I declare an environment!

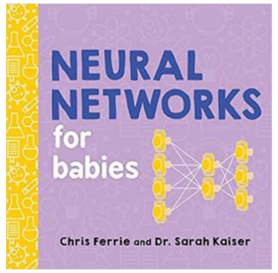
Reproducibility with and *without* Docker

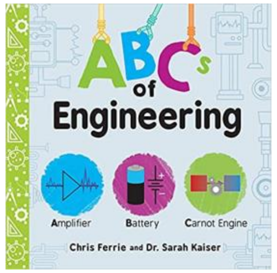
Dr. Sarah Kaiser (she/they)

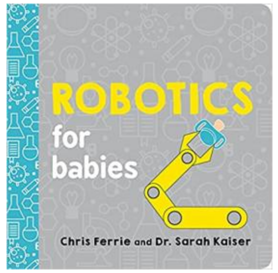
Sr. Developer Advocate @ Microsoft

Hiya! 👋

- 🛰️ Experimental Physicist by training
- 🐕 Chewie kibble delivery
- 📚 Author of books for all ages
- ⚔️ Plays anything Final Fantasy
- 🚢 Usually floating near Seattle

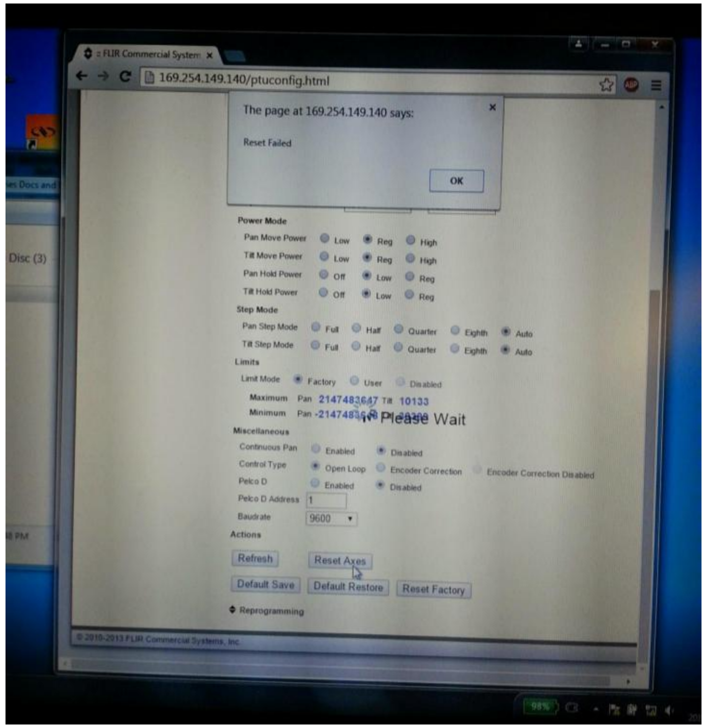






Computers do the darndest things

- Install updates
- Get lost
- Get arrested
- Fall out of orbit
- Get sat on
- Catch on fire



:: FLIR Commercial System

169.254.149.140/ptuconfig.html

The page at 169.254.149.140 says:

Reset Failed

OK

Power Mode

Pan Move Power: Low [✓ Reg] High
Tilt Move Power: Low [✓ Reg] High
Pan Hold Power: Off [✓ Low] Reg
Tilt Hold Power: Off [✓ Low] Reg

Step Mode

Pan Step Mode: Full Half Quarter Eighth [✓ Auto]
Tilt Step Mode: Full Half Quarter Eighth [✓ Auto]

Limits

Limit Mode: [✓ Factory] User Disabled
Maximum Pan 2147483647 Tilt 10133
Minimum Pan -21474836[illegible] Tilt [illegible]

Please Wait

Miscellaneous

Continuous Pan: Enabled [✓ Disabled]
Control Type: [✓ Open Loop] Encoder Correction Encoder Correction Disabled
Pelco D: Enabled [✓ Disabled]
Pelco D Address: 1
Baudrate: 9600

Actions

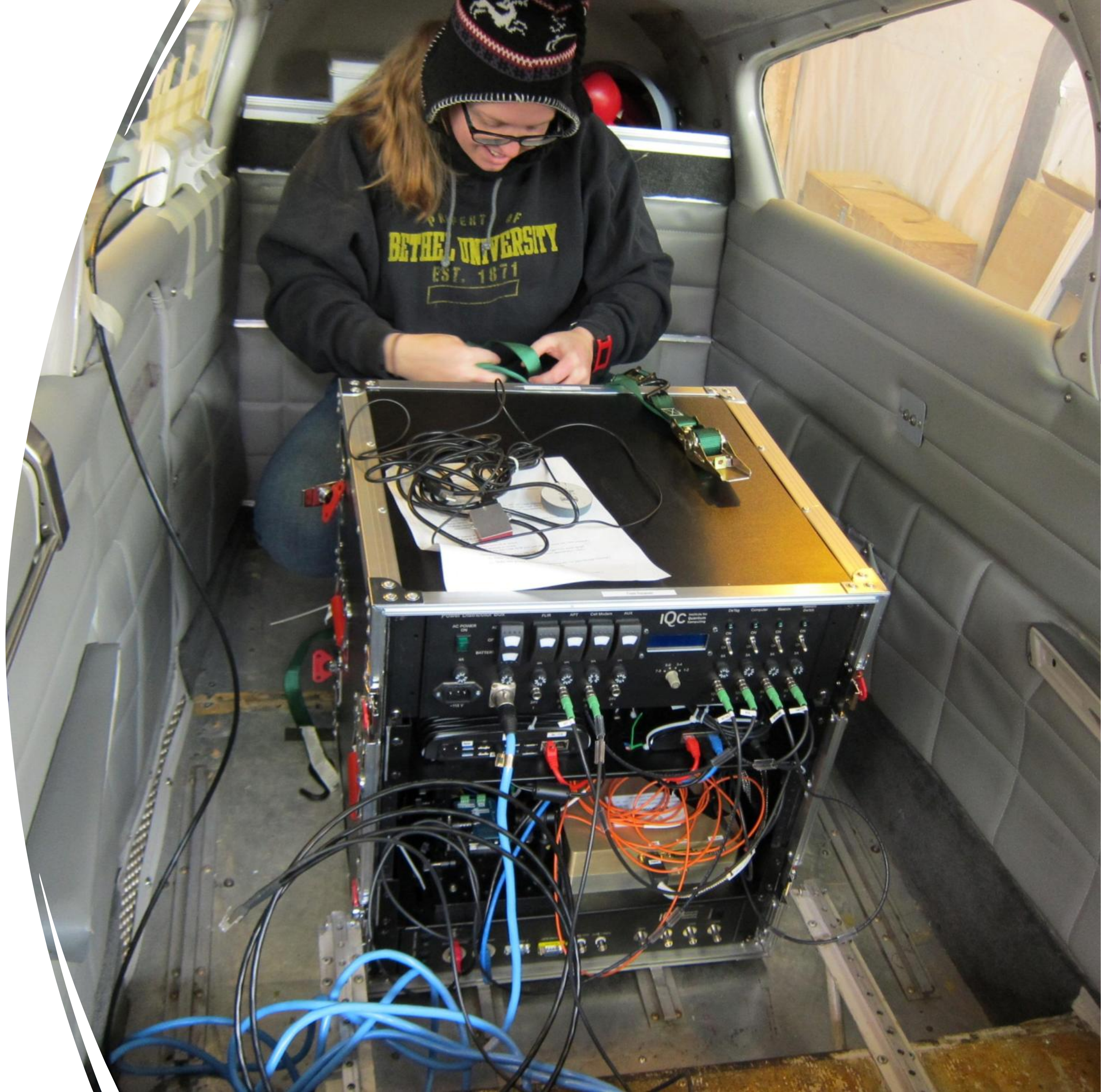
Refresh Reset Axes
Default Save Default Restore Reset Factory

Reprogramming

© 2010-2013 FLIR Commercial Systems, Inc.

What is the best way to make sure my ___ works tomorrow?

Project, lab, house, car, etc.



tl;dr

1. Good reproducible environments need **declarative** and **imperative** specs.

2. A combination of Docker and Nix can make reproducible Python projects easier!

Reproducible...

for who?

on what device?

when?

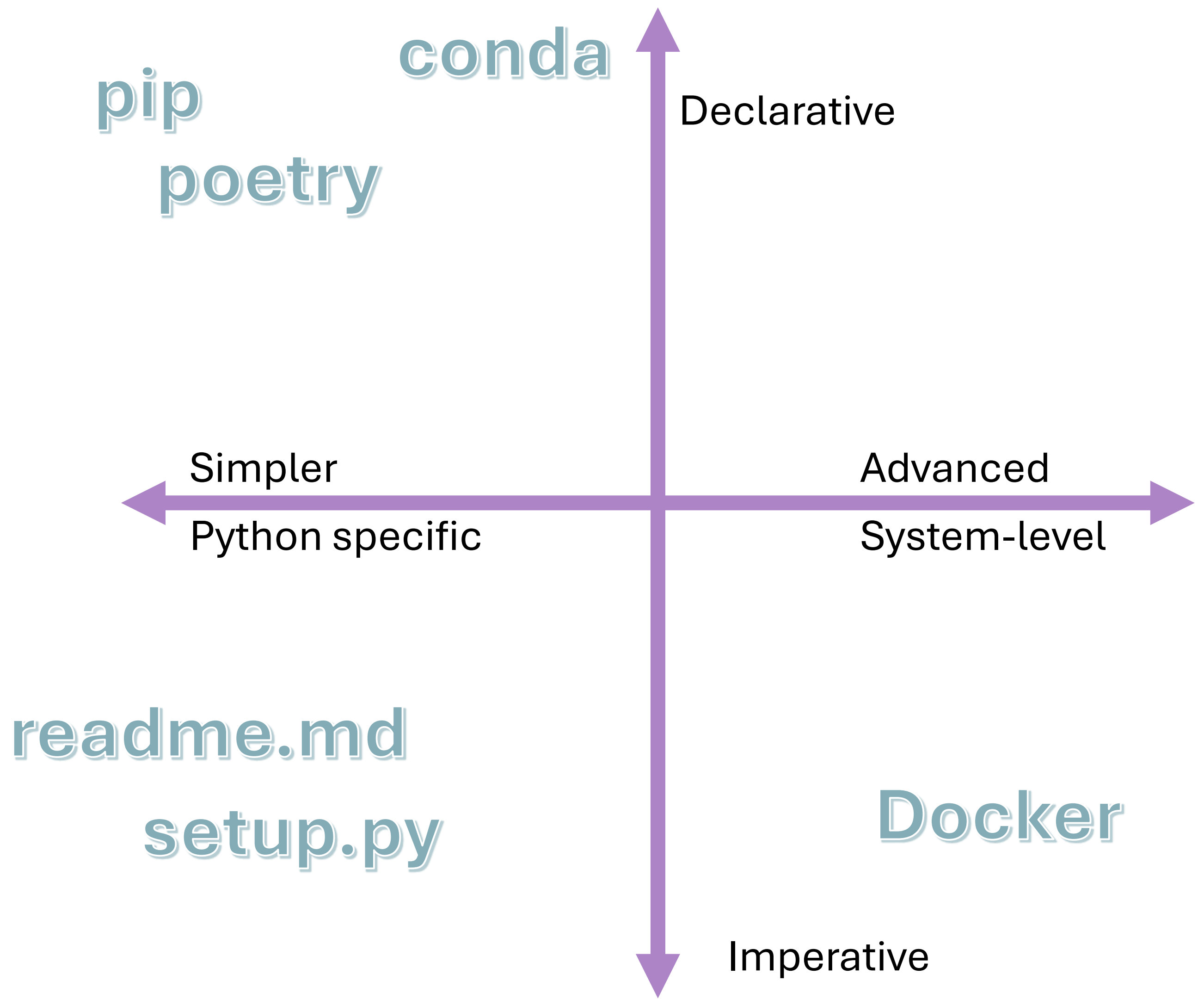
with what resources?

...Environments!

Declarative: Describe the desired final state as completely as possible

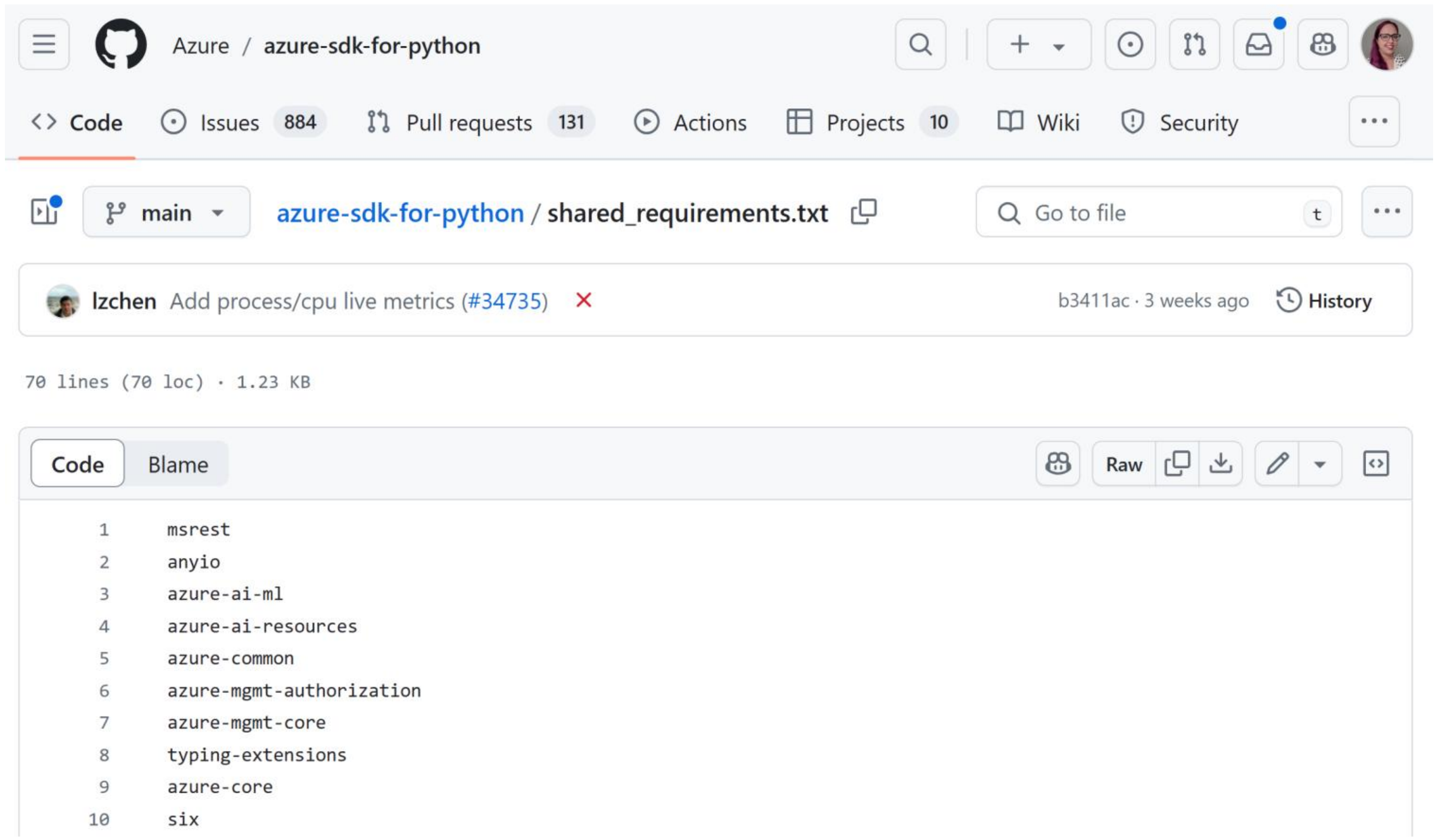
Imperative: Provide a list of steps or actions

What kinds of tools do people use?



Simple declarative: `requirements.txt`

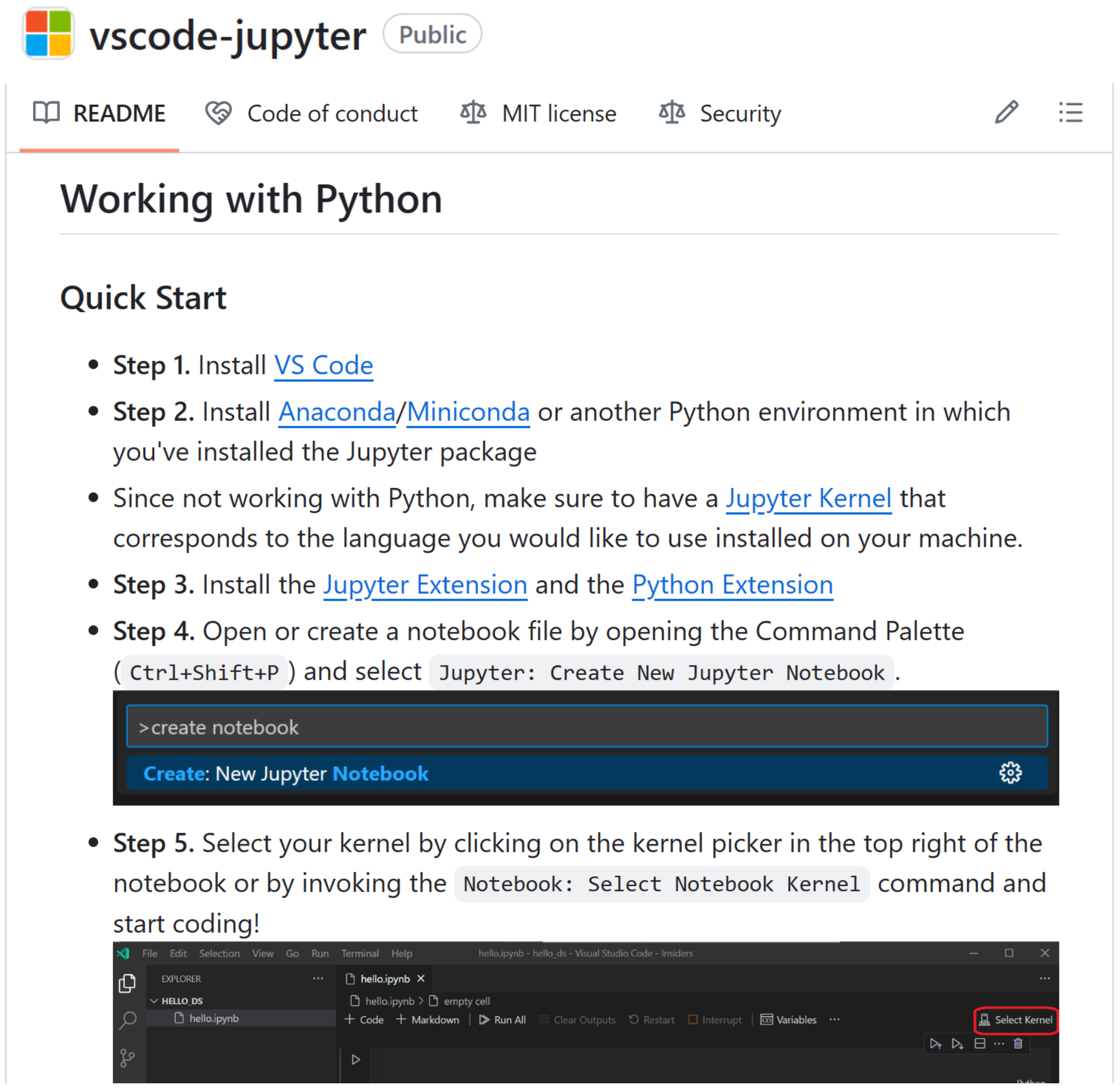
- `pyproject.toml`, `environment.yml`
- May not capture all non-Python requirements
- Can be hard to manage without additional tools


Azure / azure-sdk-for-python
Code
Issues 884
Pull requests 131
Actions
Projects 10
Wiki
Security
main
azure-sdk-for-python / shared_requirements.txt
Go to file
lzchen Add process/cpu live metrics (#34735)
b3411ac · 3 weeks ago
History
70 lines (70 loc) · 1.23 KB
Code
Blame
Raw

```
msrest
anyio
azure-ai-ml
azure-ai-resources
azure-common
azure-mgmt-authorization
azure-mgmt-core
typing-extensions
azure-core
six
```

Simple imperative: `readme.md`

- Easy to write
- Hard to maintain
- Need to test, automation testing can help



vscode-jupyter Public

README | Code of conduct | MIT license | Security

Working with Python

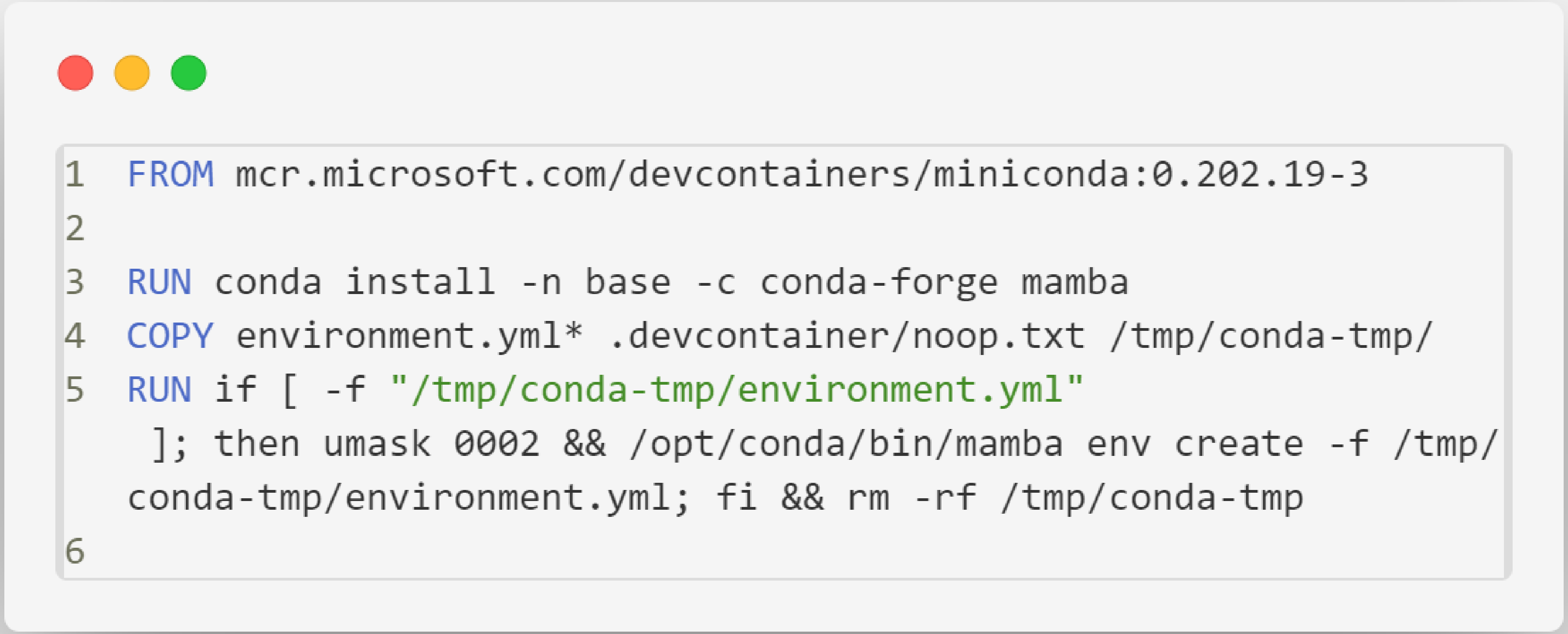
Quick Start

- **Step 1.** Install VS Code
- **Step 2.** Install Anaconda/Miniconda or another Python environment in which you've installed the Jupyter package
- Since not working with Python, make sure to have a Jupyter Kernel that corresponds to the language you would like to use installed on your machine.
- **Step 3.** Install the Jupyter Extension and the Python Extension
- **Step 4.** Open or create a notebook file by opening the Command Palette (`Ctrl+Shift+P`) and select `Jupyter: Create New Jupyter Notebook`.

- **Step 5.** Select your kernel by clicking on the kernel picker in the top right of the notebook or by invoking the `Notebook: Select Notebook Kernel` command and start coding!

File Edit Selection View Go Run Terminal Help
hello.ipynb - hello_ds - Visual Studio Code - Insiders
EXPLORER
HELLO_DS
hello.ipynb
hello.ipynb > empty cell
+ Code + Markdown Run All Clear Outputs Restart Interrupt Variables
Select Kernel

Advanced Imperative: Dockerfile



```
FROM mcr.microsoft.com/devcontainers/miniconda:0.202.19-3

RUN conda install -n base -c conda-forge mamba
COPY environment.yml* .devcontainer/noop.txt /tmp/conda-tmp/
RUN if [ -f "/tmp/conda-tmp/environment.yml"
 ]; then umask 0002 && /opt/conda/bin/mamba env create -f /tmp/
conda-tmp/environment.yml; fi && rm -rf /tmp/conda-tmp

```

Why Docker?

- Pros:
  - Supported most places
  - Easy to hack fixes in
  - Works with Dev Containers! 💖
- Cons:
  - Installing and learning Docker is not trivial
  - Easy to hack fixes in
  - Devices may not support

Something is missing…

nix.dev

Advanced Declarative: Nix

A language for declarative environments

Nix ecosystem

Nixpkgs

- The largest, most up-to-date software distribution in the world, and written in the Nix language.

NixOS

- Entire operating system that bootstraps from the nix package manager and nix-shell

nix-shell

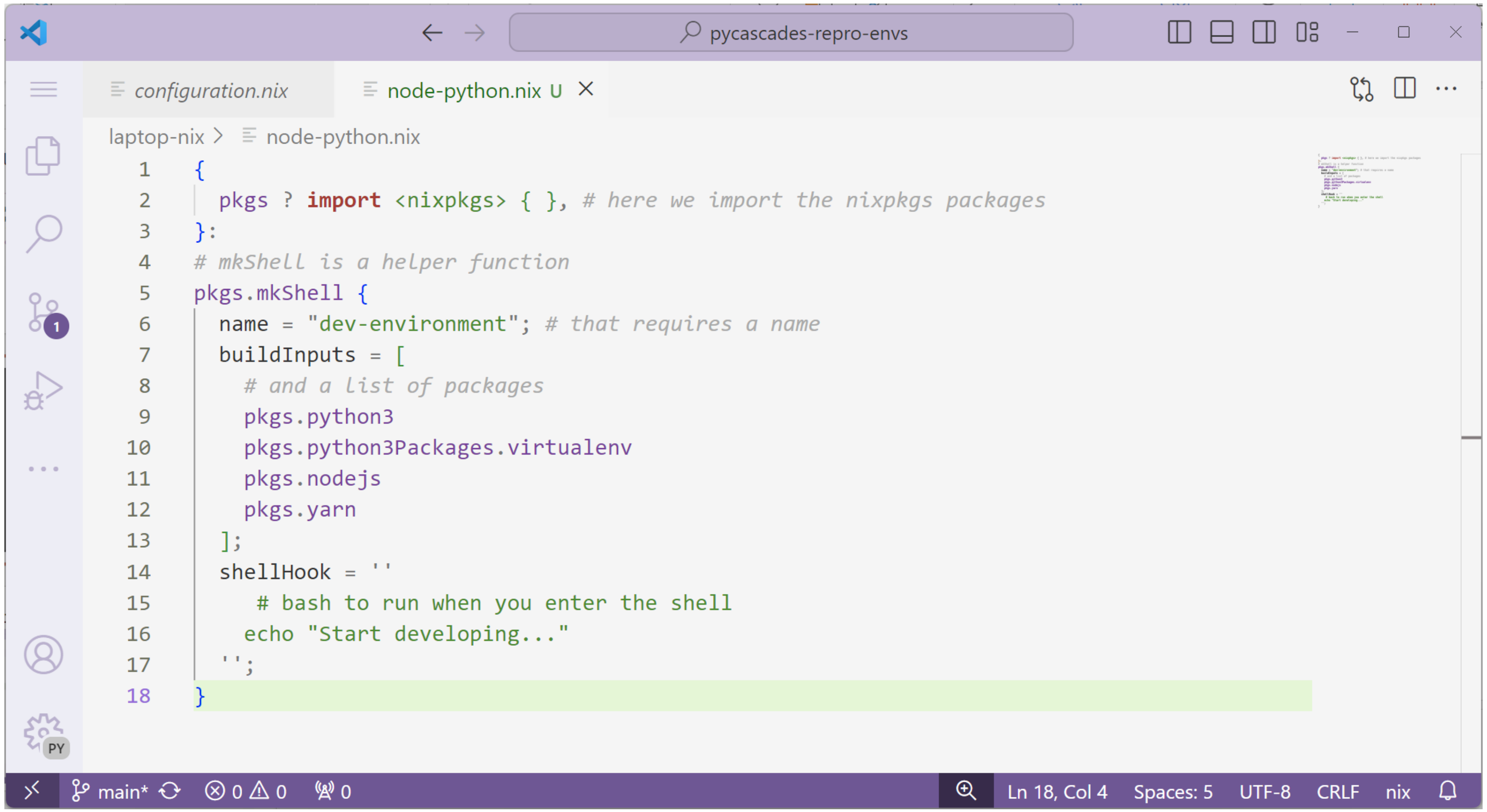
- Declarative shell environments

Advanced Declarative: `nix-shell`

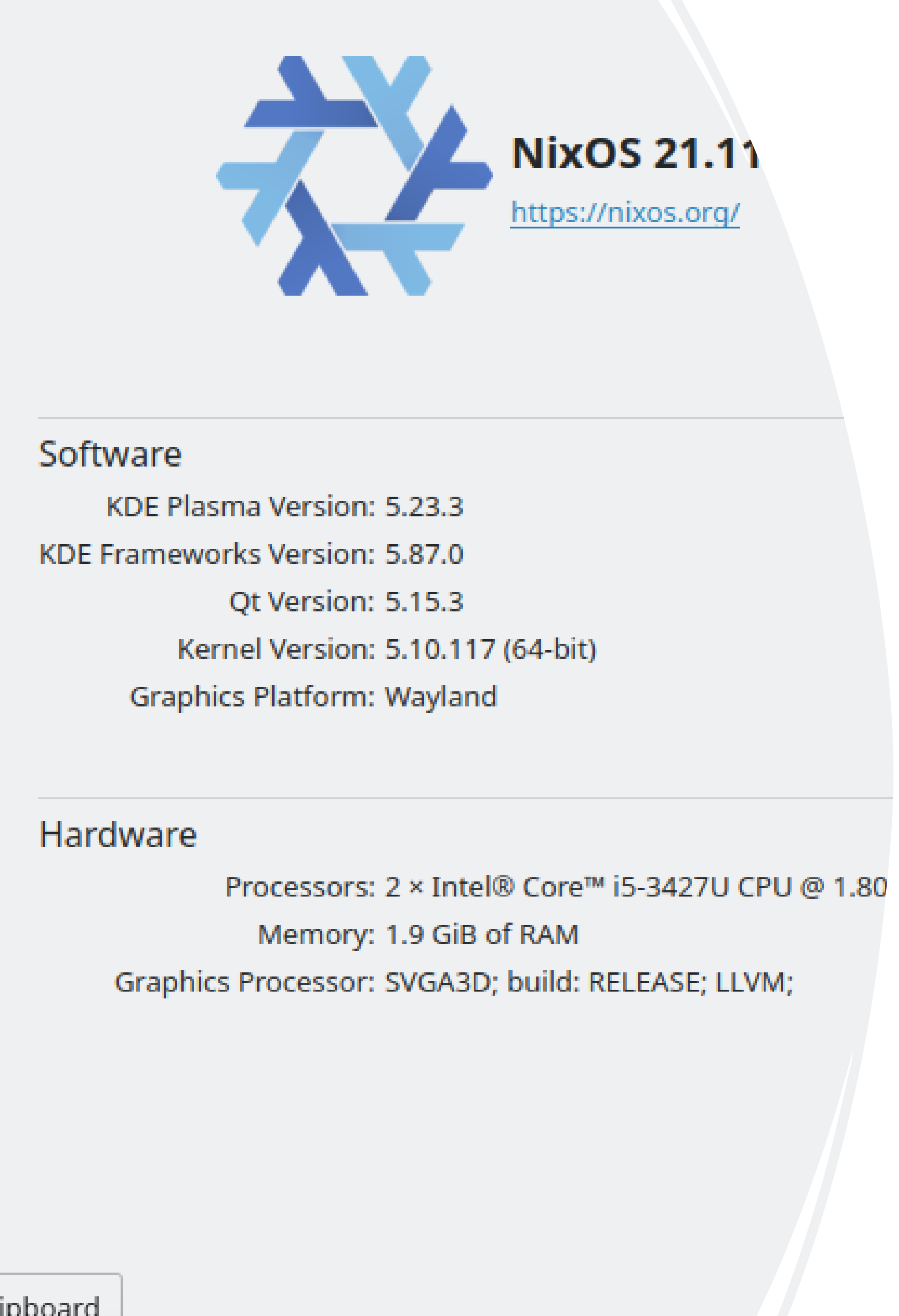
```
[nix-shell:~]$ nix-shell -p python3
this path will be fetched (11.42 MiB download, 62.64 MiB unpacked):
  /nix/store/pwy30a7siqrkki9r7xd1lksyv9fg4l1r-python3-3.10.11
copying path '/nix/store/pwy30a7siqrkki9r7xd1lksyv9fg4l1r-python3-3.10.11'
from 'https://cache.nixos.org' ...

[nix-shell:~]$ python --version
Python 3.10.11
```

[illegible]



```nix
{
  pkgs ? import <nixpkgs> { }, # here we import the nixpkgs packages
}:
# mkShell is a helper function
pkgs.mkShell {
  name = "dev-environment"; # that requires a name
  buildInputs = [
    # and a list of packages
    pkgs.python3
    pkgs.python3Packages.virtualenv
    pkgs.nodejs
    pkgs.yarn
  ];
  shellHook = ''
     # bash to run when you enter the shell
    echo "Start developing..."
  '';
}
```


NixOS 21.11

https://nixos.org/

Software

KDE Plasma Version: 5.23.3
KDE Frameworks Version: 5.87.0
Qt Version: 5.15.3
Kernel Version: 5.10.117 (64-bit)
Graphics Platform: Wayland

Hardware

Processors: 2 × Intel® Core™ i5-3427U CPU @ 1.80
Memory: 1.9 GiB of RAM
Graphics Processor: SVGA3D; build: RELEASE; LLVM;

Why Nix?

- Pros:
  - Package manager covers whole system
  - Uses similar container infra to Docker so it can interoperate
  - Efficient cache management
- Cons:
  - Not as widespread in Python community
  - Can be harder to write in the first place to get it just right

Ok, what should I choose? 😵‍💫

*Opinions are my own, your milage may vary

In a vacuum, I would go fully declarative via Nix.

In real life, why not both?!

Start with Nix:

- Make a shell.nix to specify a project environment
- Add Python and any other tools needed (e.g.: LLVM, Clang, Fortran, etc.)
- Use direnv/pipenv to activate a Nix environment when you enter a directory

add Docker when it gets hard:

-Use nix-shell in a Docker container for portability

-Describe the steps that are hard to make declarative

- Export nix package to an OCI images for interoperability

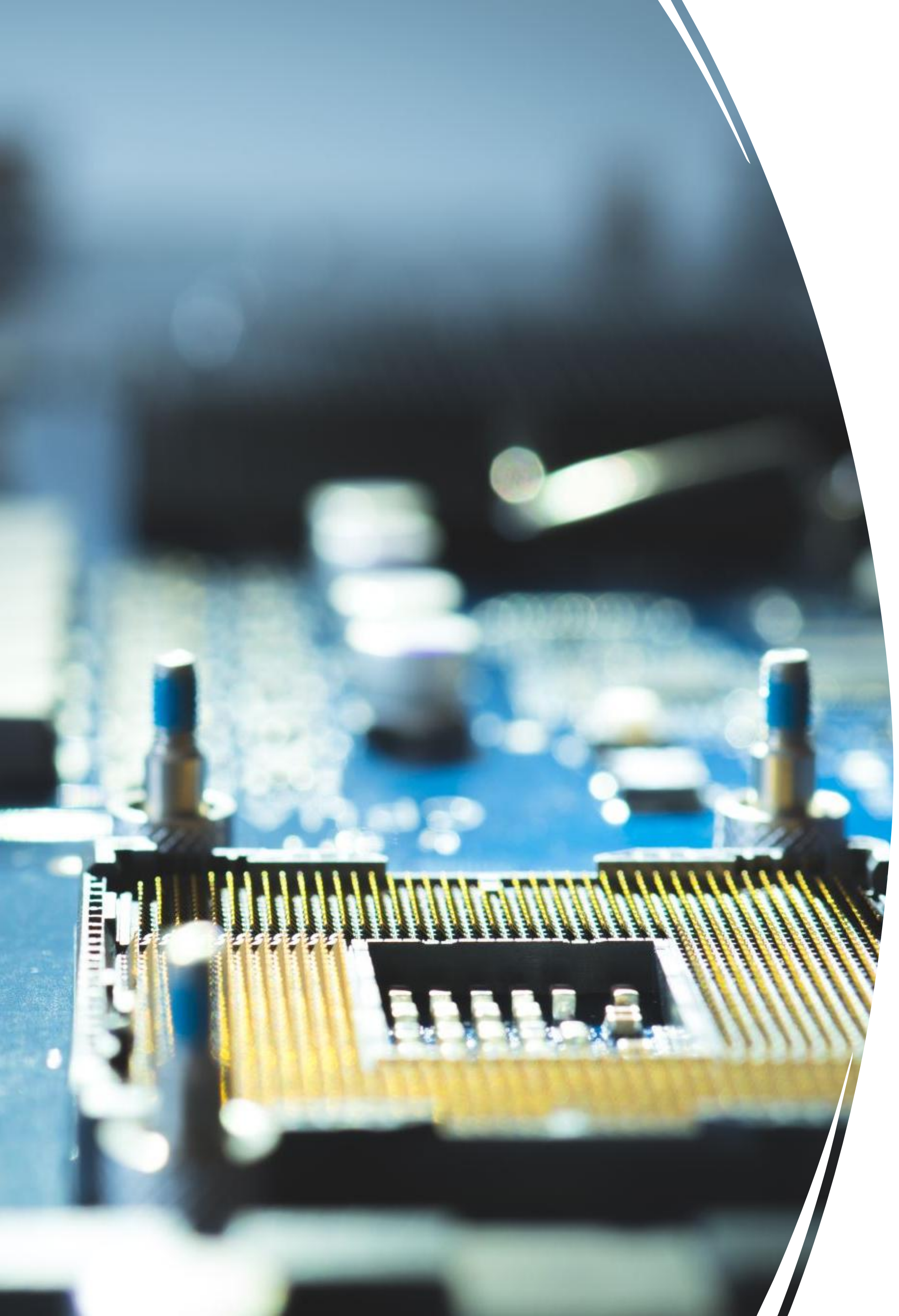
Sample python project config

```
# shell.nix
let
  pkgs = import <nixpkgs> {};
in pkgs.mkShell {
  packages = [
    (pkgs.python3.withPackages (python-pkgs: [
      # select Python packages here
      python-pkgs.pandas
      python-pkgs.requests
    ]))
  ];
}
```

Case study: My home lab

Requirements:

- Needs to be portable across devices/hardware
- Runs/hosts Docker containers and local services
  - Home Assistant
  - PiHole
  - Wiki
  - Networking
  - Etc.
- Easy to maintain and back up



The NixOS experiment

- Declare the entire operating system and put it in source control
- Needed to use docker to hack some pieces in we couldn’t get to work right
- We are still learning 😄

Sample NixOS file: **Hardware**

```nix
{ config, pkgs, ... }:

{
  imports =
    [ # Include the results of the hardware scan.
      ./hardware-configuration.nix
      <home-manager/nixos>
    ];

  # Bootloader.
  boot.loader.systemd-boot.enable = true;
  boot.loader.efi.canTouchEfiVariables = true;

  # Define your hostname.
  networking.hostName = "nixos-demo";

  # Enable networking
  networking.networkmanager.enable = true;

  services.tailscale.enable = true;
  services.flatpak.enable = true;

  # Set your time zone.
  time.timeZone = "America/Los_Angeles";
```

Sample NixOS file: **Users**

```
  # Define a user account.
  users.users.demouser = {
    isNormalUser = true;
    description = "demouser";
    extraGroups = [ "networkmanager" "wheel"
"docker"];
    packages = with pkgs; [
        discord
        slack
        signal-desktop
        element-desktop
        hyper
    ];
  };
```

Sample NixOS file: **Home manager**

```
  home-manager.users.demouser = { pkgs, ... }:
{
    home.packages = [ pkgs.atool pkgs.httpie ];
    programs.bash.enable = true;
    programs.nushell.enable = true;
    programs.starship.enable = true;
    programs.starship.settings = {
      username = {
        show_always = true;
        style_user = "bg:purple";
        style_root = "bg:purple";
        format = "[✨]($style)";
      };
      directory = {
        style = "bg:red";
        format = "[ $path ]($style)";
        truncation_length = 3;
        truncation_symbol = "…/";
      };
    };
    home.stateVersion = "23.11";
  };
```

Conclusions

Reproducable Environments

We need **declarative** and **imperative** descriptions of our projects

Docker containers are great, but can be hard to maintain

Nix is a declarative language for environments that can improve reproducibility

I declare...
an environment

Thanks! 💖

aka.ms/pycascades-repro-envs

aka.ms/pyc24

aka.ms/python-discord

mathstodon.xyz/@crazy4pi314

sckaiser.com