

Towards Reproducible Experiment Environments with Nix

Adrien Faure, Millian Poquet



Rennes 2019, January 10, 2020

Experimentations

Experimentation codes

- *ad hoc*
- Poor (no?) documentation

Problem

- Difficult replay
- Hard to maintain

How to improve it: doc + automation

scripts

- prepare inputs
- build
- run
- analyze

well-defined environments

requirements

- kernel?
- hardware?

What is Nix ?

Nix is a Package Manager

- Reproducible packages
- Reproducible software environments
- Multiple versions
- Decentralized package repositories
- Clear dependencies
- Build on my laptop, run on g5k

Main Concept

Functional paradigm applied to package management

- Functions build packages
- Inputs = dependencies, source code, build script
- Packages written in Nix expression language

No side effects

- Undeclared dependencies → fail
- New package → cannot break existing ones

Package Definition Example

```
stdenv.mkDerivation {  
  name = "chord";  
  src = fetchurl {  
    url = "https://gitlab.com/me/chord.tar.gz";  
    sha256 = "1h2jgq5pspyiskffq777nhi5rf0y8h...";  
  };  
  buildInputs = [ simgrid boost cmake ];  
}
```

Store

All packages in `/nix/store`

- Isolated packages
- *Hash(inputs, source code)-packagename*
- Package names known before build → binary cache

```
/nix/store
└─ hash-packagename
   └─ bin
      └─ packagename
   └─ lib
      └─ libpackagename.so
```

└ Nix

└ Nix tools

Nix Build: build packages

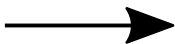
```

{ stdenv, fetchgit, cmake, simgrid, bc
stdenv.mkDerivation rec {
  pname = "chord";
  version = "0.1.0";

  src = fetchgit {
    url = "https://gitlab.inria.fr/adf";
    rev = "cbe903e7f0839794fba572e4ccf";
    sha256 = "1qpf0cn1sy2l0mrr3y1xmv2";
  };

  buildInputs = [
    cmake
    simgrid
    bc
  ];
}

```

Nix
Expression

/nix/store/awzvf...-chord-0.1.0

└ bin

└ chord

/nix/store/db2fz...-glibc-2.27

└ lib

└ libc.so.6

/nix/store/pdpmm...-simgrid-3.22.2

└ lib

└ libsimgrid.so.3.22.2

Nix shell: Virtualenv on steroids

```

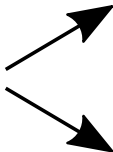
{ stdenv, fetchgit, cmake, simgrid, bc
stdenv.mkDerivation rec {
  pname = "chord";
  version = "0.1.0";

  src = fetchgit {
    url = "https://gitlab.inria.fr/adf";
    rev = "cbe903e7f8e397d4fe572eakcf";
    sha256 = "1apm6eloy2lmeeryjbmzd";
  };

  buildInputs = [
    cmake
    simgrid
    boost
  ];
}

```

Nix
Expression



```

/nix/store/db2fz...-glibc-2.27
└─ lib
  └─ libc.so.6
/nix/store/pdpmm...-simgrid-3.22.2
├─ bin
│  └─ smpicc
│  └─ smpirun
└─ lib
  └─ libsimgrid.so.3.22.2

```

```

[shell]$ echo $PATH
/nix/store/pdpmm...-simgrid-3.22.2/bin/smpicc
/nix/store/pdpmm...-simgrid-3.22.2/bin/smpirun

[shell]$ echo $LIBRARY_PATH
/nix/store/db2fz...-glibc-2.27/lib
/nix/store/pdpmm...-simgrid-3.22.2/lib

```

Package repositories

Package definitions

- Source code (in Nix)
- Stored in decentralized repositories

Official Git Repository: **NixPkgs**

<https://github.com/NixOS/nixpkgs>

- Community maintained
- +10K packages
- CI checked
- Binary caches

Conclusion

Why it is reproducible ?

- Traceable dependencies
- Automated package build
- Fixed application source
- Pinned Nixpkgs

Limitations

- No kernel version control
- Require deterministic build
- External storage (gitlab...)

<https://mpoquet.gitlabpages.inria.fr/nix-tutorial/>

Channels

A **channel** is link to branch of NixPkgs tested with continuous integration.

Channels are useful to download latest packaged version of a software.

- nixpkgs-unstable (feeling lucky?)
- nixpkgs-19.03 (current stable)
- nixpkgs-18.09 (outdated)

(Channels benefit from binary cache.)

Channels

Channels are not fully reproducible, as they are subject to updates.




























As experimenters, we will use another mechanism called pinning.

Nix - Command line interface

How do we use Nix ?

- *nix-build*: build a derivation (that will be placed to the nix store),
- *nix-env*: install a package (in your current environment) ,
- *nix search*: search for available packages.
- *nix-shell*: start a shell in the build environment of a derivation,

Existing solutions

	Module	 easybuild	 Spack	
Reproducibility				
Portability				
Multi-user				
Multiple version				
Binary packages				
Isolated build env.				
Isolated runtime env	