BUILDING ALICE SOFTWARE STACK

Giulio Eulisse (CERN)

END-TO-END SOLUTION

ALICE build infrastructure

In the last 3 months I helped revamping the infrastructure and build tool for the ALICE software stack. ***In production*** *since over one month. Old setup has been disbanded. Actual builds are driven by an in house tool which I called* ***aliBuild.*** *Testbed for some of the ideas I've been circulating since a while.*

I am not here to sell a tool

I'm here to present use-cases which I know are required for efficiently handling Continuos Integration of the software stacks of at least two experiments and how things were implemented in aliBuild. For what I am concerned the deliverable is having automated builds twice per day, pull request testing and some viable development environment for externals, not a build tool.

Reduce migration cost over "One tool shall rule them all"

I think we should try to harmonise the things we agree on rather than disagree on the things we do differently. We should reduce friction and increase reusability between the different tools out there, rather than expect one single tool will be able to solve all of our problems.

Jenkins

search

oulisse

Jenkins ▸

ENABLE AUTO REFRESH

- New Item
- People
- Build History
- Manage Jenkins
- Credentials
- My Views
- Disk usage
- Scriptler

Build Queue

No builds in the queue.

Build Executor Status

master

1 Idle

2 Idle

mesos-jenkins-0542c9f6-4d7b-4260-8392-c9ef2a831810 (offline)

mesos-jenkins-1ed29f28-ac22-44f9-9198-416e2edab5d8

1 generate-web-pages #25880

mesos-jenkins-213002e1-f38f-4fb0-a15f-3cc9fc9eee69

1 build-any-ib #2068 - aliroot-test slc7_x86-64

mesos-jenkins-66ab35c6-0bf7-41ae-87e9-76e180580728 (offline)

add description

All | Build results | My build view | +

S	W	Name ↓	Last Success	Last Failure	Last Duration
		build-any-ib	5 hr 31 min - #2063 - aliroot slc7_x86-64	5 hr 30 min - #2067 - o2 slc6_x86-64	11 min
		daily-aliphysics	45 min - #77 - AliPhysics slc5_x86-64	N/A	22 min
		generate-web-pages	11 min - #25877	1 min 12 sec - #25879	2 min 41 sec
		more-mesos-tests	15 days - #6	N/A	0.12 sec
		run-any-tests	5 hr 19 min - #1309	5 hr 19 min - #1308	1 min 8 sec
		schedule-all-ib	5 hr 35 min - #218	1 day 17 hr - #215	3 min 17 sec
		test-cern-git	14 min - #19822	4 hr 59 min - #19803	1.4 sec
		test-mesos	13 days - #68	N/A	1 min 43 sec
		update-git-branches	4 min 56 sec - #1843	3 hr 5 min - #1840	13 sec

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Over 2K build jobs in the last 3 months, 4 different architectures plus special builds.

Provisioning & scheduling

openstack™

Apache MESOS™

Configuration management / deployment

puppet labs®

ANSIBLE

docker

Continuous integration

Jenkins

Log parsing & mining

elasticsearch

Monitoring & Results

MonALISA
MONitoring Agents using a Large Integrated Services Architecture

kibana

CDash

BUILD TOOL: ALIBUILD

Open Source

*Tool itself can be found at **https://github.com/alisw/alibuild**, actual recipes to build externals are at **https://github.com/alisw/alidist**.*

Standalone

*Python is the only dependency. In particular, compared to cmsBuild, **no RPM and APT**.*

No magic, compact, maintainable

*Tool itself is **527 SLOCs of Python** + **116 SLOCs of Bash**. Recipes are simple Bash scripts with a YAML header. Challenge for Go developers: rewrite it in Go with less SLOCs. :-)*

Git based workflow

Configuration management happens in git. Carbon-copy of what cmsBuild does. IMHO, having different configurations managed as files, rather than branches, makes comparing them more difficult.

Not a CMake / SCRAM / make / (pick your favourite tool) replacement

INSTANT GRATIFICATION

```
git clone https://github.com/alisw/alibuild

git clone https://github.com/alisw/alidist

alibuild/aliBuild -d -j 40 -a slc7_x86-64 build ROOT
```

Any resemblance to other experiments naming conventions is purely fictional.

RECIPE EXAMPLE

YAML formatted metadata at the top.

```
package: O2
version: %(commit_hash)s
requires:
  - FairRoot
  - AliRoot
build_requires:
  - CMake
source: https://github.com/AliceO2Group/AliceO2
tag: master
---

cmake $SOURCEDIR -D$CMAKE_INSTALL_PREFIX=$INSTALLROOT
make -j 20
make install
```

RECIPE EXAMPLE

Bash recipe at the bottom. ***Conventions*** *over template magic / special languages.*

```
package: O2
version: %(commit_hash)s
requires:
  - FairRoot
  - AliRoot
build_requires:
  - CMake
source: https://github.com/AliceO2Group/AliceO2
tag: master
---

cmake $SOURCEDIR -D$CMAKE_INSTALL_PREFIX=$INSTALLROOT
make -j 20
make install
```

CONVENTIONS

INSTALLROOT: *the installation prefix. The build tool will create an archive based on the sole content of this directory.*

PKGNAME: *name of the current package.*

PKGVERSION: *package version, as defined in the recipe's version: field.*

PKGREVISION: *the "build iteration", automatically incremented by the build script.*

PKGHASH: *SHA1 checksum of the recipe.*

ARCHITECTURE: *an arbitrary string summarising the current build platform.*

GIT_TAG: *the Git reference to checkout.*

JOBS: *number of parallel jobs to use during compilation.*

BUILDDIR: *the working directory. E.g. the directory from where you invoke cmake. You should not write files outside this directory.*

BUILD_ROOT: *it contains BUILDDIR and the logfile for the build*

CONFIG_DIR: *directory containing all the build recipes.*

SOURCEDIR: *where the sources are cloned / unpacked.*

CONVENTIONS

For each package <PACKAGE>

<PACKAGE>_ROOT: *package installation directory.*
<PACKAGE>_VERSION: *package version.*
<PACKAGE>_REVISION: *package build number.*
<PACKAGE>_HASH: *hash of the recipe used to build the package.*

Full description of the build protocol I use:

https://github.com/alisw/alibuild#the-body

RECIPE EXAMPLE

Sources are always expected in git repositories. It simplifies source fetching logic a lot, allows automatic rebuilds when tip of a given branch changes.

```
package: O2
version: %(commit_hash)s
requires:
  - FairRoot
  - AliRoot
build_requires:
  - CMake
source: https://github.com/AliceO2Group/AliceO2
tag: master
---

cmake $SOURCEDIR -D$CMAKE_INSTALL_PREFIX=$INSTALLROOT
make -j 20
make install
```

DEPENDENCIES HANDLING

Consistent builds

When you change something in the recipe or in the tool itself, it notices and acts accordingly on a subsequent build of the tool and it's dependencies. E.g. if you change ROOT recipe and try to rebuild AliRoot, it will notice.

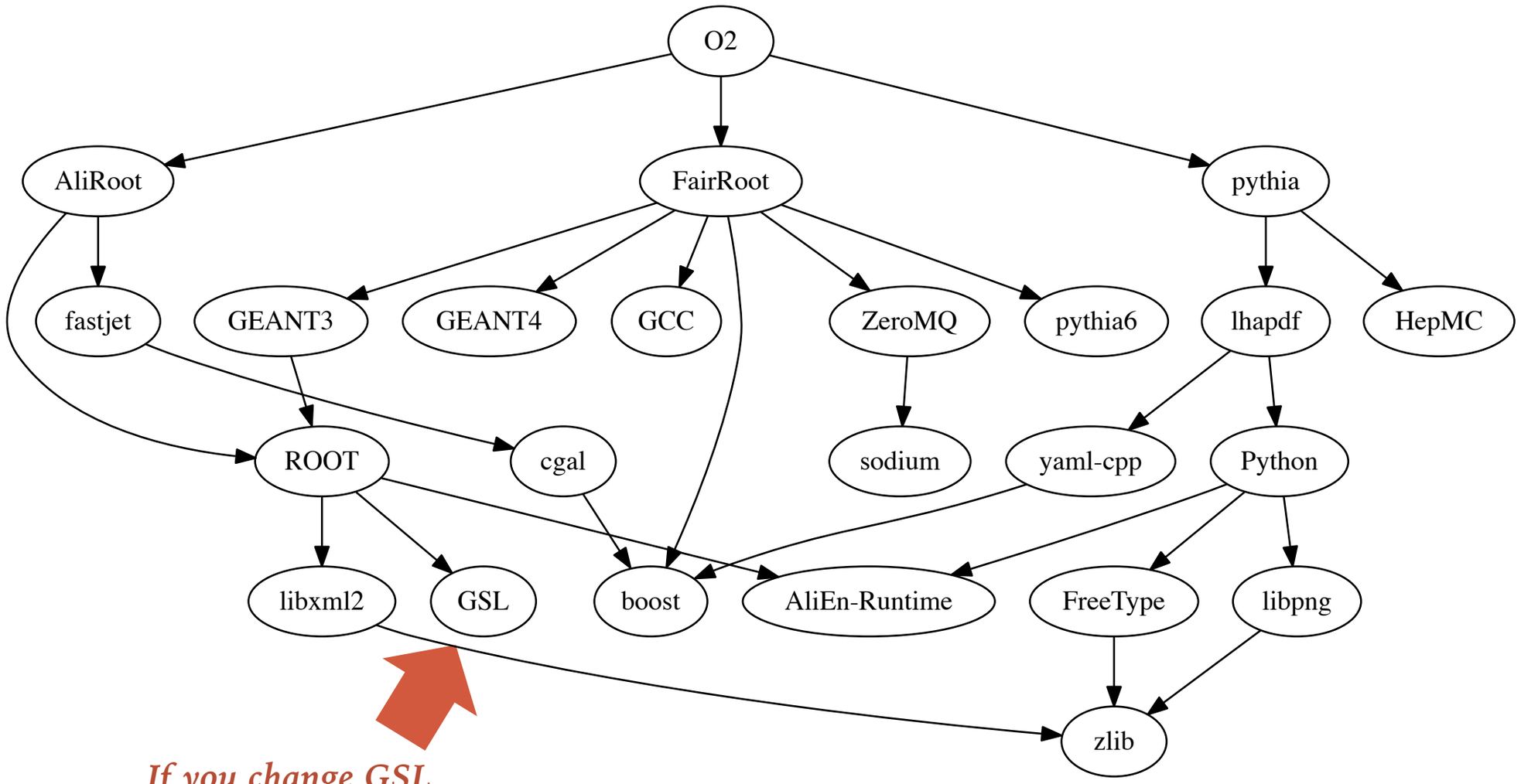
Flexible (and correct) handling of dependencies

Topological sort of the dependency graph for correct build order, even in the case of implicit dependencies. Run-time and build-time dependencies. Ability to disable dependencies. Platform (and soon Experiment) specific dependencies. Parallel installations of externals.

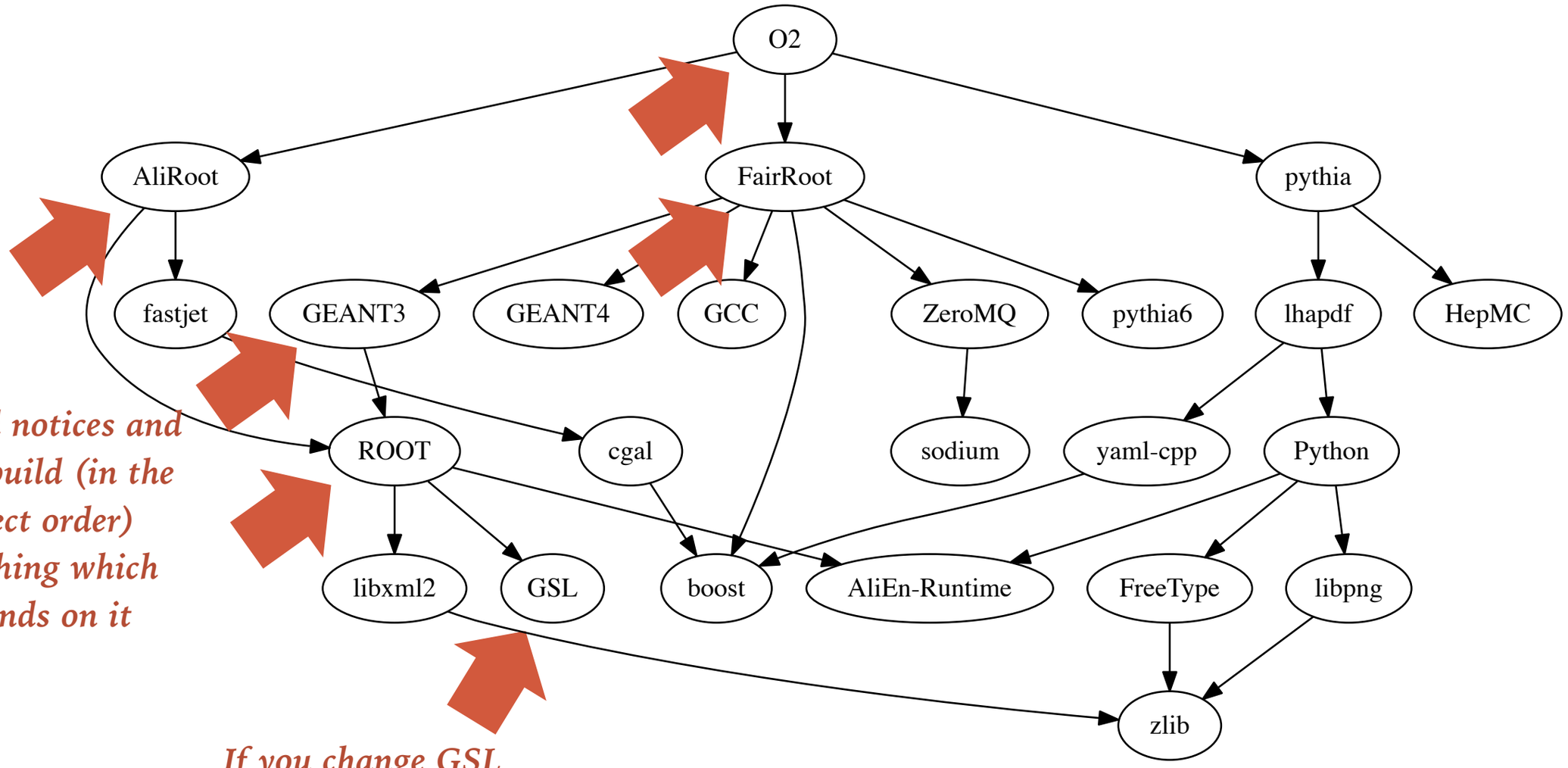
Parallel installation

The requirement to have one common namespace for all our builds is not going away any time soon. Same for the ability to reuse dependencies between different builds.

CONSISTENT BUILDS



CONSISTENT BUILDS



aliBuild notices and will rebuild (in the correct order) everything which depends on it

If you change GSL recipe...

PARALLEL INSTALLATIONS

```
../osx_x86-64/
        ../AliRoot/
                ../v1-1
                ../v2-1
                ../latest
        ../boost/
                ../latest
                ../v1.57.0-1
                ../v1.59.0-1
        ../fastjet/
                ../latest
                ../v3.1.3_1.017-1
                ../v3.1.3_1.017-2
                ../v3.1.3_1.020-1
../slc7_x86-64/...
```

Usual:

`<architecture>/<package>/<version>`

hierarchies.

Architecture is just a string, no platform auto-detection. It identifies the build host, not the installation or runtime requirements.

Changes in build recipes result in different "revisions".

BINARY REPOSITORY

Reuse builds

Just like cmsBuild packages built by one builder can be reused by other builders, ***even on a separate machine.*** *This comes handy when you cannot guarantee that you will always rebuild on the same machine. Compared to cmsBuild there are some differences:*

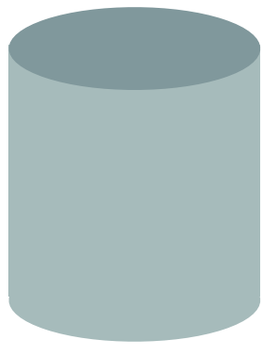
- *Binary packages are* ***standard tarballs****, not RPMs.*
- *Only rsync and some smart directory structure are needed for the object store.* ***No special APT repository needed.***
- *Packages are uploaded one by one, not in bunches. Most likely slower than cmsBuild, but much simpler as DB consistency is there without need for transactions.*

BINARY REPOSITORY

time

Build ROOT 6.X.Y-1

Build package is uploaded to store



Later rebuilds of the same recipe (on a different machine / docker container) do not rebuild but download the binary package from the store.

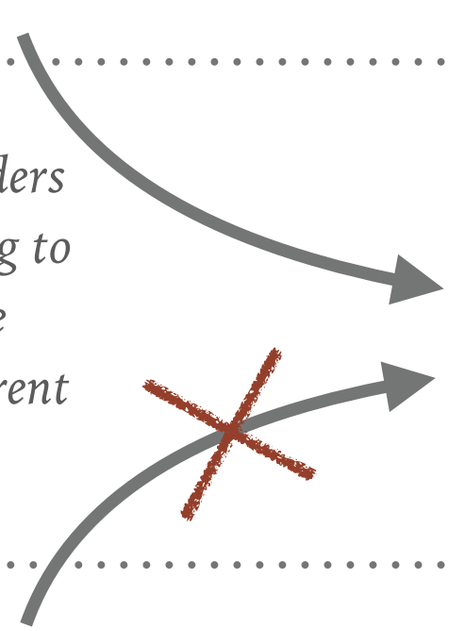
Reuse ROOT 6.X.Y-1

BINARY REPOSITORY

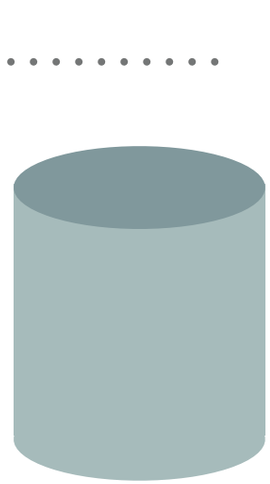
time

Build ROOT 6.X.Y-1
using recipe A

Two different builders might end up trying to upload the same package from different recipes.



Only the first one succeeds, the other one does not rebuild but repackages everything and tries again.



Build ROOT 6.X.Y-1
using recipe B

Repackage to
ROOT 6.X.Y-2

CVMFS / YUM / APT SUPPORT

time

Build ROOT 6.X.Y-1

Build package is uploaded to store

TARS

*Tarballs are converted to RPMs / DPKG via FPM, store metadata is used to construct a **standard** yum / apt repository*

aliPublish

CVMFS

YUM / RPMS

APT / DEBS

DEVELOPER MODE

Inspired by Chris Jones rant on cmsBuild. aliBuild can pick up sources from local checkouts for the builds. After the first build, one can go inside the build directory and type "make install". Handy for those who need to develop externals while improving the application code (e.g. multithreading fixes).

```
git clone https://github.com/alisw/alibuild
git clone https://github.com/alisw/alidist
git clone https://github.com/root-mirror/root ROOT
alibuild/aliBuild -a slc7_x86-64 --devel ROOT build AliRoot
...
cd sw/BUILD/ROOT-latest/ROOT
make install
```

DOCKER SUPPORT

Simplify cross platform builds

Handy for cross platform builds, i.e. using your Mac laptop to test builds in the slc7 environment. Just add "--docker" to the command line and the build will happen inside a container matching the provided architecture (another reason for ***NOT*** *detecting the local architecture).*

Create docker containers (not implemented)

It's trivial to extend the above to allow building and uploading of the containers with the results of the build itself.

ARCHITECTURE CUSTOMIZATIONS

$ARCHITECTURE

The system exports the command line provided architecture to the recipes as an environment variable.

Architecture specific dependencies

aliBuild supports architecture specific dependencies in the YAML preamble by adding a regular expression which needs to match for the requirements to be valid. E.g.:

```
name: AliRoot-tests
requires:
  - AliRoot
  - IgProf:slc7.*
```

You can then use `$<package>_ROOT` to detect if the dependency was included or not.

EXPERIMENT CUSTOMIZATIONS

aliBuild... alfaBuild... anyBuild!

*You can get experiment specific customisations (at the moment limited to a few details, like the name of the project hosting recipes) by simply using a symlink with the correct name. E.g. GSI people are experimenting with "**alfaBuild**" which uses the "**alfadist**" repository (1 week of work).*

Build customizations (idea)

*Build customisations can be driven the same way as the architecture ones. A **$FLAVOUR** can be defined as part of the build environment, depending on the tool name:*

E.g.: "aliBuild" ⇒ "FLAVOUR=ali", "alfaBuild" ⇒ "FLAVOUR=alfa".

```
name: ROOT
requires:
  - Alien: flavour=ali
  - Python: flavour=(lhcb|atlas)
```

HOW TO USE

To build a package:

```
git clone https://github.com/alisw/alibuild.git
git clone https://github.com/alisw/alidist.git
alibuild/aliBuild -d -a slc7_x86-64 -j 16 build AliRoot
```

To build a package in developer mode:

```
git clone https://github.com/root-mirror/ROOT
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --devel ROOT build AliRoot
```

To build a package in docker mode:

```
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --docker build AliRoot
```

To disable a package (and drop all its dependencies):

```
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --disable GEANT4 build O2
alibuild/aliBuild -d -a slc7_x86-64 -j 16 --disable simulation build O2
```

SOURCECODE HANDLING

Git(hub/lab) based

The tool handles directly git repositories only. This simplifies enormously the code which takes care or managing the sources and provides nice, uniform, web based views.

Benefits

- Support for "moveable" builds without extra code.

- Support for changing repository without rebuilding (assuming the commit hash is the same).

- Easy backup / proxying / mirroring of sources. Fast downloads for daily builds if local "reference" clone is available.

SOURCECODE AND PATCHES

Patches are inevitable

Some bugfixes just cannot wait for the next ROOT release. Sometimes ZeroMQ does not compile on Mac and we are the first ones to find out (e.g. yours truly: https://github.com/zeromq/libzmq/pull/1483). The ***goal is to simplify contributing upstream, not to fork****.*

Policy on how to handle external sources

Policy over tools. Current one I wrote and we use:

https://github.com/alisw/alidist#guidelines-for-handling-externals-sources

(again carbon copy of CMS one).

Policy is NOT mandatory

Of course there are cases where we cannot redistribute sources. Preferred option would be use a protected git repository, if not even is an option, "curl inside the recipe" is of course not forbidden. You simply lose the benefits of dealing with git.

PROPOSED POLICY

If Sources hosted on git, used unmodified:

- *Directly refer to the Upstream repository.*

If Sources hosted on git, need patching:

- *Fork / mirror the **relevant parts** of the Upstream repository*
- *Pick a tag / commit which will be used as `<fork-point>`, create a branch "`alice/<fork-point>`". Apply Patches on top.*

If Sources are not hosted on git:

- *Create an ALICE mirror in some agreed location, e.g. https://github.com/alisw/*
- *Import a tar-ball with one Upstream version, commit it to git, tag it with the original tag.*
- *Create a branch "`alice/<fork-point>`" and apply Patches on top.*

PROPOSED POLICY: BENEFITS

Sources history can be browsed:

https://github.com/alisw/root

Patched Sources can be pin-pointed:

https://github.com/alisw/root/tree/alice/v5-34-30

Upstream Sources can be pin-pointed:

https://github.com/alisw/root/tree/v5-34-30

Changes between w.r.t. Upstream can be diff-ed

https://github.com/alisw/root/compare/v5-34-30...alice/v5-34-30

PROPOSED POLICY: BENEFITS

Sources history can be browsed:

https://github.com/alisw/geant4/

Patched Sources can be pin-pointed:

https://github.com/alisw/geant4/tree/alice/v4.10.01.p02

Upstream Sources can be pin-pointed:

https://github.com/alisw/geant4/tree/v4.10.01.p02

Changes between w.r.t. Upstream can be diff-ed:

https://github.com/alisw/geant4/compare/v4.10.01.p02...alice/v4.10.01.p02

PERSONAL NOTES

Build protocol

*I still think that rather then worrying about the tool we should concentrate of making recipes easily usable by all our tools. Current "**build protocol**" for alibuild recipes fully described at*

https://github.com/alisw/alibuild#recipes-format

Just noticed my conventions strikingly similar to Conda's, I might even adapt a few things to make it exactly the same where possible (e.g. PKGVERSION => PKG_VERSION).

Sources protocol

I'm still convinced that embracing Git / Github for managing sources and patching is the way to go.

Different problems, different tools

IMHO tools like Homebrew, Conda, are fantastic solutions for a different problem, i.e. desktop installations. Again, finding a way to easily create Conda / Homebrew recipes from our own is more interesting, IMHO, than trying to stretch tools to our use cases.