DETERMINISTICALLY BUILT CONTAINERS

OR: HOW I LEARNED TO STOP WORRYING AND LOVE NIX

Morgan Helton

MORGAN HELTON

Software Engineer at Flox

github.com/devusb

linkedin.com/in/mohelton

morgan@flox.dev

MEET THE APPLICATION

```go
package main

import (
    "net/http"
    "os"

    "github.com/labstack/echo/v4"
)

func main() {
    e := echo.New()
    e.GET("/", func(c echo.Context) error {
        hostname, _ := os.Hostname()
        return c.JSON(http.StatusOK, map[string]string{
            "hostname": hostname,
            "message":  "hello from nix",
        })
    })
    e.Logger.Fatal(e.Start(":8080"))
}
```

OCI IMAGE ANATOMY

DOCKERFILE

```
1 FROM ubuntu:24.04
2 RUN apt update && apt install -y golang-go
3 WORKDIR /src
4 COPY . .
5 RUN go build -o /app .
6 ENTRYPOINT ["/app"]
```

LAYERS

- Every statement produces a **layer**
- Layers are **ordered** — change one, rebuild everything after it
- Layers are **additive** — you can't truly delete from a previous one

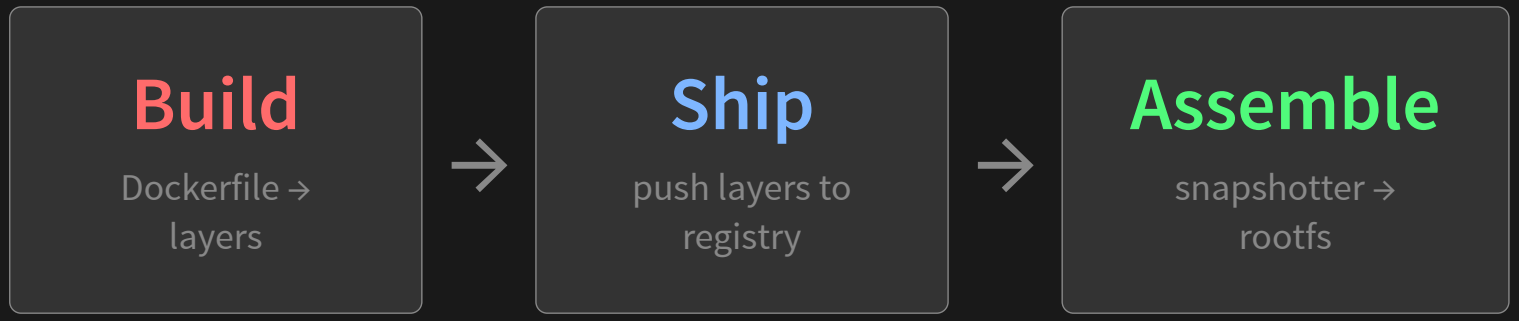
```
RUN apt update && apt install -y golang-go
RUN rm -rf /var/lib/apt/lists/*   # too late — it's in the previous layer
```

WHAT COMES ALONG FOR THE RIDE

- **1.1 GB** — build tools, package manager can end up in the final image
- **Not reproducible** — apt repos drift, tags are mutable
- Dockerfiles **don't compose** — copy-paste between projects

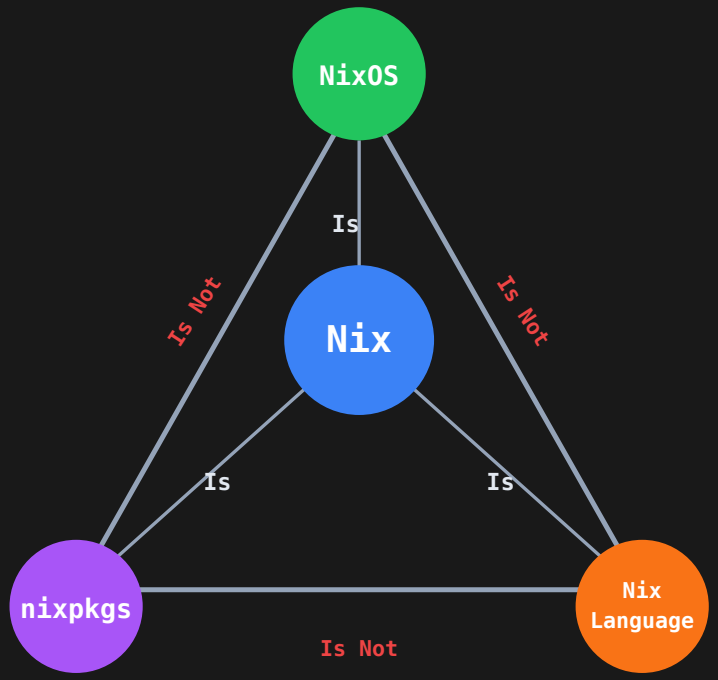
multi-stage can work around the size problem, but not the rest

CONTAINER CREATION



NIX BUILDS

WHAT IS NIX?



- **Nix** — a cross-platform package manager where every package is defined by its inputs
- **Nix Language** — the language Nix uses to describe how software is built
- **nixpkgs** — 120,000+ packages in a single monorepo
- **NixOS** — an operating system composed from nixpkgs, immutable by design

PACKAGING IN NIX

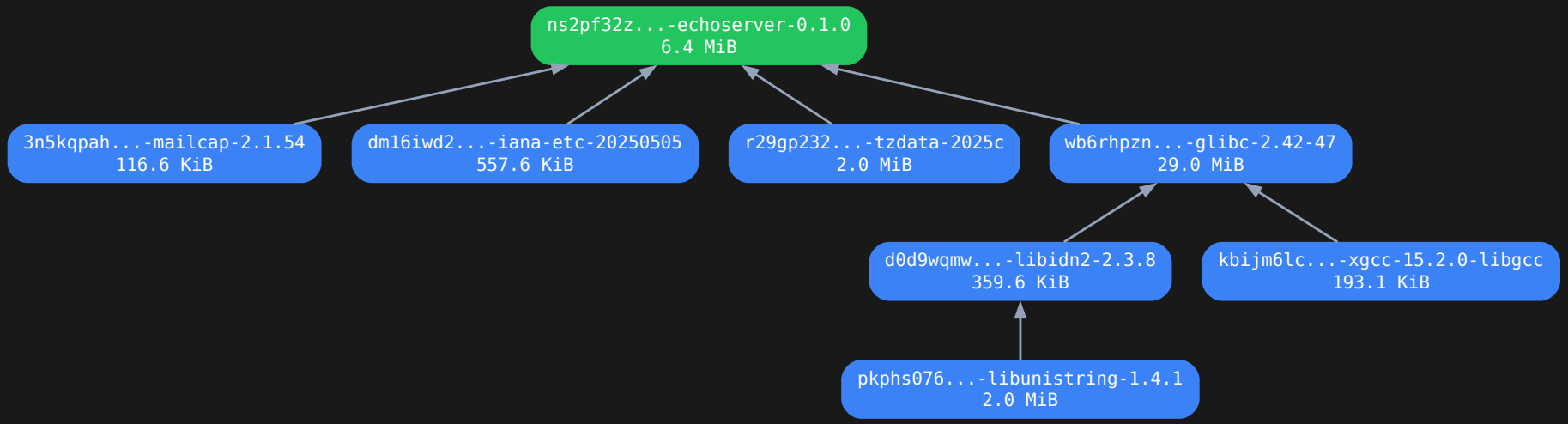
```
{ buildGoModule }:

buildGoModule {
  pname = "echoserver";
  version = "0.1.0";

  src = ./.;

  vendorHash = "sha256-JyOFhN98a5ycwKk21ey...";
}
```

STORE PATHS

```
/nix/store/ns2pf32zq4bhsf4ijv4jzid6rjfd8a5l-echoserver-0.1.0
                                          └── bin/
                                              └── echoserver (6.4 MiB)
```

- The hash `ns2pf32z...` is a **function of all inputs**
- Change any input → different hash (source, dependency, LDFLAGS)

DEPENDENCY GRAPH



Total closure: 40.6 MiB — 8 packages, every one input-addressed

WHAT THIS MEANS

- **Sandboxed build** with explicit inputs
- **Reproducible** — come back in 1, 5, 10 years, reconstruct it
- **Complete bill of materials** — the graph IS the SBOM
- **Portable** — all you need is the description + access to sources

TAKE THAT CLOSURE ANYWHERE

- **OCI image** — `dockerTools.buildImage`, `nix2container`
- **VM image** — NixOS configuration → QCOW2, VHD
- **AMI** — ship directly to AWS

YOU CAN STOP HERE

WHAT IF YOU DIDN'T HAVE TO?

JUST USE THE STORE PATHS

- The container filesystem (rootfs) is just files
- A Nix closure already describes **exactly which files** are needed
- Hand the store paths to the container runtime directly

THREE WAYS TO GET THERE

- **Flox** — mount store paths into the container at runtime
- **nix-csi** — mount the closure as a volume
- **nix-snapshotter** — replace layer unpacking

FLOX

WHAT IS FLOX?

- **Package and environment manager** — packages, env vars, services in a TOML manifest
- Familiar package manager ergonomics — backed by **Nix under the hood**
- Works on macOS and Linux, x86 and Arm
- No Nix language required

A FLOX DEV ENVIRONMENT

SHELL

```
$ go version
go: command not found
$ flox init
$ flox install go gopls delve golangci-lint

$ flox activate
flox [echoserver] $

flox [echoserver] $ which go
/nix/store/...-go-1.25.5/bin/go

flox [echoserver] $ go version
go version go1.25.5 linux/amd64
```

MANIFEST.TOML

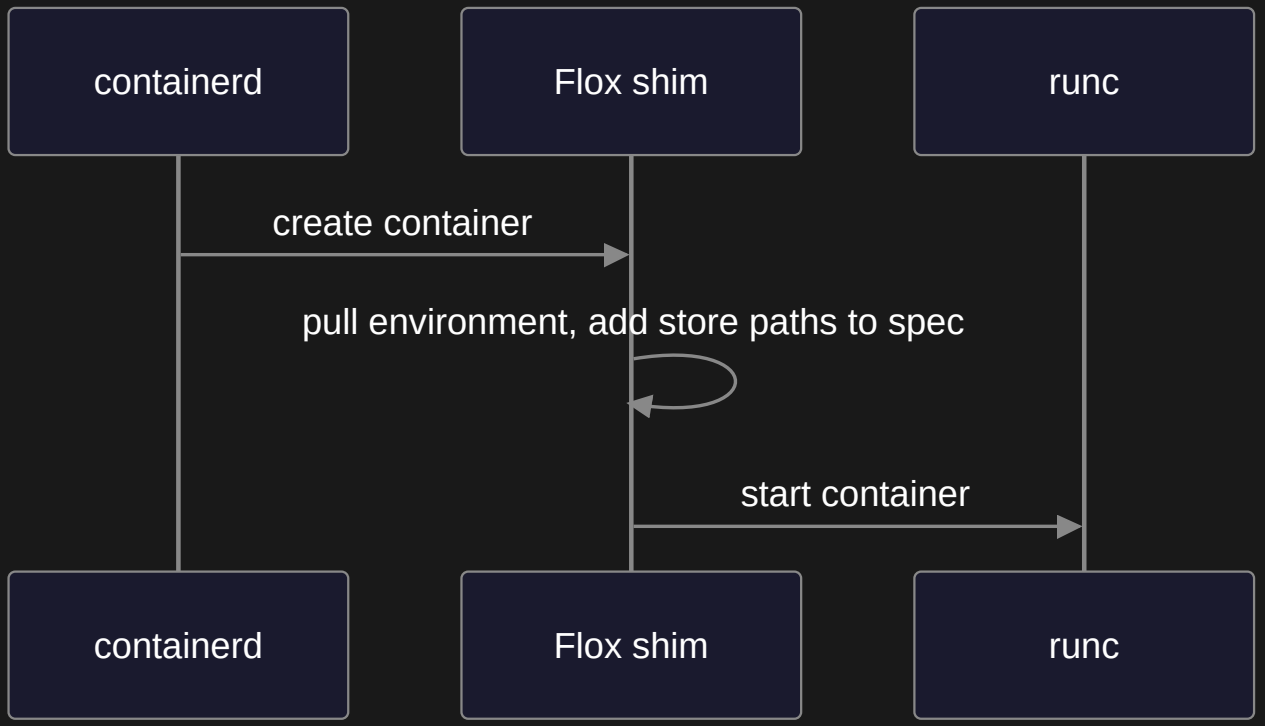
```
version = 1

[install]
go.pkg-path = "go"
gopls.pkg-path = "gopls"
delve.pkg-path = "delve"
golangci-lint.pkg-path = "golangci-lint"
```

FLOX ENVIRONMENT → CONTAINER

- Environment contains **just your app**
- **Declarative and composable**
- **Runtime shim** materializes it on the node

```
version = 1
[install]
echoserver.pkg-path = "devusb/echoserver"
```

SHIM OPERATION



FLOX IN PRACTICE

MANIFEST.TOML

```
version = 1

[install]
echoserver.pkg-path = "devusb/echoserver"
```

POD.YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: echoserver
  annotations:
    flox.dev/environment: "devusb/echoserver"
spec:
  runtimeClassName: flox
  containers:
    - name: echoserver
      image: flox/empty:latest
      command: ["echoserver"]
```

NIX-CSI

WHAT IS A CSI DRIVER?

- **Container Storage Interface** — standard K8s volume plugin
- Examples: EBS, GCP persistent disks, Longhorn
- Reference it in a pod spec, K8s handles the rest

NIX-CSI

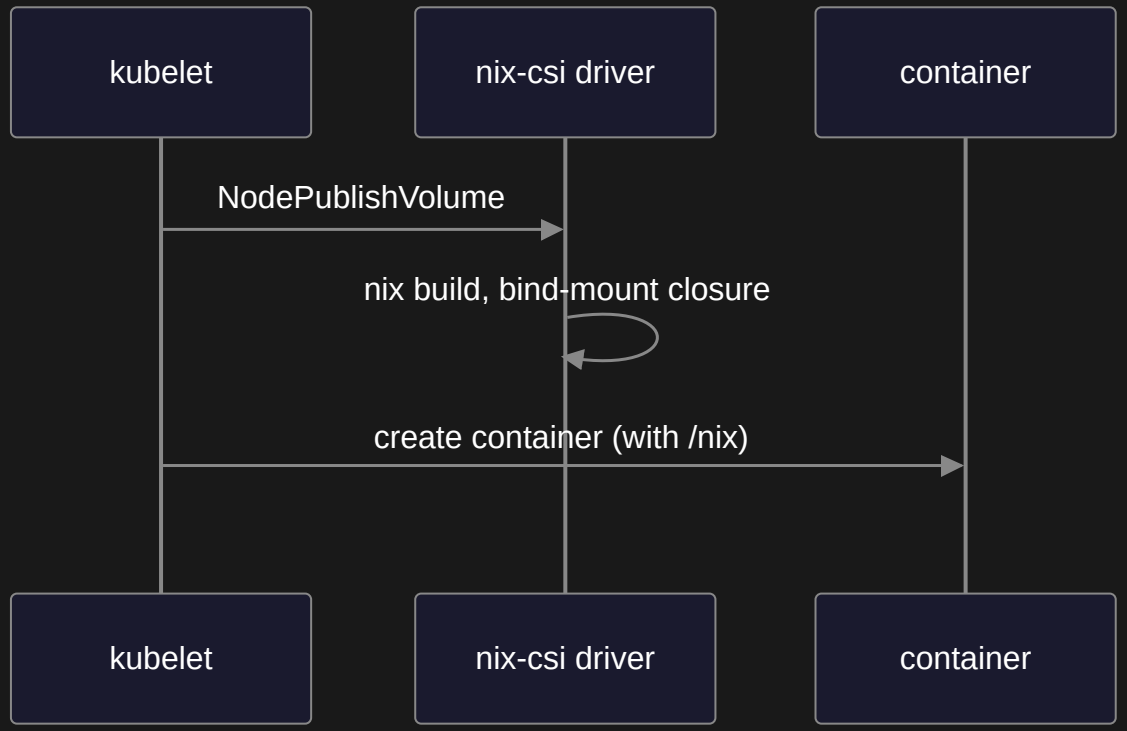
by @Lillecarl

- A CSI driver that creates **ephemeral volumes from Nix packages**
- Takes a package reference, store paths, or a Nix expression
- Builds the full closure and mounts it at `/nix`

WHAT YOU GET

- Just a **volume** — standard K8s primitive
- Runs on **any K8s cluster** — managed, self-hosted, whatever
- Includes a **cluster-level cache** — store paths shared across nodes

VOLUME CREATION



NIX-CSI POD SPEC

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: echoserver
spec:
  containers:
    - name: echoserver
      image: gcr.io/distroless/static:latest
      command: ["echoserver"]
      volumeMounts:
        - name: nix
          mountPath: /nix
          subPath: nix
  volumes:
    - name: nix
      csi:
        driver: nix.csi.store
        volumeAttributes:
          flakeRef: "github:devusb/echoserver"
```

TRADE-OFFS

- **Pure Nix** — you need to know Nix to define what gets mounted
- Packages live in `/nix`, not `/usr/bin`
- Different mechanism: volume mount vs. runtime filesystem composition

NIX-SNAPSHOTTER

A CONTAINERD SNAPSHOTTER PLUGIN

- Replaces the layer-unpacking step in containerd
- Fetches packages from a **Nix binary cache** instead of a container registry
- Each package becomes a **bind mount** — no 128-layer limit

WHY THIS ONE?

- Handles **both Nix and regular images** — passes non-Nix images through to containerd
- Can **start from an existing base image** and add Nix packages on top
- Or go pure Nix — your choice per container

NIX-SNAPSHOTTER POD SPEC

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: echoserver
spec:
  containers:
    - name: echoserver
      image: "nix:0/nix/store/...-nix-image-echoserver.tar"
      command: ["echoserver"]
```

CONSIDERATIONS

- Plugin must be installed at the **node level**, which can be difficult on some managed Kubernetes distributions
- More Nix knowledge needed than Flox, similar to nix-csi

PICKING YOUR PATH

WHEN WOULD YOU PICK EACH?

- **Flox** — you don't want to learn Nix, you control your nodes
- **nix-csi** — you want managed K8s portability
- **nix-snapshotter** — you need to mix OCI and Nix in the same container

SHIP LIGHTER

- Nix gives you **complete knowledge** of what your app needs
- Every dependency is **input-addressed** — same inputs, same output, every time
- That's a **complete bill of materials** for free
- For heavy workloads, skip the image — **reuse the packages directly**

RESOURCES

- **Flox K8s** — docs.flox.dev/docs/k8s/intro
- **nix-csi** — github.com/Lillecarl/nix-csi
- **nix-snapshotter** — github.com/pdtpartners/nix-snapshotter
- **Learn Nix** — fzakaria.com/2024/07/05/learn-nix-the-fun-way.html