**Defining packages in Nix**

Defined using a custom functional language

- Knowledge of this is not required for most users

Packages are kept in a directory containing a hash of:

- package source via a SHA256 hash
- build configuration
- each dependency’s hashes all the way to the libc

The hash uniqueness ensures:

- Many versions/configurations without conflicts
- No ambiguity: same install location iff same build

Example: Build both ROOT and XRootD with different Python and gcc versions

- Results in four different install directories for each package:

```
# gcc 6
# Python 2
# gcc 7
# Python 3
```

Main upstream repository of packages is nixpkgs:

- Includes support for most build systems
- Many helper functions to minimise boilerplate
- Various “channels” for stable and unstable releases

Steps to add a new package:

- Create a file defining the source and dependencies
- Add one line to all-packages.nix

Default build script splits the build into phases:

```
unpackPhase
patchPhase
configurationPhase
checkPhase
installPhase
installCheckPhase
fixupPhase
```

```
buildPhase
checkPhase
installPhase
installCheckPhase
```

- Default: Run ./configure.sh if present
- Dependencies can automatically override (i.e. cmake)

Build script is flexible, phases can be easily overridden

Automatic tweaks for languages and build systems

Total flexibility without any boilerplate

[1] https://nixos.org/nixpkgs/

**Defining environments**

Environments can also be defined using Nix

- Get the build environment for a package
- Make a meta package of symlinks (buildEnv)

Packages can easily define setup hooks

- Arbitrary shell script that is sourced automatically
- Can be used to easily add environment variables

```
# See the HSF packaging group’s "testdrive" for an example of using buildEnv to define a deep stack.
```

---

**Requirements for HEP packaging**

**Nix**

- A “purely functional package manager”
- Works with Linux and other Unix systems (including macOS)
- Supports 1686, x86_64 and aarch64 (experimental) including cross-compilation
- Everything is kept in the store directory (default: /nix/store)
- Designed to support many conflicting software versions/configurations
- Preexisting community with 0(4,000) package definitions

Nix has a very strong focus on:

- Reproducibility: All dependencies should be explicitly defined and build tools should not look in locations outside of Nix.
- Software must be stable for long periods (much longer than a LTS OS)

---

**What is Nix?**

- The source to build the package which can be downloaded via http, ftp, git, sun, cvs and other. The hash is as a dependencies

- Software packaging and distribution for LHCb using Nix

---

**Full recipe for building the base “LHCb” software application**

Here is a complete nix expression which allows the base application of the LHC软件 package to be built.

```
<table>
<thead>
<tr>
<th>[ file ]</th>
<th>fetchurl</th>
<th>fetchtar</th>
<th>nixurl</th>
<th>mirrsources</th>
<th>fetchattr</th>
<th>installPhase</th>
<th>buildPhase</th>
<th>checkPhase</th>
<th>installPhase</th>
<th>checkPhase</th>
</tr>
</thead>
<tbody>
<tr>
<td>python 2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>python 2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>python 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>python 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gavd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libgit2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Testing Nix within LHCb**

- Approximately 20 separate packages
- Distributed as binary releases on CMVFS

**Changing the store directory**

- Changed to /cmvfs/lhcbdev.cern.ch/nix/
- Would be an essential feature for LHCb

**Custom Hydra instance dramatically improved the Nix experience**

- Changing the store directory requires a full rebuild (slow!)
- Host on CERN OpenStack, back by Postgres DB instance
- Connect via SSH to docker containers on faster build machines
- Managing and scaling a “cluster” of build machines was easy

**Forcing nixpkgs**

- Makes deep customisation easier
- Successfully auto-rebasings the fork to track upstream changes
- Hydra monitors for and keeps track of automatically builds changed packages
- Will setup a system to pull relevant changes upstream

**Building LHCb reconstruction software (Brunel)**

- Depends on 4 other LHCb packages
- Many external dependencies, most were already available
- Some minor tweaks were needed

**Oracle Instant Client**

- Licensing issues prevent Nix from downloading
- Had to manually import source
- Enables builds of non-free software
- Missing derivations: Boost, COOL, CORAL, CLHEP, frontier, pcapparser, RELAX, Reflex, VOT, XrootD
- Most were trivial to define
- CMake

- Closed source build system that depends on glibc
- Once identified easy to fix using patchelf

---

**Providing binary caches with Hydra**

- Building deep stacks locally is time consuming and issue prone
- Mitigate this with binary caches
  - Static web servers serving signed tarballs
  - Request file using the package hash

**Hydra**

- A continuous build system
- Deep integration with Nix
- Builds periodically, after every commit or for releases
- Scalable from a single machine to a entire cluster (via SSH)
- Can serve binaries directly or use plugins to export (e.g. S3)
- Mitigations for common issues (bad workers, network, ...) Can also provide continuous integration
- Also used by some CNOU projects

[1] https://nixos.org/hydra

---

**Summary**

The LHCb stack can be built within Nix!

- HSF packaging WG is considering Nix

**Benefits:**

- Environments are exactly defined and reproducible
- Independent from the host OS
- Hydra could replace Jenkins for CI/CD needs

**Disadvantages:**

- Not fully backwards compatible yet...
  - Store directory can be changed to be on CMVFS
  - Could use containers & user namespaces instead