

Painless Transposition of Reproducible Distributed  
Environments with NixOS Compose  
OAR-BATSIM-NXC REGALE Workshop 2023

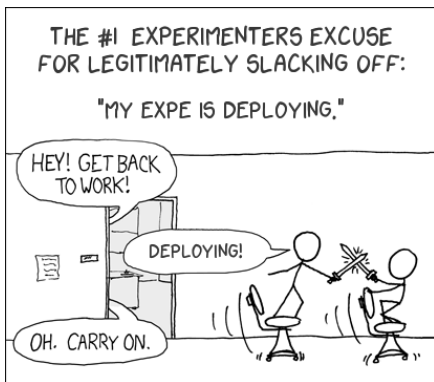
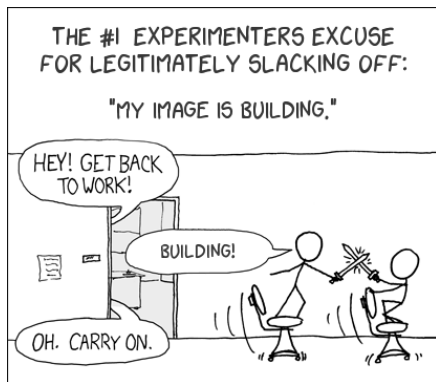
Quentin GUILLOTEAU, Jonathan BLEUZEN, Millian POQUET,  
Olivier RICHARD

Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG

2023-02-24

# Motivation

Setting up Distributed Environments for Distributed Experiments  
 ↪ **Difficult, Time-consuming** and **Iterative** process



⇒ **Does not encourage good reproducibility practices**

# The Reproducibility Problem

## Different Levels of Reproducibility

- 1 Repetition:** Run exact same experiment
- 2 Replication:** Run experiment with different parameters
- 3 Variation:** Run experiment with different environment

↔ **Share the experimental environment and how to build/modify it**

## How to share a Software Environment in HPC?

- Containers? ~> need Dockerfile to rebuild/modify.  
↔ might not be repro (e.g., apt update, curl, commit)
- Modules? ~> cluster dependent. how to modify?
- Spack? ~> share through modules...

# Nix and NixOS

## The Nix Package Manager

- Functional Package Manager
- Packages are functions
  - Inputs = dependencies
  - Body of function = how to build
- No side-effect
- ( $\simeq$ ) Solves Dependencies Hell
- Reproducible by design



## The NixOS Linux Distribution

- Based on Nix
- Declarative approach
- Complete description of the system (kernel, services, pkgs)

# How to store the packages?

## Usual approach: Merge them all

- Conflicts
- PATH=/usr/bin

```

/usr
├── bin
│   └── myprogram
└── lib
    ├── libc.so
    └── libmylib.so
  
```

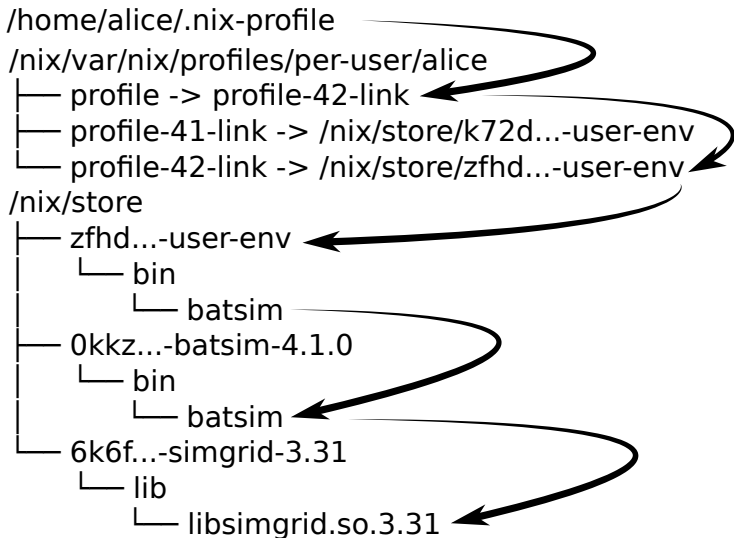
## Nix approach: Keep them separated

- + Pkg variation
- + Isolated
- + Well def. PATH
- + Read-only

```

/nix/store
├── y9zg6ryffgc5c9y67fcmfdkyyiivjzpj-glibc-2.27
│   └── lib
│       └── libc.so
└── nc5qbagn3wqfg2lv1gwj3r3bn88dpqr8-mypkg-0.1.0
    ├── bin
    │   └── myprogram
    ├── lib
    │   └── libmylib.so
  
```

## Nix Profiles





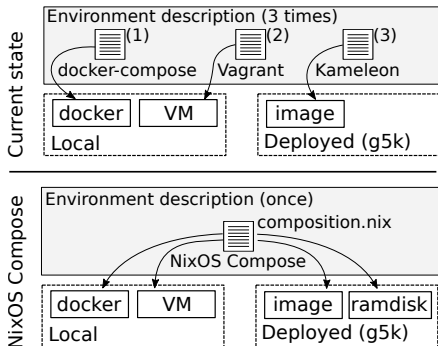
# NixOS Compose - Introduction

## Goal

Use **Nix(OS)** to reduce friction for the development of **reproducible distributed environments**

## The Tool

- Python + Nix ( $\approx 4000$  l.o.c.)
- One Definition  
↔ Multiple Platforms
- Build and Deploy
- **Reproducible by design**





# NixOS Compose - Terminology

## Transposition

Capacity to deploy a **uniquely defined environment** on several platforms of different natures (flavours, see later).

## Role

**Type of configuration** associated with the mission of a node.

Example: One Server and several Clients.

## Composition

Nix expression describing the NixOS **configuration of every role** in the environment.

## NixOS Compose - Composition Example: K3S

```

1 { pkgs, ... }:
2 let k3sToken = "df54383b5659b9280aa1e73e60ef78fc";
3 in {
4   nodes = {
5     server = { pkgs, ... }: {
6       environment.systemPackages = with pkgs; [
7         k3s gzip
8       ];
9       networking.firewall.allowedTCPPorts = [
10        6443
11      ];
12      services.k3s = {
13        enable = true;
14        role = "server";
15        package = pkgs.k3s;
16        extraFlags = "--agent-token ${k3sToken}";
17      };
18    };
19    agent = { pkgs, ... }: {
20      environment.systemPackages = with pkgs; [
21        k3s gzip
22      ];
23      services.k3s = {
24        enable = true;
25        role = "agent";
26        serverAddr = "https://server:6443";
27        token = k3sToken;
28      };
29    };
30  };
31 }

```

Diagram illustrating the NixOS Compose configuration for K3S, showing the role and associated packages, ports, and services.

The configuration is structured as follows:

- Role:** The role is defined as "server" for the server node and "agent" for the agent node.
- Packages:** The packages installed are k3s and gzip.
- Ports:** The allowed TCP ports are 6443.
- Services:** The services are k3s, which is enabled and configured with the role, package, and extra flags.

# NixOS Compose - Flavours = Target Platform + Variant

`docker` - local and virtual

Generate a `docker-compose` configuration.

`vm-ramdisk` - local and virtual

In memory QEMU Virtual Machines.

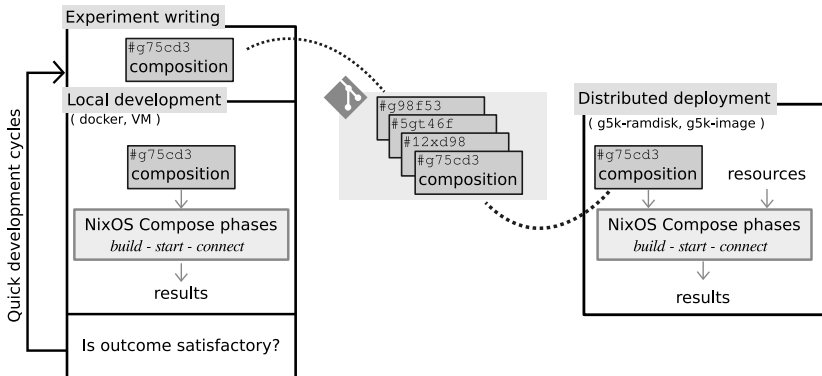
`g5k-ramdisk` - distributed and physical

`initrds` deployed in memory without reboot on G5K (via `kexec`).

`g5k-image` - distributed and physical

Full system tarball images on G5K via `Kadeploy`.

# NixOS Compose - Workflow



# NixOS Compose - Workflow for experiment setup

## Step 1: Setting up the environment

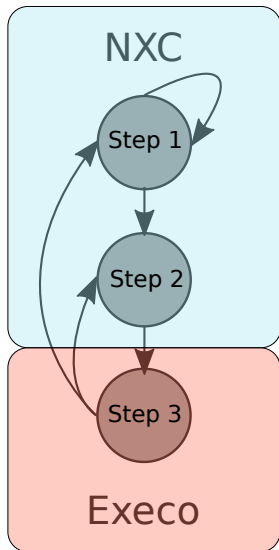
- Add the needed packages
- Open ports
- Define `systemd` services

## Step 2: Setting up experimental bricks

- Gather command lines as scripts
- Package those scripts

## Step 3: Script the experiment

- NXC provides an interaction with Execo
- Use Execo to use the "bricks"



# Your Turn!

## Take Home Message

NXC helps with setting up (reproducible) distributed exper

## What will you do now?

- 1 Get a Grid'5000 account
- 2 Install NixOS Compose
- 3 Get familiar with concepts
- 4 Setup environment for testing NFS performances

<https://tinyurl.com/NXCRegale>