Smarkets

Frustration free* C++17 with Nix

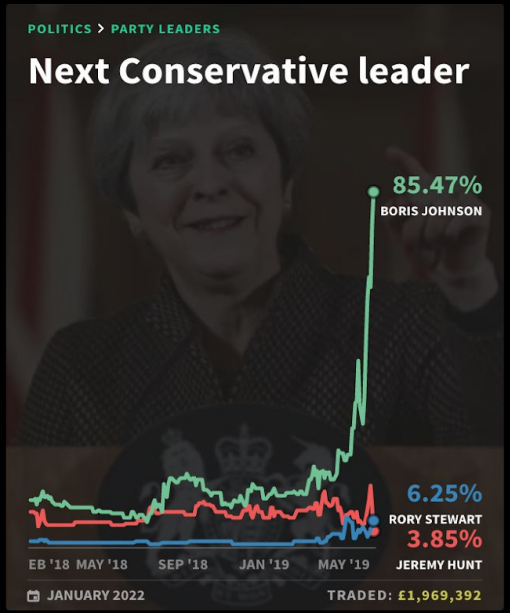
Alexander Schmolck

C++ Meetup at Smarkets, June 2019

Smarkets Limited [GB] smarkets.com/politics
Search Google
Vivaldi.com Travel Deals: Fl... start.me YouTube News pin Merge Request...

POLITICS > PARTY LEADERS

Next Conservative leader



JANUARY 2022 TRADED: £1,969,392

POLITICS > NEXT UK GENERAL ELECTION

Year of next General Election

2019	2020
35.97% 2.78	32.26% 3.1

+4 CONTRACTS

JANUARY 2022 TRADED: £215,678

POLITICS > US PRESIDENTIAL ELECTION 2020

Next President

Donald Trump	Joe Biden
45.87% 2.18	16.67% 6

+38 CONTRACTS

NOVEMBER 2020 TRADED: £212,246

POLITICS > DONALD TRUMP

Trump exit date

2020 or later	2019
92.59% 1.08	6.45% 15.5

JANUARY 1 TRADED: £184,392

POLITICS > US PRESIDENTIAL ELECTION 2020

Democratic Nominee for President

Joe Biden	Elizabeth Warren
29.41% 3.4	14.29% 7

+39 CONTRACTS

NOVEMBER 2020 TRADED: £135,003

POLITICS > NEXT PRIME MINISTER

Next Prime Minister after Theresa May

Boris Johnson	Andrea Leadsom
81.97% 1.22	7.41% 13.5

+93 CONTRACTS

JULY 2022 TRADED: £80,291

Popular Categories

POPULAR EVENTS

- West Indies 321 / Bangladesh 91 — LIVE NOW — TRADED: £175,593
- China PR (W) vs. Spain (W) — IN 1 HOUR — TRADED: £40,468
- South Africa (W) vs. Germany (W) — IN 1 HOUR — TRADED: £31,514
- Open Championship 2019 — JULY 18 — TRADED: £65,120
- Next Conservative leader — JANUARY 2022 — TRADED: £1,969,392
- Japan vs. Chile — IN 8 HOURS — TRADED: £99,349
- Next Prime Minister after Theresa May — JULY 2022 — TRADED: £80,291
- 15:45 - Catterick — EVENT ENDED — TRADED: £87,845
- Seats after the next General Election — JANUARY 2022 — TRADED: £61,516

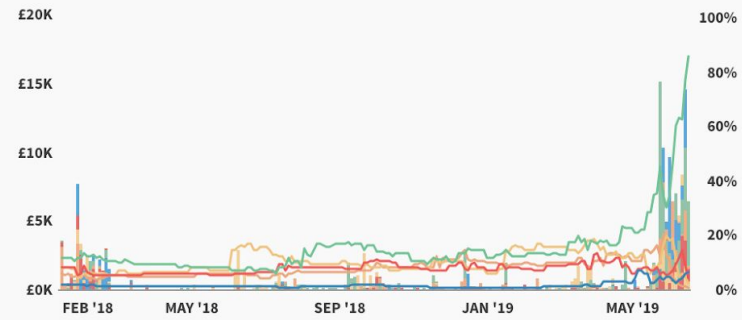
Politics 53
Politics 37
Brexit 7
Chancellor of the Exchequer 1
Conservative Leadership contest 2
House of Commons Speaker 1
London Mayoral Election 1
Next Cabinet Member to leave 1
Next Prime Minister 1
Next UK General Election 7
Party Leaders 10
Scottish Independence 1
The Independent Group for Change 5
ropean Politics 6

JANUARY 1, 2022 1 MARKET AVAILABLE

Next permanent leader of the Conservative Party, after Theresa May?

TRADED: £1,969,851 P&L: £1.10 / -£0.80

ALL 12 MONTHS 6 MONTHS 1 MONTH



CONTRACT	LAST TRADED PRICE	105.85%			97.87%		
Boris Johnson -£0.80 TRADE OUT	85.47%	94.34% £331	90.91% £423	86.96% £208	85.47% £29	84.75% £34	82.64% £9
Rory Stewart £1.10	6.45%	7.41% £9	7.14% £3	6.45% £2	6.06% £27	5.88% £25	5.71% £29

YOUR BETS

ALL ACTIVE

BET	STAKE	ODDS	MATCHED	
Next Conservative leader / Next permanent leader of the Conservative Party, after Theresa May?				
Against Boris Johnson	£1.10 £0.80	57.98% (AVG.)	£1.10	2
Next President / Winner of 2020 Presidential Election				
For Donald Trump	£0.05	42.74%	£0.05	1
For Andrew Yang	£30.00	2.50%	£30.00	1
Democratic Nominee for President / Elected Democratic Party presidential nominee for the 2020 US presidential election				
For Tulsi Gabbard	£13.00	2.17%	£13.00	1
Next Prime Minister after Theresa May / Next Prime Minister after Theresa May				
For Michael Gove	£0.06	14.71%	£0.06	1

SHOW MORE SHOW ALL

ALL ACTIVITY

Our story

Smarkets is a betting exchange, with a rapidly growing tech team. We use a variety of DBs, languages (mostly Python on the server, JS for frontend, C++ and Erlang for core backend functionality: exchange, settlements and market state).

2016 DevEnv was terrible.

Exchange team C++

- moving from erlang and homebrew protocols to C++ and FIX (finance standard)
- only modern C++17
- event sourced architecture, using disruptor pattern
- dispatch on event std::variant
  - poor man's Haskell-style pattern matching
- property based testing with rapid_check
- extensive use of type_safe to create safer numeric types for IDs, prices, money etc.
- aim for straightforward, direct code

C++ specific challenges

- dependency and build tooling not great
- large and complex eco-system; clang/llvm has best tooling, but large and complex beast
- a lot of implicit state and dynamic lookup (magic paths like /include, ld.conf, RPATH, ...)
- want to be close to bleeding edge

Andy Boot 12:24 PM

I honestly don't care about the details. But I envisage a world where a new joiner is given a computer with all things required to run hanson already installed. And we all use the same versions. People call me crazy.

Can we solve this problem with

Vagrant

Can we solve this problem with

Vagrant

- Anyone here ever used Vagrant and didn't regret it?
- Literally often took a week to be able to build master again after a git pull
- So people did not pull!
- Super slow (virtualization, but ESP file system mounting both slow ***and*** flaky)

Can we solve this problem with

Docker?

Can we solve this problem with

Docker?

- easy to get started with
- hard to get working reliably (has gotten more robust in recent years)
- not declarative, not reproducible
- caching is heuristic and therefore often wrong; everyone runs into apt-get --upgrade problem eventually.
- Only support simple linear deps; not dependency trees (invalidates too much)
- network/process isolation is useful but half baked (cf bsd jails)
- mostly just used as a glorified static linker
- but fair amount of overhead (docker run bash vs bash; networking etc.)

Can we solve this problem with

Kubernetes?

Can we solve this problem with

Kubernetes?

- Moving dev env to k8s was a major step forward
- two years later we still only have a few services in prod
- works great if you can use GCP, not so much on AWS (yet)
- Complex to run reliably
- Still no declarative builds/provisioning
- Slow for dev, especially if using minikube

Can we solve this problem with

Nix?

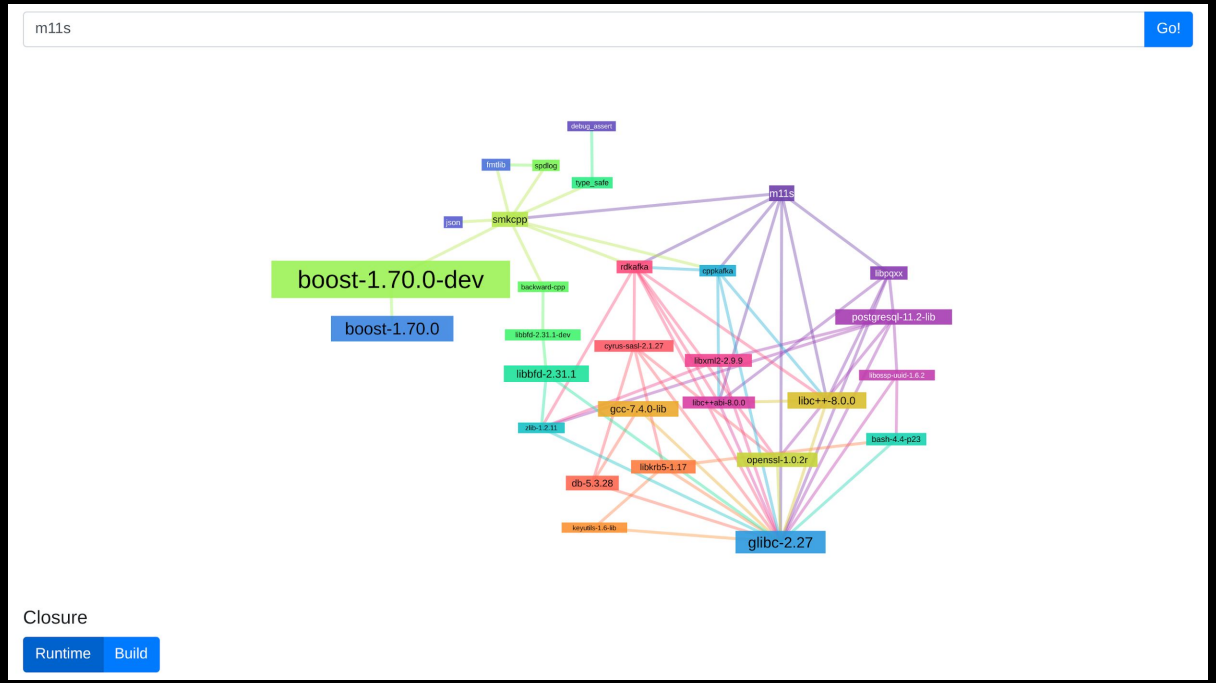
- the ultimate build system/package manager
- declarative: built on pure functions, or at least semi pure-functions that take well defined inputs (not just global env)
- no "nix-build clean" (unlike [c]make etc. build is always correct)
- if you didn’t declare it it’s not there (even if it used to be there before)
- rollbacks, transactionality, isolation
- single language for everything (your DB setup and even migrations, python, haskell, C++, vim, ...)
- even single file to declare whole machine if you want to
- zero overhead
- small, completely reproducible docker images (without any docker tooling!)

We can solve this problem with

Nix!

- trivially customize software versions and build params
- do DB migrations as build artifacts (thanks to perfect caching, clean DB setup <1s from several minutes)
- treat tooling as build artifact (nix-shell only, does not bloat production)
  - vim-based IDE functionality (refactor, complete, goto def, doc lookup, code-format on save)
  - db shells, code-formatters, debug utilities for kafka protobuf topics etc.
  - grafanix for depency analysis!
- build docker images that don't suck
  - concisely and declaratively
  - small docker image
  - k8s for prod, but don't need it for dev

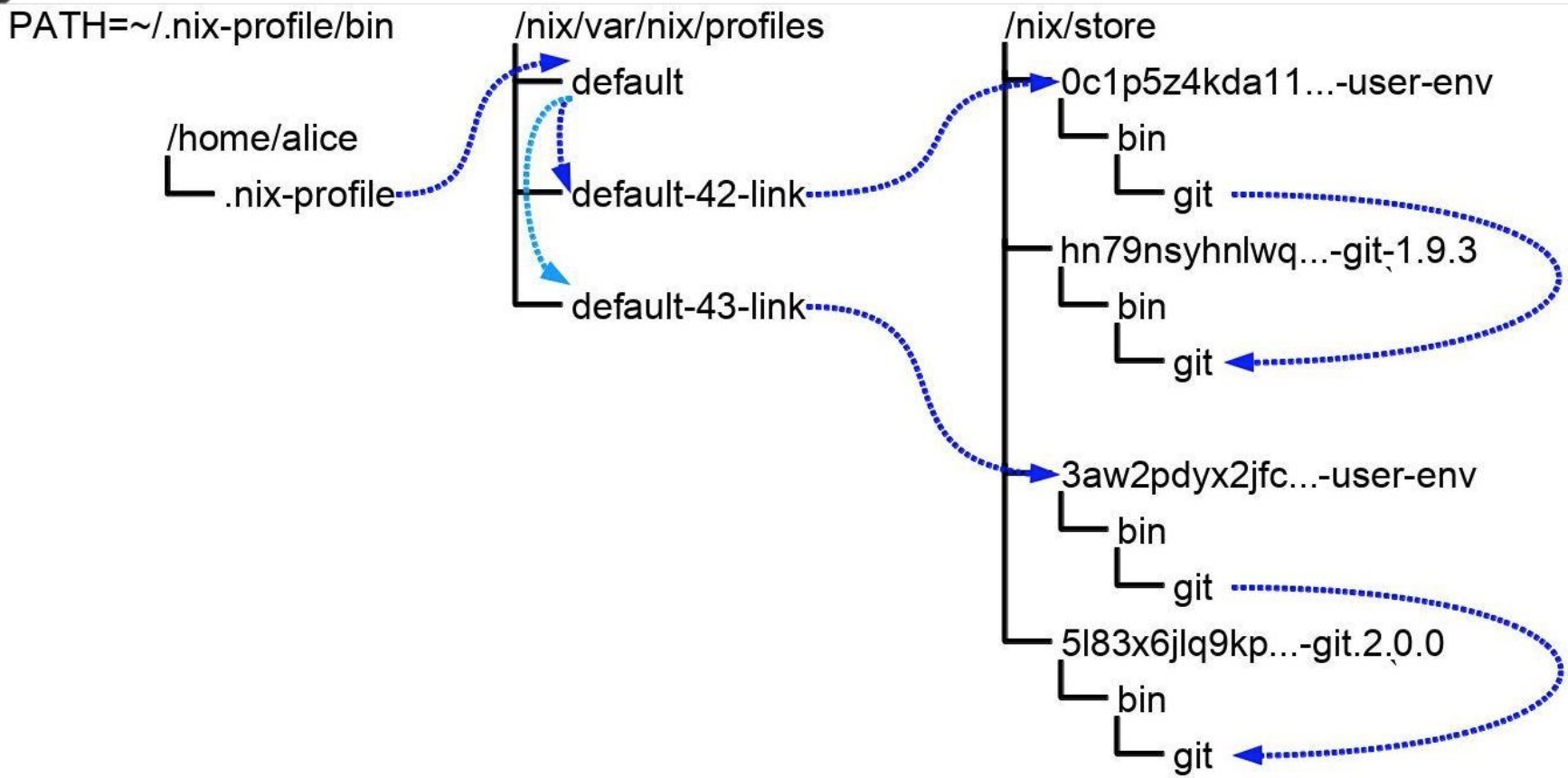
github.com/stolyaroleh/grafanix can visualise your package dependencies!



- works with all languages
- shows why things depend on each other
- draws messy graphs
- made with

Example package

```
with import <nixpkgs> {};
stdenv.mkDerivation {
 name = "fmt";
 src = fetchFromGitHub {
   owner = "fmtlib";
   repo = "fmt";
   rev = "9e554999ce02cf86fcdfe74fe740c4fe3f5a56d5";
   sha256 = "1hl9s69a5ql5nckc0ifh2fzlgsgv1wsn6yhqkpnrhasqkhj0hgv4";
 };
 # Use CMake to build this package
 nativeBuildInputs = [ cmake ];
}
```


PATH=~/.nix-profile/bin
/home/alice
.nix-profile
/nix/var/nix/profiles
default
default-42-link
default-43-link
/nix/store
0c1p5z4kda11...-user-env
bin
git
hn79nsyhnlwq...-git-1.9.3
bin
git
3aw2pdyx2jfc...-user-env
bin
git
5l83x6jlq9kp...-git.2.0.0
bin
git

Demo Time!

Demo Time!

Fresh Env in AWS from scratch

```
./make_nodes.py  --just-do-it --realm testing -p nix-test-box-1 -t exchange
-i m5.large -c 1 -s private
H=$(awsteam exchange |awk '/nix-test-box/ && $0=$2')
scp -rp ~/demo-aws-creds ~/demo-ssh-config "$H":
ssh -tA "$H" mv demo-ssh-config .ssh
ssh -tA "$H" mv demo-aws-config .aws
REPO=git@git.corp.smarkets.com:smarkets/smarkets.git
ssh -tA "$H" git clone --branch meetup --depth 1 "$REPO"
ssh -tA "$H" ./smarkets/backend/nix/install.sh
ssh -tA "$H" bash -lc '"cd smarkets/services/exchange/oms && nix-shell"'
```

C++

C++ specific challenges and nix

- implicit state/hardcoded paths antithetical
- but C++ compiler without include path and stdlib not much use
- clang++ is wrapped in nix to "know about" standard libraries implicitly
- means cmake nevers sees these as flags and they're not in compile_commands.json
- this confuses cquery and other tooling
- we work around this by patching compile-commands.json to add the magic paths manually, but better solution would be to define toolchain explicitly (WIP)

Tooling

C++

Build tool	Compiler	Linting, formatting and IntelliSense
CMake	Clang 8 with libc++ and Coroutines TS!	clang-tidy, clang-format and cquery

Can we avoid ABI issues?

C++

```
let
  cppOverlay = import ./overlay.nix {};
in
  import ../nixpkgs {
    overlays = [
      cppOverlay
    ];
  }
```

Can we avoid ABI issues?

C++

```
self: pkgs: with pkgs; {
  # replace GCC and libstdc++ with Clang and libc++
  stdenv = libcxxStdenv;
}
```

It's a bit tricky...

C++

```
self: pkgs: with pkgs; {
  # replace GCC and libstdc++ with Clang and libc++
  stdenv = libcxxStdenv;
}

libcxxStdenv = if stdenv.isDarwin then stdenv else lowPrio llvmPackages.libcxxStdenv;
llvmPackages.libcxxStdenv = overrideCC stdenv buildLlvmTools.libcxxClang;
```

It's a bit tricky... On Linux

C++

```
self: pkgs: with pkgs; {
  # replace GCC and libstdc++ with Clang and libc++
  stdenv = libcxxStdenv;
}

libcxxStdenv = lowPrio llvmPackages.libcxxStdenv;
llvmPackages.libcxxStdenv = overrideCC stdenv buildLlvmTools.libcxxClang;
```

We can avoid ABI issues!

C++

```
let
  cppOverlay = import ./overlay.nix {
    # libcxxStdenv is defined in terms of stdenv.
    # get it from a copy of nixpkgs to prevent infinite recursion.
    vanillaNixpkgs = import ../nixpkgs {};
  };
in
  import ../nixpkgs {
    overlays = [
      cppOverlay
    ];
  }
```

We can avoid ABI issues!

C++

```
{ vanillaNixpkgs }:
self: pkgs: with pkgs; {
  # things we depend on, but don't want to (or can't) recompile using clang
  inherit (vanillaNixpkgs) cmake python27 ...;
  # use Clang 7
  stdenv = vanillaNixpkgs.llvmPackages_7.libcxxStdenv;
  # and define our own C++ packages
  smkcpp = callPackage ../../../packages/smkcpp/smkcpp.nix {};
}
```

CMake: Building with Nix

C++

```
stdenv.mkDerivation {
  name = "smkcpp";
  src = lib.cleanSource ./.;
  configurePhase = ''
    cmake -Bbuild -DCMAKE_INSTALL_PREFIX="$out" -H.
    cd build
  '';
}
```

C++

```
stdenv.mkDerivation {
  name = "smkcpp";
  src = lib.cleanSource ./.;
  nativeBuildInputs = [ cmake ];
}
```

C++

```
cmakeProject {
  name = "smkcpp";
  src = lib.cleanSource ./.;
}
```

CMake: Building with Nix

C++

```
{ cmakeProject, fetchJSON, debug_assert }:
cmakeProject {
  name = "type_safe";
  src = fetchJSON ./type_safe.src.json;
  cmakeBuildType = "Release";
  propagatedBuildInputs = [ debug_assert ];
  patches = [ ./type_safe.tests.patch ];
}
```

C++

```
project("smkcpp")
add_library(smkcpp lib.cpp)
install(TARGETS smkcpp DESTINATION lib)
```

```
project("p12r")
add_executable(p12r main.cpp)
target_link_libraries(p12r ???)
```

C++

```
project("smkcpp")
add_library(smkcpp lib.cpp)
install(TARGETS smkcpp DESTINATION lib EXPORT smkcpp-targets)
install(EXPORT smkcpp-targets
        FILE smkcpp-config.cmake
        DESTINATION "${CMAKE_INSTALL_PREFIX}")

project("p12r")
find_package(smkcpp REQUIRED CONFIG)
add_executable(p12r main.cpp)
target_link_libraries(p12r smkcpp)
```

Editor integration

C++

```
smkCmakeProject {
  name = "smkcpp";
  src = lib.cleanSource ./.;
}

# ^ like cmakeProject, but also includes
# a customized vim and C++ Language Server
nativeBuildInputs = nativeBuildInputs ++ [
  cquery smkCppVim
];
```

CI

Andy Boot 12:18 PM

CI should not break when things don't change.
Nothing changed and CI broke.
Our CI is not robust. It needs to be.

3 1

Building docker images!

CI

```
{ dockerTools, makeSymlink, smarkets-base-image }:
service:
dockerTools.buildImage {
  name = service.name;
  fromImage = smarkets-base-image;
  tag = "latest";
  contents = [
    service
    (makeSymlink "${service}/bin" "app")
  ];
  config = {
    ...
  };
}
```

2 functions replaced 13 Dockerfiles

CI

```
m11s-image = cppImage m11s;
p12r-image = cppImage p12r;
emc2-image = cppImage emc2;
oms-image = cppImage oms;
mds-image = cppImage mds;
# ...
```

Binary caching using an S3 bucket

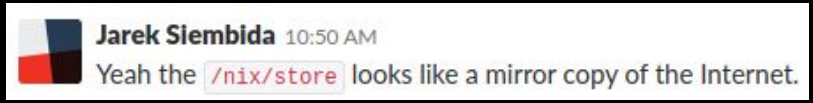
CI

```
# nix.conf
substituters = s3://smarkets-artefact-cache?region=eu-west-1 https://cache.nixos.org
```

Binary caching using an S3 bucket

CI

```
# nix.conf
substituters = s3://smarkets-artefact-cache?region=eu-west-1 https://cache.nixos.org
```



...and 300 LoC of Python

CI

```
usage: cache-upload [-h] [--everything] [--all-images] [--all-packages]

optional arguments:
  -h, --help          show this help message and exit
  --everything        Build everything: cache all packages and upload all
                      images to docker registry. It will try to avoid extra
                      work by checking what was uploaded last time. This is
                      what CI uses.
  --all-images        Build and upload all images to docker registry (release).
  --all-packages      Build and cache all packages (debug + release). It will
                      try to avoid extra work by checking what was uploaded
                      last time.
```

Running integration tests locally is as easy as

CI

```
./ci.sh
```

Running integration tests locally is as easy as

CI

```
#!/usr/bin/env nix-shell
#!nix-shell --expr "(import ../. {}).callPackage ./ci.nix {}" -i bash

# remove temp directory, kill processes/containers started there
cleanup ci-tmp

mkdir ci-tmp && cd ci-tmp && eval "$buildPhase" && python ci.py "$@"
```

Running integration tests locally is as easy as

CI

```
#!/usr/bin/env nix-shell
#!nix-shell --expr "(import ../. {}).callPackage ./ci.nix {}" -i bash

# remove temp directory, kill processes/containers started there
cleanup ci-tmp

mkdir ci-tmp && cd ci-tmp && eval "$buildPhase" && python ci.py "$@"
```

Conclusion

- Technologically Nix is decades ahead of docker, ansible, pip, [c]make, etc.
- Only way to dependably, declaratively and reproducible specify, compile and provision cross-lang artefacts
- Super fast and efficient compared to k8s/docker
- However, docker takes 30mins to be productive (regret only sets in days later at the first badly cached apt-get --upgrade)
- Nix has a steep learning curve
- Needs work on social acceptance (and training)