ALICE & LIM

Giulio Eulisse

CONSTRAINTS

Ownership

ALICE requires full ownership of the infrastructure and process which is critical for data-taking.

Github

ALICE hosts software on Github, so Github integration is a must.

End-user "laptop" support

*Traditionally ALICE users have always being able to develop easily on their laptops. While we like a lot managed infrastructures (analysis trains!) we will always need to support the "**laptop**" **usecase**, without need for a shared filesystem.*

LCG RELEASES AND NIGHTLIES

Production usecase

*Due to the fact that the release process is integrated with **AliEn** and the physics in general, **we do not foresee moving to LCG releases for production**, not in the medium term at least.*

End user usecase

*We already have working integration of our stack on top of LCG releases for **SWAN (D. Berzano - 2016),** foreseen to be used for QA activities (M. Ivanov).*

*We have some ongoing work to make sure we can use LCG views to **speedup user builds on lxplus** and alike resources.*

Customisation

*We need to maintain ownership on parts of our software stack configuration (e.g. ROOT, GEANT4). A mechanism to perform **customisations of a predefined configuration without any interaction / negotiation** with the outside world is mandatory. A mechanism like Nix "overlays" or AliBuild "defaults" is a requirement for our configuration needs.*

DOCKER DISTRIBUTIONS

Container usage

Just like every other experiments we do use containers for things like builds or release validation and we are experimenting to use them in production jobs (e.g. Singularity@GSI, Docker@HLT).

Not a software distribution mechanism!

CVMFS is and remains our baseline *for production installations. We can create containers with our software inside, however we will always prefer mounting CVMFS inside the container and use it to deliver software installations.*

BUILD INFRASTRUCTURE AND TOOLS

Jenkins

*Used for **building releases** and some ancillary tasks. **Maintenance / load is low.** We do not see any advantage to move to a different instance in the medium term. We might consider Jenkins-X (because of the Kubernetes integration).*

CI

*Builds are currently performed by a set of **stateful microservices**. This is to make sure we can reuse previous builds and do incremental rebuilds when testings PRs. We might want to consider moving from Mesos + Apache Aurora for the deployment to Openstack + Kubernetes.*

Packaging tools

*AliBuild serves well the need of the whole collaboration with little / no effort by now. We foresee some minor developments to fully support **LCG View-like environments as "system view"**.*

AFS

No need.

FURTHER COLLABORATION IDEAS

LIM build infrastructure as a Github App

If the LIM infrastructure could be exploited in a way similar to Travis, but with local hardware, I suspect we could exploit it more than with current approach (on premises Travis Enterprise, of course would be best).

LIM Artefact store

Managing logs, tarballs repository is by far our biggest operational overhead for the builds. Having a proper artefact store, centrally managed, would be something we would look into especially for CI.

Porting efforts

We are always very eager to hear about porting effort, e.g. clang, ARM or Mac. Separating that part of the meeting from the discussion about what goes into the release would probably be useful.

Mac / ARM hardware

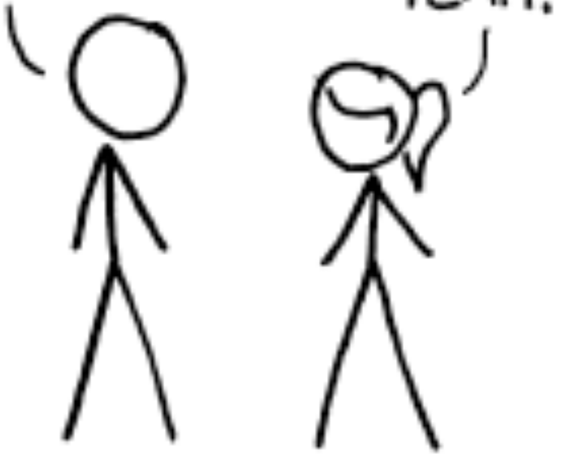
Having the possibility to use centralised Mac hardware would be greatly appreciated.

HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.