Nix:

A Package Manager for Reproducible Environments

dav009

dav009

Slides, samples and commands:
https://github.com/dav009/nixtalk

Ec2 instance, lets boot it

The problem

⭐ Reproducibility

Reproducibility sits at the core of

Software engineering:
- **Same code leads to the same Artefact**
- **Running the same Artefact in the same environment**

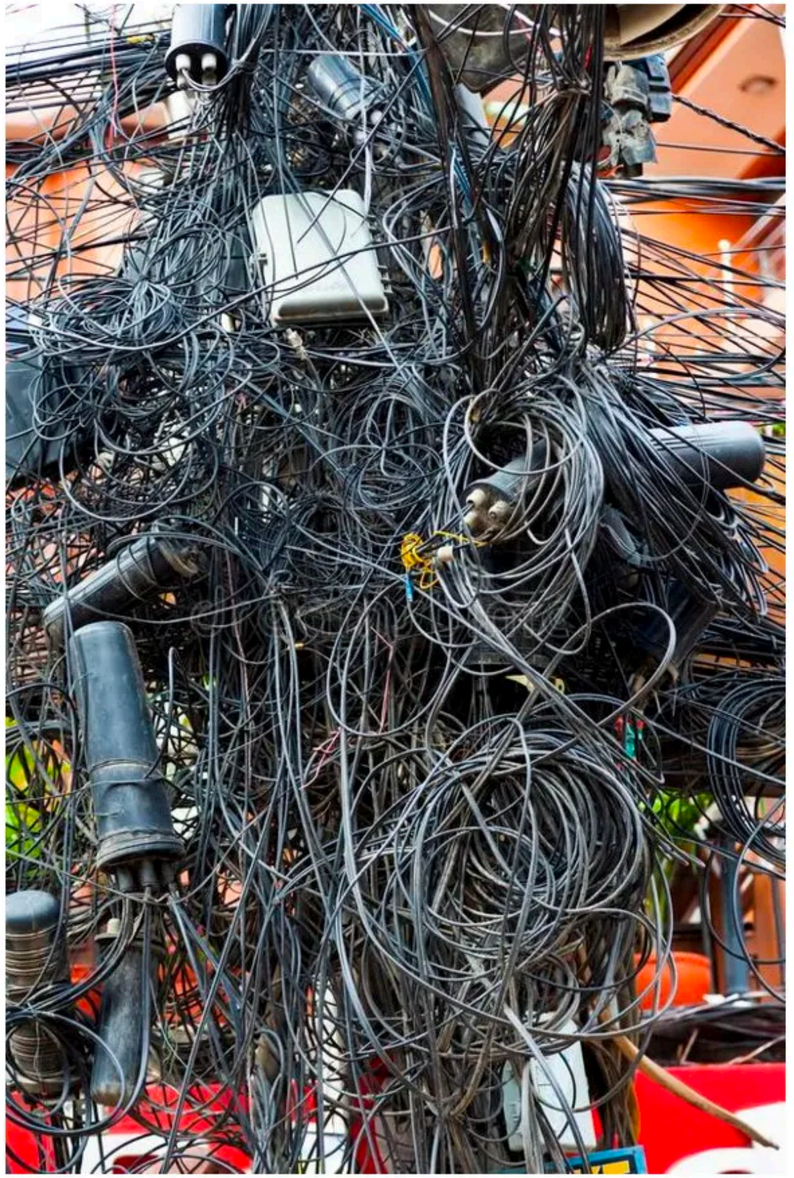
Machine learning:
- **Same data + Same code = Same results**
- **Same transformations in model training and model serving**

Reproducibility is hard

Symptoms of reproducibility issues:

First week in your new job?

Case 1: Spend your first week setting up your local dev environment



Case 2:

Case 3:

You want to contribute to that OSS project

but.. setting up the local environment will take you a few hours

Disparate environments fighting

Case 4:

Disparate environments / multiple moving parts:

1. Local development environments

2. Production environment
   - container image

3. CI environment
   - Github action

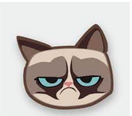
Symptom

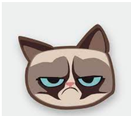
You update a version of some library
now you have to make changes to these three environments

Case 5:

You installed a global dependency: e.g: brew install or apt-get install

It fixed your problem of the moment 🥳

Later to find out you broke something 

Worst case: You end up breaking your entire system 

Case 6:

Multiple versions of X in different projects

e.g:
Project 1 uses Golang v1
Project 2 uses Golang v2
…

Hard to work on a project without breaking another

…Current solutions..

Language specific environment tooling

Promise: Isolated environments

- **Python**: conda, pyvenv, pipx..
- **Javascript**: npm ci…

Issue: these tools have limited scope

Example: Python case

many c-level libraries
- how do you install this isolated from the rest of your system?

Current state of Python isolated environment tools


pip
Conda
mamba
pyenv
asdf
Python Launcher for Windows
Python Launcher for Unix
venv
virtualenv
virtualenv wrapper
pip-tools
Pipenv
pdm
Poetry
Hatch
Pyflow
Flit
build
distutils
setuptools
Twine
tox
nox
pipx
direnv
python-dotenv
Python package installation
Virtual environment management
Python installation management
Non-Python package installation
Executable management
Resolve & lock dependencies
pyproject.toml Configuration
Build packages
Build PEP 517 packages with pyproject.toml
Publish Packages to PyPI
env variable auto activation
Test automation
Isolated envs for scripts
simple creation
conta-lock
deprecated
dotenv plugin

Tools

	pip	Conda	mamba	pyenv	asdf	Python Launcher for Windows	Python Launcher for Unix	venv	virtualenv	virtualenv wrapper	pip-tools	Pipenv	pdm	Poetry	Hatch	Pyflow	Flit	build	distutils	setuptools	Twine	tox	nox	pipx	direnv	python-dotenv	
Python package installation	✓	✓	✓								✓	✓	✓	✓	✓												Python package installation
Virtual environment management		✓	✓	✓			✓	✓ simple creation	✓	✓		✓		✓	✓							✓	✓		✓		Virtual environment management
Python installation management		✓	✓	✓	✓											✓											Python installation management
Non-Python package installation		✓	✓		✓																						Non-Python package installation
Executable management				✓	✓	✓	✓																				Executable management
Resolve & lock dependencies		✓ conda-lock	✓ conda-lock								requirements.in ▼ requirements.txt	Pigfile ▼ Pigfile.lock (JSON)	pyproject.toml ▼ pdm.lock (TOML)	pyproject.toml ▼ poetry.lock (TOML)	pyproject.toml or hatch.toml ▼ poetry.lock (TOML)	pyproject.toml ▼ pyflow.lock (??)										Resolve & lock dependencies
pyproject.toml Configuration											✓		✓	✓	✓	✓	✓										Project Metadata (PEP 621)
Build packages	✓												✓	✓	✓	✓	✓	✓	✓ deprecated	✓							Build sdist & wheel packages
Build PEP 517 packages with pyproject.toml	✓												✓	✓	✓	✓	✓	✓									Build packages with pyproject.toml (PEP 517)
Publish Packages to PyPI													✓	✓	✓	✓	✓				✓						Publish Packages to PyPI
env variable auto activation												✓	✓	✓ dotenv plugin	✓										✓	✓	env variable auto activation
Test automation															✓							✓	✓				Test automation
Isolated envs for scripts																✓								✓			Isolated envs for scripts
	pip	Conda	mamba	pyenv	asdf	Python Launcher for Windows	Python Launcher for Unix	venv	virtualenv	virtualenv wrapper	pip-tools	Pipenv	pdm	Poetry	Hatch	Pyflow	Flit	build	distutils	setuptools	Twine	tox	nox	pipx	direnv	python-dotenv	

Tools

Features

Tools

Features

Example:

Non python-package installation

Now you are thinking:

🤔 but we have container images (aka Docker images) ?

You have one point:

Container Images are pure love 🤩

You have one point:

Container Images are pure love 🤩

But Dockerfiles..uhg..

Dockerfiles are source of frustrations..

Dockerfiles

“Ah John (machine learning eng) worked on a MVP on that! ~1 year ago…”

“Ah John (machine learning eng) worked on a MVP on that! ~1 year ago…”

The project has a dockerfile
it should be runnable..right? Right?

John’s dockerfile 😡

```
FROM ubuntu:latest

RUN apt-get update && apt-get install fukuoka

CMD [“fukuoka”]
```

John’s dockerfile 😡

FROM ubuntu:latest 

Latest ? When? When “docker build” was run

RUN apt-get update && apt-get install fukuoka

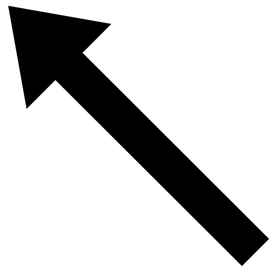
CMD [“fukuoka”]

John’s dockerfile 😡

```
FROM ubuntu:latest

RUN apt-get update && apt-get install fukuoka

CMD [“fukuoka”]
```



It won’t compile , it will fetch packages from a repository:

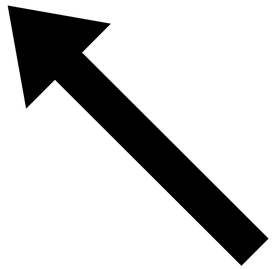
- package can dissapear
- No explicit version of the package
- Side effects : different on every run

John’s dockerfile 😡

FROM ubuntu:latest

RUN apt-get update && apt-get install fukuoka

CMD [“fukuoka”]



The binary that “Fukuoka” packages install can change, or can be in different path

John’s dockerfile 😡

FROM ubuntu:latest

RUN apt-get update && apt-get install fukuoka

CMD [“fukuoka”]

Dockerfiles are imperative, we are telling step by step what happens..

I came here for Nix..

Think of Nix as a potential solution to the mentioned issues

At least the ones I consider important:

At least the ones I consider important:

New team member joins my team, I want that person to be productive from day 1

At least the ones I consider important:

New team member joins my team, I want that person to be productive from day 1

No breaking environments due to different versions for different projects?
just run nix and get everything set up

At least the ones I consider important:

New team member joins my team, I want that person to be productive from day 1

No breaking environments due to different versions for different projects?
just run nix and get everything set up

Changed jobs? Got a new laptop?
just run nix and get everything set up

The “Nix” confusion

The “Nix” confusion

“Nix” might refer to three things:

The “Nix” confusion

“Nix” might refer to three things:

1. Nix OS
2. ⭐ Nix Package Manager
3. Nix Language

The “Nix” confusion

“Nix” might refer to three things:

1. Nix OS 
2. ⭐ Nix Package Manager
3. Nix Language

An OS distribution, think of it like an “ubuntu”

The “Nix” confusion

“Nix” might refer to three things:

1. Nix OS
2. ⭐ Nix Package Manager
3. Nix Language

The “Nix” confusion

“Nix” might refer to three things:

1. Nix OS
2. ⭐ Nix Package Manager
3. Nix Language

A “programming language” to tell nix how to build packages

Nix Package manager

Nix Package manager

- It is a binary, it can be installed in unix / OSX

- think of it as a “replacement for: brew or apt-get”

Nix Package manager

- It is a binary, it can be installed in unix / OSX

- think of it as a “replacement for: brew or apt-get”

We can use the package manager:

1. Install a package that is available in nix packages
2. Build our own packages

Nix Package manager demo

```
nix build nixpkgs#hello
nix run nixpkgs#cowsay -- hola
nix-shell --pure -p go_1_18
nix-shell --pure -p go_1_18 which
nix-shell   --pure -p python310 aws python310Packages.numpy

nix profile install nixpkgs#cowsay
```

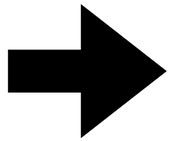
Nix Project demo

```
nix run github:dav009/nixsample
nix develop github:dav009/nixsample
nix build github:dav009/nixsample
nix build github:dav009/nixsample#docker
```

But how does this work?

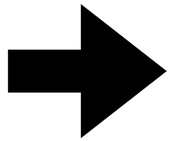
Nix lang
build instructions

Nix lang
build instructions



<derivation>

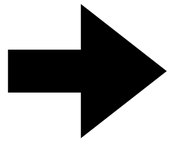
Nix lang
build instructions



<derivation>

Recipe :
- where are the inputs
- what commands to call
- calculates hash (store path)

Nix lang
build instructions



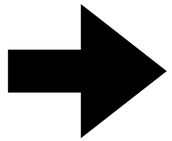
<derivation>

Recipe :
- where are the inputs
- what commands to call
- calculates hash (store path)

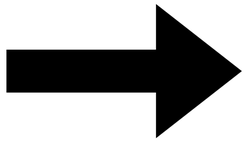
cmd: nix show-derivation

Language agnostic build instructions

Nix lang
build instructions



<derivation>



/nix/store/<hash>

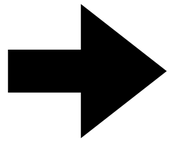
Recipe :
- where are the inputs
- what commands to call
- calculates hash (store path)

XXXXXXXX LANG XXX

cmd: nix show-derivation

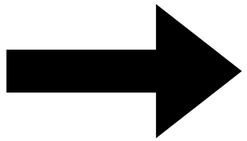
Language agnostic build instructions

Nix lang
build instructions



<derivation>

Recipe :
- where are the inputs
- what commands to call
- calculates hash (store path)



/nix/store/<hash>

cmd: nix show-derivation

Language agnostic build instructions

cmd: nix build

Output of executing the
steps in the derivation

The magic is:

We can know ahead of time where the path of something is..

Idempotence (no side effects)

Same inputs = Same outputs

Be careful with impure derivations!

Nix repl

..Nix basics..

Container images with nix

- Smaller container images:
  - We only add exactly what we need

- We don’t tell how to build a container image
  - We describe what packages the container image should have
  - Nix figures out what needs to be in the container

- We can rebuild the container image: reproducibility

Nix Flakes

Predefined outputs/structure for a nix file

…Just a standard..

Nix Flakes

```
nix flake show
nix develop
nix run .
```

Nix Flakes: Structure

Inputs:
- packages

Outputs:
- defaultPackage : `nix build`
- defaultApp: `nix run`
- devShell : `nix develop`
- you can specify any arbitrary outputs if you want

Nix Flakes: Structure

scratchflake.nix (modified)

```
{
  description = "Dummy hello world";

  # packages ..
  inputs.nixpkgs.url = "nixpkgs/nixos-unstable";

  # other flakes / packages
  inputs.flake-compat = {
    url = "github:edolstra/flake-compat";
    flake = false;
  };

  outputs = { self, nixpkgs, ... }:
    let

    in {

      packages = {

        # HERE SOME CODE
      };

      apps = {
        # HERE SOME CODE
      };

      defaultPackage = # here some code

      defaultApp = # here some code

      devShell = # here some code

    };
}
```

Nix Flakes: Structure

scratchflake.nix (modified)

```
{
  description = "Dummy hello world";

  # packages ..
  inputs.nixpkgs.url = "nixpkgs/nixos-unstable";

  # other flakes / packages
  inputs.flake-compat = {
    url = "github:edolstra/flake-compat";
    flake = false;
  };

  outputs = { self, nixpkgs, ... }:
    let

    in {

      packages = {

        # HERE SOME CODE
      };

      apps = {
        # HERE SOME CODE
      };

      defaultPackage = # here some code

      defaultApp = # here some code

      devShell = # here some code

    };
}
```

Inputs:
- repositories with packages
- “implicit” the code in your git repository

Nix Flakes: Structure

scratchflake.nix (modified)

```
{
  description = "Dummy hello world";

  # packages ..
  inputs.nixpkgs.url = "nixpkgs/nixos-unstable";

  # other flakes / packages
  inputs.flake-compat = {
    url = "github:edolstra/flake-compat";
    flake = false;
  };

  outputs = { self, nixpkgs, ... }:
    let

    in {

      packages = {

        # HERE SOME CODE
      };

      apps = {
        # HERE SOME CODE
      };

      defaultPackage = # here some code

      defaultApp = # here some code

      devShell = # here some code

    };
}
```

Outputs

Nix Flakes: Structure

```
  outputs = { self, nixpkgs, ... }:
    let

    in {

      packages = {

        # HERE SOME CODE
      };

      apps = {
        # HERE SOME CODE
      };

      defaultPackage = # here some code

      defaultApp = # here some code

      devShell = # here some code

    };
}
```

Nix Flakes: Structure

```
  outputs = { self, nixpkgs, ... }:
    let

    in {

      packages = {

        # HERE SOME CODE
      };

      apps = {
        # HERE SOME CODE
      };

      defaultPackage = # here some code

      defaultApp = # here some code

      devShell = # here some code

    };
}
```

Generate artifact for our app e.g:
- build a binary given golang code
- build a docker file given our binary

Nix Flakes: Structure

```
outputs = { self, nixpkgs, ... }:
  let

  in {

    packages = {

      # HERE SOME CODE
    };

    apps = {
      # HERE SOME CODE
    };

    defaultPackage = # here some code

    defaultApp = # here some code

    devShell = # here some code

  };
}
```

Generate artifact for our app e.g:
- build a binary given golang code
- build a docker file given our binary

Reference the build we have in packages and expose it as a runnable

Nix Flakes: Structure

```
outputs = { self, nixpkgs, ... }:
  let

  in {

    packages = {

      # HERE SOME CODE
    };

    apps = {
      # HERE SOME CODE
    };

    defaultPackage = # here some code

    defaultApp = # here some code

    devShell = # here some code

  };
}
```

Generate artifact for our app e.g:
- build a binary given golang code
- build a docker file given our binary

Reference the build we have in packages and expose it as a runnable

Define a dev environment with all the deps we might need to :
test, run an edit , lap servers, enviers..

So far you should be wondering:

you never pinned any versions..liar..

flake.lock

“The above example will continue to work a decade from now because all transitive dependencies are fully pinned by the NixOS specification.”

Flake templates

- predefined templates for projects + Flakes..
- saving your from writing boilerplate..

Quick demo!

Recap: Flakes

- we only define how to build our app once
- we then use that same definition to:
  - Create a local dev environment
  - Build container image

Recap: Flakes

- we only define how to build our app once
- we then use that same definition to:
  - Create a local dev environment
  - Build container image

- We use nix in CI (devshell)
- We use nix in local development (devshell)
- Any changes to anything (code, dependencies) generates a new hash..

Recap: Flakes

- we only define how to build our app once
- we then use that same definition to:
  - Create a local dev environment
  - Build container image

- We use nix in CI (devshell)
- We use nix in local development (devshell)
- Any changes to anything (code, dependencies) generates a new hash..

CI = local dev = built app = app in container image

Misc 🤯

🤯 NixOS

- want to test your new configuration changes before applying them?

🤯 NixOS

- want to test your new configuration changes before applying them?

- you can spawn a VM with your new config before you roll out changes

🤯 NixOS

- want to test your new configuration changes before applying them?

- you can spawn a VM with your new config before you roll out changes

nixos-rebuild build-vm

🤯 NixOS

when you make a change to your system you can roll it back

when you restart
you can pick what “version” of your system you want to run

🤯 nix-darwin

manage your setup as NixOS
https://github.com/LnL7/nix-darwin

🤯 nix-darwin

manage your setup as NixOS
https://github.com/LnL7/nix-darwin

you can describe your OSX setup:

apps, packages, the sensibility of the trackpad, networking, screen resolutions, keybindin, services ..

🤯 nix-darwin

manage your setup as NixOS
https://github.com/LnL7/nix-darwin

you can describe your OSX setup:

apps, packages, the sensibility of the trackpad, networking, screen resolutions, keybindin, services ..

Got a new laptop?
just run: darwin-rebuild switch

🤯 NixOS & NixPkg’s Lightweight integration tests

🤯 NixOS & NixPkg's Lightweight integration tests

NixOS integration test system

- Spawn vms
- Build your package
- Build is successful ? Then test passes
- Super lightweight!

https://www.haskellforall.com/2020/11/how-to-use-nixos-for-lightweight.html

🤯 NixOS & NixPkg's Lightweight integration tests

NixOS integration test system

- Spawn vms
- Build your package
- Build is successful ? Then test passes
- Super lightweight!

https://www.haskellforall.com/2020/11/how-to-use-nixos-for-lightweight.html

Compact (everything of your test is in one place!)

🤯 Cross compile

you can ask nix to build your artefact remotely

🤯 Cross compile

you can ask nix to build your artefact remotely

for any architecture

🤯 Copy cmd

🤯 Copy cmd

```
nix copy --to ssh-ng://my-remote-host  "nixpkgs#apachakafka_3_3"
```

🤯 Copy cmd

```
nix copy --to ssh-ng://my-remote-host  "nixpkgs#apachakafka_3_3"
```

Copies everything required to run Apache Kafka v3.3 to a remote host with Nix installed

🤯 Copy cmd

```
nix copy --to ssh-ng://my-remote-host  "nixpkgs#apachakafka_3_3"
```

Copies everything required to run Apache Kafka v3.3 to a remote host with Nix installed

because Nix packages know their dependencies

The future 

_ hype train _

Nix has taken off in popularity

improving the developer experience by creating abstractions on top

tools that abstract the complexities of Nix are being built!

https://devenv.sh/

- Abstract some complexities
  - let you serve services aka : docker-compose
    - I want to run a Kafka cluster when my devshell starts

https://github.com/flox/flox

- Simple Virtualenv using nix under the hood

flox install -e demo hello

Final thoughts 🤔

Is this tech ready for the spotlight?

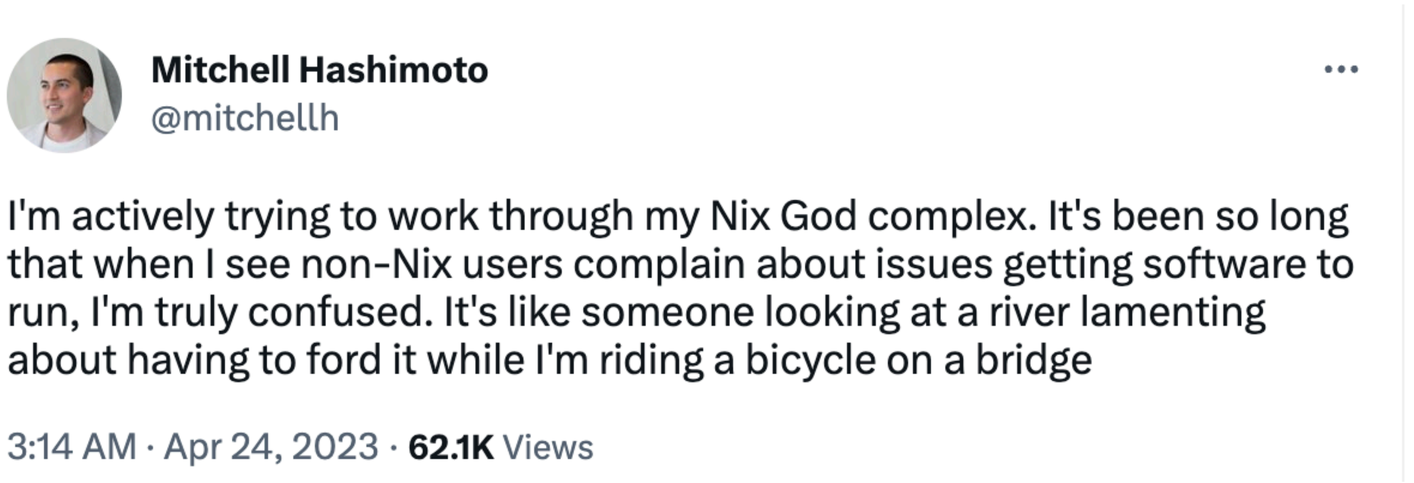
IMO yes!

Is this something that can make my team more productive?

yes! But.. it is worth waiting for the tooling that is being built on top

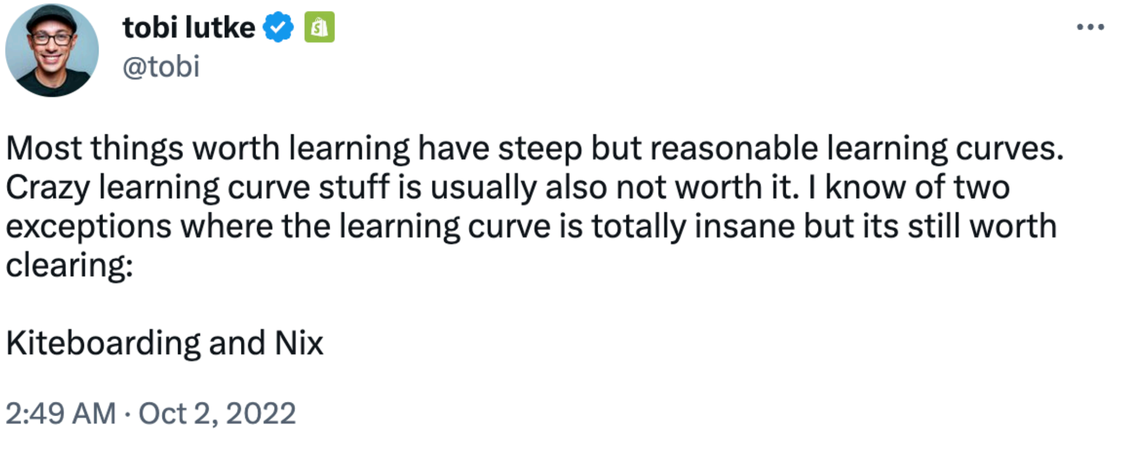
Should you give it a try?

You should!



Should you give it a try?

You should!



Recommended resources:

- **https://xeiaso.net/**
- **“Nix under the hood” https://www.youtube.com/watch?v=GMQPzv3Sx58&t=1801s**
- **“Nix in Production” - https://leanpub.com/nixos-in-production**
- **“Nix pills” https://nixos.org/guides/nix-pills/**