



	<div data-bbox="824 184 1572 604"><h1>Reproducible builds and why NixOS matters</h1><p>Tired of "works on my machine"?</p><p>NixOS Foundation</p><p>mmkaram</p><p>1</p></div>
<p>About me:</p> <ul style="list-style-type: none">- In my 1B term- From seattle- Like to play squash and read- Have been using nixos- Am involved in the opencv and prisma space concerning nixos, if you ever use the nixos wiki concerning those two programs I probably wrote whatever text you're reading	<div data-bbox="824 642 1572 1062"><h2>About me</h2><ul style="list-style-type: none">- Name: Mahdy Karam- Term: 1B- Hometown: Seattle, WA, USA- Interests: Squash, Reading- Nixos User since Dec 2023- OpenCV, Prisma<p>2</p></div>
<p>Plan for today:</p> <ul style="list-style-type: none">- I'm going to go pretty in depth on why nixos might be useful, and only after that will I explain what it is and how it can solve common problems- Then I'll try and point you all to some places to learn more if you'd like	<div data-bbox="824 1100 1572 1520"><h2>Agenda</h2><ul style="list-style-type: none">- Introduction: Why is NixOS important/how can it be used- Nix ecosystem<ul style="list-style-type: none">- Nix language- Nixpkgs- NixOS- Nix shell- Home-manager- Flakes- Further reading<p>3</p></div>
<p>Congrats! You got a job at LooWaterAISoft as a development operations engineer! The product team has spent 3 months making an AI Web3 crypto platform using nextJS, and it's your job to deploy it!</p> <p>As a devops engineer you have a lot of infrastructure choices to make. But those decisions almost always depend on what you're trying to optimize for with your product. So let's take a look at the spec</p>	<div data-bbox="824 1558 1572 1978"><h2>Setting the stage</h2><ul style="list-style-type: none">- Congratulations! You got a job at the hottest startup on the block, LooWaterAISoft, as a DevOps engineer!- You are tasked with deploying the NextJS application that the software team has spent all of Q1 '25 coding up.- How would you go about setting up this production infrastructure?<p>4</p></div>

[List spec]

So I actually made an app with nextjs recently and stress tested on a semi average system and I was able to hit 750 ish requests per second without sacrificing usability. But this will always depend on exactly what kind of app you're making, we'll just use this as the number for now, it isn't actually that important.

Ok we have the spec, we know the limits of our app, how do we set this up.

Setting the stage: Spec

Management at LooWaterAISoft has given you minimum requirements for your site rollout

- 500,000 users on the waitlist
- 10,000 users expected to be using the site at a time

5

[cont] The most logical way of doing it would be to set up a set of machines all running the same app and distribute the traffic between them to handle all the number of requests we'll be getting.

Ok, we set up one machine but that can only handle 750 RPS, we'll need more machines, remember, we have half a million customers to manage, so we're going to need a lot of VMs.

[animation run x13]

The problem now is how we set them all up, each machine needs an OS, npm, all npm packages used by the nextjs server and all other dependencies the product team decided we need. We could manually install all of those, but that seems like a chore, which leads us to...

[next slide] the problem...

Setting the stage: Availability



6

[cont] we need multiple identical machines to keep up with demand.

Keeping in mind, that with a massive fleet of these, drift and inconsistencies in our setup are likely to be issues. So how do we solve this?

The problem

We need multiple identical nodes to keep up with demand.

7

That wasn't actually a rhetorical question.
[Crowd work]: Can anyone tell me the industry standard solution for reproducible builds and multi instancancng?
[animation run when answered]

Docker!

[animation run]. [Explain VMs and docker]

- [
- VMs virtualize hardware and run a whole OS under the hood
 - Containers run in a sandbox on top of an engine that interacts with the host OS, you generally don't need to worry about hardware at all with containers
-]

Now that we understand what a docker container is, here's a simple example of a NextJS app packaged with Docker

[animation run]

[explain the lines]

This seems reproducible, we've got a specific node version and everything! What more could we want!

[animation run]

Well, this image isn't nearly as reproducible as one may think,

Can anyone guess why? [crowd work]

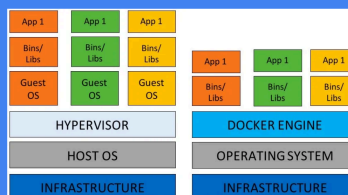
Issue 1: The `node:20-alpine` image is a *tag*, not an immutable identifier. While it points to a specific image *today*, it can be updated in the future with new packages or security patches. This means a `docker build` run today might not produce the exact same result tomorrow.

Issue 2: Packages can be installed differently depending on the system, a specific example that I've encountered was trying to install the tensorflow python package with docker on the CS Club servers. [what is tf, cs club server didn't have avx encoding, build failed]. [opposite can happen, package gets built differently leading to an error]

So one way (and what I have found to be the best way of solving this) is NixOS, and the NixOS ecosystem. At the foundation of which is Nix. THIS image was taken from the nixos wiki and says that nix is a pure and functional build system, but that combination of words seems to trip beginners up sometimes, so we'll start by talking about Nix in it's most barebones form, the language.

The solution?

Docker!



```
FROM node:20-alpine AS deps
WORKDIR /app
COPY package.json package-lock.json
RUN npm install

FROM node:20-alpine AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN npm run build

FROM node:20-alpine AS runner
WORKDIR /app

# Copy only necessary files
COPY --from=builder /app/next.config.ts ./
COPY --from=builder /app/public ./public
COPY --from=builder /app/package.json ./
COPY --from=builder /app/next ./next
COPY --from=builder /app/node_modules ./node_modules

EXPOSE 3000
CMD ["npm", "start"]
```

8

The illusion of reproducibility

Issue 1

```
FROM node:20-alpine AS deps
```

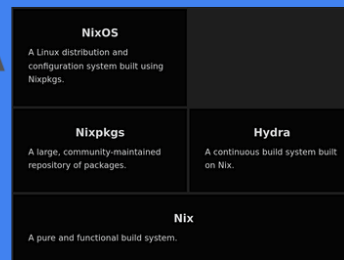
Issue 2

```
COPY package.json package-lock.json
RUN npm install
```

9

The real solution

The NixOS ecosystem stack



10

What's so so special about the nix language. Well, what differentiates it from your Pythons and Javas and C++s is that its a functional programming language, not imperative. What does that mean?

Everything is a function, it's in the name.

Scopes are limited to the function you're in

There is no state, and there are no side effects

[Give an example of side effects in java]

With nix and other functional languages, we can guarantee that given the same inputs, a function will always spit out the same output. There are some hopefully clear benefits of this when it comes to packaging and reproducibility. But I'm getting ahead of myself, let's see the language in action!

Nix

What is Nix?

- A pure and functional programming language

11

Nix language basics

```
2 + 2
# 4

90.89 - 45.9
# 44.99

true || false
# true

true && false
# false

if true then "this is true" else "this is false"
# "this is true"
```

```
text = ["hello" "alt-tab" "!"]
text
# ["hello" "alt-tab" "!"]

attrSet = {name="Waterloo"; faculty="Math";}
attrSet
# {name="Waterloo"; faculty="Math";}

add = a: b: a + b
add 2 2
# 4

add = {a,b}: a + b
add {a=2; b=2;}
# 4
```

12

I changed the definition of nix for simplicity in the last slide but I promised I'd get to explaining what exactly a build system is. A build system is a blueprint that we can follow to create an app, or as NixOS calls them, derivations. A really simple build system would be something like what you do for CS classes, like have your code in a folder, then compile it, then upload it to whatever online grading system your prof is using.

[animation run: show package.nix]

[explain package.nix]

Nix build system/derivations

Click to add test

```
{pkgs ? import <nixpkgs> {}}:
pkgs.stdenv.mkDerivation {
  name = "hello-script";

  src = ./.;

  installPhase = ''
    mkdir -p $out/bin
    cp $src/hello.sh $out/bin/hello
    chmod +x $out/bin/hello
  '';

  ## hello.sh
  ##!/bin/bash
  #echo "Hello AltTab!"
```

13

So, what is nixpkgs. Nixpkgs is just a giant repository of derivations that anyone can access. It's hosted on github repository, so anyone can also contribute to nixpkgs, just with a few caveats and instructions that you'd have to follow.

[animation run]

[Explain number of packages]

[animation run]

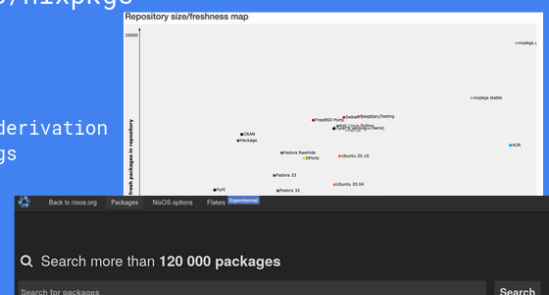
Nixpkgs doesn't only contain packages though, it also contains functions that you can use to make packages. That might seem a little confusing, so let's go into the demo so I can show you what I mean and so we can build the derivation we made in the previous slide!

Derivations/nixpkgs

Another demo!

Recap:

- Creating a derivation using Nixpkgs functions



13

[show the package.nix in vm]
[build package.nix with 'sudo nix-build package.nix']
[explain the nix store]
[animation run]
[recap]

Ok, so we have a derivation, but I want it on my OS, I want every user on my system to be able to access my awesome hello world package and permanently add it to the nix store, how can I tell my OS to run this derivation and add it to my system path. Well, with the open source Nix Operating System, it couldn't be easier. Looks like it's time for another demo
[animation run]
[integrate the package.nix into configuration.nix]
Ok that concludes the my explanation of the nixos stack diagram, hopefully you can see how this OS built around a reproducible system but so far we've been talking about this in the context of a production environment, but there are a couple of other things I want to go over before we call it that may convince you to use it in situations that go beyond the server.

NixOS

Another another demo!

Recap:

- Adding our Nix derivation to our NixOS configuration

14

Have you ever wanted your programming packages and libraries to go away when you're trying to use your system day to day? Have you ever wanted to run different versions of the same package in different codebases?

By creating a shell.nix file you can use the mkShell function included in nixpkgs and specify build inputs you want in a temporary shell. All you need to do run nix-shell in the same directory as the shell.nix file and a temporary shell will be created with your specified dependencies. No more dependency hell. Write it once. Use it forever. You can even share it with your co maintainers to make sure everyone's dev environment will work exactly the same.

Nix shell (mkShell)

- Don't want your build dependencies to clutter up the rest of your system?

```
{pkgs ? import <nixpkgs> {}}:
pkgs.mkShell {
  buildInputs = [
    pkgs.python311
    pkgs.python311Packages.numpy
  ];
}
```

15

Next, home manager. We've all had some bashrc configuration or app theme choice that we have to reconfigure every time we reinstall that specific program or migrate to a new device. Home manager is a nixos module that we can use to declaratively write configuration files that live in our home directory, hence the name.

Home manager

Manage your dotfiles with nix

```
{
  programs.git = {
    enable = true; # allow home manager to mess with git
    userName = "makarab";
    userEmail = "mahdykaram@outlook.com";
    extraConfig.init = {
      defaultBranch = "main"; # tell git to name all default branches main instead of master
    };
  };
}
```

16

The keen eyes among you will have noticed a few flaws with the reproducibility of NixOS. The first is that there are multiple channels of nixpkgs. There's 24.11, the current stable version NixOS unstable, the rolling release model for derivations Nix Darwin, for macOS (yes, the nix package manager can be used on OSX) So if I share my config files with a friend and they're on a different channel, the packages they'll get will be different, this is by design

The second issue is that say my friend and I ARE on the same channel, nixOS unstable for example, if I run nixOS-rebuild switch on January 1st 2025, and my friend runs it on February 1st, the derivation files may have changed in the nixpkgs repository.

Thankfully there are solutions for both these issues, and it's called flakes!

[animation run]

Flakes allow us to pin our nixpkgs and other inputs to a specific commit, making all builds truly reproducible.

[animation run]

Here's an example of the flake.lock file that our flake will create. In fact, this is the flake.lock that my NixOS system produces, the exact commit I'm building from is contained in the narHash attribute

[animation run]

With flakes, the dream of a truly reproducible system becomes a reality, and we can finally report to our superiors at LooWaterAISoft that we've found a way to handle all our future traffic, and spin up new machines with ease, without having to worry about things breaking as we scale.

For the record I haven't even scratched the surface on what you can do with nixOS, this is like maybe 15%. [Add more cool stuff here]

Read the wiki! It's a great resource and has answered basically all the questions I've ever had about NixOS. And if you peek for long enough, you may find some pages that I've authored

The Vimjoyer youtube channel is great, he makes quick and easily digestible videos about nix that are also super fun. In fact his latest video is about how you can set up a reproducible minecraft server with nix, which I thought was pretty cool.

Finally, my config, it doesn't really need to be my config but a lot of nixOS users backup their configs to github so you can learn a lot of some niche uses of nixOS

Flakes

Pin dependencies to a specific commit

```
{ "nodes": {
  "nixpkgs": {
    "locked": {
      "lastModified": 1741114848,
      "narHash": "sha256-8j3yJQ6z42e/ty7s1tHbF4LIqGUYaudgsXg/+qSCUyvg=",
      "owner": "nixos",
      "repo": "nixpkgs",
      "rev": "7611ae8d5db655461da5ffe0e7373d730b0d5f8",
      "type": "github"
    },
    "original": {
      "owner": "nixos",
      "ref": "nixos-unstable-small",
      "repo": "nixpkgs",
      "type": "github"
    }
  }
}
```

17

References

"Nix Ecosystem - NixOS Wiki." Accessed March 23, 2025.
https://wiki.nixos.org/wiki/Nix_ecosystem.
"Server Operating System Market Volume, Share | Analysis, 2032." Accessed March 21, 2025.
<https://www.fortunebusinessinsights.com/server-operating-system-market-106601>.

19