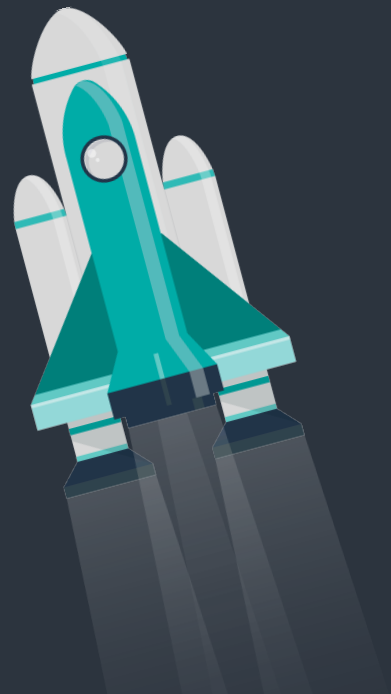
Nix for Monorepos

Niklas Korz

About Me

- Website: https://korz.dev
- Co-founder and tech lead at alugha
- Rustacean
- Nix journey started in December 2022
- Runs NixOS on servers, desktops and Raspberry Pis

1. Motivation
2. Building and Caching
3. Containers and Deployments
4. Further Reading

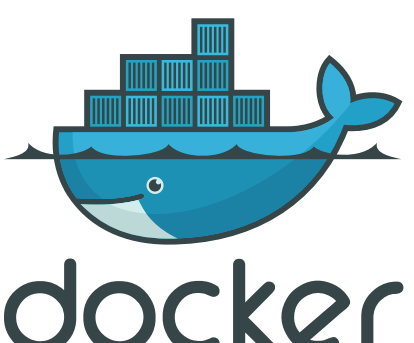


Why Monorepos?

- Coordination of multi-service changes
- Shared dependencies without hosting a registry
- Single source for deployments
- All is well, if it works…

CI at alugha: pre 2023

- ~1000 lines of Gitlab CI YAML
  - Path-based change filters for rebuilding services
  - Dependency caching based on lockfile hashes
- 17 Dockerfiles (217 lines)
  - One Dockerfile per service
  - Some base images for layer sharing



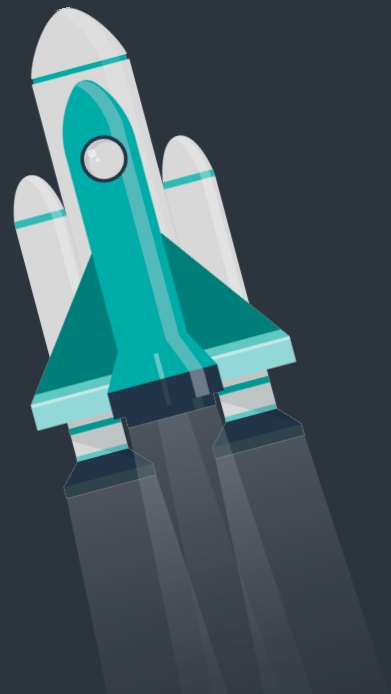
Enter: Nix

- Declarative package manager
- Functional configuration language
- Also a build tool
- Nixpkgs: more than 60′000 up to date packages
  - Debian & Ubuntu: ~20′000
  - Arch User Repository (AUR): ~25′000

CI at alugha: 2023+

- ~1000 lines of Nix (excluding auto-generated files)
- 641 lines of Python scripts
- 86 lines of Gitea Actions YAML
- Fine-grained build cache
- No more hacky change detection

1. Motivation
2. **Building and Caching**
3. Containers and Deployments
4. Further Reading



A basic Nix derivation

```
my-cpp-program = pkgs.stdenv.mkDerivation {
  name = "my-cpp-program";
  src = ./src;
  buildPhase = "c++ -o my-cpp-program main.cpp";
  installPhase = ''
    mkdir -p $out/bin
    cp my-cpp-program $out/bin/
  '';
};
```

A basic Nix derivation

```
→ nix build .#my-cpp-program
evaluating derivation '.#packages.aarch64-darwin.my-cpp-
program'
[1/1 built]

→ file result/bin/my-cpp-program
result/bin/my-cpp-program: Mach-O 64-bit executable arm64
```

Building Go programs with Nix

```
my-go-program = pkgs.buildGoModule {
  pname = "my-go-program";
  version = "0.5.0";
  src = ./my-go-src;
  buildInputs = with pkgs; [ olm libsignal-ffi ];
  vendorHash = "sha256-sa6M9rMrI7fa8!!...";
};
```

Building Rust libraries with Nix

```
libsignal-ffi = pkgs.rustPlatform.buildRustPackage {
  pname = "libsignal-ffi";
  version = "0.39.2";
  src = ./libsignal-ffi-src;
  cargoLock.lockFile = libsignal-ffi-src/Cargo.lock;
  cargoBuildFlags = [ "-p" "libsignal-ffi" ];
};
```

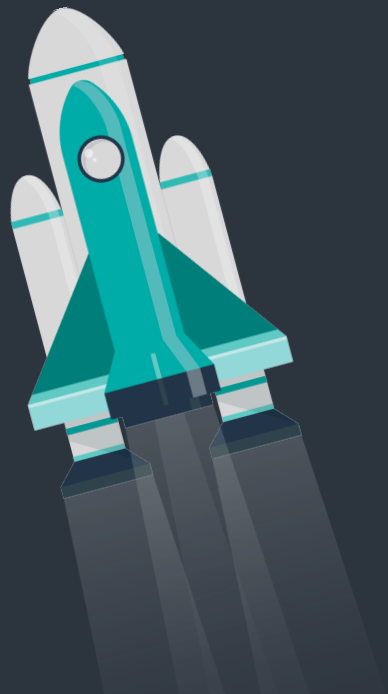
cargo2nix

```
let rustPkgs = pkgs.rustBuilder.makePackageSet {
  rustVersion = "latest";
  packageFun = import ./Cargo.nix;
}; in {
  manifests = rustPkgs.workspace.manifests { };
  proxypress = rustPkgs.workspace.proxypress { };
  traffic-tracker = rustPkgs.workspace.traffic-tracker { };
}
```

How Nix improved our build process

- Batteries included for all popular languages
- Bring your own abstractions, or the community's
- Push to and pull from binary caches
- Language-agnostic dependency graph

1. Motivation
2. Building and Caching
3. **Containers and Deployments**
4. Further Reading



Our pipeline: a summary

1. Check Nix cache status of current commit's images
2. Rebuild uncached images
3. Push newly built images to container registry
4. Deploy Kubernetes manifests with new image tags

build-and-deploy
Erfolg

Set up job 1s

Check out repository code 4s

Build 6m29s

```
1  Logging into attic
3  Configuring attic cache
8
9  Found: alugha2go, attic, auth, credits, events, extract-data, goserver, graphql-server, import-data, kju, maint
   enance-cron, manifests, manifests-go, migration, nginx, nocontent, notification-cron, payment-cron, prisma-migr
   ations, proxypress, public-api, rss, sitemaps, skynet, traffic-tracker, usersync, webhooks
10 Missing: web
11
12 Building package web
18 Uploading image code.alugha.dev/alugha/web:r65jhn5s5xld5zshg4qi4m4p2lgpiizw
19
20 Deploying: web
21 Setting mapping
22 Installing/upgrading Helm release
23 namespace:        "alugha"
24 release name:     "review-project-663-web"
25 chart directory:  "charts/web"
26 registry image:   "code.alugha.dev/alugha/web"
27 version:          "r65jhn5s5xld5zshg4qi4m4p2lgpiizw"
```

Complete job 0s

Dockerfile

```
FROM golang:1.21
COPY my-go-src .
RUN go mod download
# TODO: How do we install olm and libsignal-ffi? 🙀
RUN go build -o /bin/my-go-program .
ENTRYPOINT [ "/bin/my-go-program" ]
```

Nixpkgs dockerTools

```
pkgs.dockerTools.buildImage {
  name = "my-go-image";
  tag = "latest";
  config.Entrypoint = [
    "${my-go-program}/bin/my-go-program"
  ];
}
```

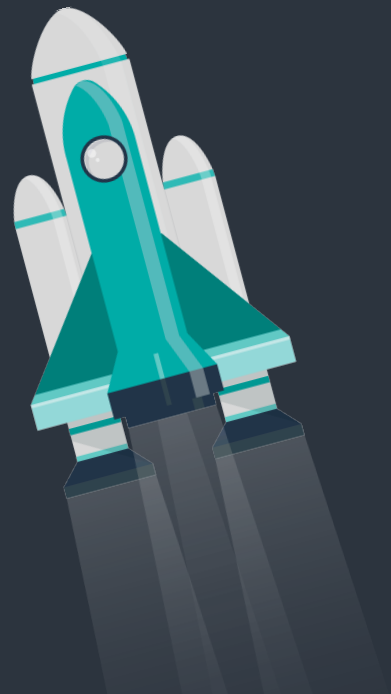
nix2container

```
nix2container.buildImage {
  name = "my-go-image";
  config.Entrypoint = [
    "${my-go-program}/bin/my-go-program"
  ];
  layers = [
    (nix2container.buildLayer { deps = [ glibc ]; })
    (nix2container.buildLayer { deps = [ libsignal-ffi ]; })
  ];
}
```

Summary

- Nix lets you abstract many parts of your build pipeline
- Language-independent dependency graph
- OCI images connect the Nix and Kubernetes world
- Separation of concerns: container images don’t care how packages are built

1. Motivation
2. Building and Caching
3. Containers and Deployments
4. **Further Reading**



Further Reading

- https://nix.dev
- Tor Hovland: Building Nix flakes from Rust workspaces
- Adam Hoese: Announcing Gomod2nix
- Luc Perksins: Deploying Nix-built containers to Kubernetes
- PDT Partners: nix-snapshotter

Further Reading

- Xe Iaso: Nix is a better Docker image builder than Docker's image builder
- Peter Kolloch: Nix & Docker: Layer explicitly without duplicate packages!
- Hopefully soon on my blog (korz.dev): Nix for Monorepos

Questions?

- Download: https://dl.korz.dev/nixda-monorepos.pdf
- Contact me on…
  - Matrix: @niklaskorz:korz.dev
  - Mastodon: @niklaskorz@rheinneckar.social
  - Email: contact@korz.dev