ALIBUILD
2.5 YEARS AFTER

Giulio Eulisse (CERN)

WHAT IS ALIBUILD?

aliBuild (http://alisw.github.io/alibuild/) is the in-house packaging / build / setup tool for ALICE Experiment. It's actually generic enough to be considered a standalone product.

It's used to build the whole software stack and to prepare the environment for the user work area. Target audience includes the end-user, not merely the librarian.

It was already presented to this audience back in 2015, see https://indico.cern.ch/event/457365/contributions/1957114/

TIMELINE

*aliBuild was started in **July 2015** as part of an effort to modernise ALICE build infrastructure.*

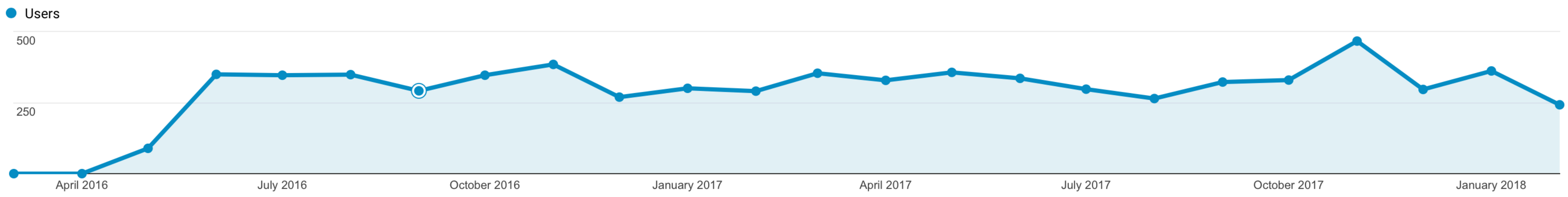
*By **September 2015**, the whole infrastructure entered into production and the old system was switched off.*

*By **May 2016**, aliBuild became only end-user (i.e. analysis user) tool for setting up their build environment.*

***Effort needed**: 2.5 months for the tool and the initial recipes, 2.5 months for setting up the infrastructure (Mesos, Jenkins, deployment to CVMFS, etc). Continuous "central" effort of 1 FTE (aggregated) to improve build recipes, fix bugs, improve the tool, baby-sit infrastructure.*

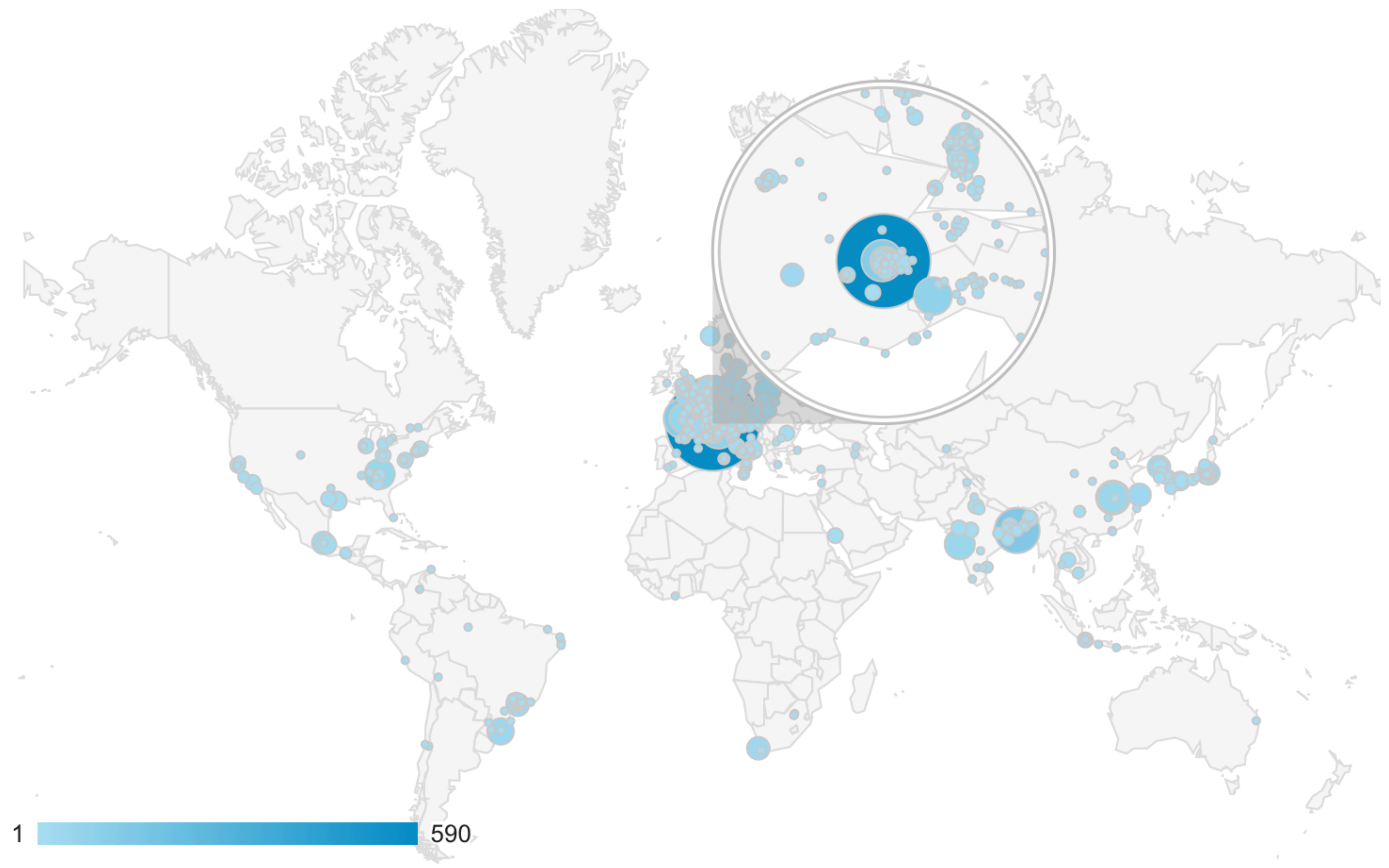
*Take away message: **the problem is not the tool**, and to certain extent, at least for big experiments, not even the maintenance cost, if you do the right choices. What follows is my personal take on what I would redo and what I would not redo.*

USAGE

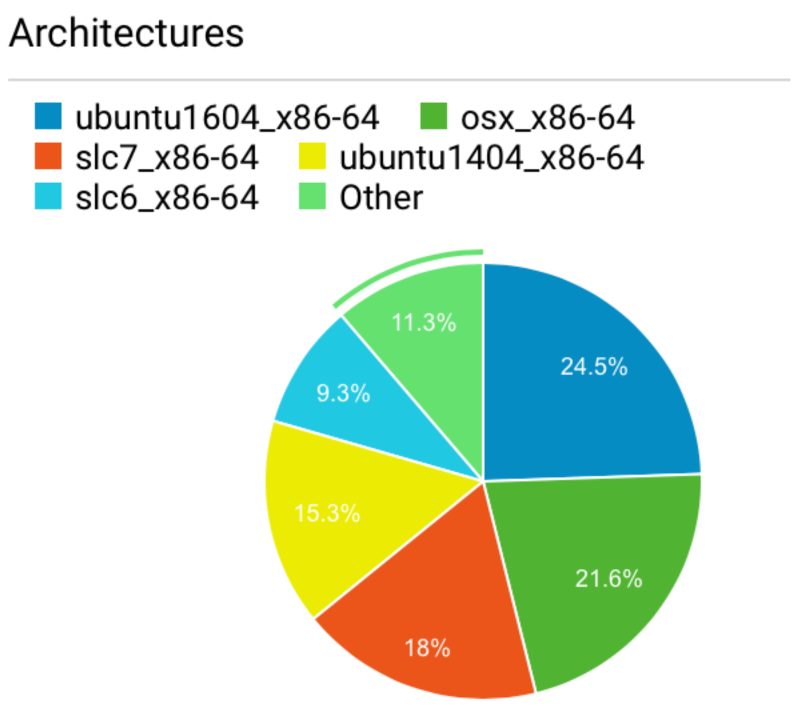


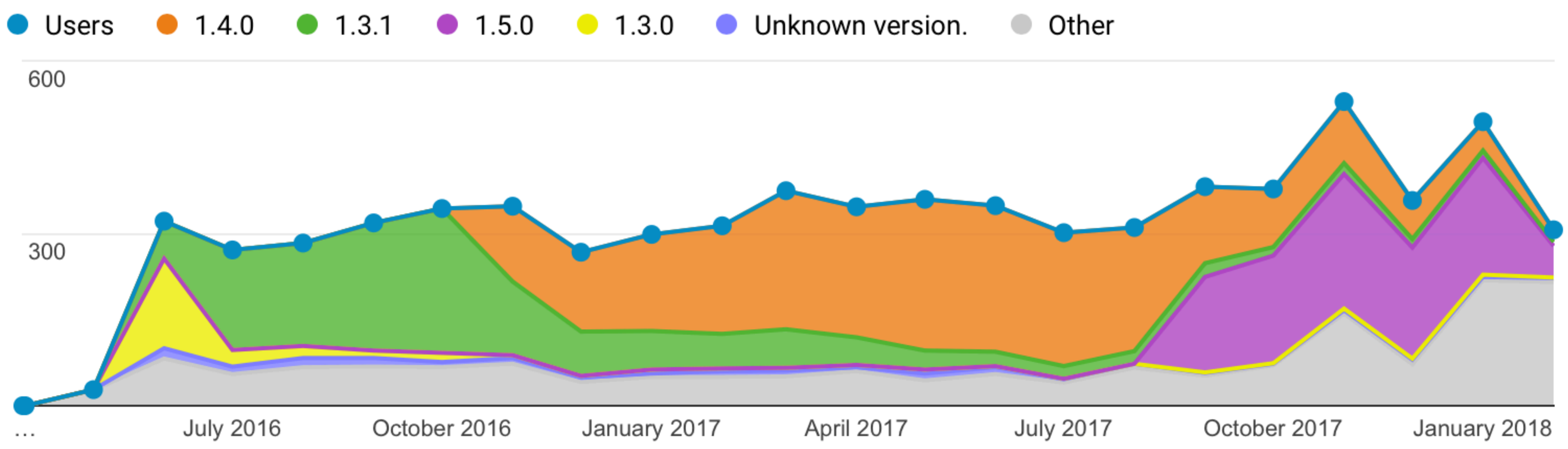
"Users" mean separate home directories.

A given user might invoke aliBuild multiple times because he wants to upgrade to a new AliRoot version on his laptop.



ARCHITECTURES AND VERSIONS





THE RIGHT CHOICES: KEEP IT SIMPLE

Git: *Relying on git / Github for the sources simplifies fetch / rebuild logic enormously and provides a nice way to patch externals.*

- *If the project has a git repository use it.*
- *If the project does not have a git repository, import the tarballs of the versions you need in your own repo.*

BASH: *Relying on bash for the actual build recipes allowed a few dozens of non-experts to contribute. As much as I love Nix, for example, that would have not been possible there.*

YAML: *aliBuild metadata is in YAML, together with the choice above, this makes it more widely accessible than using a custom language for the recipes, while keeping it easy to parse (Hello, cmsBuild!).*

Conventions over template magic: *the recipes have no magic template language behind, simply conventions on the environment, e.g. $BUILDDIR.*

Use tarballs as file format: *RPMS and DPKGs are generated in a subsequent step, no direct integration with those tools.*

Being able to read recipes, tell what they are doing immediately and even to cut and paste snippets into a shell is frankly priceless. We do not need a single tool, we need to be able to read easily each other recipes.

THE RIGHT CHOICES: DEPENDENCY MANAGEMENT

No Central "Configuration": *configuration arises from the set of dependencies, allowing us to threat recipes in isolation and easily customise behaviour locally.*

Avoid recipe inheritance: *using inheritance to save a few keystrokes when doing "configure, make, make install" obfuscates code and does not make it more maintainable.*

CSS-like customization of recipes: *aliBuild allows customising builds via a CSS-like set of modifier grouped in a so called "default". This turned out to be a much better way to scale similar configurations than using multiple branches. E.g.: "--defaults root6", "--defaults daq".*

THE RIGHT CHOICES: AOB

A setup tool: *aliBuild can be used to build the whole stack, but also allows people to simply setup their workarea and continue from there using standard tools like make and module. Once the workarea is ready, users can still use aliBuild to build or switch to simply do:*

```
cd $WORKAREA; <modify file>; make install; <run>
```

Embrace homebrew, Redhat-devtools, Ubuntu: *when targeting end- users' installations, people expect as much as possible of their system tools to be reused. Maybe we should put more focus in contributing natively to those de-facto standards, rather than trying to build the whole stack on our own.*

Integration with Google Analytics: *(opt-in) information about your audience.*

Easy development builds: *locally cloned sources get preferred over whatever is specified in the build recipes. Useful if you are a contributor to ROOT / FairRoot / AliceO2 at the same time.*

THINGS I WOULD IMPROVE, IF I HAD TIME

Python: *by far the greatest source of runtime errors came from interaction with the user python installation (Hello, Conda! Hello Python 3). Yes @sbinet, I would rewrite it in Go if I had the time. ;-)*

Distributing cached build artefacts to end users: *while the rsync based cache works very well for CI infrastructure, I would probably rewrite it to use e.g. S3 in order to scale to ~500 users.*

Generate "module" files automatically: *right now each recipe has a big blob at the end which creates the environment file to be user by the "module" package. Automation there might actually*

Rethink docker integration: *rather than natively supporting docker from aliBuild, I would probably write a custom provisioner for something like* ***https://packer.io***

Embrace CVMFS more: *just like laptop users expect their system installation to be used without them noticing, lxplus users expects CVMFS integration. Having more integrated support for it in aliBuild is a long-standing feature request.*

nix-style overlays: *if you have a PhD in CS, Nix rocks. IMHO, there is no doubt about it being the most advanced, elegant and powerful build system I've ever seen. One feature from it I might end up stealing is the ability to have "overlay" recipe repositories that have precedence over the standard one.*

THINGS I WOULD DO, IF I WAS COMPLETELY BORED

- Assuming GVFS was not yet available on Linux / Mac: write an automated converter from aliBuild to nix-overlays and force the whole collaboration to move to Nix.
- Assuming GVFS was available on Linux / Mac: I would write an automated converter to Google bazel.io, ditch cmake and simply use bazel for everything.