

# What the Nix?!

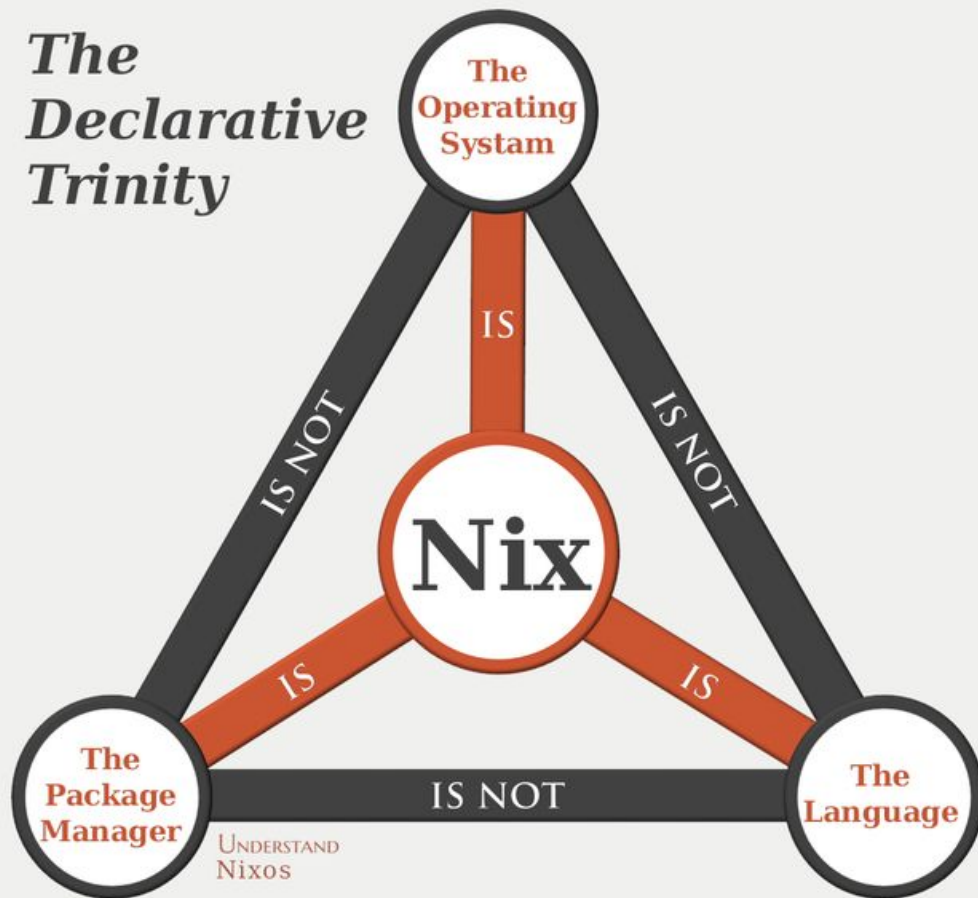
Bryan Honof—Software Engineer @ Tweag

# What is Nix?!





# The Declarative Trinity



# The Package Manager



```
$ terraform --version
```

```
Command 'terraform' not found, but can be installed with:  
sudo snap install terraform
```

```
$ nix-shell -p terraform
```

```
...
```

```
[nix-shell:~]$ terraform --version
```

```
Terraform v1.2.2  
on linux_amd64
```

# The Language



Or “JSON with functions”

```
{  
  "names": [ "Bryan", "Tweag", "We" ],  
  "whoLovesNix": [ "Bryan loves Nix!", "Tweag loves Nix!", "We loves Nix!" ]  
}
```

```
{  
  names = [ "Bryan" "Tweag" "We" ];  
  loveNix = a: "${a} loves Nix!";  
  whoLovesNix = builtins.map loveNix names;  
}
```

# The Operating System



```
{ config, pkgs, ... }:  
  
{  
  services.httpd.enable = true;  
  services.httpd.adminAddr = "bryan.honof@tweag.io";  
  services.httpd.virtualHosts.localhost.documentRoot = "/memes";  
}
```

**Why do we Nix?!**





**Why ~~do we~~ | Nix?!**













`nixos-rebuild --rollback switch`





**How do we Nix?!**





# curl -L https://nixos.org/nix/install | sh -s -- --daemon

NixOS 22.11 released. [Announcement](#)

## Reproducible builds and deployments.

Nix is a tool that takes a unique approach to package management and system configuration. Learn how to make reproducible, declarative and reliable systems.

[Download](#) [Get started](#)

```

$ # Hi!
$ # Wondering how to start using Nix?
$ # Here are a few examples:
$ node -e "console.log(1+1)"
node: command not found
$ # Interesting, no node on this machine
$ # No problem with Nix!
$ nix-shell -p node; "console.log(1+1)"
(nix-shell) $ node -e "console.log(1+1)"
2
(nix-shell) $ # And now the environment has node.
(nix-shell) $ # Without cluttering the user environme
$ # Typing "nix-shell -p ..." each time can be tedious
$ # We can write every change in shell.nix
$ cat -n shell.nix
1 { pkgs ? import <nixpkgs> {} # here we import
2 };
3 pkgs.mkShell { # mkShell is a h
4   name="dev-environment"; # that requires
5   buildInputs = [ # and a list of

```

**Reproducible**

Nix builds packages in isolation from each other. This ensures that they are reproducible and don't have undeclared dependencies, so **if a package works on one machine, it will also work on another.**

**Declarative**

Nix makes it **trivial to share development and build environments** for your projects, regardless of what programming languages and tools you're using.

**Reliable**

Nix ensures that installing or upgrading one package **cannot break other packages**. It allows you to **roll back to previous versions**, and ensures that no package is in an inconsistent state during an upgrade.

Yes, I know... Don't pipe directly into bash.

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

```
pkgs.mkShell {  
  packages = [  
    pkgs.terraform  
    pkgs.skopeo  
    pkgs.kubectl  
  ];  
}
```

### \$ nix-shell

these paths will be fetched (0.07 MiB download, 0.20 MiB unpacked):

/nix/store/...-skopeo-1.8.0

/nix/store/...-terraform-1.2.2

...

copying path '/nix/store/...-skopeo-1.8.0' from 'https://cache.nixos.org'...

...

[nix-shell:~]\$



<https://nixos.org/learn.html>  
<https://nix.dev/>

# What?! You want more?



## A Gentle Introduction to Nix



2023-02-06, 14:00–14:50, B.3.037

Do you have a colleague or friend that just can't stop talking about Nix? How it's reproducible, the future of software. Why it's so much better than the other tools out there. He just won't stop talking about it. And now, you've given in. You decided to give it a try, but where to start?

If the above is you, this talk is for you! Bryan will give his best attempt at giving a gentle introduction to Nix. After this talk, you hopefully understand Nix's values, and where to find more help if you happen to need it.

## Using Nix to generate Docker images



2023-02-07, 12:30–12:35, D Aud

In this ignite talk, I will expose quickly the benefits of not writing Dockerfiles but rather using Nix expressions to generate Docker images.