Build Outside the (Sand)Box

Guillaume ‘layus’ Maudoux

NixCon, October 2019, Brno

Introduction

Two years ago

Now

Q: Can we build individual nix packages incrementally ?

A: No. Not with the default sandbox.

Ouch !

Q: Can I quickly test changes to {firefox, gtk, libreoffice, qt, gcc, ...} derivations ?

A: No. Not with the default sandbox.

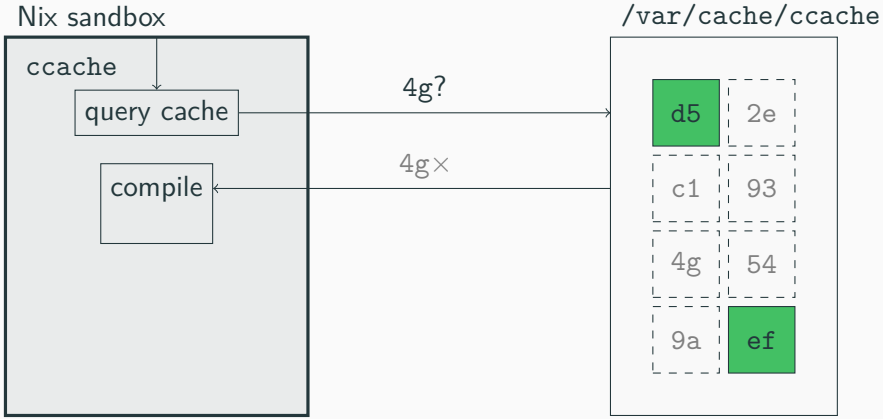
First attempt: `ccache`

What is `ccache` ?

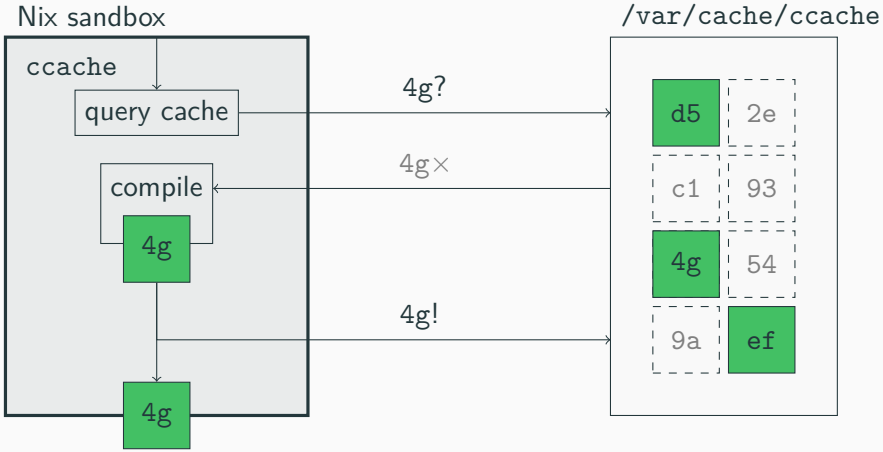
Compiler **Cache**

- «Developed and maintained (since 2010) by Joel Rosdahl. Was originally written by Andrew “Tridge” Tridgell.»
- Stores previous compiler invocations with their results.

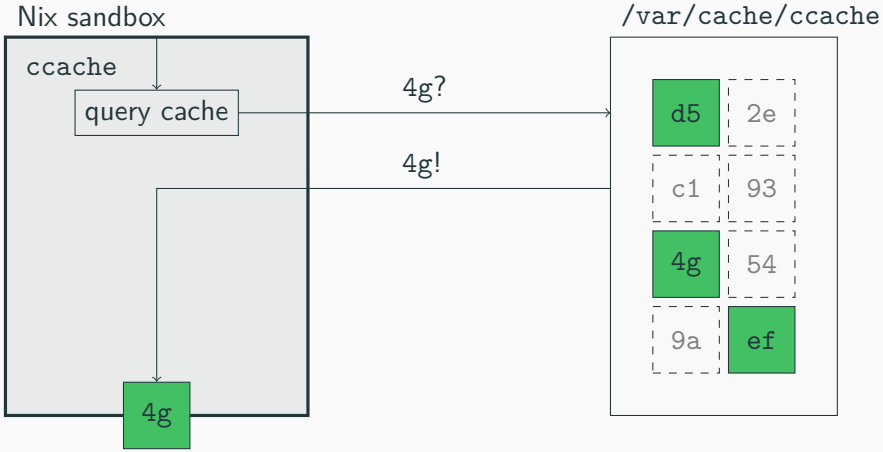
CCache miss


Nix sandbox
ccache
query cache
compile
4g?
4g×
/var/cache/ccache
d5
2e
c1
93
4g
54
9a
ef

CCache miss


Nix sandbox
ccache
query cache
compile
4g
4g
4g?
4g×
4g!
/var/cache/ccache
d5
2e
c1
93
4g
54
9a
ef

CCache hit


Nix sandbox
ccache
query cache
4g
4g?
4g!
/var/cache/ccache
d5
2e
c1
93
4g
54
9a
ef

Turns out this is already possible, in nixpkgs

```
ccacheStdenv = overrideCC stdenv ccacheWrapper;

# Wrap cc.cc
ccacheWrapper =
  cc.override {
    cc = ccache.links {
      inherit extraConfig;
      unwrappedCC = cc.cc;
    };
  };
```

Turns out this is already possible, in NixOS

```
programs.ccache.enable = true;

programs.ccache.cacheDir = "/var/cache/ccache"; # by default.

# Packages that will have access to CCache.
programs.ccache.packageNames = [ "firefox-unwrapped" "wxGTK30" "qt48" ...
```

CCache demo usage

DEMO

CCache demo usage

- `nix-build -A i3`
  - Timings added.

CCache demo usage

- `nix-build -A i3`
  - Timings added.
- `nix-build -A i3-ccache`
  - Building with ccache, with cold cache

CCache demo usage

- `nix-build -A i3`
  - Timings added.
- `nix-build -A i3-ccache`
  - Building with ccache, with cold cache
- `nix-build -A i3-ccache.again`
  - Building with ccache, with hot cache

CCache demo usage

i3	i3-ccache	i3-ccache.again	phase
0.227s	0.225s	0.250s	unpackPhase
0.425s	0.418s	0.420s	patchPhase
8.732s	**9**.431s	**7**.614s	configurePhase
10.444s	**15**.091s	**6**.199s	buildPhase
0.004s	0.003s	0.003s	glibPreInstallPhase
0.091s	0.092s	0.090s	installPhase
0.003s	0.003s	0.003s	glibPreFixupPhase

Is this a good thing ?

Yes !

No !

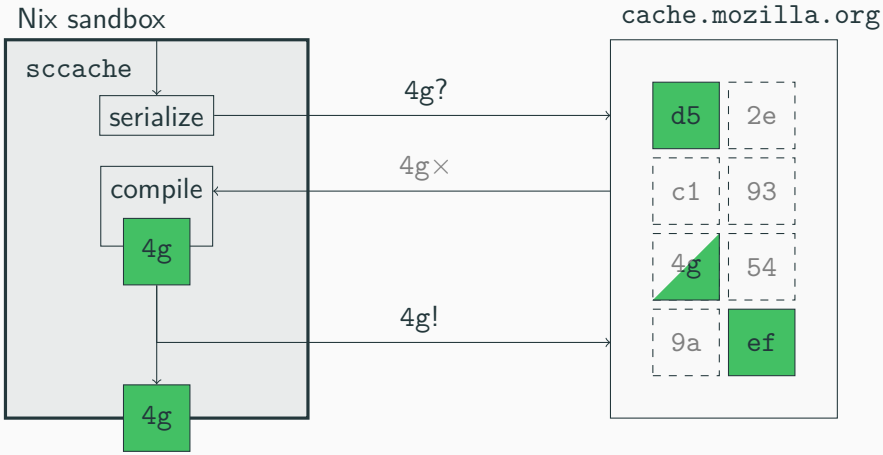
Second attempt: `sccache`

What is `sccache` ?

Shared **CCache**

- A mozilla product
- Supports C/C++ and Rust
- CCache + a network cache protocol / encoding.

An `sccache` example



Networking in the sandbox ?!

Networking in the sandbox ?!

No.

Networking in the sandbox ?!

No. Not supported.

Networking in the sandbox ?!

No. Not supported.

Just disable the sandbox for now,
but restricted network access could be implemented.

Note: Disabling the sandbox is not yet possible with chroot stores.

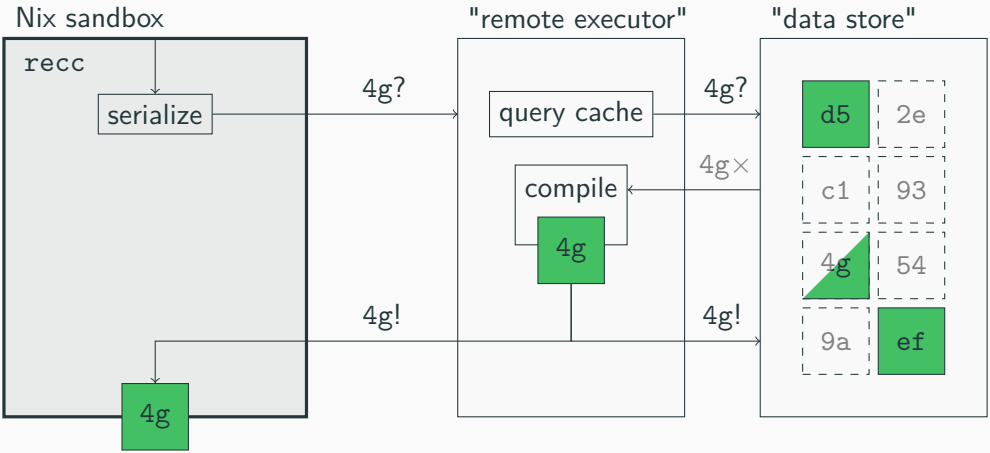
Third attempt: `recc`

What is `recc` ?

Remote-**E**xecution **C** **C**ompiler

- Developed at Bloomberg
- Compiler wrapper client for the Remote Execution API
- Work-around to benefit from without migrating to {BuildStream, Bazel, Buck}-like build systems.

recc design


Nix sandbox
recc
serialize
4g?
"remote executor"
query cache
compile
4g
4g?
4g×
"data store"
d5
2e
c1
93
4g
54
9a
ef
4g!
4g!
4g

`recc` demo usage

- `nix-build -A i3-recc-small` *Cold cache*
- `nix-build -A i3-recc-small.again` *Hot cache*
- `nix-build -A i3-recc-small` *Hot cache, on the builder*

Note: Single threaded
Remote worker is slow

`recc` demo usage

`i3-recc-small`	`i3-recc-small.again`	local	build phase
0.230s	0.240s	0.198s	unpackPhase
0.416s	0.419s	0.241s	patchPhase
8.507s	8.495s	8.105s	configurePhase
231.068s	150.807s	131.113s	buildPhase
0.003s	0.003s	0.002s	glibPreInstallPhase
0.092s	0.094s	0.090s	installPhase
0.003s	0.003s	0.002s	glibPreFixupPhase

This does not work as-is.

- We need the build inputs on the worker.
- Nix design gives some guarantees on correctness, even tough the worker has no sandbox.
- Impure errors are cached !
- We could automate this and generalize it.

The Good, the Bad and the Ugly Impurities

The Good, the Bad and the Ugly Impurities

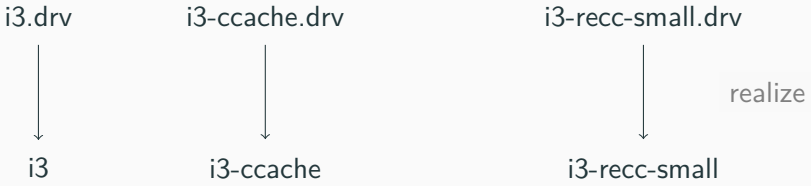
Q: Are such derivations **reproducible** ?

Q: Are such derivations **pure** ?

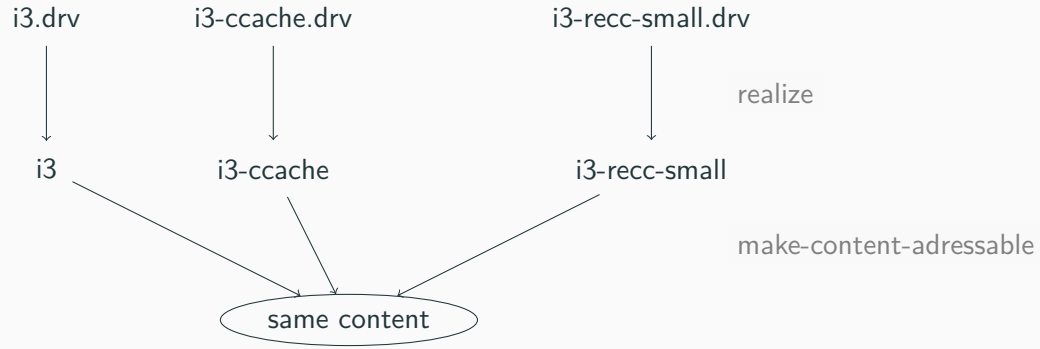
Q: Are such builds **hermetic** ?

A: Demonstration ! With “experimental” nix.

Different intentions, same content



Different intentions, same content



Conclusion

What we have now (but that still needs tuning).

Take away messages

- Breaching the sandbox does not necessarily bring impurities.
- Bringing incremental builds inside derivations is possible.
- Maybe we could consider trusting other tools than Nix ?

Questions

Questions

- Should we trust other build systems ? Or is pure, nix-only isolation preferable ?

Questions

- Should we trust other build systems ? Or is pure, nix-only isolation preferable ?
- Do you have idea and/or opinions about opening the sandbox for other build systems ? What about *Bazel*, *Gradle*, *Recc* or *Nix* itself (recursively) ?

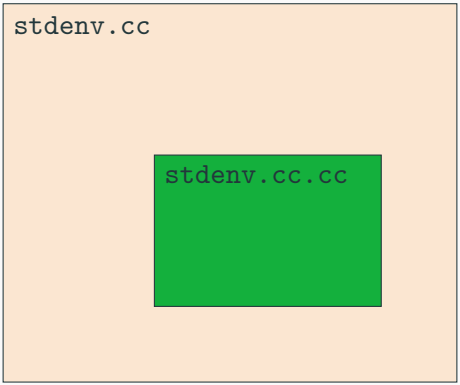
Questions

- Should we trust other build systems ? Or is pure, nix-only isolation preferable ?
- Do you have idea and/or opinions about opening the sandbox for other build systems ? What about *Bazel*, *Gradle*, *Recc* or *Nix* itself (recursively) ?
- Can we make this generic ?

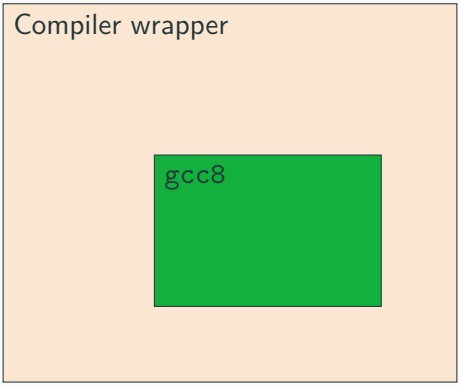
The `again` trick

```
forever = drv:
  let
    drv2 = drv.overrideAttrs (oldAttrs: {
      postFixup = oldAttrs.postFixup + "\n true";
    });
  in
    drv // { again = forever drv2; };
```

Wrapping `cc.cc` but not `cc`



Wrapping `cc.cc` but not `cc`



Wrapping `cc.cc` but not `cc`