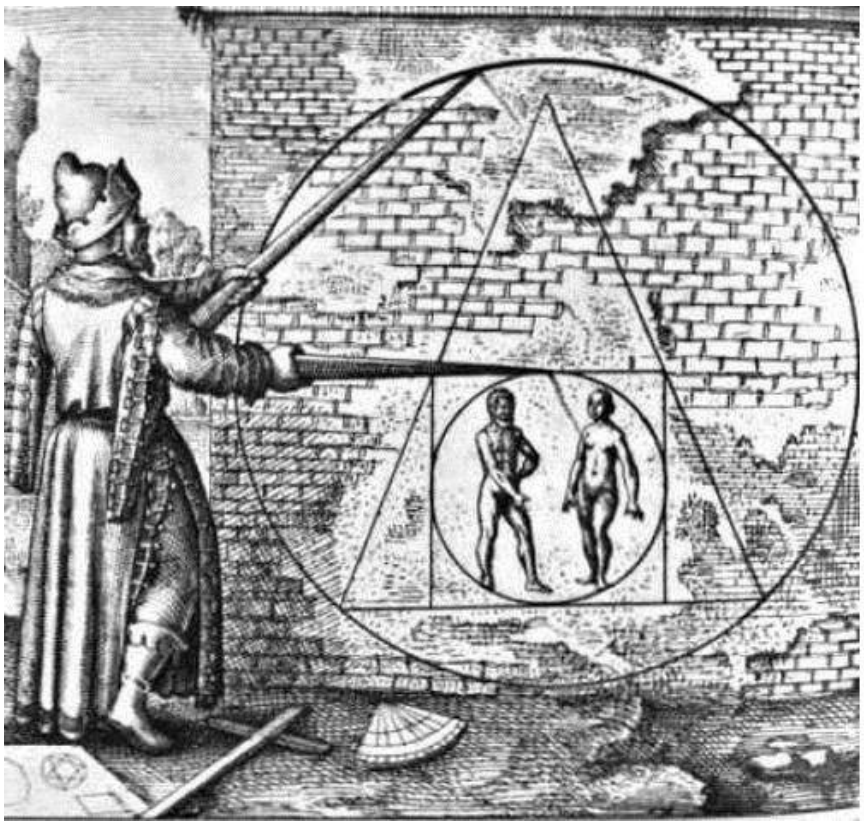
Philosopher's Stone NixOS Demo

Jacek Galowicz

Content

- Philosopher's Stone Certified Image Builds
- NixOS Philosopher's Stone System Image Content
  - System Content High Level
  - System Content package view
  - Build Variants
- Integration Tests
- Demo Session

Philosopher's Stone



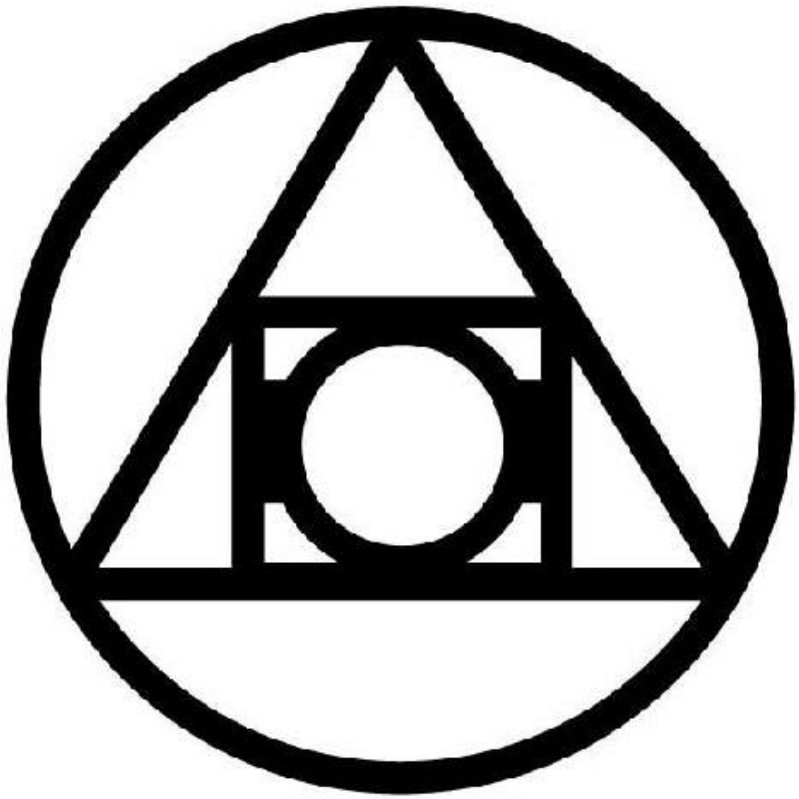


Figure 1: Philosopher's Stone and the Hermetic Seal of Light

Philosopher's Stone Certified Image Builds

BSI imposes **integrity** requirements on product/image builds

This project is a demo on how to build such images with nix/NixOS

Typical Secure Network Product Requirements

- Bootable Image(s) with preconfigured Linux system(s)
- Pre-installed selection of packages
- Pre-configured services
- Minimized build
- Some packages are built...
  - from local source
  - from remote source with local **patches**
- Build flow shall be both reproducible and fast
- **Offline** build capability (integrity requirement)
  - must be *exportable* for evaluating parties

What is NixOS?

- FOSS GNU/Linux Distribution
- Toolbox for building all kinds of system images
- Focus on **reproducible** builds and deployments
- Declarative package *and* system configuration
- Hybrid source/binary packaging mechanism
- Atomic system deployment, upgrade, rollback
- Simple toolchain maintenance included
- Hermetic builds *per package*
- Independence from complex Single-Point-of-Truth CIs

Figure 2: Official NixOS Logo

Why NixOS? (1)

R$\frac{13}{\text{EPRODUCIBILIT}}$Y: NixOS

Is NixOS Reproducible?

Tracking: `nixos-unstable`'s `iso_minimal` job for `x86_64-linux`.

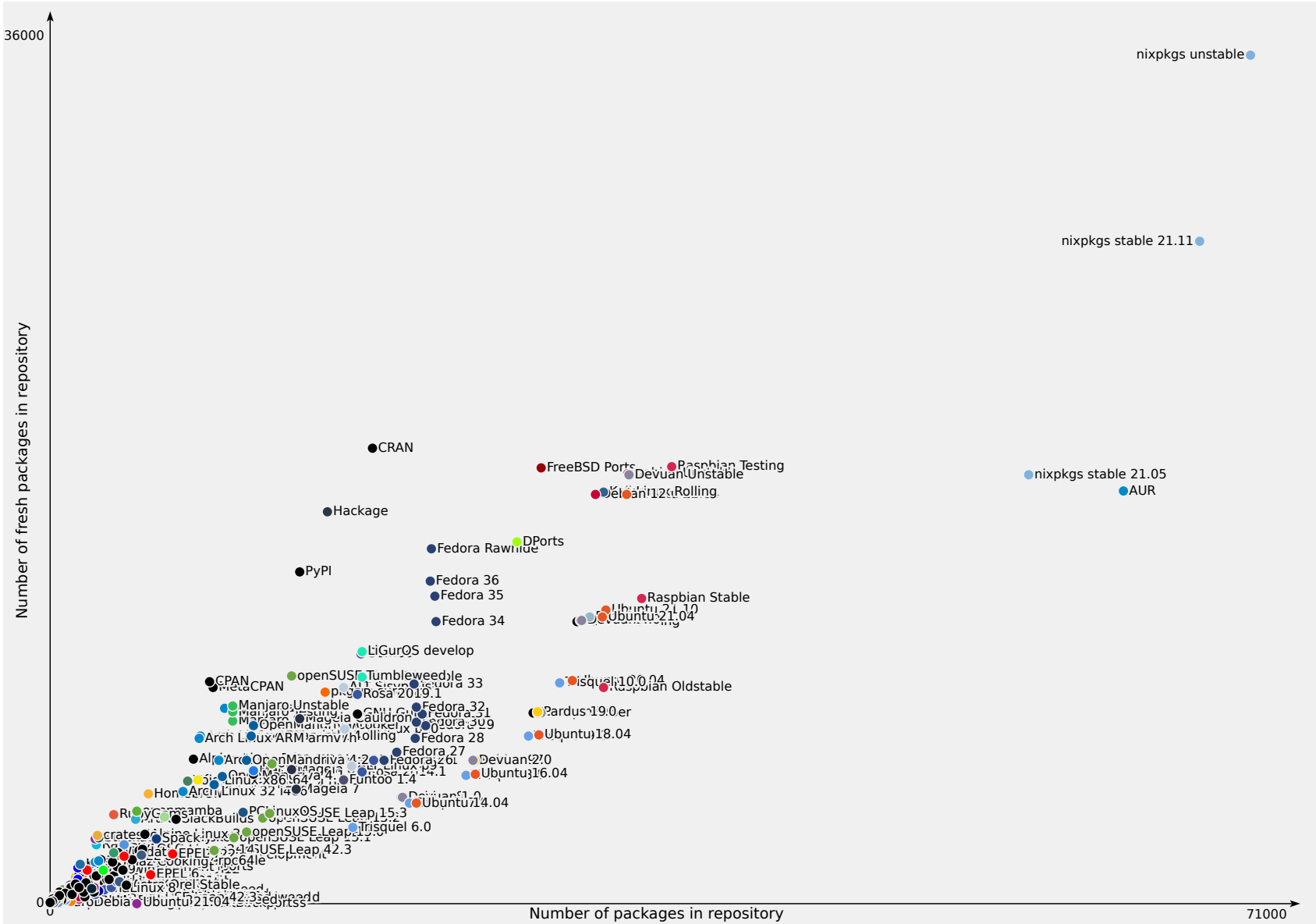
Build via:

```
git clone https://github.com/nixos/nixpkgs.git
cd nixpkgs
git checkout 42c2003e5a0c21b1222e2e17f95c2cc926852ebe
nix-build ./nixos/release-combined.nix -A nixos.iso_minimal.x86_64-linux
```

1570 out of 1572 (99.87%) paths in the minimal installation image are reproducible!

(Website screenshot is from 2022-03-24)

Why NixOS? (2)



(Graph is from 2022-03-24)

What are we going to build with it?

Example System: High-Level View

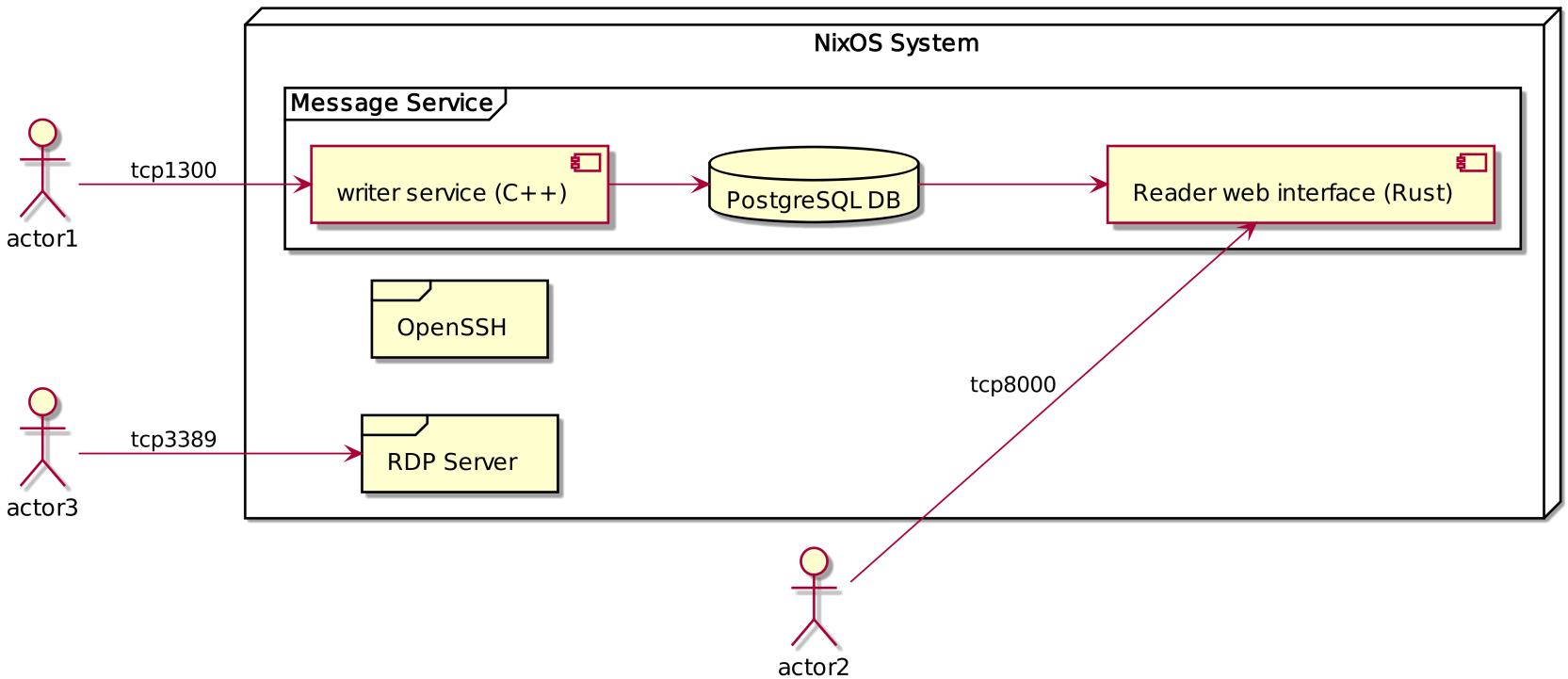


Figure 3: NixOS system running *some* example services

Example System: Package View

Examples for how to *package* and *configure* a custom application service

- Message Server Writer
  - C++ app
  - Listens on port 1300, waits for messages
- Message Server Reader
  - Rust app
  - HTTP Service, listens on port 8000, prints messages

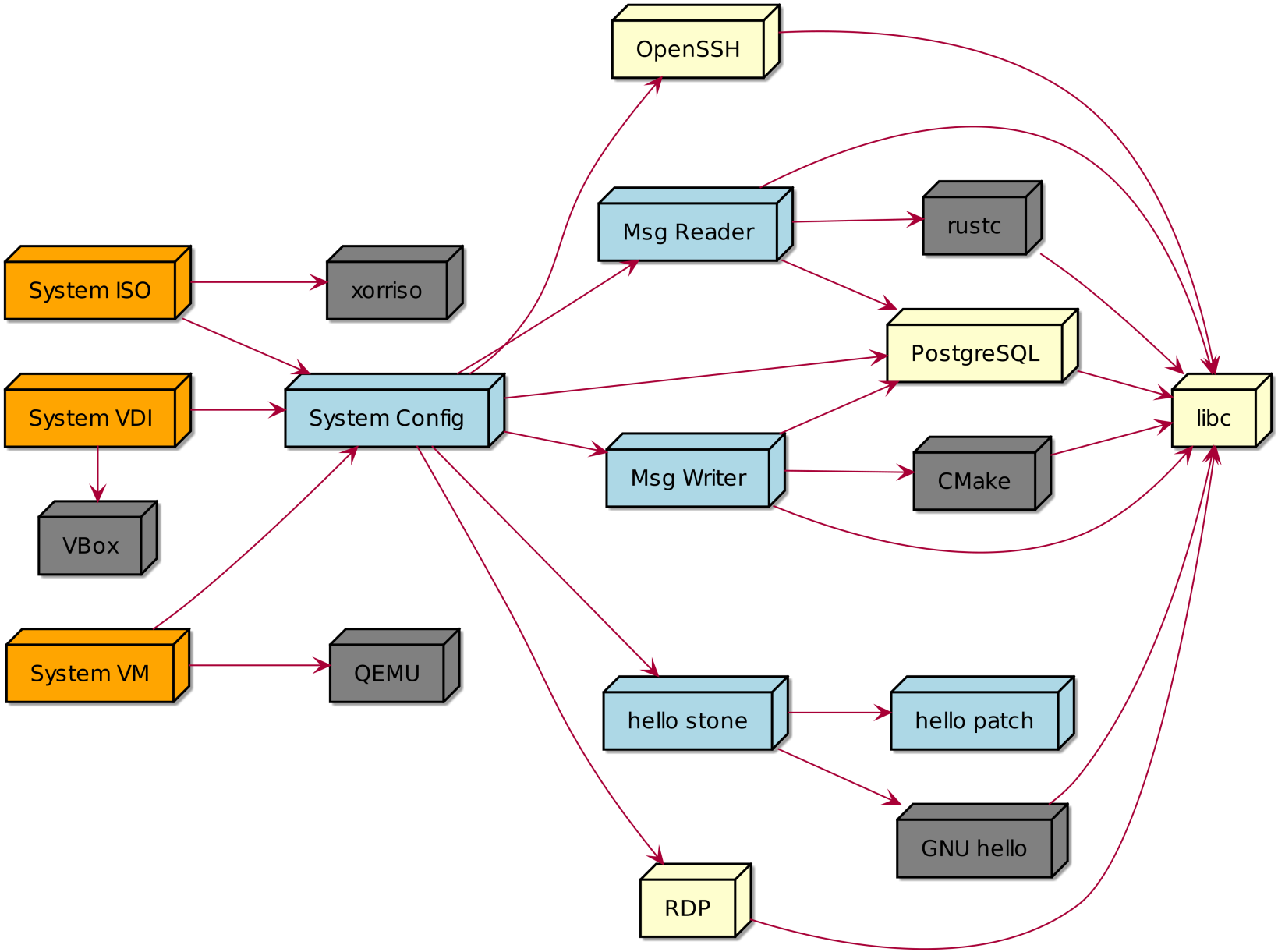
minimal demo:

```
[stone@nixos:~]$ echo -n "hello world" | nc localhost 1300
ok
[stone@nixos:~]$ curl localhost:8000
2021-08-11: hello world
```

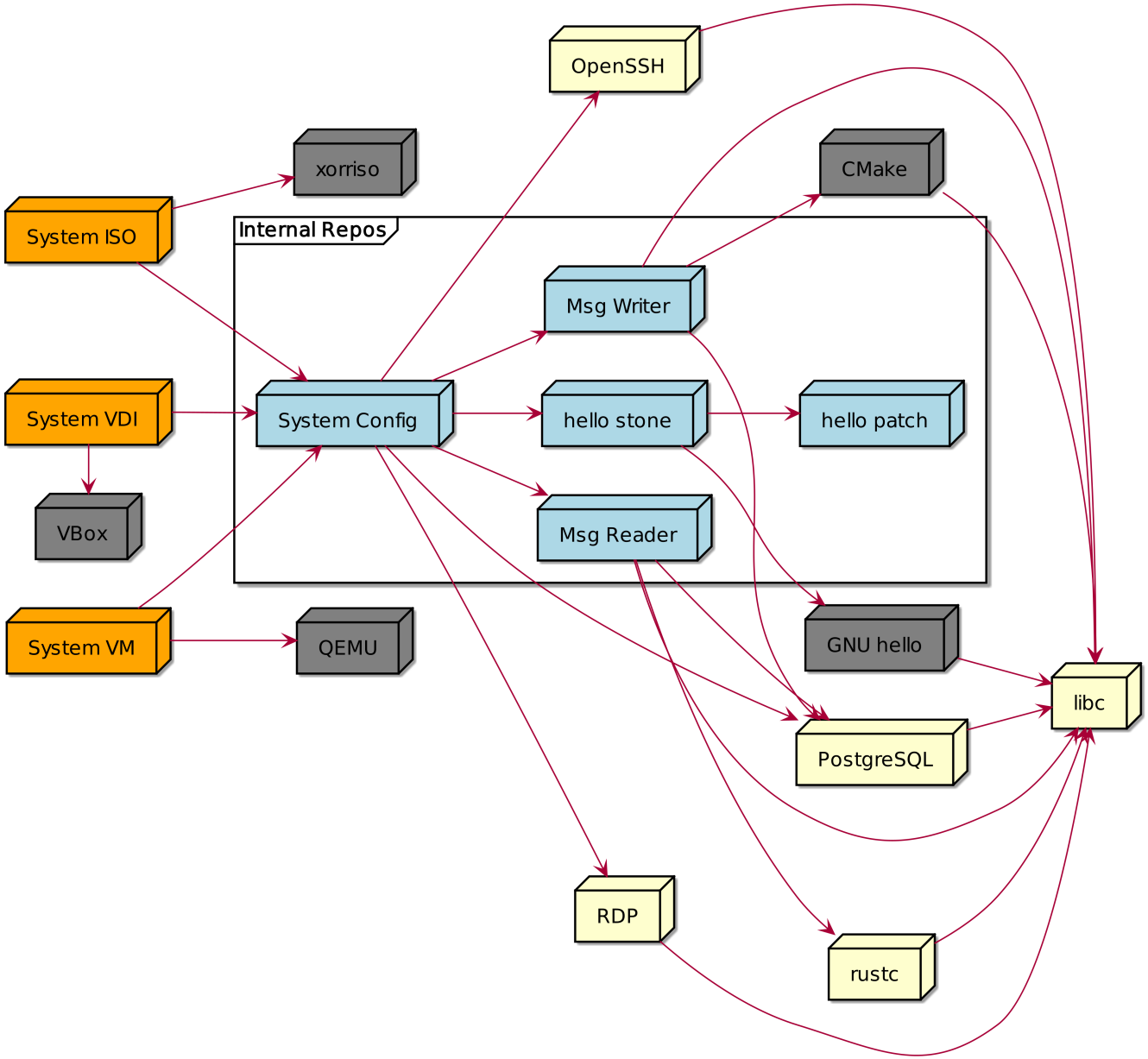
Build Variants

- System config is *composable*
- Any system config can be transformed into different builds:
  - bootable ISO
    - live system
    - installer
  - runnable *shallow* NixOS-VM
  - integration test
  - Virtualbox VDI, Amazon AMI, Google Cloud Image, Azure, ...

Dependency Graph



Dependency Graph Mapped to Company Structures

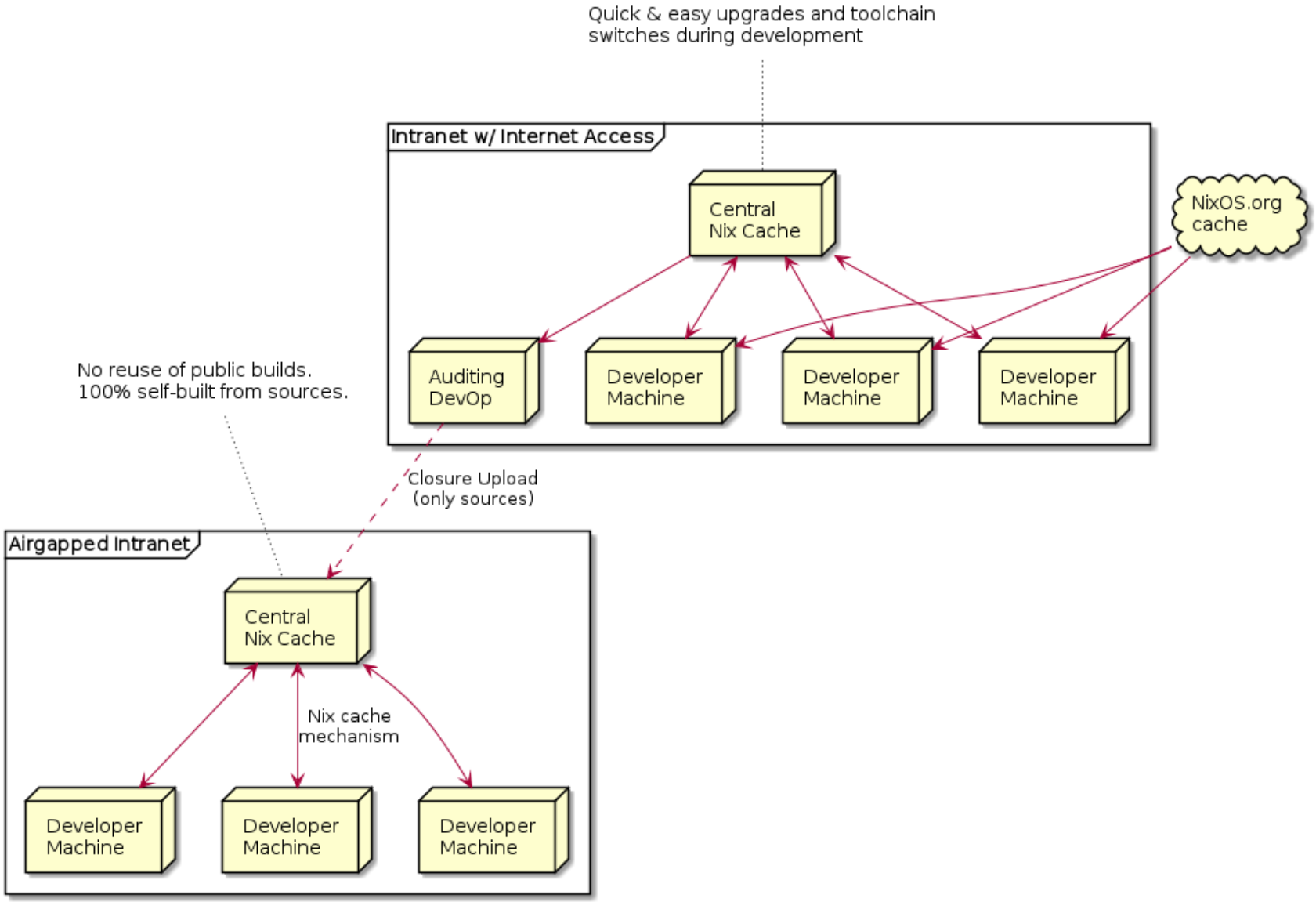


Demo Session

Bottom-up: How to...

- build the message-service packages
  - developer workflow
  - packaging workflow
- patch an external package and repackage it
- define a NixOS system image
- build multiple image configuration $\times$ variants
- integration test a running service in a VM
- rebuild the whole thing on an air-gapped system

Integrating Nix Caches w/ Integrity Requirements



References from Demo Session

C++ and Cartesian build product variants:

https://blog.galowicz.de/2019/04/17/tutorial_nix_cpp_setup
https://blog.galowicz.de/2018/02/27/managing_libraries_with_nix
https://github.com/tfc/nix_cmake_example/

Nix(OS) Documentation:

- https://nixos.org/manual/nixos/stable
- https://nixos.org/manual/nix/stable
- https://nixos.org/manual/nixpkgs/stable

NixOS Wiki: https://nixos.wiki/

Nix.dev Community Tutorials: https://nix.dev/

Nix Overlays: https://nixos.wiki/wiki/Overlays

Summary

The code on github:

https://github.com/tfc/philosophers-stone-nixos