

Incremental Package Builds

Guillaume Maudoux
@layus

NixCon 2017



Louvain-la-Neuve hold a huge bike event



@layus

laïus (🇫🇷) a long, vague and emphatic speech

@layus

laïus (🇫🇷) a long, vague and emphatic speech

layus a nixpkgs contributor

@lays

laïus (🇫🇷) a long, vague and emphatic speech

lays a nixpkgs contributor



I contributed to build a castle in France



© Guédelon

So I...

- ▶ like building stuffs

So I...

- ▶ like building stuffs
- ▶ started a PhD on incremental builds

So I...

- ▶ like building stuffs
- ▶ started a PhD on incremental builds
- ▶ want your feedback on this presentation

Incremental package builds

Incremental

adjective [...] occurring in **especially small increments.**

Incremental

adjective [...] occurring in **especially small increments.**

noun the amount or degree by which something changes;

Incremental build systems

- ▶ Works better with small steps
- ▶ Can reuse older build products
- ▶ Can detect what needs to be built

Plan

Firefox – Works better with small steps

i3 – Can reuse older build products

Nix store – Can detect what needs to be built

Firefox – Small steps

Building Firefox takes very long

50s	unpack	1 min
1s	patch	
156s	configure	2 min 30 s
4006s	build	1 hour 6 min
31s	install	
8s	fixup	



We need checkpoints!



We need checkpoints!



We need checkpoints!



We need checkpoints!



Trivial idea

We could split each package phase in a different derivation...

Trivial idea

We could split each package phase in a different derivation...

- ▶ Only a small, local fix
- ▶ Not clean for /nix/store
- ▶ Already done by external wrappers (firefox)

Incremental builds is about small steps

- ▶ Nix is incremental at the package level.
- ▶ Nix is a package manager

Incremental builds is about small steps

- ▶ Nix is incremental at the package level.
- ▶ Nix is a package manager
- ▶ Build systems are incremental at command level.
- ▶ let's use that!

Build systems details – make

1. pros:

- ▶ well known

2. cons:

- ▶ requires previous builds as an input
- ▶ uses timestamps to detect changes

Build systems details – ccache

ccache memoizes compiler invocations

sccache can share it's cache on the network

nix is much like sccache

Build systems details – nix-make

 [edolstra / nix-make](#)

[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Insights](#)

Experimental Nix build management stuff



Branch: [master](#) ▾ [New pull request](#)

 **edolstra** Produce more sensible store path names

 [examples](#) * `dependencyClosure` now allows a search path, e.g.,

 [lib](#) Produce more sensible store path names

Build systems details – nix-make

```
let {
  inherit (import ../../lib) compileC link;

  hello = link {
    objects = compileC {
      main = ./hello.c;
    };
  };

  body = [hello];
}
```

Build systems details – nix-make

1. pros:

- ▶ very small steps
- ▶ compatible with nix

2. cons:

- ▶ every intermediate .o file ends up in the store
- ▶ requires to port projects to nix-make

Build systems details – bazel

 **Bazel** Documentation Contribute    Search 

{Fast, Correct} - Choose two

Build and test software of any size, quickly and reliably



[GET BAZEL](#) [GET STARTED](#)

Build systems details – bazel

1. pros:

- ▶ caches arbitrary commands
- ▶ uses sandboxing to guarantee correctness

2. cons:

- ▶ conflicts with nix

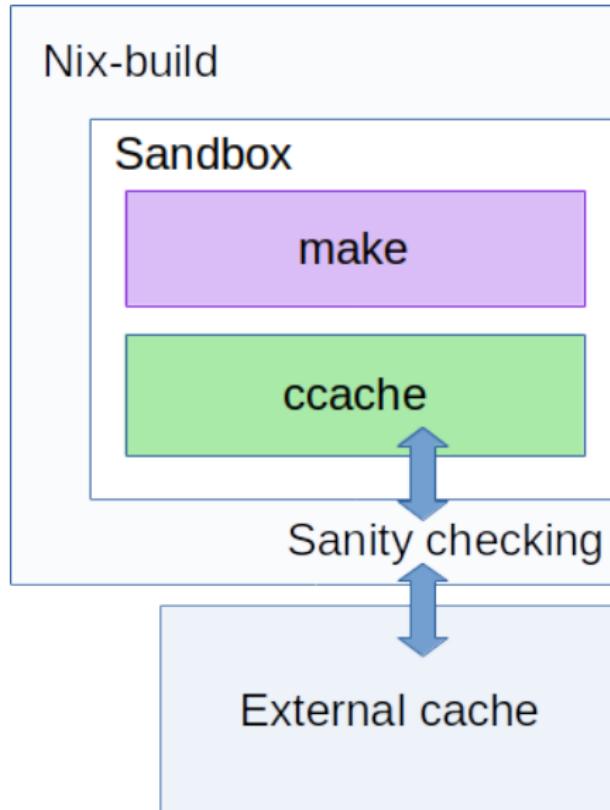
`bazel = nix + build system`

Intermediate solution

- ▶ Use caching in the build system
- ▶ Control caching from nix-build

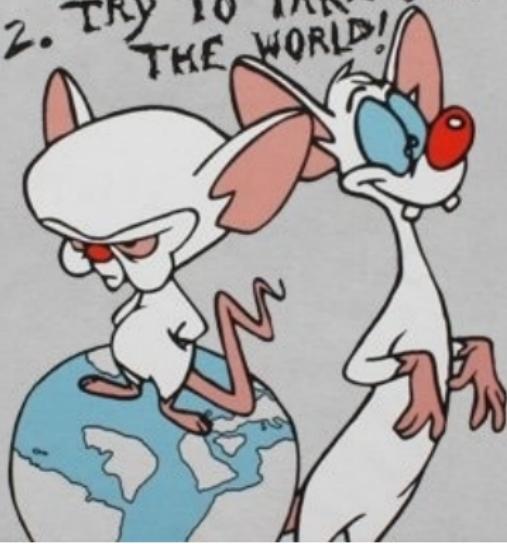
Easy to implement, if we trust the build system.

Allow caching inside nix-build



TO DO LIST:

1. WAIT FOR TONIGHT
2. TRY TO TAKE OVER
THE WORLD!



still a work in progress

i3 – Reusing old builds

The issue

Requirements

- ▶ Need to patch i3 for an annoying bug
- ▶ Can only be tested on this machine
- ▶ Needs to be included in NixOS config

Typical debug session

- ▶ build a custom version (`nix-shell`)
- ▶ write an overlay for that package (`nix-build`)
- ▶ modify nixos to use it, and use debug flags
(`nixos-rebuild`)
- ▶ test and restart

Current options

We need an easy solution to achieve simple, local package development:

Current options

We need an easy solution to achieve simple, local package development:

1. Mount the nix store RW

Current options

We need an easy solution to achieve simple, local package development:

1. Mount the nix store RW
2. Insert a symlink in the store

Current options

We need an easy solution to achieve simple, local package development:

1. Mount the nix store RW
2. Insert a symlink in the store
3. Configure services with paths outside the store

Current options

We need an easy solution to achieve simple, local package development:

1. Mount the nix store RW
2. Insert a symlink in the store
3. Configure services with paths outside the store

Current options

We need an easy solution to achieve simple, local package development:

1. Mount the nix store RW
2. Insert a symlink in the store
3. Configure services with paths outside the store

Hacky !!

Existing tools

`git rebase --interactive` brings you right at the conflicting commit, in a fake environment

`nix-shell -A` setups the right environment to build a package. But you *can not* write to the store

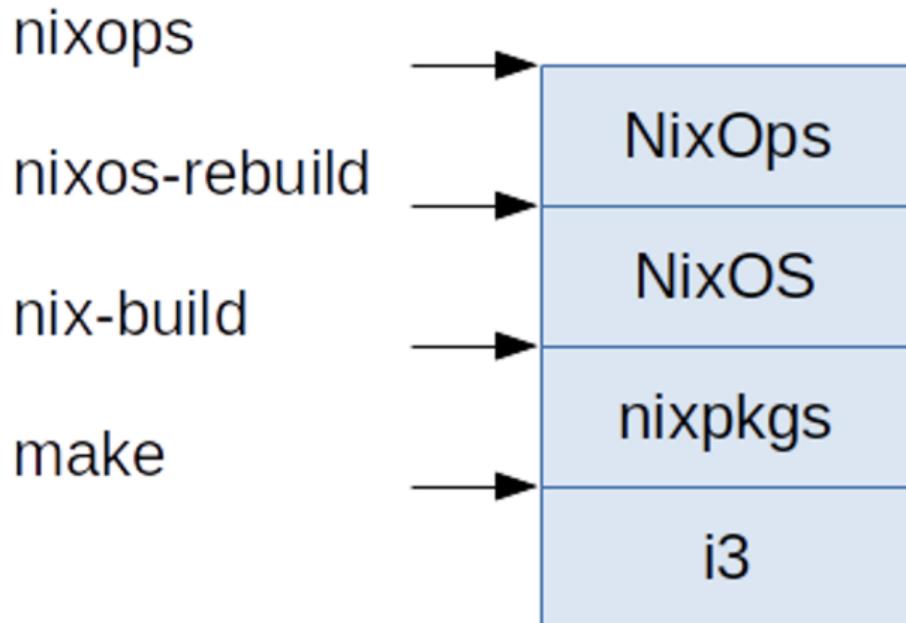
Solving the nix-shell issue

To make nix-shell more handy, it could generate a random \$out on each invocation, and allow writes to that store location.

We want a trade-off between correctness and ease of use.

```
nix build --hack i3
```

Nix* “OSI” model



nix-shell is a zipper

nix-shell
make



Going further

```
nixos-rebuild test --hack i3
```

Drop me in a shell where I can patch i3, then use that for this nixos version.

And even further

We can even use caching when implemented! A “clean” nix-build could reuse results cached by a “hacky” build.

And even further

We can even use caching when implemented! A “clean” nix-build could reuse results cached by a “hacky” build.

TL;DR: Never compile the same build step twice.

And even further

We can even use caching when implemented! A “clean” nix-build could reuse results cached by a “hacky” build.

TL;DR: Never compile the same build step twice.

Well, *if* implemented.

Nix store – Track changes

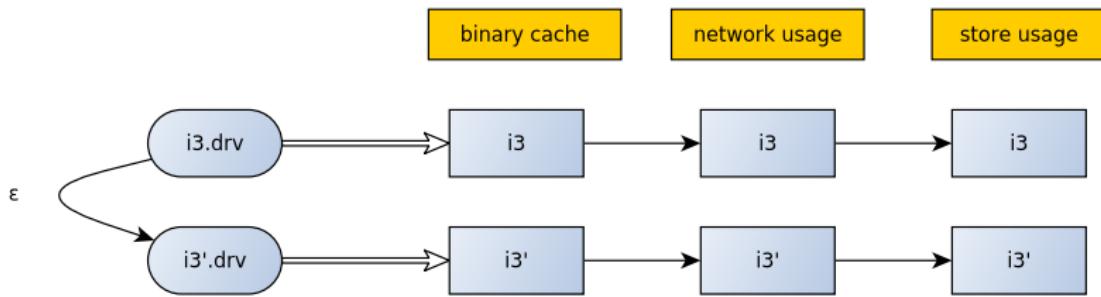
/nix/store has many similar packages

```
diffoscope /nix/store/4w573p7v5jjjkph5ndzmbwg55v2ids1y-poppler-data-0.4.7 /nix/
--- /nix/store/4w573p7v5jjjkph5ndzmbwg55v2ids1y-poppler-data-0.4.7
+++ /nix/store/ccc9py7hfgd5v4q7hk2fv3l4rjllh12i-poppler-data-0.4.7
  lib
    pkgconfig
      poppler-data.pc
      @@ -1,8 +1,8 @@
-poppler_datadir=/nix/store/4w573p7v5jjjkph5ndzmbwg55v2ids1y-poppler-data-0.
+poppler_datadir=/nix/store/ccc9py7hfgd5v4q7hk2fv3l4rjllh12i-poppler-data-0.

  Name: poppler-data
  Description: Encoding files for use with poppler
  Version: 0.4.7

-Cflags: -DPOPPLER_DATADIR=/nix/store/4w573p7v5jjjkph5ndzmbwg55v2ids1y-poppl
+Cflags: -DPOPPLER_DATADIR=/nix/store/ccc9py7hfgd5v4q7hk2fv3l4rjllh12i-poppl
```

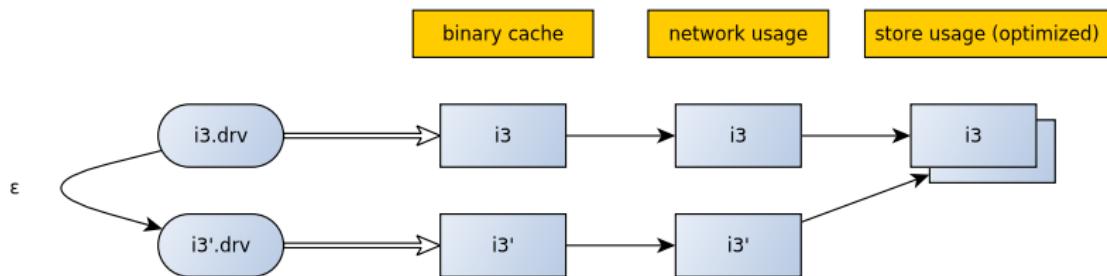
/nix/store has many similar packages



Nix store --optimize

```
$ du -shc /nix/store/{X,Y,Z}-poppler-data-0.4.7
12M  /nix/store/4w573p7v5jjjkph5ndzmbwg55v2ids1y-poppler-data-0
60K   /nix/store/ccc9py7hfgd5v4q7hk2fv3l4rjllh12i-poppler-data-0
60K   /nix/store/pshrgbbmvkxp6lf4hzwn04560brf52lp-poppler-data-0
13M    total
```

Nix store --optimize



Mass rebuilds use a lot of network.

Every week, unstable branch is merged into master.

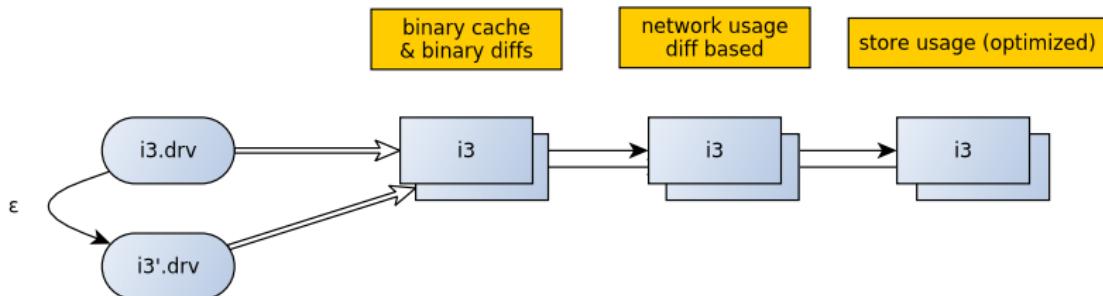
Every week, nixpkgs master receives mass-rebuild commits.

Updating once per month makes you download a full new distro each time.

Can we do better than that ?

Binary diffs of substitutes

By storing and sharing diffs of binary packages, we could save bandwidth and hydra space.



Content addressed storage

We can even go further for derivations whose only difference is their \$out.

NixOS / rfcs

Watch 35

Code Pull requests 11 Projects 0 Insights

Intensional Store #17

Open wmortens wants to merge 2 commits into NIXOS:master from wmortens:master

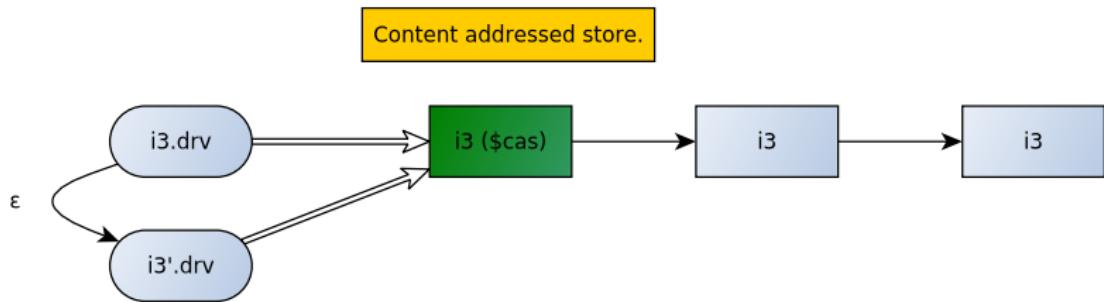
Conversation 12 Commits 2 Files changed 1

wmortens commented on 11 Aug

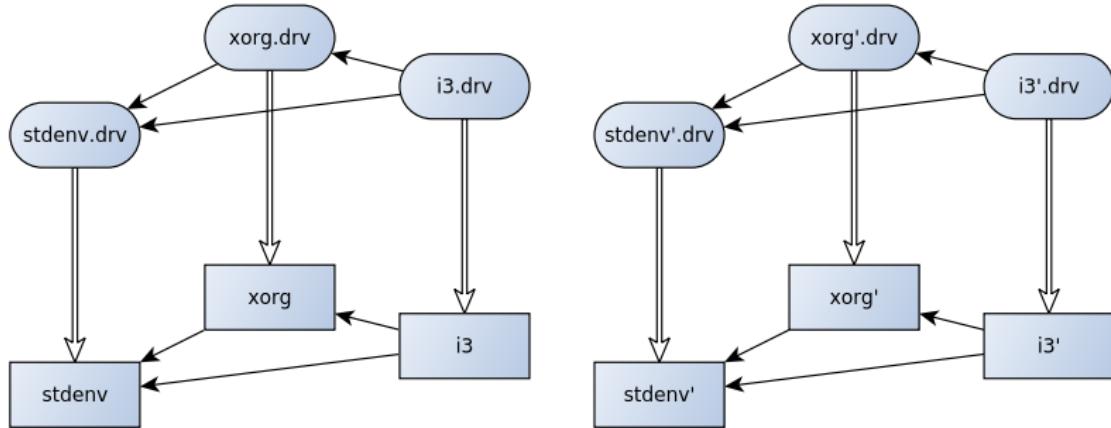
A practical proposal to store build products under their output hash instead of their input hash.

4 1

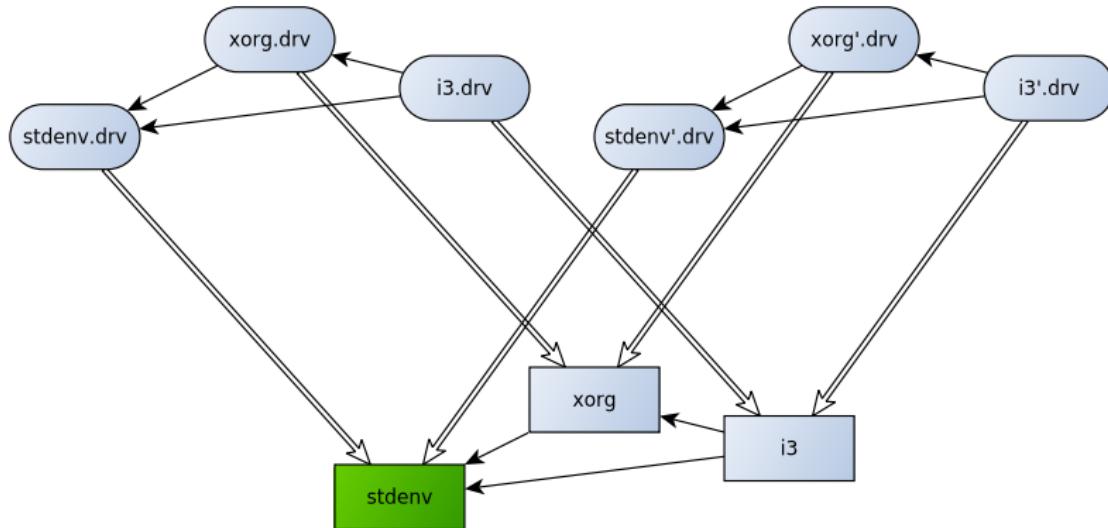
Content addressed storage



CAS & change propagation



CAS & change propagation



Content addressed storage

1. Pros:

- ▶ Does not propagate changes to dependencies
- ▶ Less compiling
- ▶ Faster updates
- ▶ Not too difficult to implement
- ▶ Outsourcing reproducibility is easy

2. Cons:

- ▶ Changes Nix
- ▶ Real impact is unknown

CAS & refactorings

The screenshot shows a GitHub repository page for `NixOS/nixpkgs`. The top navigation bar includes links for `This repository`, `Search`, `Pull requests`, `Issues`, `Marketplace`, and `Explore`. On the right side of the header are icons for notifications, a plus sign, and a user profile.

The main content area displays a pull request titled "Just strip everything by default, fully tested version :-)" with the ID `#15339`. The status bar indicates the PR is `Open` and it wants to merge 4 commits from `layus:fix-strip` into `NixOS:staging`. Below this, there are sections for `Conversation 35`, `Commits 4`, and `Files changed 11`. A summary of code changes shows `+125 -37` with a progress bar consisting of green and red squares.

A comment by `layus` dated 10 May 2016 is shown, stating: "This is the same PR as #15087, but fully tested and modified to take into account the discovered bugs. Stdenv now supports the `dontStripPath` attribute which is the black-list version of the former `stripDebugList` and `stripAllList` (maintained for compatibility).".

On the right side of the comment, there are sections for `Reviewers` (which is empty) and `Assignees` (with a profile picture for `vcunat`).

CAS & Cached builds

Caching builds can make them less stable
(reproducible)

Content addressed store would help to catch
unstable builds

Conclusion

There are still a lot of possible improvements to nix.

I have started to work on cached builds, and CAS is under RFC.

Comments welcome.

Questions ?