Introduction to the Nix Package Manager

Millian Poquet

2021-05-12 — Datamove (Inria) seminar

Why Nix?

Control your software environment!

- Programs/libraries/scripts/configurations + versions

Why is it important for us?

- Use/develop/test/distribute software
  - Manually install many dependencies? No, just type `nix-shell`
  - Shared env for whole team (tunable) and test machines
  - Bug only on my machine? Means this is hardware or OS related
- Reproducible research
  - Repeat experiment in exact same environment
  - Introduce or test variation

What is Nix?

Nix: package manager

- Download and *install* packages
- Shell into well-defined environment (like `virtualenv`)
- Transactional (rollback works)
- Cross-platform: Linux, macOS, Windows (WSL)

Nix: programming language

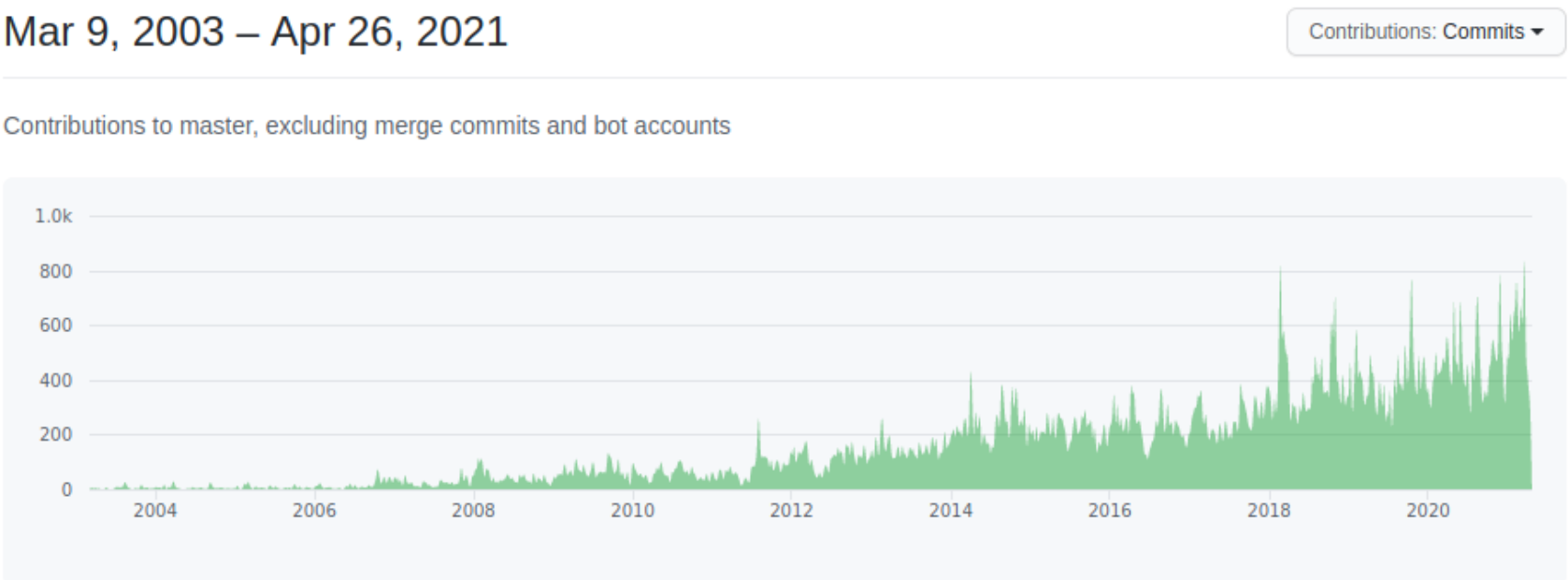
- Define packages
- Define environments (set of packages)
- Functional, DSL

NixOS: Linux distribution

- Declarative system configuration
- Uses the Nix language
- Transactional (rollback still works)

Nix in numbers

- Started in 2003
- Nix [1]: 10k commits, 28k C++ LOC
- Nixpkgs [2]: 285k commits, 55k packages [3]



1. https://github.com/NixOS/nix
2. https://github.com/NixOS/nixpkgs
3. https://repology.org/repositories/statistics

Presentation summary

Traditional package store (dpkg, rpm, pacman...)

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

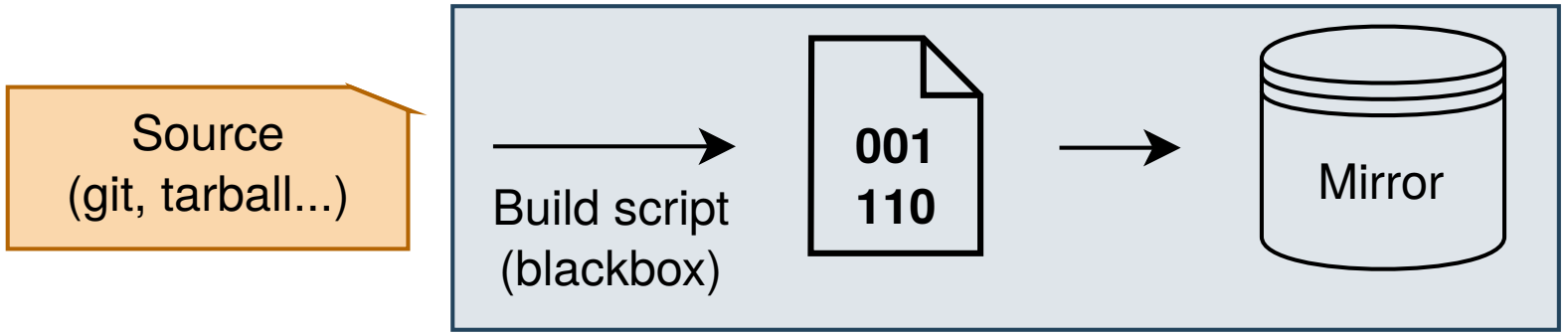
- All packages merged together
- Name conflict → cannot store multiple versions
  (or manual hack for each package)
- Use the *default* environment all the time
  - Bins in default dirs (`/bin/` or `/usr/bin/`) or hacked (debian)
  - Libs in default dirs (`/lib/` or `/usr/lib/`) or hacked (debian)
  - Vague file dependency (require `libmylib.so`)

Nix store

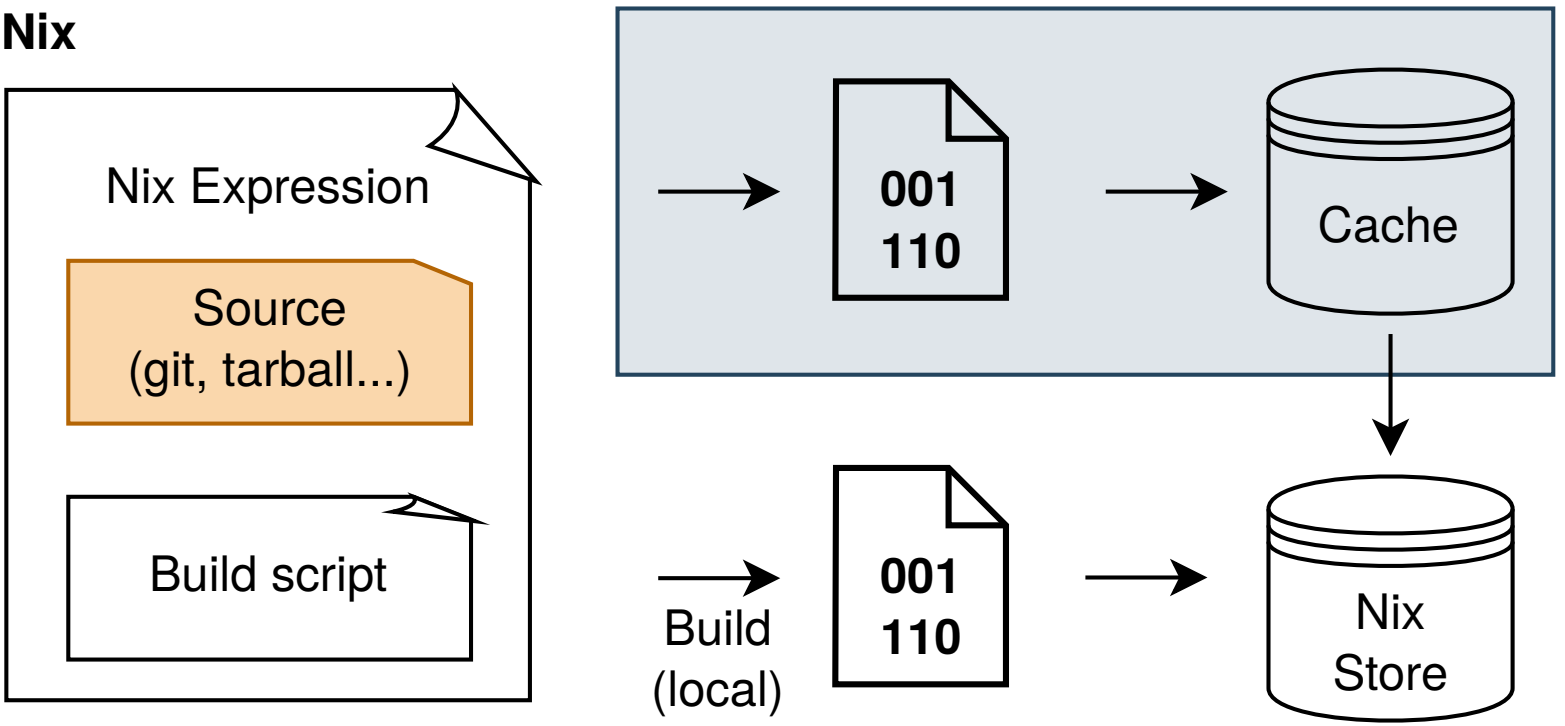
- All packages kept separated + links
- Store path: hash of inputs + package name
- Name conflicts?
  - Can *store* multiple versions
  - Cannot *enable* them simultaneously (in the same env)
- Precise dependency management
  - `DT_RUNPATH` set in ELFs (still hackable via `LD_LIBRARY_PATH`)
  - `PYTHONPATH`-like wrappers for interpreted scripts

Package build workflow

Traditional package managers



Nix



Main ideas on building a Nix package

Build in a ~~jail~~ sandbox

- pure env variables
- no network access (src fetched by Nix, not by user code)
- no ipc
- isolated filesystem

Build phases

- unpack
- patch
- configure
- build
- check
- install

Nix package example: intervalset.nix

```
1  { stdenv, fetchgit, meson, ninja, pkgconfig, boost, gtest }:
2
3  stdenv.mkDerivation rec {
4    pname = "intervalset";
5    version = "1.2.0";
6    src = fetchgit {
7      url = "https://gitlab.inria.fr/batsim/intervalset.git";
8      rev = "v${version}";
9      sha256 = "1ayj6jjznbd0kwacz6dki6yk4rxdssapmz4gd8qh1yq1z1qbjqgs";
10   };
11   buildInputs = [ meson ninja pkgconfig boost gtest ];
12   # configurePhase = "meson build";
13   # buildPhase = "meson compile -C build";
14   # checkPhase = "meson test -C build";
15   # installPhase = "meson install -C build";
16 }
```

Package variation: `override` for inputs

```
1  packages = rec {
2    intervalset = pkgs.callPackage ./intervalset.nix { };
3    intervalset-as-debian = intervalset.override {
4      boost = boost-167;
5      meson = meson-049;
6    };
7
8    boost-176 = ...;
9    boost-167 = ...;
10   boost = boost-176;
11
12   meson-058 = ...;
13   meson-049 = ...;
14   meson = meson-058;
15 };
```

Package variation: overrideAttrs for attributes

```
1  packages = rec {
2    intervalset = pkgs.callPackage ./intervalset.nix { };
3    intervalset-110 = intervalset.overrideAttrs (old: rec {
4      version = "1.1.0";
5      src = pkgs.fetchgit {
6        url = "https://framagit.org/batsim/intervalset.git";
7        rev = "v${version}";
8        sha256 = "0kksrr1l9gv7fg6rdjz39ph9l6smy74jsahyaj6pmpi1kzs33qva";
9      };
10   });
11   intervalset-local = intervalset.overrideAttrs (old: rec {
12     version = "local";
13     src = "/home/user/projects/intervalset";
14     mesonBuildType = "debug";
15   });
16 };
```

Define an environment: `mkShell`

```
1  { kapack ? import
2  (fetchTarball "https://github.com/oar-team/nur-kapack/archive/master.tar.gz"){}
3  }:
4  rec {
5    pkgs = kapack.pkgs;
6    expe-packages = [batsim-pinned batsched-pinned kapack.batexpe];
7    expe-shell = pkgs.mkShell rec {
8      name = "my-experiment-env";
9      buildInputs = expe-packages;
10   };
11   expe-docker = pkgs.dockerTools.buildImage {
12     name = "my-experiment-docker-env";
13     tag = "latest";
14     contents = expe-packages;
15   };
16   batsim-pinned = ...;
17   batsched-pinned = ...;
18   });
19 }
```

Command-line usage

```
~/proj/nur-kapack master $ nix-build . -A batsim-master
/nix/store/hbb03gjlk3v5hhv28iwkag2qf4ylxahg-batsim-master
~/proj/nur-kapack master $ tree result
result
└── bin
    └── batsim

1 directory, 1 file
~/proj/nur-kapack master $ ./result/bin/batsim --version
commit e4e9ae5614e04a44ac42dcb8358af35650d34453 (built by Nix from master branch)
~/proj/nur-kapack master $ which meson ninja batsim
meson not found
ninja not found
batsim not found
~/proj/nur-kapack master $ nix-shell -A batsim-master --command zsh                FAIL: 1
(nix-shell) ~/proj/nur-kapack master $ which meson ninja batsim
/nix/store/1ivl94srr4rrpdsdzby4mrjwjpv1zx3g-meson-0.55.1/bin/meson
/nix/store/hcd27k2wkllz33sk2id5jfd2ij34zpbf-ninja-1.10.1/bin/ninja
batsim not found
(nix-shell) ~/proj/nur-kapack master $                                             FAIL: 1
~/proj/nur-kapack master $                                                         FAIL: 1
```

- `nix-build -A <attr>` builds `derivation <attr>`
- `nix-shell -A <attr>` enters into the environment of attribute `<attr>` (build env for `derivation`, described env for `mkShell`)
- `nix-shell --command <cmd>` runs `<cmd>` inside an environment

Nix critique

Strengths

- No missing dependencies, local build likely works anywhere
- No boilerplate: Nix package = information needed to build it
- `nix-shell` = multi-language `virtualenv`
- Minimal size docker container generation is trivial
- Distributed Nix expressions — *e.g.*, nur-kapack [4]

Weaknesses

- Contaminant: dependencies must be expressed in Nix
- Learning curve — entry-level doc [5] improved a lot recently
- Implicit behaviors to build packages (looks magic at first sight)
- If used to dev: change in practice (for the greater good)
- Turing complete considered harmful — Guix/Spack do worse

4. `https://github.com/oar-team/nur-kapack`
5. `https://nixos.org`

Take home message

Nix in a nutshell

- Define pure packages (build in sandbox)
- Control and isolate your environments

Steep learning curve, but most likely worth it

- If you want to make sure your code runs in 5 years
- If you want to escape dependency hell

Additional resources

- Nix official website [6] — install, getting started...
- Tutorial on Nix for reproducible experiments [7]
- Nix pills [8] — AKA how nix works

6. https://nixos.org
7. https://nix-tutorial.gitlabpages.inria.fr/nix-tutorial
8. https://nixos.org/guides/nix-pills