

Learn Nix the Fun Way

Why I love Nix, and you should too

Farid Zakaria · farid.m.zakaria@gmail.com

fzakaria.github.io/learn-nix-the-fun-way

github.com/fzakaria/learn-nix-the-fun-way · [02108e5](#)
[/nix/store/3x9v5nfy2x31cx9c5n5hnjlds9r48si7-source](#)

Hello!

Welcome to my talk on learning Nix the fun way.

This slide deck can be found online at the following link. I welcome improvements, edits and corrections.

Hello 🖐️

My name is Farid Zakaria, and I'm a software engineer at [Meta](#), with a PhD from UC Santa Cruz.

- Long-time Nix user (circa 2018)
- Introduced Nix at many companies, including Confluent & Google, and an active member of their internal communities
- Written a lot of Nix software and projects, such as [nixmultiverse.com](#), [omnibin.dev](#), [trynix.dev](#) and [omniflake.com](#), to name a few
- Daily drive NixOS on my wonderful [frame.work](#) laptop



farid.m.zakaria@gmail.com
linkedin.com/in/fmzakari
x.com/fmzakari

Hello 🖐️

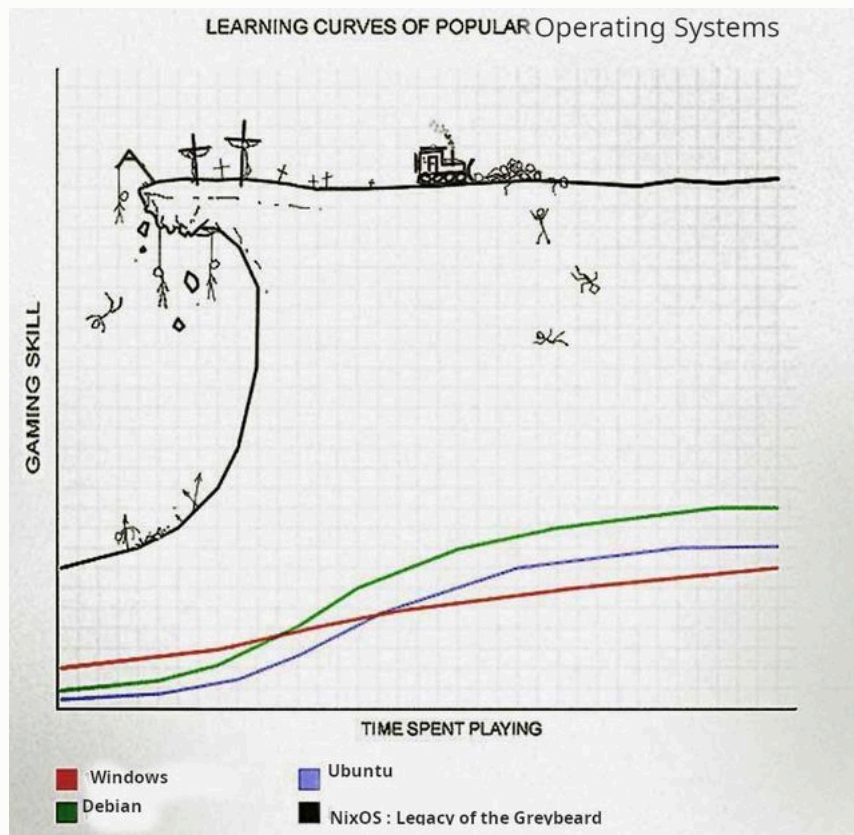
Great, so before we begin, let me introduce myself.

My name is Farid Zakaria, and I'm a software engineer at Meta.

I've been using Nix in various capacities for quite a while now. I've presented at a few previous incarnations of Nix conventions, and I've had the pleasure of bringing Nix into many of the companies I've worked at, including Confluent and Google.

I'm very bullish on Nix and it has been a source of mostly joy, and it was additionally the inspiration for a lot of my PhD research.

Disclaimer



Disclaimer

I have given Nix talks a few times, individually and to groups -- including to engineering groups. I've noticed that explaining the rationale for Nix, and even Nix itself, is challenging. It's an active area of discussion within the community. Even reflecting on my own journey of learning and loving Nix, it took a few iterations, so I'm pretty empathetic to those that feel they might not understand what the big deal is or find it too complex.

So the disclaimer is that some of what I demonstrate may be complicated or tough to understand. What I do hope to convey however is that there is a reason for this complexity. Software is complex, and aiming for the levels of guarantees that Nix provides puts you face to face with that complexity.

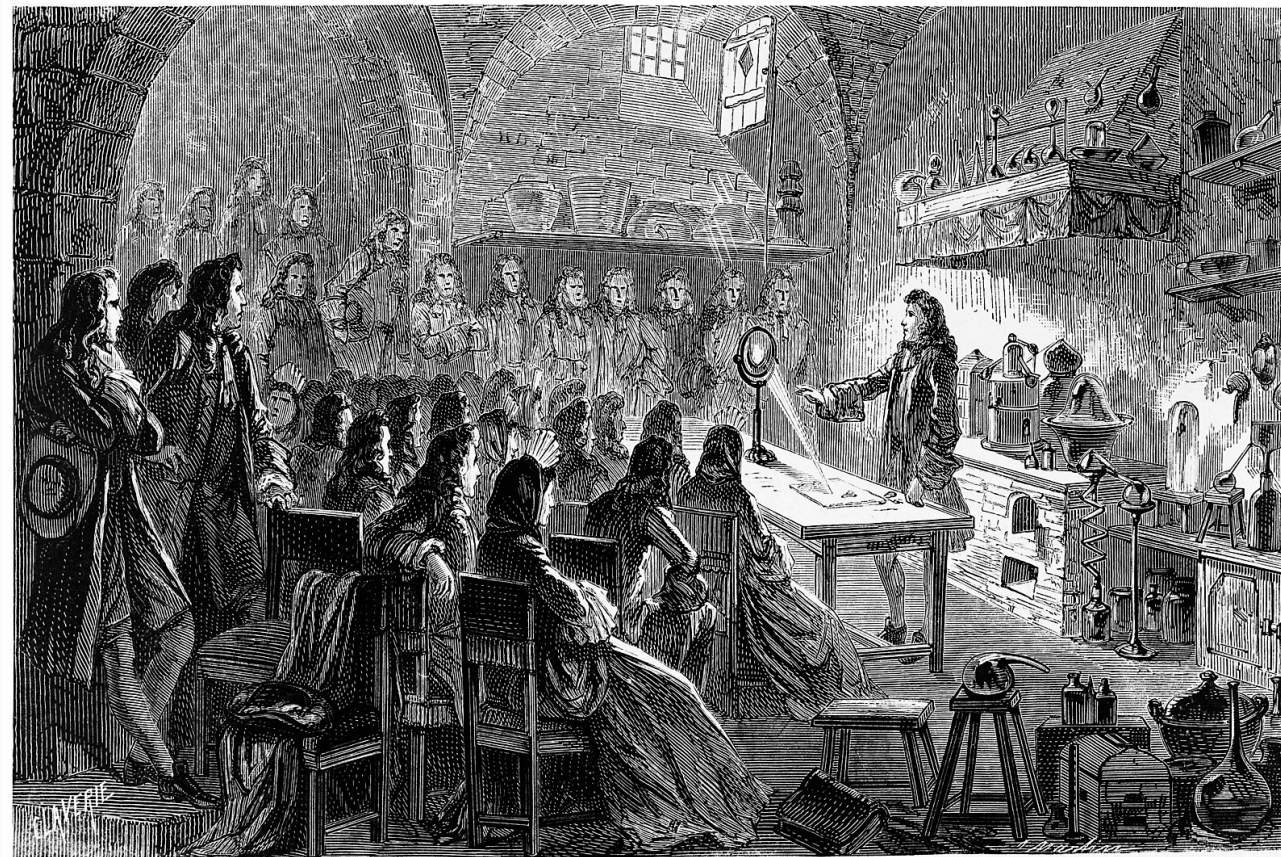
Please feel free to ask me questions during the talk. I have kept the slides light on written text, but there are snapshots of code. I have all of the code handy, and if it's helpful I'm willing to DO IT LIVE and go beyond what is shown.

Why I love Nix, and you should too! 💖

Disclaimer

Okay, so this is the alternate working title for this talk.

I want to share why I love Nix. Nix has brought a lot of joy and simplification to my software life, and oftentimes learning it can feel challenging without much insight into what is the purpose.



Disclaimer

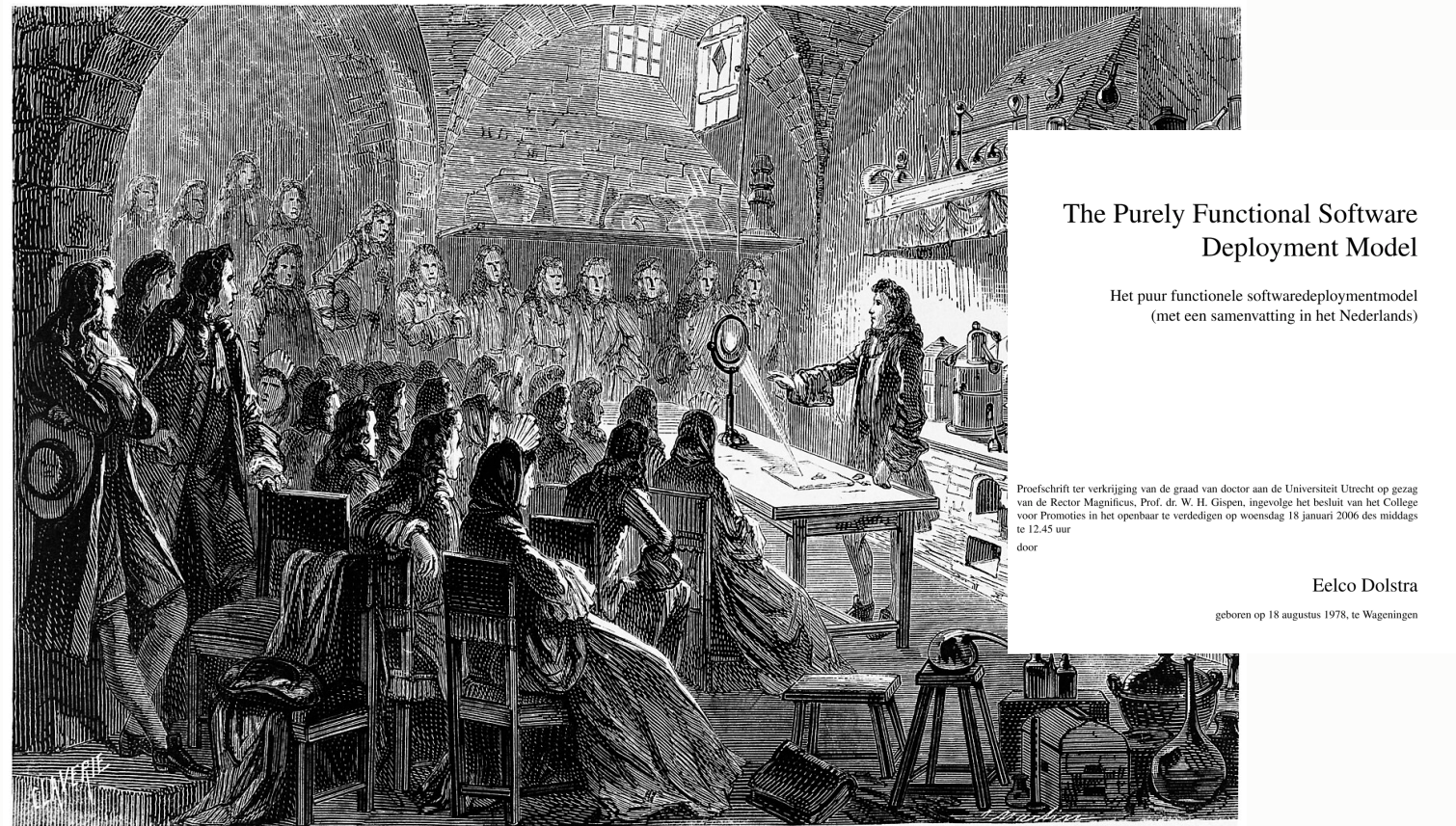
I've given a lot of talks about Nix. I've given talks to meetup groups, friends and plenty internally at the companies I've worked at.

This is how I've always pictured myself giving a talk about Nix.

It's always a noble pursuit where I impart the beautiful knowledge of that which is Nix.

**** click ****

I tend to always begin these talks at the beginning, by discussing Eelco's PhD, The Purely Functional Software Deployment Model, where he first shared Nix with the world in 2006.



Disclaimer

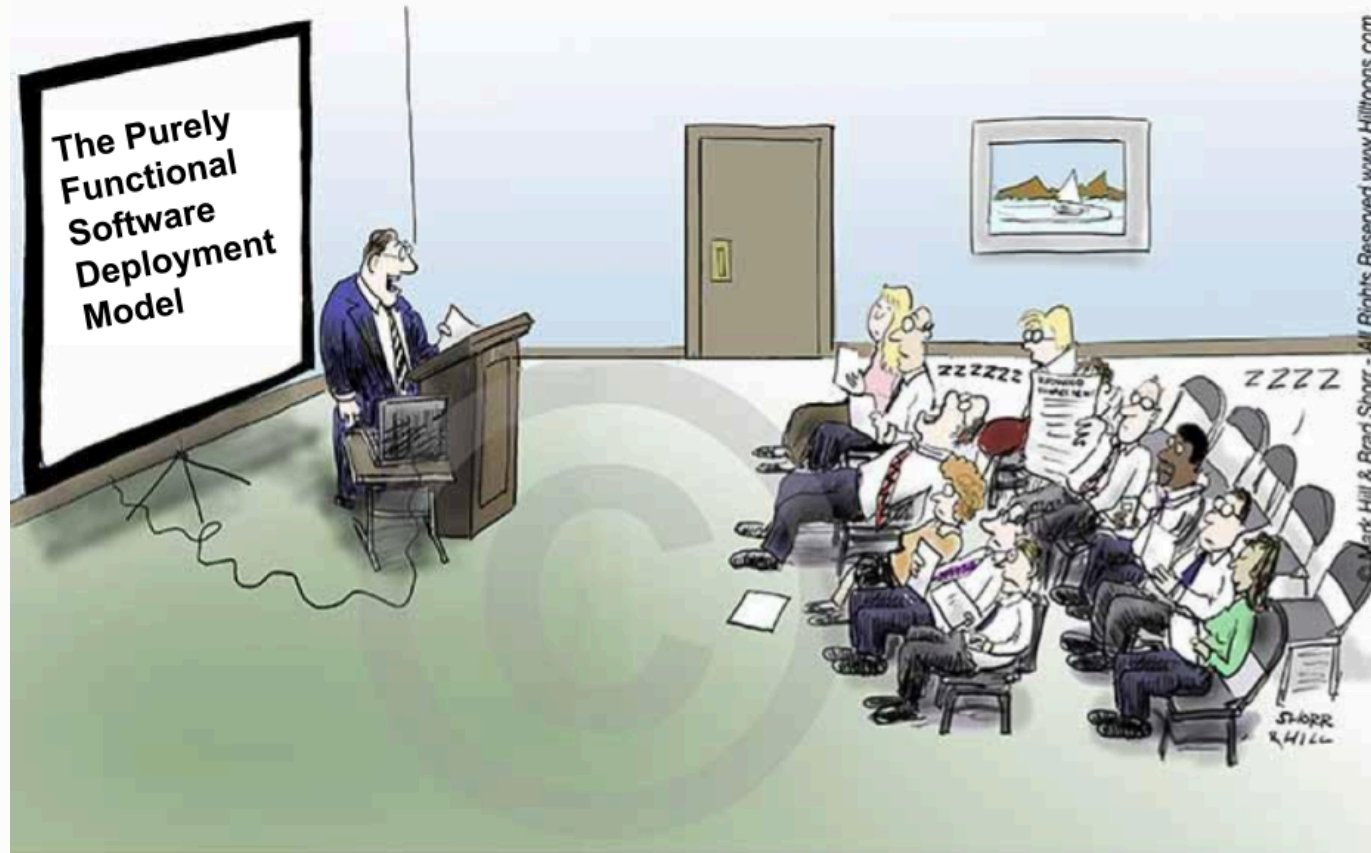
I've given a lot of talks about Nix. I've given talks to meetup groups, friends and plenty internally at the companies I've worked at.

This is how I've always pictured myself giving a talk about Nix.

It's always a noble pursuit where I impart the beautiful knowledge of that which is Nix.

**** click ****

I tend to always begin these talks at the beginning, by discussing Eelco's PhD, The Purely Functional Software Deployment Model, where he first shared Nix with the world in 2006.



Disclaimer

In retrospect, this slide better depicts how the talks have gone. People in attendance are at best totally bored by the nuance and complexity of what I'm discussing.

**** click ****

At worst, I've immediately turned off potential new Nix-enthusiasts by jumping straight into academic and dense vocabulary about the ideas and technology that powers Nix.



Disclaimer

In retrospect, this slide better depicts how the talks have gone. People in attendance are at best totally bored by the nuance and complexity of what I'm discussing.

**** click ****

At worst, I've immediately turned off potential new Nix-enthusiasts by jumping straight into academic and dense vocabulary about the ideas and technology that powers Nix.

Let's do it different this time.

Disclaimer

The motivation of this talk is to do it different this time.

**** click ****

I want to impart the benefits of using Nix by what I believe to be learning Nix the fun way.

Let's do it different this time.

Let's learn Nix the fun way.

Disclaimer

The motivation of this talk is to do it different this time.

**** click ****

I want to impart the benefits of using Nix by what I believe to be learning Nix the fun way.

What is my IP?

```
#!/usr/bin/env bash
> curl -s http://httpbin.org/get | \
    jq --raw-output .origin
```

What is my IP?

Let's begin with something seemingly innocuous. Here is a small script to determine the IP of your current machine.

This should look pretty standard, I'm hoping, to most people, and probably reminiscent of a lot of scripts we might have written ourselves that stitch a few programs together.

**** click ****

And it does what it says.

What is my IP?

```
#!/usr/bin/env bash  
> curl -s http://httpbin.org/get | \  
jq --raw-output .origin
```

```
73.231.52.39
```

What is my IP?

Let's begin with something seemingly innocuous. Here is a small script to determine the IP of your current machine.

This should look pretty standard, I'm hoping, to most people, and probably reminiscent of a lot of scripts we might have written ourselves that stitch a few programs together.

**** click ****

And it does what it says.

The same script, in Nix

```
{
  system ? builtins.currentSystem,
  pkgs ?
    let
      # NixOS 24.11
      rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
      url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
    in
      import (fetchTarball url) {inherit system;},
}:
derivation {
  name = "what-is-my-ip";
  builder = "/bin/sh";
  args = [
    "-c"
    ''
      ${pkgs.coreutils}/bin/mkdir -p $out/bin
      echo '#!/bin/sh' > $out/bin/what-is-my-ip
      echo '${pkgs.curl}/bin/curl -s http://httpbin.org/get | \
        ${pkgs.jq}/bin/jq --raw-output .origin' \
        >> $out/bin/what-is-my-ip
      ${pkgs.coreutils}/bin/chmod +x $out/bin/what-is-my-ip
    ''
  ];
  system = builtins.currentSystem;
  outputs = ["out"];
}
```

The same script, in Nix

Here is what rewriting that same script in the most raw form of Nix might look like.

I know I just showed you a new language, but for a tiny second just suspend questions you may have. The only take-away from the language would be that it looks semi-similar to other languages you may have used; it has a somewhat JSON-like feel to it.

**** click ****

Here is the important part of this Nix recipe, and it looks similar-ish to what we had before.

The same script, in Nix

```
{
  system ? builtins.currentSystem,
  pkgs ?
    let
      # NixOS 24.11
      rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
      url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
    in
      import (fetchTarball url) {inherit system;},
}:
derivation {
  name = "what-is-my-ip";
  builder = "/bin/sh";
  args = [
    "-c"
    ,
    "${pkgs.coreutils}/bin/mkdir -p $out/bin
    echo '#!/bin/sh' > $out/bin/what-is-my-ip
    echo '${pkgs.curl}/bin/curl -s http://httpbin.org/get | \
    ${pkgs.jq}/bin/jq --raw-output .origin' \
    >> $out/bin/what-is-my-ip
    ${pkgs.coreutils}/bin/chmod +x $out/bin/what-is-my-ip
    ,
  ];
  system = builtins.currentSystem;
  outputs = ["out"];
}
```

The same script, in Nix

Here is what rewriting that same script in the most raw form of Nix might look like.

I know I just showed you a new language, but for a tiny second just suspend questions you may have. The only take-away from the language would be that it looks semi-similar to other languages you may have used; it has a somewhat JSON-like feel to it.

**** click ****

Here is the important part of this Nix recipe, and it looks similar-ish to what we had before.

The same script, in idiomatic Nix

```
{
  system ? builtins.currentSystem,
  pkgs ?
    let
      rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
      url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
    in
      import (fetchTarball url) {inherit system;},
}:
pkgs.writeShellScriptBin "what-is-my-ip" ''
  ${pkgs.curl}/bin/curl -s http://httpbin.org/get | \
  ${pkgs.jq}/bin/jq --raw-output .origin
''
```

The same script, in idiomatic Nix

But of course, this is not what you'd actually write in Nix. The following is a little more idiomatic and a lot more terse.

Here it much more resembles the original script.

**** click ****

For those that might want to follow along now or later, it's important to keep note of this commit value, and that I'm building this on an amd64 Linux machine.

The same script, in idiomatic Nix

```
{
  system ? builtins.currentSystem,
  pkgs ?
    let
      rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
      url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
    in
      import (fetchTarball url) {inherit system;},
}:
pkgs.writeShellScriptBin "what-is-my-ip" ''
  ${pkgs.curl}/bin/curl -s http://httpbin.org/get | \
    ${pkgs.jq}/bin/jq --raw-output .origin
''
```

The same script, in idiomatic Nix

But of course, this is not what you'd actually write in Nix. The following is a little more idiomatic and a lot more terse.

Here it much more resembles the original script.

**** click ****

For those that might want to follow along now or later, it's important to keep note of this commit value, and that I'm building this on an amd64 Linux machine.

The same script, in idiomatic Nix

```
{
  system ? builtins.currentSystem,
  pkgs ?
    let
      rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
      url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
    in
      import (fetchTarball url) {inherit system;},
}:
pkgs.writeShellScriptBin "what-is-my-ip" ''
  ${pkgs.curl}/bin/curl -s http://httpbin.org/get | \
  ${pkgs.jq}/bin/jq --raw-output .origin
''
```

The same script, in idiomatic Nix

But of course, this is not what you'd actually write in Nix. The following is a little more idiomatic and a lot more terse.

Here it much more resembles the original script.

**** click ****

For those that might want to follow along now or later, it's important to keep note of this commit value, and that I'm building this on an amd64 Linux machine.

Let's build it

```
> nix build -f what-is-my-ip.nix --print-out-paths
```

Let's build it

Let's build it!

**** click ****

We get some output path, okay cool.

**** click ****

That output path has the script we created.

**** click ****

We can execute that script and it works as expected.

Let's build it

```
> nix build -f what-is-my-ip.nix --print-out-paths

/nix/store/y2g0ijqqiyi9vvr9xgmvvgb1xqflqzav-what-is-my-ip
```

Let's build it

Let's build it!

**** click ****

We get some output path, okay cool.

**** click ****

That output path has the script we created.

**** click ****

We can execute that script and it works as expected.

Let's build it

```
> nix build -f what-is-my-ip.nix --print-out-paths

/nix/store/y2g0ijqqiyi9vvr9xgmvvgbllxqflqzav-what-is-my-ip

> /nix/store/y2g0ijqqiyi9vvr9xgmvvgbllxqflqzav-what-is-my-ip/bin/what-is-my-ip
```

Let's build it

Let's build it!

**** click ****

We get some output path, okay cool.

**** click ****

That output path has the script we created.

**** click ****

We can execute that script and it works as expected.

Let's build it

```
> nix build -f what-is-my-ip.nix --print-out-paths

/nix/store/y2g0ijqqiyi9vxr9xgmvvgbllxqflqzav-what-is-my-ip

> /nix/store/y2g0ijqqiyi9vxr9xgmvvgbllxqflqzav-what-is-my-ip/bin/what-is-my-ip

73.231.52.39
```

Let's build it

Let's build it!

**** click ****

We get some output path, okay cool.

**** click ****

That output path has the script we created.

**** click ****

We can execute that script and it works as expected.

What did we build?

```
> cat /nix/store/y2g0ijqqiyi9vvr9xgmvvglxqflqzav-what-is-my-ip/bin/what-is-my-ip
```

What did we build?

What is the script we produced though? Why did we need Nix?

**** click ****

The output of our script looks very similar now to the original variant we had before, except everything seems to be prefixed with some long string.

What did we build?

```
> cat /nix/store/y2g0ijqqiyi9vxr9xgmvvglxqflqzav-what-is-my-ip/bin/what-is-my-ip

#!/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash
/nix/store/7p8814g62bybwa8blmaqr9piv3vjwan5-curl-8.11.1-bin/bin/curl -s http://httpbin.org/get | \
/nix/store/jhw7aklgrc23125ysglmc72y8zq0lndw-jq-1.7.1-bin/bin/jq --raw-output .origin
```

What did we build?

What is the script we produced though? Why did we need Nix?

**** click ****

The output of our script looks very similar now to the original variant we had before, except everything seems to be prefixed with some long string.

What is `/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip?`

What did we build?

Although I promised not to go too academic into Nix, this part is important.

**** click ****

What is that long prefix string ahead of our built script, and similarly found within the script before each of the tools?

**** click ****

There's a handy Nix command to show us the meaning of that prefix.

What is `/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip?`

Why do we care about `y2g0ijqqiyi9vxr9xgmvvgblxqflqzav?`

What did we build?

Although I promised not to go too academic into Nix, this part is important.

**** click ****

What is that long prefix string ahead of our built script, and similarly found within the script before each of the tools?

**** click ****

There's a handy Nix command to show us the meaning of that prefix.

What is `/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip`?

Why do we care about `y2g0ijqqiyi9vxr9xgmvvgblxqflqzav`?

```
> nix derivation show /nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip
```

What did we build?

Although I promised not to go too academic into Nix, this part is important.

click

What is that long prefix string ahead of our built script, and similarly found within the script before each of the tools?

click

There's a handy Nix command to show us the meaning of that prefix.

Hashed into y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav

```
{
  "/nix/store/rw89fpljini3yx09v83ds4hyl4945lmk-what-is-my-ip.drv": {
    "args": [
      "-e",
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
    "env": {
      "buildCommand": "<build command>",
      "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
      "name": "what-is-my-ip",
      "out": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav-what-is-my-ip",
      "stdenv": "/nix/store/cf464y41p2x3lh1qvbg6678lc3f8zbd6-stdenv-linux",
      "system": "x86_64-linux",
      "text": "<contents of script removed for brevity>"
    },
    "inputDrvs": {
      "/nix/store/7k0msqyp2dm021sdj0qjgpkzff8xhqzr-bash-5.2p37.drv": {},
      "/nix/store/a9dyisrdm25k6cprjvypprxs3n8b283-jq-1.7.1.drv": {},
      "/nix/store/crgvallixc0kpg62whvkiv6x38p15nqr-curl-8.11.1.drv": {},
      "/nix/store/ycj0m56p8b0rv9v78mvgfa6xhm31rw3-stdenv-linux.drv": {}
    },
    "inputSrcs": [
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "name": "what-is-my-ip",
    "outputs": {
      "out": {
        "path": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav-what-is-my-ip"
      }
    },
    "system": "x86_64-linux"
  }
}
```

Hashed into y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav

The whole purpose of Nix is to kind of build this file. This is the recipe describing what it takes to build our IP script.

**** click ****

It has a build command.

**** click ****

The text of our script, which may also be the source code if it's a little more involved.

**** click ****

The tools we depend on to make the script function, like cURL and jq.

**** click ****

The system we're running on, since Nix also supports different architectures and operating systems.

**** click ****

All of this information is hashed to calculate this prefix.

There's a really mind-bending property though.

Hashed into y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav

```
{
  "/nix/store/rw89fpljini3yx09v83ds4hyl4945lmk-what-is-my-ip.drv": {
    "args": [
      "-e",
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
    "env": {
      "buildCommand": "<build command>",
      "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
      "name": "what-is-my-ip",
      "out": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav-what-is-my-ip",
      "stdenv": "/nix/store/cf464y41p2x3lh1qvbg6678lc3f8zbd6-stdenv-linux",
      "system": "x86_64-linux",
      "text": "<contents of script removed for brevity>"
    },
    "inputDrvs": {
      "/nix/store/7k0msqyp2dm021sdj0qjgpkzff8xhqzr-bash-5.2p37.drv": {},
      "/nix/store/a9dyisrdm25k6cprjvypppxrs3n8b283-jq-1.7.1.drv": {},
      "/nix/store/crgvallixc0kpg62whvkiv6x38p15nqr-curl-8.11.1.drv": {},
      "/nix/store/ycj0m56p8b0rv9v78moggfa6xhm31rww3-stdenv-linux.drv": {}
    },
    "inputSrcs": [
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "name": "what-is-my-ip",
    "outputs": {
      "out": {
        "path": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav-what-is-my-ip"
      }
    },
    "system": "x86_64-linux"
  }
}
```

Hashed into y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav

The whole purpose of Nix is to kind of build this file. This is the recipe describing what it takes to build our IP script.

**** click ****

It has a build command.

**** click ****

The text of our script, which may also be the source code if it's a little more involved.

**** click ****

The tools we depend on to make the script function, like cURL and jq.

**** click ****

The system we're running on, since Nix also supports different architectures and operating systems.

**** click ****

All of this information is hashed to calculate this prefix.

There's a really mind-bending property though.

Hashed into y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

```
{
  "/nix/store/rw89fpljini3yx09v83ds4hyl4945lmk-what-is-my-ip.drv": {
    "args": [
      "-e",
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
    "env": {
      "buildCommand": "<build command>",
      "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
      "name": "what-is-my-ip",
      "out": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip",
      "stdenv": "/nix/store/cf464y41p2x3lh1qvbg6678lc3f8zbd6-stdenv-linux",
      "system": "x86_64-linux",
      "text": "<contents of script removed for brevity>"
    },
    "inputDrvs": {
      "/nix/store/7k0msqyp2dm021sdj0qjgpkzff8xhqzr-bash-5.2p37.drv": {},
      "/nix/store/a9dyisrdm25k6cprjvypprxs3n8b283-jq-1.7.1.drv": {},
      "/nix/store/crgvallixc0kpg62whvkiv6x38p15nqr-curl-8.11.1.drv": {},
      "/nix/store/ycj0m56p8b0rv9v78mggfa6xhm31rww3-stdenv-linux.drv": {}
    },
    "inputSrcs": [
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "name": "what-is-my-ip",
    "outputs": {
      "out": {
        "path": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip"
      }
    },
    "system": "x86_64-linux"
  }
}
```

Hashed into y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

The whole purpose of Nix is to kind of build this file. This is the recipe describing what it takes to build our IP script.

**** click ****

It has a build command.

**** click ****

The text of our script, which may also be the source code if it's a little more involved.

**** click ****

The tools we depend on to make the script function, like cURL and jq.

**** click ****

The system we're running on, since Nix also supports different architectures and operating systems.

**** click ****

All of this information is hashed to calculate this prefix.

There's a really mind-bending property though.

Hashed into y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

```
{
  "/nix/store/rw89fpljini3yx09v83ds4hyl4945lmk-what-is-my-ip.drv": {
    "args": [
      "-e",
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
    "env": {
      "buildCommand": "<build command>",
      "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
      "name": "what-is-my-ip",
      "out": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip",
      "stdenv": "/nix/store/cf464y4lp2x3lh1qvbg6678lc3f8zbd6-stdenv-linux",
      "system": "x86_64-linux",
      "text": "<contents of script removed for brevity>"
    },
    "inputDrvs": {
      "/nix/store/7k0msqyp2dm021sdj0qjgpkzff8xhqzr-bash-5.2p37.drv": {},
      "/nix/store/a9dyisrdm25k6cprjvypprxs3n8b283-jq-1.7.1.drv": {},
      "/nix/store/crgvallixc0kpg62whvkiv6x38p15nqr-curl-8.11.1.drv": {},
      "/nix/store/ycj0m56p8b0rv9v78moggfa6xhm31rww3-stdenv-linux.drv": {}
    },
    "inputSrcs": [
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "name": "what-is-my-ip",
    "outputs": {
      "out": {
        "path": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip"
      }
    },
    "system": "x86_64-linux"
  }
}
```

Hashed into y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

The whole purpose of Nix is to kind of build this file. This is the recipe describing what it takes to build our IP script.

**** click ****

It has a build command.

**** click ****

The text of our script, which may also be the source code if it's a little more involved.

**** click ****

The tools we depend on to make the script function, like cURL and jq.

**** click ****

The system we're running on, since Nix also supports different architectures and operating systems.

**** click ****

All of this information is hashed to calculate this prefix.

There's a really mind-bending property though.

Hashed into y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

```
{
  "/nix/store/rw89fpljini3yx09v83ds4hyl4945lmk-what-is-my-ip.drv": {
    "args": [
      "-e",
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
    "env": {
      "buildCommand": "<build command>",
      "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
      "name": "what-is-my-ip",
      "out": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip",
      "stdenv": "/nix/store/cf464y41p2x3lh1qvbg6678lc3f8zbd6-stdenv-linux",
      "system": "x86_64-linux",
      "text": "<contents of script removed for brevity>"
    },
    "inputDrvs": {
      "/nix/store/7k0msqyp2dm021sdj0qjgpkzff8xhqzr-bash-5.2p37.drv": {},
      "/nix/store/a9dyisrdm25k6cprjvypprxs3n8b283-jq-1.7.1.drv": {},
      "/nix/store/crgvallixc0kpg62whvkiv6x38p15nqr-curl-8.11.1.drv": {},
      "/nix/store/ycj0m56p8b0rv9v78mggfa6xhm31rww3-stdenv-linux.drv": {}
    },
    "inputSrcs": [
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "name": "what-is-my-ip",
    "outputs": {
      "out": {
        "path": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip"
      }
    },
    "system": "x86_64-linux"
  }
}
```

Hashed into y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

The whole purpose of Nix is to kind of build this file. This is the recipe describing what it takes to build our IP script.

**** click ****

It has a build command.

**** click ****

The text of our script, which may also be the source code if it's a little more involved.

**** click ****

The tools we depend on to make the script function, like cURL and jq.

**** click ****

The system we're running on, since Nix also supports different architectures and operating systems.

**** click ****

All of this information is hashed to calculate this prefix.

There's a really mind-bending property though.

Hashed into y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav

```
{
  "/nix/store/rw89fpljini3yx09v83ds4hyl4945lmk-what-is-my-ip.drv": {
    "args": [
      "-e",
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
    "env": {
      "buildCommand": "<build command>",
      "builder": "/nix/store/gwgqdl0242ymlikq9s9s62gkp5cvyal3-bash-5.2p37/bin/bash",
      "name": "what-is-my-ip",
      "out": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav-what-is-my-ip",
      "stdenv": "/nix/store/cf464y41p2x3lh1qvbg6678lc3f8zbd6-stdenv-linux",
      "system": "x86_64-linux",
      "text": "<contents of script removed for brevity>"
    },
    "inputDrvs": {
      "/nix/store/7k0msqyp2dm021sdj0qjgpkzff8xhqzr-bash-5.2p37.drv": {},
      "/nix/store/a9dyisrdm25k6cprjvypprxs3n8b283-jq-1.7.1.drv": {},
      "/nix/store/crgvallixc0kpg62whvkiv6x38p15nqr-curl-8.11.1.drv": {},
      "/nix/store/ycj0m56p8b0rv9v78moggfa6xhm31rw3-stdenv-linux.drv": {}
    },
    "inputSrcs": [
      "/nix/store/v6x3cs394jgqfbi0a42pam708flxaphh-default-builder.sh"
    ],
    "name": "what-is-my-ip",
    "outputs": {
      "out": {
        "path": "/nix/store/y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav-what-is-my-ip"
      }
    },
    "system": "x86_64-linux"
  }
}
```

Hashed into y2g0ijqqiyi9vxr9xgmvvgb1xqflqzav

The whole purpose of Nix is to kind of build this file. This is the recipe describing what it takes to build our IP script.

**** click ****

It has a build command.

**** click ****

The text of our script, which may also be the source code if it's a little more involved.

**** click ****

The tools we depend on to make the script function, like cURL and jq.

**** click ****

The system we're running on, since Nix also supports different architectures and operating systems.

**** click ****

All of this information is hashed to calculate this prefix.

There's a really mind-bending property though.

Extremely strong guarantees

Seeing `y2g0ijqqiyi9vxr9xgmvvglxqflqzav` means we have *extremely* strong guarantees of the software graph down to the commit and build instructions.



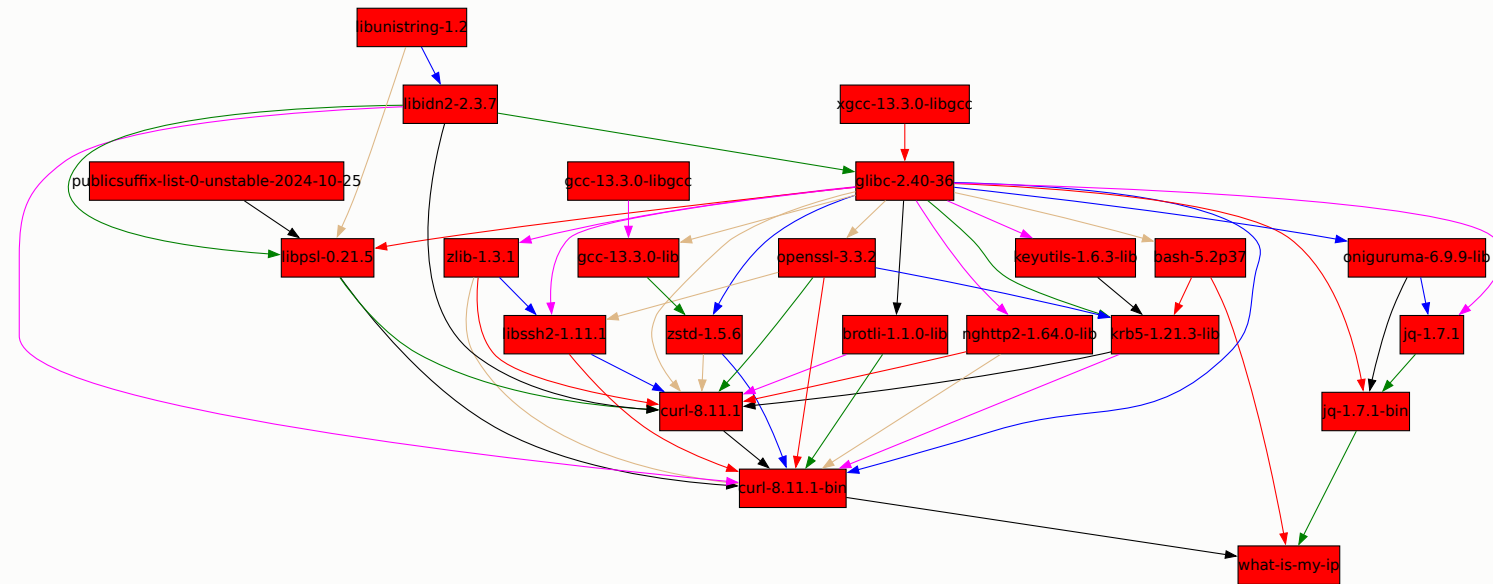
Extremely strong guarantees

That recipe earlier contained the tools we needed, and the source code; each one also had a unique hash prefix.

There's a level of recursion here that means repeatedly seeing this hash is an incredibly strong property of reproducibility.

It means not only was our script text the same and tool dependencies the same, but the same applies to each tool itself and so forth.

Once you know your full software graph, everything is a permutation on that



```
> nix-store --query --graph /nix/store/y2g0ijqqiyi9vxr9xgmvvglxqflqzav-what-is-my-ip
```

Once you know your full software graph, everything is a permutation on that

In Nix, you spend considerable time wiring things up, and the purpose is to completely model your software as a graph.

Turns out that once you have a graph, everything you might want to do in software is a permutation or transformation on the graph.

Here we have the full dependency graph of our seemingly innocuous script. It's a little more involved than originally thought.

Permutation #1: a developer shell

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  pkgs = import (fetchTarball url) {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
in
  pkgs.mkShell {
    packages = [what-is-my-ip];

    shellHook = ''
      echo "Hello, Nix!"
    '';
  }
```

Permutation #1: a developer shell

Okay, here is another Nix snippet. Again ignore the actual syntax, but know that what we are defining here is a shell environment.

**** click ****

We import our same script and add it to the packages that we want present in our shell.

Permutation #1: a developer shell

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  pkgs = import (fetchTarball url) {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
in
  pkgs.mkShell {
    packages = [what-is-my-ip];

    shellHook = ''
      echo "Hello, Nix!"
    '';
  }
```

Permutation #1: a developer shell

Okay, here is another Nix snippet. Again ignore the actual syntax, but know that what we are defining here is a shell environment.

**** click ****

We import our same script and add it to the packages that we want present in our shell.

Permutation #1: a developer shell

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  pkgs = import (fetchTarball url) {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
in
  pkgs.mkShell {
    packages = [what-is-my-ip];

    shellHook = ''
      echo "Hello, Nix!"
    '';
  }
```

Permutation #1: a developer shell

Okay, here is another Nix snippet. Again ignore the actual syntax, but know that what we are defining here is a shell environment.

**** click ****

We import our same script and add it to the packages that we want present in our shell.

Permutation #1: a developer shell

```
> nix-shell what-is-my-ip-shell.nix
```

Remember y2g0ijqqiyi9vxr9xgmvvqblxqflqzav

Permutation #1: a developer shell

Let's enter the shell.

**** click ****

We see the shell hook we wrote. That's just an aid to demonstrate we entered the shell.

**** click ** ** click ****

My PATH now contains what-is-my-ip, and importantly at the exact same hash prefix string we've built before.

**** click ****

The script also works and runs as we expect.

Permutation #1: a developer shell

```
> nix-shell what-is-my-ip-shell.nix
```

```
Hello, Nix!
```

Remember [y2g0ijqqiyi9vxr9xgmvvqblxqflqzav](#)

Permutation #1: a developer shell

Let's enter the shell.

**** click ****

We see the shell hook we wrote. That's just an aid to demonstrate we entered the shell.

**** click ** ** click ****

My PATH now contains what-is-my-ip, and importantly at the exact same hash prefix string we've built before.

**** click ****

The script also works and runs as we expect.

Permutation #1: a developer shell

```
> nix-shell what-is-my-ip-shell.nix
```

```
Hello, Nix!
```

```
[nix-shell]
```

```
> which what-is-my-ip
```

Remember [y2g0ijqqiyi9vxr9xgmvvqblxqflqzav](#)

Permutation #1: a developer shell

Let's enter the shell.

**** click ****

We see the shell hook we wrote. That's just an aid to demonstrate we entered the shell.

**** click ** ** click ****

My PATH now contains what-is-my-ip, and importantly at the exact same hash prefix string we've built before.

**** click ****

The script also works and runs as we expect.

Permutation #1: a developer shell

```
> nix-shell what-is-my-ip-shell.nix

Hello, Nix!

[nix-shell]
> which what-is-my-ip

/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip/bin/what-is-my-ip
```

Remember y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

Permutation #1: a developer shell

Let's enter the shell.

**** click ****

We see the shell hook we wrote. That's just an aid to demonstrate we entered the shell.

**** click ** ** click ****

My PATH now contains what-is-my-ip, and importantly at the exact same hash prefix string we've built before.

**** click ****

The script also works and runs as we expect.

Permutation #1: a developer shell

```
> nix-shell what-is-my-ip-shell.nix

Hello, Nix!

[nix-shell]
> which what-is-my-ip

/nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip/bin/what-is-my-ip

[nix-shell]
> /nix/store/y2g0ijqqiyi9vxr9xgmvvgblxqflqzav-what-is-my-ip/bin/what-is-my-ip
73.231.52.39
```

Remember y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

Permutation #1: a developer shell

Let's enter the shell.

**** click ****

We see the shell hook we wrote. That's just an aid to demonstrate we entered the shell.

**** click ** ** click ****

My PATH now contains what-is-my-ip, and importantly at the exact same hash prefix string we've built before.

**** click ****

The script also works and runs as we expect.

Reproducible developer environments ✓



Reproducible developer environments ✓

Wow. Do we all realize what we just did?

We just created a simple way to bring reproducible developer environments to projects.

No need for containers. No need for namespaces. Easy to enter and easy to validate thanks to that prefix hash.

We actually know everything about our code now.

Reproducible developer environments ✓

We actually know everything about our code now.

**** click ****

Because we know the full dependency closure, we can do binary or source deployments.

**** click ****

We simply copy the necessary store paths to another machine...

**** click ****

...and that machine doesn't even need Nix installed.

**** click ****

Profit.

We actually know everything about our code now.

We can now do binary or source deployments 
since we know the full dependency closure of our tool(s).

Reproducible developer environments ✓

We actually know everything about our code now.

**** click ****

Because we know the full dependency closure, we can do binary or source deployments.

**** click ****

We simply copy the necessary store paths to another machine...

**** click ****

...and that machine doesn't even need Nix installed.

**** click ****

Profit.

We actually know everything about our code now.

We can now do binary or source deployments 
since we know the full dependency closure of our tool(s).

We simply copy the necessary `/nix/store` paths to another machine

Reproducible developer environments ✓

We actually know everything about our code now.

**** click ****

Because we know the full dependency closure, we can do binary or source deployments.

**** click ****

We simply copy the necessary store paths to another machine...

**** click ****

...and that machine doesn't even need Nix installed.

**** click ****

Profit.

We actually know everything about our code now.

We can now do binary or source deployments 
since we know the full dependency closure of our tool(s).

We simply copy the necessary `/nix/store` paths to another machine ~~with Nix installed~~.

Reproducible developer environments ✓

We actually know everything about our code now.

**** click ****

Because we know the full dependency closure, we can do binary or source deployments.

**** click ****

We simply copy the necessary store paths to another machine...

**** click ****

...and that machine doesn't even need Nix installed.

**** click ****

Profit.

We actually know everything about our code now.

We can now do binary or source deployments 🚀🔧📦
since we know the full dependency closure of our tool(s).

We simply copy the necessary `/nix/store` paths to another machine ~~with Nix installed~~.

Profit 😊💰

Reproducible developer environments ✓

We actually know everything about our code now.

**** click ****

Because we know the full dependency closure, we can do binary or source deployments.

**** click ****

We simply copy the necessary store paths to another machine...

**** click ****

...and that machine doesn't even need Nix installed.

**** click ****

Profit.

Permutation #2: a deployment

```
> nix copy --to ssh://dennard.soe.ucsc.edu \  
$(nix build -f what-is-my-ip.nix --print-out-paths)
```

Remember [y2g0ijqqiyi9vxr9xgmvvqblxqflqzav](#)

Permutation #2: a deployment

Here is one such command to copy the graph we saw earlier to another machine. In this example we are leveraging a nix copy command, but you could do something as dumb as scp or rsync.

**** click ****

The items are copied over. Now let's login to the machine.

**** click ****

The exact store path we want is present there and runs. The fact that it runs means all of its dependencies, who also have hash prefix strings, are also present.

Permutation #2: a deployment

```
> nix copy --to ssh://dennard.soe.ucsc.edu \  
$(nix build -f what-is-my-ip.nix --print-out-paths)  
  
> ssh dennard.soe.ucsc.edu  
Last login: Mon Mar  3 05:02:57 2025 from 73.231.52.39
```

Remember [y2g0ijqqiyi9vxr9xgmvvqblxqflqzav](#)

Permutation #2: a deployment

Here is one such command to copy the graph we saw earlier to another machine. In this example we are leveraging a nix copy command, but you could do something as dumb as scp or rsync.

**** click ****

The items are copied over. Now let's login to the machine.

**** click ****

The exact store path we want is present there and runs. The fact that it runs means all of its dependencies, who also have hash prefix strings, are also present.

Permutation #2: a deployment

```
> nix copy --to ssh://dennard.soe.ucsc.edu \  
$(nix build -f what-is-my-ip.nix --print-out-paths)  
  
> ssh dennard.soe.ucsc.edu  
Last login: Mon Mar  3 05:02:57 2025 from 73.231.52.39  
  
[dennard]  
> /nix/store/y2g0ijqqiyi9vxr9xgmvvgbllxqflqzav-what-is-my-ip/bin/what-is-my-ip  
128.114.53.24
```

Remember y2g0ijqqiyi9vxr9xgmvvgbllxqflqzav

Permutation #2: a deployment

Here is one such command to copy the graph we saw earlier to another machine. In this example we are leveraging a nix copy command, but you could do something as dumb as scp or rsync.

**** click ****

The items are copied over. Now let's login to the machine.

**** click ****

The exact store path we want is present there and runs. The fact that it runs means all of its dependencies, who also have hash prefix strings, are also present.

Binary (or source) deployments ✓



Binary (or source) deployments ✓

Wow. We just did a deployment of not only our "code", which was our script, but all of its necessary dependencies.

This blows my mind. I can't tell you the number of times in practice I had to coordinate the runtime needed on the host or some other base library being present.

That's awesome! Unfortunately, our company uses Docker / Kubernetes 😞

That's awesome! Unfortunately, our company uses Docker / Kubernetes 😞

Well, maybe that's a bit aggressive for deployments, and you have already drunk the kool-aid of Kubernetes.

**** click ****

No problem! Since we know the graph, everything we'd want is merely a permutation on our graph.

Here is another snippet, and again ignore the syntax, but know this is creating an OCI compliant image.

**** click ****

We set the command of the image to be our what-is-my-ip package.

That's awesome! Unfortunately, our company uses

Docker / Kubernetes 🙄

No problem! Again, once we know the graph of our software everything is just a permutation.

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  pkgs = import (fetchTarball url) {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
in
  pkgs.dockerTools.buildImage {
    name = "what-is-my-ip-docker";
    config = {
      Cmd = ["${what-is-my-ip}/bin/what-is-my-ip"];
    };
  }
```

**That's awesome! Unfortunately, our company uses Docker /
Kubernetes** 🙄

Well, maybe that's a bit aggressive for deployments, and you have already drunk the kool-aid of Kubernetes.

**** click ****

No problem! Since we know the graph, everything we'd want is merely a permutation on our graph.

Here is another snippet, and again ignore the syntax, but know this is creating an OCI compliant image.

**** click ****

We set the command of the image to be our what-is-my-ip package.

That's awesome! Unfortunately, our company uses

Docker / Kubernetes 😞

No problem! Again, once we know the graph of our software everything is just a permutation.

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  pkgs = import (fetchTarball url) {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
in
  pkgs.dockerTools.buildImage {
    name = "what-is-my-ip-docker";
    config = {
      Cmd = ["${what-is-my-ip}/bin/what-is-my-ip"];
    };
  }
```

That's awesome! Unfortunately, our company uses Docker / Kubernetes 😞

Well, maybe that's a bit aggressive for deployments, and you have already drunk the kool-aid of Kubernetes.

**** click ****

No problem! Since we know the graph, everything we'd want is merely a permutation on our graph.

Here is another snippet, and again ignore the syntax, but know this is creating an OCI compliant image.

**** click ****

We set the command of the image to be our what-is-my-ip package.

Permutation #3: a container

```
> nix build -f what-is-my-ip-docker.nix --print-out-paths
```

Remember y2g0ijqqiyi9vxr9xgmvvqblxqflqzav

Permutation #3: a container

Let's build the image.

**** click ****

The output is an OCI tarball.

**** click ****

We can load that tarball into the docker daemon as normal.

**** click ****

Running the image works as we'd expect as well.

**** click ****

Inspecting the image, we see yet again that same prefix hash on our package for the command to execute -- which continues to give us comfort about the version and state of our software.

Permutation #3: a container

```
> nix build -f what-is-my-ip-docker.nix --print-out-paths

/nix/store/vwlbi7m9l23cm6k jy1skh4fskh02c5zc-docker-image-what-is-my-ip-docker.tar.gz
```

Remember [y2g0ijqqiyi9vxr9xgmvvgblxqflqzav](#)

Permutation #3: a container

Let's build the image.

click

The output is an OCI tarball.

click

We can load that tarball into the docker daemon as normal.

click

Running the image works as we'd expect as well.

click

Inspecting the image, we see yet again that same prefix hash on our package for the command to execute -- which continues to give us comfort about the version and state of our software.

Permutation #3: a container

```
> nix build -f what-is-my-ip-docker.nix --print-out-paths

/nix/store/vwlbi7m9l23cm6k jy1skh4fskh02c5zc-docker-image-what-is-my-ip-docker.tar.gz

> docker load /nix/store/vwlbi7m9l23cm6k jy1skh4fskh02c5zc-docker-image-what-is-my-ip-docker.tar.gz
Loaded image: what-is-my-ip-docker:vwlbi7m9l23cm6k jy1skh4fskh02c5zc
```

Remember y2g0ijqqiyi9vxr9xgmvvgblxqflqzav

Permutation #3: a container

Let's build the image.

click

The output is an OCI tarball.

click

We can load that tarball into the docker daemon as normal.

click

Running the image works as we'd expect as well.

click

Inspecting the image, we see yet again that same prefix hash on our package for the command to execute -- which continues to give us comfort about the version and state of our software.

Permutation #3: a container

```
> nix build -f what-is-my-ip-docker.nix --print-out-paths

/nix/store/vwlbi7m9l23cm6kjylskh4fskh02c5zc-docker-image-what-is-my-ip-docker.tar.gz

> docker load /nix/store/vwlbi7m9l23cm6kjylskh4fskh02c5zc-docker-image-what-is-my-ip-docker.tar.gz
Loaded image: what-is-my-ip-docker:vwlbi7m9l23cm6kjylskh4fskh02c5zc

> docker run -it what-is-my-ip-docker:vwlbi7m9l23cm6kjylskh4fskh02c5zc
73.231.52.39
```

Remember y2g0ijqqiyi9vxr9xgmvvqblxqflqzav

Permutation #3: a container

Let's build the image.

click

The output is an OCI tarball.

click

We can load that tarball into the docker daemon as normal.

click

Running the image works as we'd expect as well.

click

Inspecting the image, we see yet again that same prefix hash on our package for the command to execute -- which continues to give us comfort about the version and state of our software.

Permutation #3: a container

```
> nix build -f what-is-my-ip-docker.nix --print-out-paths

/nix/store/vwlbi7m9l23cm6kjylskh4fskh02c5zc-docker-image-what-is-my-ip-docker.tar.gz

> docker load /nix/store/vwlbi7m9l23cm6kjylskh4fskh02c5zc-docker-image-what-is-my-ip-docker.tar.gz
Loaded image: what-is-my-ip-docker:vwlbi7m9l23cm6kjylskh4fskh02c5zc

> docker run -it what-is-my-ip-docker:vwlbi7m9l23cm6kjylskh4fskh02c5zc
73.231.52.39

> docker inspect what-is-my-ip-docker:vwlbi7m9l23cm6kjylskh4fskh02c5zc --format "{{.Config.Cmd}}"
[/nix/store/y2g0ijqqiyi9vxr9xgmvvqblxqflqzav-what-is-my-ip/bin/what-is-my-ip]
```

Remember y2g0ijqqiyi9vxr9xgmvvqblxqflqzav

Permutation #3: a container

Let's build the image.

click

The output is an OCI tarball.

click

We can load that tarball into the docker daemon as normal.

click

Running the image works as we'd expect as well.

click

Inspecting the image, we see yet again that same prefix hash on our package for the command to execute -- which continues to give us comfort about the version and state of our software.

Cool! 😎

Nix + Docker integrate perfectly. The image produced has only the files exactly necessary to run the tool provided, effectively distroless.

You can do the same thing for any deployment format!

- OCI images
- tarballs
- self-extracting ZIP
- rpm
- deb
- Flatpak / AppImage

Cool! 😎

Nix and Docker integrate perfectly. The image has exactly the files necessary to run the tool -- effectively distroless.

And the same trick works for any deployment format: tarballs, RPMs, debs, AppImages. They are all just another permutation of the graph.

A Linux distribution is simply a collection of packages.

Cool! 😎

Which brings us to the big one. A Linux distribution is simply a collection of packages.

Permutation #4: an operating system

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  nixpkgs = fetchTarball url;
  pkgs = import nixpkgs {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
  nixos = import "${nixpkgs}/nixos" {
    configuration = {
      users.users.alice = {
        isNormalUser = true;
        # enable sudo
        extraGroups = ["wheel"];
        packages = [
          what-is-my-ip
        ];
        initialPassword = "swordfish";
      };

      system.stateVersion = "24.05";
    };
  };
in
  nixos.vm
```

Permutation #4: an operating system

Here is a whole NixOS machine described in Nix.

**** click ****

Our same script, added to the packages for a user named alice.

**** click ****

And we ask for it as a virtual machine.

Permutation #4: an operating system

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  nixpkgs = fetchTarball url;
  pkgs = import nixpkgs {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
  nixos = import "${nixpkgs}/nixos" {
    configuration = {
      users.users.alice = {
        isNormalUser = true;
        # enable sudo
        extraGroups = ["wheel"];
        packages = [
          what-is-my-ip
        ];
        initialPassword = "swordfish";
      };

      system.stateVersion = "24.05";
    };
  };
in
  nixos.vm
```

Permutation #4: an operating system

Here is a whole NixOS machine described in Nix.

**** click ****

Our same script, added to the packages for a user named alice.

**** click ****

And we ask for it as a virtual machine.

Permutation #4: an operating system

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  nixpkgs = fetchTarball url;
  pkgs = import nixpkgs {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
  nixos = import "${nixpkgs}/nixos" {
    configuration = {
      users.users.alice = {
        isNormalUser = true;
        # enable sudo
        extraGroups = ["wheel"];
        packages = [
          what-is-my-ip
        ];
        initialPassword = "swordfish";
      };

      system.stateVersion = "24.05";
    };
  };
in
  nixos.vm
```

Permutation #4: an operating system

Here is a whole NixOS machine described in Nix.

**** click ****

Our same script, added to the packages for a user named alice.

**** click ****

And we ask for it as a virtual machine.

Permutation #4: an operating system

```
let
  rev = "5ef6c425980847c78a80d759abc476e941a9bf42";
  url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
  nixpkgs = fetchTarball url;
  pkgs = import nixpkgs {};
  what-is-my-ip = import ./what-is-my-ip.nix {inherit pkgs;};
  nixos = import "${nixpkgs}/nixos" {
    configuration = {
      users.users.alice = {
        isNormalUser = true;
        # enable sudo
        extraGroups = ["wheel"];
        packages = [
          what-is-my-ip
        ];
        initialPassword = "swordfish";
      };

      system.stateVersion = "24.05";
    };
  };
in
  nixos.vm
```

Permutation #4: an operating system

Here is a whole NixOS machine described in Nix.

**** click ****

Our same script, added to the packages for a user named alice.

**** click ****

And we ask for it as a virtual machine.

Permutation #4: an operating system

```
> nix build -f what-is-my-ip-vm.nix --print-out-paths
```

Remember [y2g0ijqqiyi9vxr9xgmvvqblxqflqzav](#)

Permutation #4: an operating system

Build it.

**** click ****

A VM.

**** click ** ** click ****

Boot it, and log in as alice.

**** click ****

Our script is on alice's PATH, it resolves to the very same store path, and it runs.

Permutation #4: an operating system

```
> nix build -f what-is-my-ip-vm.nix --print-out-paths  
  
/nix/store/k6vqp2hn0gh7r5dmc61f5knqgnqn53wz-nixos-vm
```

Remember y2g0ijqqiyi9vxr9xgmvvqblxqflqzav

Permutation #4: an operating system

Build it.

**** click ****

A VM.

**** click ** ** click ****

Boot it, and log in as alice.

**** click ****

Our script is on alice's PATH, it resolves to the very same store path, and it runs.

Permutation #4: an operating system

```
> nix build -f what-is-my-ip-vm.nix --print-out-paths

/nix/store/k6vqp2hn0gh7r5dmc61f5knqgnqn53wz-nixos-vm

> QEMU_KERNEL_PARAMS=console=ttyS0 \
  /nix/store/k6vqp2hn0gh7r5dmc61f5knqgnqn53wz-nixos-vm/bin/run-nixos-vm -nographic; reset
```

Remember y2g0ijqqiyi9vxr9xgmvvqblxqflqzav

Permutation #4: an operating system

Build it.

**** click ****

A VM.

**** click ** ** click ****

Boot it, and log in as alice.

**** click ****

Our script is on alice's PATH, it resolves to the very same store path, and it runs.

Permutation #4: an operating system

```
> nix build -f what-is-my-ip-vm.nix --print-out-paths

/nix/store/k6vqp2hn0gh7r5dmc61f5knqgnqn53wz-nixos-vm

> QEMU_KERNEL_PARAMS=console=ttyS0 \
  /nix/store/k6vqp2hn0gh7r5dmc61f5knqgnqn53wz-nixos-vm/bin/run-nixos-vm -nographic; reset

... long boot sequence ...

<<< Welcome to NixOS 24.11pre-git (x86_64) - ttyS0 >>>

nixos login: alice
Password:
```

Remember y2g0ijqqiyi9vxr9xgmvvqblxqflqzav

Permutation #4: an operating system

Build it.

**** click ****

A VM.

**** click ** ** click ****

Boot it, and log in as alice.

**** click ****

Our script is on alice's PATH, it resolves to the very same store path, and it runs.

Permutation #4: an operating system

```
> nix build -f what-is-my-ip-vm.nix --print-out-paths

/nix/store/k6vqp2hn0gh7r5dmc61f5knqgnqn53wz-nixos-vm

> QEMU_KERNEL_PARAMS=console=ttyS0 \
  /nix/store/k6vqp2hn0gh7r5dmc61f5knqgnqn53wz-nixos-vm/bin/run-nixos-vm -nographic; reset

... long boot sequence ...

<<< Welcome to NixOS 24.11pre-git (x86_64) - ttyS0 >>>

nixos login: alice
Password:

[alice@nixos:~]$ which what-is-my-ip
/etc/profiles/per-user/alice/bin/what-is-my-ip

[alice@nixos:~]$ realpath $(which what-is-my-ip)
/nix/store/y2g0ijqqiyi9vvr9xgmvvqblxqflqzav-what-is-my-ip/bin/what-is-my-ip

[alice@nixos:~]$ what-is-my-ip
73.231.52.39
```

Remember y2g0ijqqiyi9vvr9xgmvvqblxqflqzav

Permutation #4: an operating system

Build it.

**** click ****

A VM.

**** click ** ** click ****

Boot it, and log in as alice.

**** click ****

Our script is on alice's PATH, it resolves to the very same store path, and it runs.

So... what is Nix?

Permutation #4: an operating system

We've now taken one tiny script and turned it into a developer shell, a deployment, a container and an operating system. So let's step back: what actually is Nix?

A language, and a command-line tool

```
# hello-world.nix
derivation {
  name = "hello-world";
  # POSIX requires this file to be present
  # so we can depend on it but not on /bin/bash
  builder = "/bin/sh";
  args = [ "-c" "echo 'hello world' > $out" ];
  system = builtins.currentSystem;
}
```

```
# hello-world-writeText.nix
let pkgs = import <nixpkgs> {};
in
pkgs.writeText "hello-world" "hello world"
```

```
> nix build --file hello-world.nix --json | jq -r '.[].outputs.out'
/nix/store/dg2rpsw16yfiw3h2ma14j5qh7a6m08kk-hello-world

> cat /nix/store/dg2rpsw16yfiw3h2ma14j5qh7a6m08kk-hello-world
hello world
```

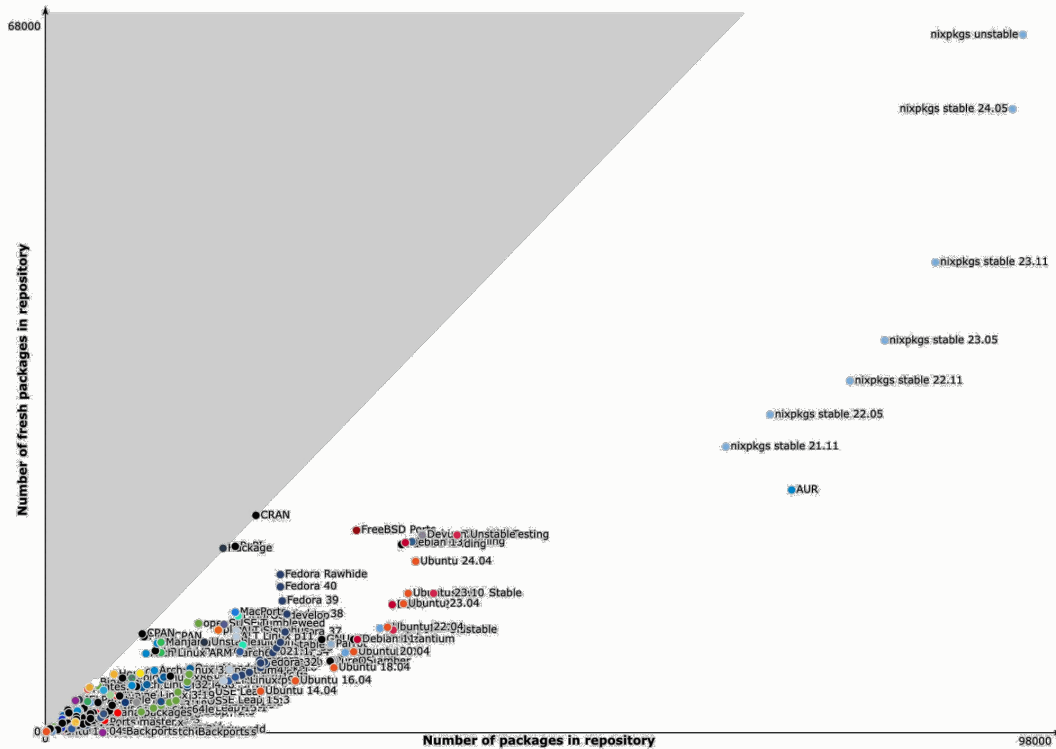
A language, and a command-line tool

Nix is everything, which is part of why it's hard to explain. It began as Eelco Dolstra's PhD research, and I highly recommend the read.

It's a language, depicted here. It's used ultimately to create what the language refers to as 'derivations', but you can think of them as build recipes. The language resembles JSON a little, with the added bonus that it's a pure, functional, lazy, untyped language. It's kind of like a poor version of Haskell. It's also the command line tool itself, called nix, with which I can do a bunch of stuff like build these recipes or create developer environments, as we saw.

A package collection: nixpkgs

Repository size/freshness map



github.com/NixOS/nixpkgs

Repository size vs.
freshness, from
repology.org

A package collection: nixpkgs

It's a package collection, known as nixpkgs. On repology's map of every package repository, nixpkgs is out on its own: the largest, and among the freshest.

GitHub publishes a yearly report known as Octoverse, and nixpkgs has been in the top 10 across multiple dimensions such as contributors, activity and pull requests.

Another fun fact: this whole package set can be bootstrapped from a 357-byte binary seed.

One recipe, many permutations

```
> nix run nixpkgs#hello  
Hello, world!
```

One recipe, many permutations

Because nixpkgs is a graph, a package is not one thing, it's a recipe you can permute.

click

Here's GNU hello, built normally.

click

The same hello, statically linked against musl. One attribute.

click

Cross-compiled for RISC-V. It won't run on my x86 laptop, which is the point.

click

Or for Windows, and we can run it under wine. Same recipe, every time; just a different path through the graph.

One recipe, many permutations

```
> nix run nixpkgs#hello
Hello, world!

> nix build nixpkgs#hello --no-link --print-out-paths
/nix/store/m1r53pnm6hnjwyjmxska24y8amvlpjp-hello-2.12.1
```

One recipe, many permutations

Because nixpkgs is a graph, a package is not one thing, it's a recipe you can permute.

click

Here's GNU hello, built normally.

click

The same hello, statically linked against musl. One attribute.

click

Cross-compiled for RISC-V. It won't run on my x86 laptop, which is the point.

click

Or for Windows, and we can run it under wine. Same recipe, every time; just a different path through the graph.

One recipe, many permutations

```
> nix run nixpkgs#hello
Hello, world!

> nix build nixpkgs#hello --no-link --print-out-paths
/nix/store/m1r53pnm6hnjwyjmxska24y8amvlpjp-hello-2.12.1

> nix build nixpkgs#pkgsStatic.hello --no-link --print-out-paths
/nix/store/d3scdlvhylfvjgwg2ydbhdlss7zpcr-hello-static-x86_64-unknown-linux-musl-2.12.1
```

One recipe, many permutations

Because nixpkgs is a graph, a package is not one thing, it's a recipe you can permute.

click

Here's GNU hello, built normally.

click

The same hello, statically linked against musl. One attribute.

click

Cross-compiled for RISC-V. It won't run on my x86 laptop, which is the point.

click

Or for Windows, and we can run it under wine. Same recipe, every time; just a different path through the graph.

One recipe, many permutations

```
> nix run nixpkgs#hello
Hello, world!

> nix build nixpkgs#hello --no-link --print-out-paths
/nix/store/mlr53pnnm6hnjwyjmxska24y8amvlpjp-hello-2.12.1

> nix build nixpkgs#pkgsStatic.hello --no-link --print-out-paths
/nix/store/d3scdlvhylfvjgjpg2ydbhdlss7zpcr-hello-static-x86_64-unknown-linux-musl-2.12.1

> nix build nixpkgs#pkgsCross.riscv64.hello --no-link --print-out-paths
/nix/store/p2fhzrphxmd8if2mf7b28biawj7walf1-hello-riscv64-unknown-linux-gnu-2.12.1
> /nix/store/p2fhzrphxmd8if2mf7b28biawj7walf1-hello-riscv64-unknown-linux-gnu-2.12.1/bin/hello
bash: ../bin/hello: cannot execute binary file
```

One recipe, many permutations

Because nixpkgs is a graph, a package is not one thing, it's a recipe you can permute.

click

Here's GNU hello, built normally.

click

The same hello, statically linked against musl. One attribute.

click

Cross-compiled for RISC-V. It won't run on my x86 laptop, which is the point.

click

Or for Windows, and we can run it under wine. Same recipe, every time; just a different path through the graph.

One recipe, many permutations

```
> nix run nixpkgs#hello
Hello, world!

> nix build nixpkgs#hello --no-link --print-out-paths
/nix/store/mlr53pnm6hnjwyjmxska24y8amvlpjp-hello-2.12.1

> nix build nixpkgs#pkgsStatic.hello --no-link --print-out-paths
/nix/store/d3scdlvhylfvjgjjwgp2ydbhdlss7zpcr-hello-static-x86_64-unknown-linux-musl-2.12.1

> nix build nixpkgs#pkgsCross.riscv64.hello --no-link --print-out-paths
/nix/store/p2fhzrphxmd8if2mf7b28biawj7walf1-hello-riscv64-unknown-linux-gnu-2.12.1
> /nix/store/p2fhzrphxmd8if2mf7b28biawj7walf1-hello-riscv64-unknown-linux-gnu-2.12.1/bin/hello
bash: ../bin/hello: cannot execute binary file

> nix build nixpkgs#pkgsCross.mingw64.hello --no-link --print-out-paths
/nix/store/b78fyx3z62pdww8a3nk8vfa3cb2api6h-hello-x86_64-w64-mingw32-2.12.1
> wine64 /nix/store/b78fyx3z62pdww8a3nk8vfa3cb2api6h-hello-x86_64-w64-mingw32-2.12.1/bin/hello.exe
Hello, world!
```

One recipe, many permutations

Because nixpkgs is a graph, a package is not one thing, it's a recipe you can permute.

click

Here's GNU hello, built normally.

click

The same hello, statically linked against musl. One attribute.

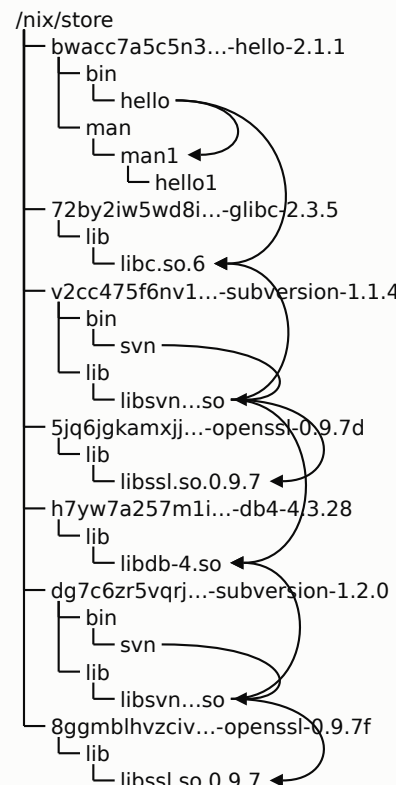
click

Cross-compiled for RISC-V. It won't run on my x86 laptop, which is the point.

click

Or for Windows, and we can run it under wine. Same recipe, every time; just a different path through the graph.

A store



Source: [Eelco Dolstra, The Purely Functional Software Deployment Model \(2006\)](#), redrawn

A store

It's a store. This is actually a very important distinction that separates Nix from traditional software installations: it forgoes the Filesystem Hierarchy Standard, the well-known directories like `/lib` and `/etc` that we all know and love.

By forgoing that limitation, Nix is capable of having limitless variants of a single package, since they are no longer confined to a single key space. Furthermore, dependencies of Nix software only exist within other store entries, forming a DAG.

An operating system

```
eudoxia@sextant:~$ neofetch

eudoxia@sextant
-----
OS: NixOS 22.11.3681.2b1bba76a13 (Raccoon) x86_64
Host: LENOVO 20Y5002MAU
Kernel: 5.15.106
Uptime: 2 hours, 7 mins
Packages: 934 (nix-system), 1431 (nix-user)
Shell: bash 5.1.16
Resolution: 2560x1600
DE: none+spectrum
WM: spectrum
Theme: Adwaita [GTK3]
Icons: Adwaita [GTK3]
Terminal: urxvt
Terminal Font: terminus
CPU: 11th Gen Intel i7-11800H (16) @ 4.600GHz
GPU: NVIDIA GeForce RTX 3050 Ti Mobile
GPU: Intel TigerLake-H GT1 [UHD Graphics]
Memory: 3218MiB / 15727MiB
```

An operating system

It's an operating system. Once you have solved producing software packages, a Linux distribution is merely a collection of software packages and a kernel.

Nix Language

Nix Operating System

Nix Package Manager

Modeling software as a directed acyclic graph.

Nix Ecosystem

Nix Build System

Nix Stores

Nix Command Line Tools

Developer Environments

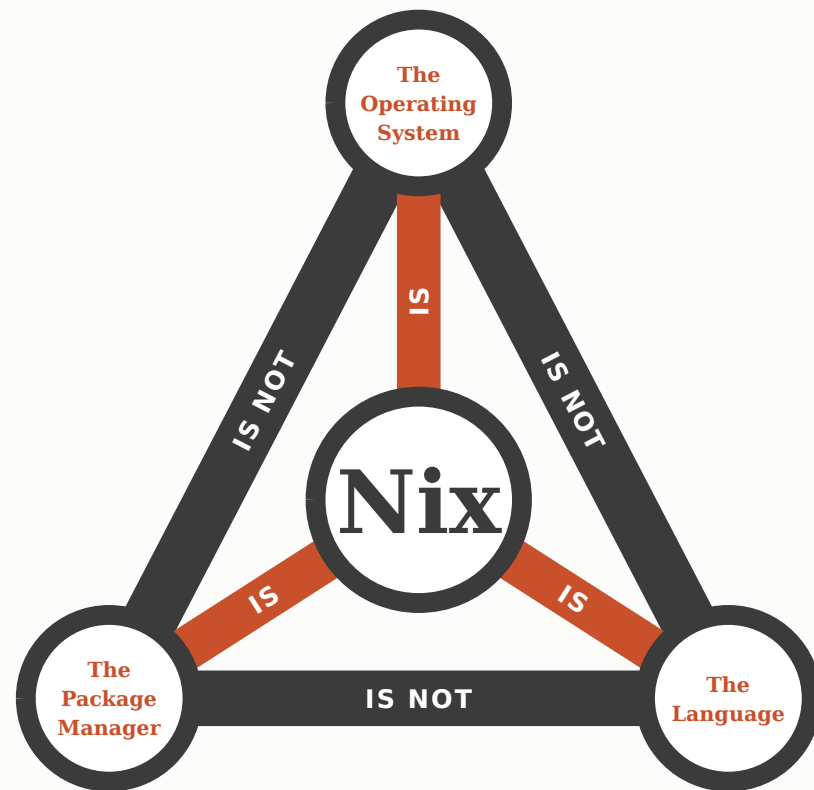
An operating system

Fundamentally though, Nix is the idea of modeling all software as a directed acyclic graph with an immense level of fidelity.

It's an idea that says: what if every part of our build system was accounted for?

The language, the store, the package set, the OS, the dev shells -- they're all just ways of building, storing and walking that graph.

The declarative trinity



Source: Understand NixOS, redrawn

The declarative trinity

Which is why the naming is so confusing. Nix the language, Nix the package manager, and NixOS the operating system are all called Nix, but none of them is the other. What they share is the graph.

Under the hood

How does any of this actually work?

The declarative trinity

For the last part, let's go one level down and look at the mechanism.

I'm going to go over very briefly how software is laid out traditionally, hopefully illuminate how it's utter madness, and then tie the thread to see how Nix solves it.

A recurring theme: Docker, specifically containers, are not a solution to any of this. Containers are just a snapshot at a particular state.

A story about Ruby

```
> ls /usr/bin/ruby*  
/usr/bin/ruby  /usr/bin/ruby3.0  /usr/bin/ruby3.1
```

A story about Ruby

I'm going to pick Ruby, as it's an innocuous piece of software. It's written in C, present on every distribution, and would seemingly have a very small set of dependencies.

Here are the Rubies on a typical machine.

click

Which one is this? What was it built with?

A story about Ruby

```
> ls /usr/bin/ruby*  
/usr/bin/ruby  /usr/bin/ruby3.0  /usr/bin/ruby3.1  
  
> /usr/bin/ruby3.1 --version  
ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [x86_64-linux-gnu]
```

A story about Ruby

I'm going to pick Ruby, as it's an innocuous piece of software. It's written in C, present on every distribution, and would seemingly have a very small set of dependencies.

Here are the Rubies on a typical machine.

**** click ****

Which one is this? What was it built with?

Building Ruby

1. Install the prerequisite dependencies for building the CRuby interpreter:
 - C compiler

For RubyGems, you will also need:

- OpenSSL 1.1.x or 3.0.x / LibreSSL
- libyaml 0.1.7 or later
- zlib

If you want to build from the git repository, you will also need:

- autoconf - 2.67 or later
- gperf - 3.1 or later
 - Usually unneeded; only if you edit some source files using gperf
- ruby - 2.5 or later
 - We can upgrade this version to system ruby version of the latest Ubuntu LTS.

2. Install optional, recommended dependencies:

- libffi (to build fiddle)
- gmp (if you wish to accelerate Bignum operations)
- libexecinfo (FreeBSD)
- rustc - 1.58.0 or later, if you wish to build YJIT.

From Ruby's [building_ruby.md](#)

Building Ruby

Ruby's own build instructions. A C compiler, OpenSSL, libyaml, zlib -- and if you build from git, autoconf, gperf, and a Ruby to build Ruby. Plus optional extras.

Where does each of these come from on your machine? What version?

What does it actually load?

```
> libtree /usr/bin/ruby3.1 -p
/usr/bin/ruby3.1
├─ /lib/x86_64-linux-gnu/libruby-3.1.so.3.1
│   ├─ /lib/x86_64-linux-gnu/libz.so.1
│   ├─ /lib/x86_64-linux-gnu/libcrypt.so.1
│   └─ /lib/x86_64-linux-gnu/libgmp.so.10
├─ /lib/x86_64-linux-gnu/libcrypt.so.1
├─ /lib/x86_64-linux-gnu/libgmp.so.10
└─ /lib/x86_64-linux-gnu/libz.so.1
```

What does it actually load?

Here is what the installed ruby resolves at runtime. Everything comes from one shared /lib, by name only. Whatever libz.so.1 happens to be there, wins.

...and everything it loads, loads

```
> libtree /usr/bin/ruby3.1 -vvv
/usr/bin/ruby3.1
├── libruby-3.1.so.3.1 [ld.so.conf]
│   ├── libz.so.1 [ld.so.conf]
│   │   └── libc.so.6 [ld.so.conf]
│   │       └── ld-linux-x86-64.so.2 [ld.so.conf]
│   ├── ld-linux-x86-64.so.2 [ld.so.conf]
│   ├── libc.so.6 [ld.so.conf]
│   │   └── ld-linux-x86-64.so.2 [ld.so.conf]
│   ├── libm.so.6 [ld.so.conf]
│   │   ├── libc.so.6 [ld.so.conf]
│   │   │   └── ld-linux-x86-64.so.2 [ld.so.conf]
│   │   └── ld-linux-x86-64.so.2 [ld.so.conf]
│   ├── libcrypt.so.1 [ld.so.conf]
│   │   └── libc.so.6 [ld.so.conf]
│   │       └── ld-linux-x86-64.so.2 [ld.so.conf]
│   └── libgmp.so.10 [ld.so.conf]
│       └── libc.so.6 [ld.so.conf]
│           └── ld-linux-x86-64.so.2 [ld.so.conf]
├── libc.so.6 [ld.so.conf]
│   └── ld-linux-x86-64.so.2 [ld.so.conf]
├── libm.so.6 [ld.so.conf]
│   ├── libc.so.6 [ld.so.conf]
│   │   └── ld-linux-x86-64.so.2 [ld.so.conf]
│   └── ld-linux-x86-64.so.2 [ld.so.conf]
├── libcrypt.so.1 [ld.so.conf]
│   └── libc.so.6 [ld.so.conf]
│       └── ld-linux-x86-64.so.2 [ld.so.conf]
├── libz.so.1 [ld.so.conf]
│   └── libc.so.6 [ld.so.conf]
│       └── ld-linux-x86-64.so.2 [ld.so.conf]
└── ld-linux-x86-64.so.2 [ld.so.conf]
```

...and everything it loads, loads

And with the full verbose tree, every library pulls in more libraries, all resolved the same way: by name, from a single global directory.

Ruby, as Nix sees it

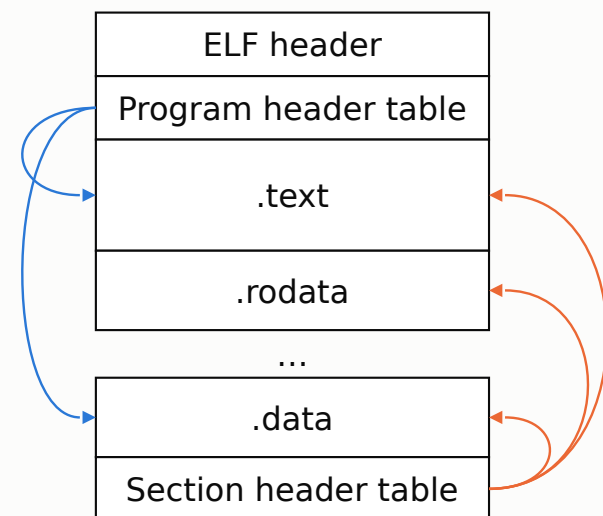


```
> nix-store --query --include-outputs --graph \  
$(nix-store --query --deriver $(nix eval nixpkgs#ruby.outPath --raw))
```

Ruby, as Nix sees it

And this is Ruby's full build graph in nixpkgs, every input accounted for. This is the thing Nix makes you model -- and once you have it, everything else we saw today falls out of it.

Quick primer: what's a binary?



```
$ readelf -a $(which ruby) | grep NEEDED
0x0000000000000001 (NEEDED) Shared library: [libruby-2.7.so.2.7]
0x0000000000000001 (NEEDED) Shared library: [libc.so.6]
$ readelf -a libruby-2.7.so.2.7 | grep NEEDED
0x0000000000000001 (NEEDED) Shared library: [libpthread.so.0]
0x0000000000000001 (NEEDED) Shared library: [librt.so.1]
0x0000000000000001 (NEEDED) Shared library: [libgmp.so.10]
0x0000000000000001 (NEEDED) Shared library: [libdl.so.2]
0x0000000000000001 (NEEDED) Shared library: [libcrypt.so.1]
0x0000000000000001 (NEEDED) Shared library: [libm.so.6]
0x0000000000000001 (NEEDED) Shared library: [libc.so.6]
0x0000000000000001 (NEEDED) Shared library: [ld-linux-x86-64.so.2]
```

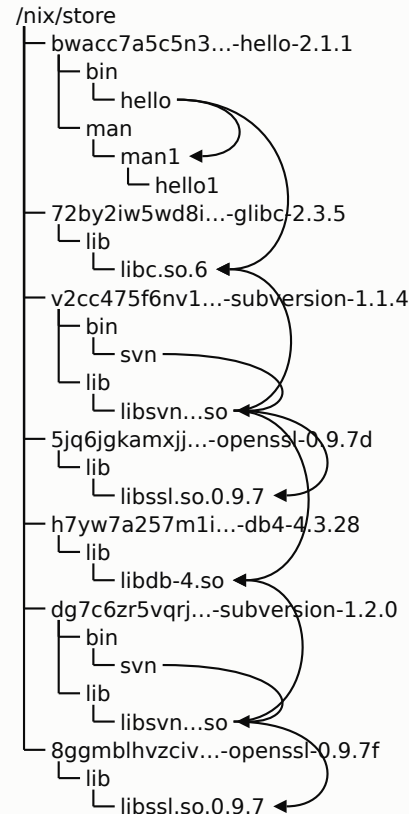
Source: [Surueña, CC BY-SA 3.0](#), redrawn

Quick primer: what's a binary?

A quick primer on what a binary is in Linux. All binaries on Linux are in the ELF format. It's an incredibly generic format.

Specifically, a dynamic binary must have an interpreter set, which is always the dynamic linker, and can list the libraries it needs. Note that NEEDED entries are just names -- not paths, and not versions.

An alternative to the FHS model



```
$ patchelf --print-rpath ruby | tr ':' '\n'
/nix/store/yyhykbfxfj6jfc62x4f9a2awfbch605fd-ruby-2.6.6/lib
/nix/store/6hhqb94ilrdmh0cx5bhdqb92m6bqkvdj-zlib-1.2.11/lib
/nix/store/90illc73xfs933d06daq6d41njs8yh66-glibc-2.32-37/
```

Source: [Eelco Dolstra, The Purely Functional Software Deployment Model \(2006\)](#), redrawn

An alternative to the FHS model

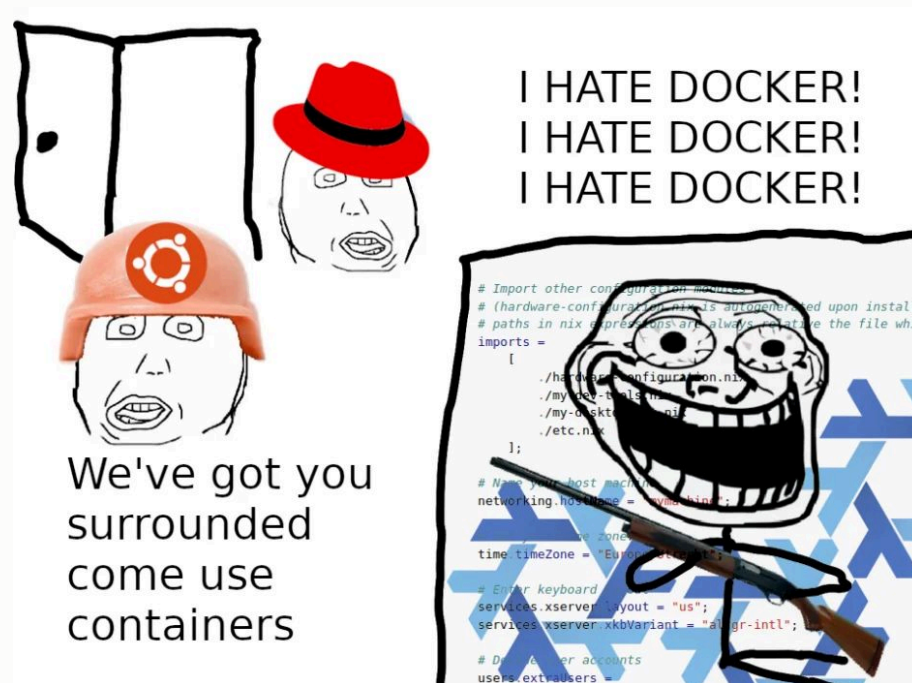
This slide won't do the concept justice at all, so I apologize; I could probably spend multiple hours talking about it.

In order to add reproducibility to the world of software management, a new concept was developed in the early 2000s as part of a PhD thesis. The idea was to nest all software underneath a single root, include a cryptographic hash in the name so similar packages could live side by side, and restrict all dependencies to other entries in the root, or store.

Dynamic linking relies on a feature to specify an RPATH on the binary, which supersedes the FHS search. Each NEEDED name now resolves against exact store paths.

Similar to RPATH, the developer environments set the PATH variable to isolate access to the current terminal.

When you discover NixOS



https://www.reddit.com/r/NixOS/comments/p132mz/when_you_discover_nixos/

When you discover NixOS

And that's how it feels once it clicks.

Fin.

Thank you!

fzakaria.github.io/learn-nix-the-fun-way

fzakaria.com/2024/07/05/learn-nix-the-fun-way.html

When you discover NixOS

Thank you! The slides and every example are online; questions welcome.