Nix Package Manager

Michal Gorlas, Nedas Adamavicius, Kornel Gorski

Agenda

- Theory
- Practical
- NixOS

Agenda

- **Theory**
- Practical
- NixOS

Learning Objectives

LO1: The audience is aware of the concepts behind Nix Package Manager

LO2: How Nix compares to alternatives

- Ideal cases for Nix
- When Nix might be overkill

LO3: How to use Nix Flakes in a real project (Golang/Python/TypeScript)

Motivation

- A recurring difficulty in modern software development is reproducibility (1), (2), (3)
- One of the tools that has gained noticeable traction over the past decade for tackling this issue is Nix

(1) https://arxiv.org/abs/2402.00424
(2) https://zenodo.org/records/15315531
(3) https://doi.org/10.1109/MS.2021.3073045

What is Nix?

- Created as PhD project in 2003 by Eelco Dolstra. (4)
- The concept is to treat package management as a purely functional transformation: a package is built by a function whose result is immutable and identified by a hash of all its inputs.
- The "recipes" for Nix packages are written in Nix language.
- Nix language is a declarative, purely functional, lazily evaluated, dynamically typed.

(4) https://edolstra.github.io/pubs/nspfssd-lisa2004-final.pdf

Nix comparison

Nix

VS

Docker

GNU Guix

Nix vs Docker

Nix (The Builder)

- **Focus: Reproducible Builds**
- **How:** Declarative, functional, isolated dependencies
- **Result:** Byte-for-byte identical software

Docker (The Shipper)

- **Focus: Reproducible Runtime**
- **How:** Imperative script (`Dockerfile`), bundles OS snapshot
- **Result:** Runs the same *image* everywhere

Use **Nix to build** your app, use **Docker to package** the result

Nix vs GNU Guix

Nix / NixOS

- **Language:** Custom Nix (DSL)
- **Philosophy: Pragmatic** (Allows non-free software)
- **Ecosystem: Massive** & Evolving (Flakes)
- **Init System:** `systemd`

GNU Guix / Guix System

- **Language: Guile Scheme** (a Lisp)
- **Philosophy: Strict** (100% Free Software)
- **Ecosystem:** Smaller & Pure
- **Init System:** `GNU Shepherd`

Ideal cases for Nix

- Hybrid cloud/multi-cloud deployments
    - For describing VMs, containers, or bare-metal machines in a single Nix expression and deploy the same configuration on AWS, GCP, Azure, or on-premise hardware with a single command
- Continuous integration pipelines
    - For having every commit in a deterministic sandbox, catching build-time regressions early, and being able to publish the resulting store paths as immutable artefacts for later deployment. Example: Google Caliptra (5)
- Developer workstations and reproducible dev environments
    - To avoid "works on my machine" cases, and allow for faster onboarding

(5) https://github.com/chipsalliance/caliptra-sw/tree/main/ci-tools/github-runner

When Nix might be overkill

- **Reproducibility isn't real-world priority**
    - E.g., if environment drift rarely causes any issues
- **Dependency isolation is not a pain point**
    - E.g., projects that already work fine with language-specific managers (e.g., pip, npm, apt)
    - Or those with simple dependency stacks
- **Declarative configs aren't business-critical**
    - E.g., projects with stable and rarely changing setups

What are Nix Flakes?

Definition: A modern addition to the Nix ecosystem providing a standardized way to manage projects.

Core Concepts:

Reproducible builds for projects.

Integration with Git repositories.

A clear and portable interface for Nix-based projects.

Components:

flake.nix file for project definitions.

Specify inputs such as unstable/stable nixpkgs or home manager

Specify outputs like dev environments, applications, containers, and libraries.

Reproducibility for projects

Ever heard this:

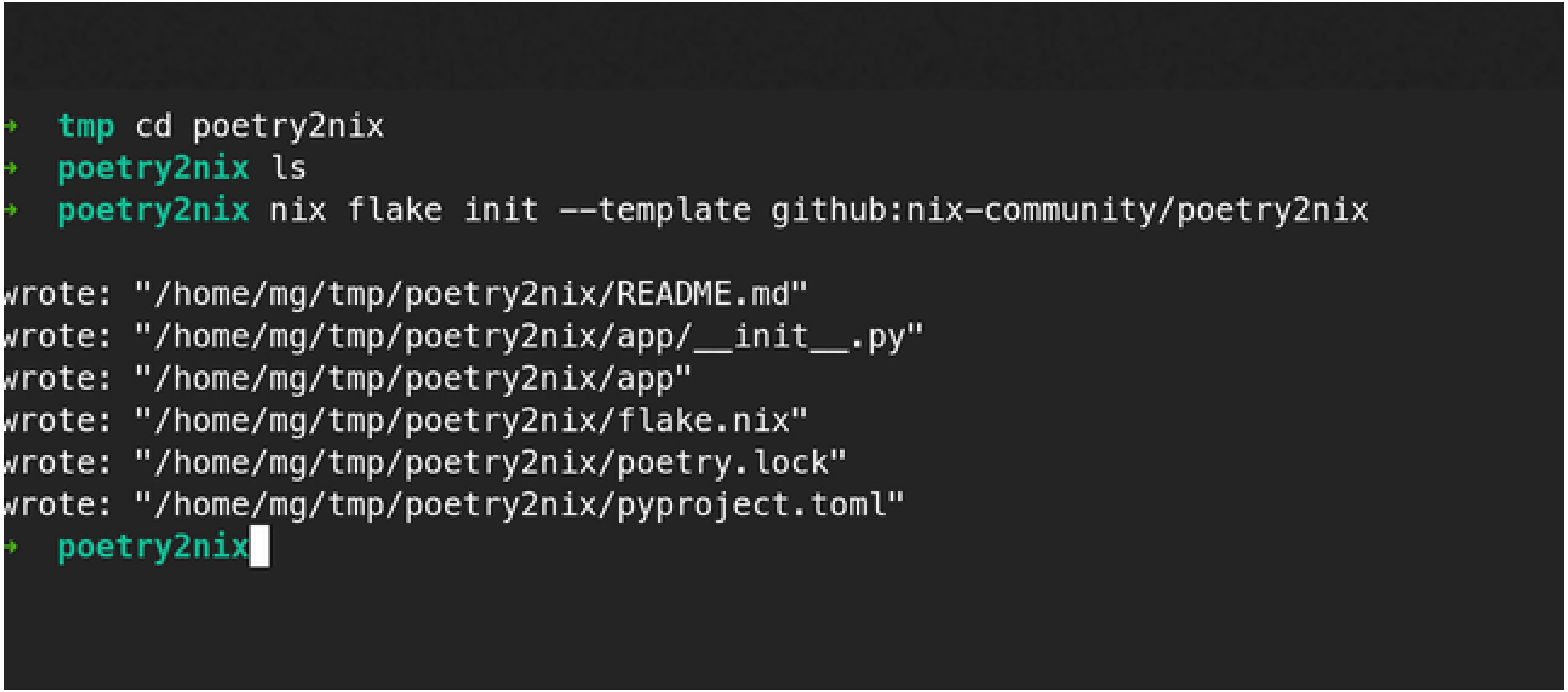
Out of 14k Nix packages, 99.94% have been proved to be fully reproducible, independent of the machine (1).

Thus:

if *nix flake .#* "works on my machine" => *nix flake .#* works everywhere

(1) https://arxiv.org/abs/2402.00424

Git integration



```
➜  tmp cd poetry2nix
➜  poetry2nix ls
➜  poetry2nix nix flake init --template github:nix-community/poetry2nix

wrote: "/home/mg/tmp/poetry2nix/README.md"
wrote: "/home/mg/tmp/poetry2nix/app/__init__.py"
wrote: "/home/mg/tmp/poetry2nix/app"
wrote: "/home/mg/tmp/poetry2nix/flake.nix"
wrote: "/home/mg/tmp/poetry2nix/poetry.lock"
wrote: "/home/mg/tmp/poetry2nix/pyproject.toml"
➜  poetry2nix
```

Minimal *flake.nix*

```
{
  description = "A very basic flake";

  inputs = {
    nixpkgs.url = "github:nixos/nixpkgs?ref=nixos-unstable";
  };

  outputs = { self, nixpkgs }: {

    packages.x86_64-linux.hello = nixpkgs.legacyPackages.x86_64-linux.hello;

    packages.x86_64-linux.default = self.packages.x86_64-linux.hello;

  };
}
```

Inputs

- Flake inputs are Nix dependencies that a flake needs to be built. Each input in the set can be pulled from various sources, such as Github, generic git repositories, and even your filesystem. (1)
- Inputs can modify each other's inputs to make sure that, for example, multiple dependencies all rely on the same version of nixpkgs. This is done via the *inputs.<input>.follows* attribute.

(1) https://zero-to-nix.com/concepts/flakes/#inputs

Outputs

Flake outputs are what a flake produces as part of its build. Each flake can have many different outputs simultaneously, including but not limited to:

- Nix packages
- Nix development environments
- NixOS configurations
- Nix templates

Flake outputs are defined by a function, which takes an attribute set as input, containing each of the inputs to that flake (named after the chosen identifier in the inputs section). (1)

(1) https://zero-to-nix.com/concepts/flakes/#outputs

Agenda

- Theory
- **Practical**
- NixOS

Exercises

Language	**Difficulty**	**Owner**
Golang	Very easy	Michal
Python	Easy	Nedas
TypeScript	Easy	Kornel

Start here: **github.com/FontysVenlo/esd-workshop-nix-enjoyers**

Agenda

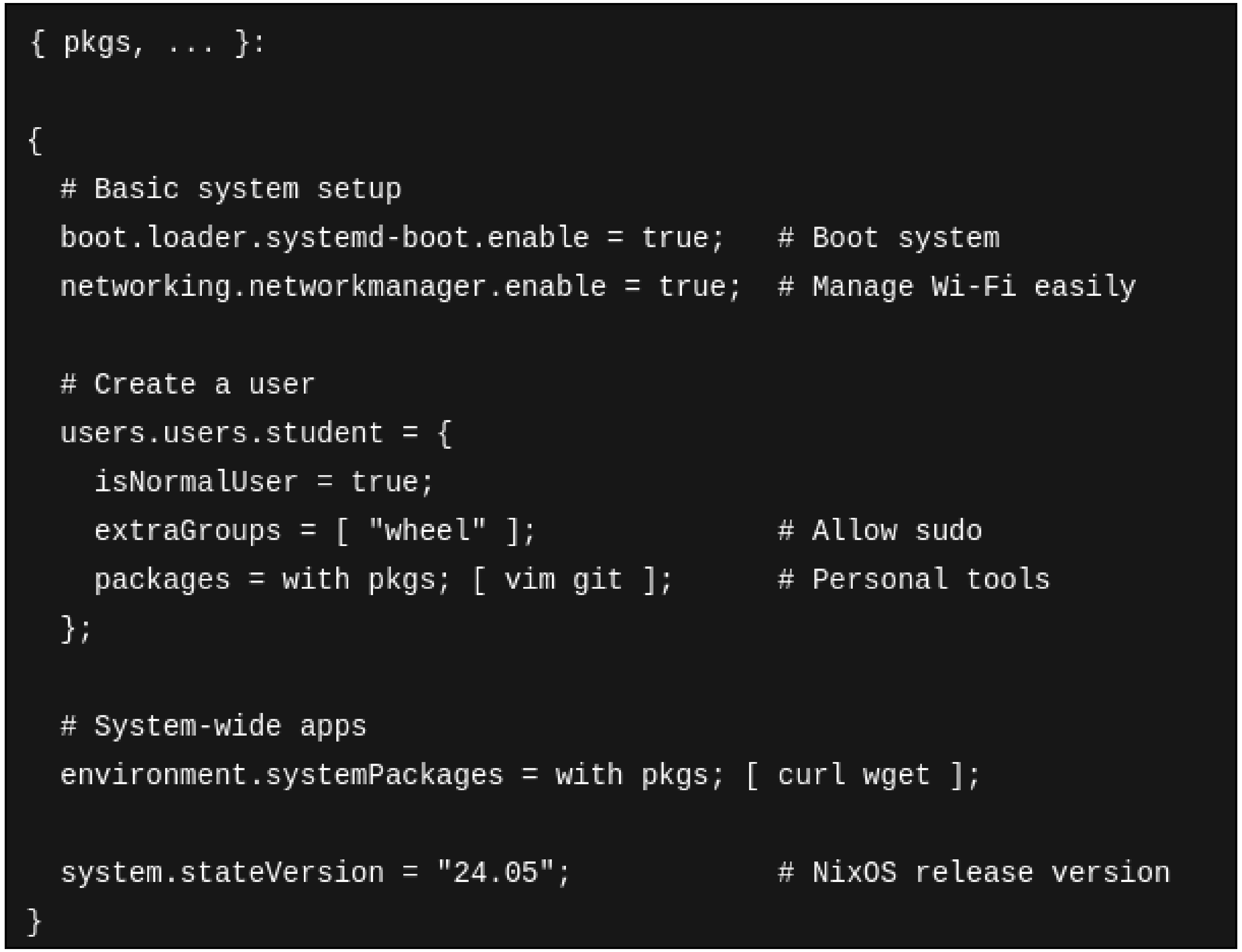
- Theory
- Practical
- **NixOS**

The OS-as-Code Vision

NixOS is a **Linux distro built on the Nix package manager**, where the entire OS – from kernel to services and packages – is defined **declaratively** in configuration files rather than set up manually.

The OS-as-Code Vision

NixOS is a **Linux distro built on the Nix package manager**, where the entire OS – from kernel to services and packages – is defined **declaratively** in configuration files rather than set up manually.



```
{ pkgs, ... }:

{
  # Basic system setup
  boot.loader.systemd-boot.enable = true;   # Boot system
  networking.networkmanager.enable = true;  # Manage Wi-Fi easily

  # Create a user
  users.users.student = {
    isNormalUser = true;
    extraGroups = [ "wheel" ];              # Allow sudo
    packages = with pkgs; [ vim git ];      # Personal tools
  };

  # System-wide apps
  environment.systemPackages = with pkgs; [ curl wget ];

  system.stateVersion = "24.05";            # NixOS release version
}
```

Reflect on your past projects, where would you use Nix?

Thanks Mr. Chaaya for inspiration