputs
  nativeBuildInputs = [ libxml2.dev libxslt.dev cython ];
  buildInputs = [ libxml2 libxslt zlib ];

  # tests are meant to be ran "in-place" in the same directory as src
  doCheck = false;

  pythonImportsCheck = [ "lxml" "lxml.etree" ];

  meta = with lib; {
    description = "Pythonic binding for the libxml2 and libxslt libraries";
    homepage = "https://lxml.de";
    license = licenses.bsd3;
    maintainers = with maintainers; [ jonringer sjourdois ];
  };
}
```

https://github.com/NixOS/nixpkgs/blob/master/pkgs/development/python-modules/lxml/default.nix


libunistring-0.9.10
libidn2-2.3.0
glibc-2.31
zlib-1.2.11
bzip2-1.0.6.0.1
ncurses-6.2
xz-5.2.5
bash-4.4-p23
expat-2.2.8
libxml2-2.9.10
sqlite-3.32.3
gdbm-1.18.1
openssl-1.1.1g
libffi-3.3
readline-6.3p08
libxslt-1.1.34
python3-3.6.11
python3.6-lxml-4.5.2

Pinning the Package Set

```
let nixpkgs = (builtins.fetchTarball {
    # Descriptive name to make the store path easier to identify
    name = "nixos-unstable-2020-08-21";
    # Commit hash for nixos-unstable as of 2020-08-21
    url = "https://github.com/nixos/nixpkgs/archive/c59ea8b8a0e7f927e7291c14ea6cd1bd3a16ff38.tar.gz";
    # Hash obtained using `nix-prefetch-url --unpack <url>`
    sha256 = "1ak7jqx94fjhc68xh1lh35kh3w3ndbadprrb762qgvcfb8351x8v";
});
in { pkgs ? import nixpkgs {} }:

pkgs.mkShell {
    buildInputs = [
        (pkgs.python38.withPackages (ps: with ps; [ lxml GeoIP flask ]))
    ];
}
```

Tooling and Ecosystem

Many other tools included

Nix graph database as a platform, Nixpkgs as a standard library.
Functions and data-structures can be composed to provide:

- Vim/Emacs configuration tools and plugins
- Cross-compilation tool-chains
- Docker image creation tools – Check out nixery.dev !
- OS image building tools
- Container/VM based system test tools

MachNix

Pros:

- Easy to choose providers on a per-package basis.
- Pulls packages from nixpkgs, PyPi (sdist or wheels).
- Mostly automatic dependency resolution with requirements files.

Cons:

- Giant (~200MB) pypi-dep-db download.

```
let
  mach-nix = import (builtins.fetchGit {
    url = "https://github.com/DavHau/mach-nix/";
    ref = "refs/tags/2.3.0";
  });
in
mach-nix.mkPython {
  requirements = ''
    pillow
    numpy
    requests
  '';
}
```

poetry2nix

Pros:

- Built into nixpkgs, no need to install
- Consumes existing poetry TOML and lockfile
- Poetry lockfile already contains hashes
- Supports building from local or remote source
  - Local source relative to nix file (./.)
  - Remote source from web (fetchTarball)

poetry.nix

```
{ pkgs ? import <nixpkgs> {} }:

let pythonEnv = pkgs.poetry2nix.mkPoetryEnv {
    python = pkgs.python38;
    projectDir = ./.;
};
in pkgs.mkShell {
    buildInputs = [ pythonEnv ];
}
```

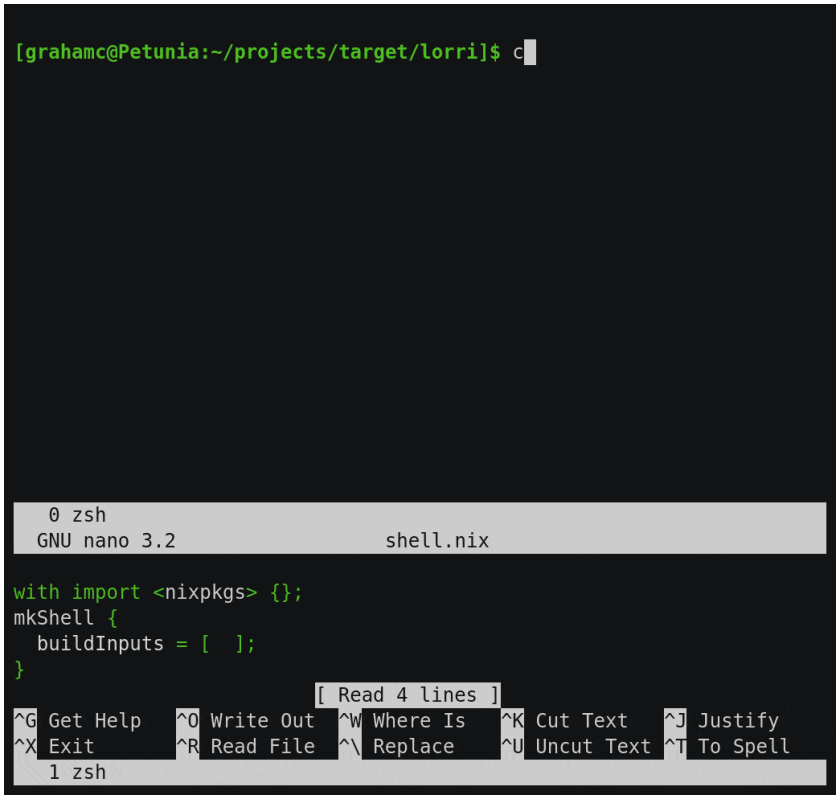
```
{ pkgs ? import <nixpkgs> {} }:

let pythonApp = pkgs.poetry2nix.mkPoetryApplication {
    projectDir = fetchTarball {...};
};
in pkgs.mkShell {
    buildInputs = [ pythonApp ];
}
```

Lorri/direnv

- Switches to your nix-shell environment when you cd into a project directory

```
$ nix-env --install direnv lorri
$ lorri init
$ direnv allow
```

- Lorri daemon will rebuild/reload your shell environment when shell dependencies change



Deploying to docker

Docker image can be cross-built from mac

```
$ nix-build docker.nix \
  -A docker -o image-out
```

Nix packs one dependency per layer

```
$ ./image-out | docker load
Loaded image: mypythonapp:…
```

Run docker container just like any other

```
$ docker run --rm -it mypythonapp
Hello PyCon 2020
```

docker.nix

```
let
    overlay = self: super: {
        mypythonapp = self.poetry2nix.mkPoetryApplication {
            projectDir = ./.;
            python = self.python3;
        };
    };
    hostPkgs  = import <nixpkgs> { overlays = [ overlay ]; };
    linuxPkgs = import <nixpkgs> {
        overlays = [ overlay ]; system = "x86_64-linux"; };
in
{
    inherit (hostPkgs) mypythonapp;
    docker = hostPkgs.dockerTools.buildLayeredImage {
        name = "mypythonapp";
        contents = [ linuxPkgs.mypythonapp ];
        config.Cmd = [ "mypythonapp" ];
    };
}
```

Example code at https://github.com/twoolie/pyconau2020

Fantastic Resources

- https://nix.dev – An opinionated guide for developers getting things done using the Nix ecosystem.
- https://nixos.org/manual/nix/stable/ – The Nix Manual is incredibly comprehensive
- https://nixos.org/guides/nix-pills/index.html – Step-by-Step guide from the ground up
- https://nixos.wiki/wiki/Python – The Nix Wiki page for python has many tips and examples
- https://github.com/nix-community/poetry2nix – The poetry2nix upstream project
- https://discourse.nixos.org/ - NixOS/Nixpkgs forum and organizing platform. Ask many questions!
- Using Nix for Repeatable Python Environments | SciPy 2019 | Daniel Wheeler
- Beyond Python packaging with Nix | PyCon PL 2020 | Asko Soukka

Thanks for Watching

Questions?