Building, testing and distributing common software for the LHC experiments

Patricia Mendez Lorenzo
for the EP-SFT/SPI project
Overview

This talk relates to this part

Core libraries and external packages non-HEP

Applications
Event, Detector, Calibration
Exp. specific packages
Simulation, Data Mgt.
Distributed analysis....

And here????
What does the baseline contain? → ca. 400 packages

Including:

- Common EP-SFT projects: ROOT and Geant4
- 55 MC generators and 18 Grid MW packages
- Common tools: CMake, gcc, clang...

How do we provide it? → in LCG releases: LCG_XX

<table>
<thead>
<tr>
<th>Full release volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>~500k directories</td>
</tr>
<tr>
<td>~65k links</td>
</tr>
</tbody>
</table>

- **Packaged** → EOS: /eos/project/l/lcg/www/lcgpackages
  - https://lcgpackages.web.cern.ch/lcgpackages
  - tar files, RPMS and docker containers

- **Expanded** → CVMFS: /cvmfs/sft.cern.ch/lcg/releases
  - 10X2 platforms for both python 2 and 3
Our platform declaration

Platform = `{ARCH}-${OS}-${COMPILER}-${BUILD_TYPE}`

(Definition agreed by the HSF Platform naming convention)

<table>
<thead>
<tr>
<th>Compilers</th>
<th>Architectures (ARCH)</th>
<th>Operating Systems (OS)</th>
<th>Build type</th>
</tr>
</thead>
<tbody>
<tr>
<td>● gcc62</td>
<td>● x86_64(+avx2+fma)</td>
<td>● slc6</td>
<td>● Release</td>
</tr>
<tr>
<td>● gcc7</td>
<td></td>
<td>● Centos7</td>
<td>● Debug</td>
</tr>
<tr>
<td>● gcc8</td>
<td></td>
<td>● Ubuntu16</td>
<td></td>
</tr>
<tr>
<td>● clang60</td>
<td></td>
<td>● Ubuntu18</td>
<td></td>
</tr>
<tr>
<td>● native (ubuntu)</td>
<td></td>
<td>● Mac</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● arm64</td>
<td></td>
</tr>
</tbody>
</table>

(Under testing in the so-called nightly builds)

See G. Amadio, Robust Linux Binaries, Hall 3 12th July 10:30
Our challenges

- Reproducibility
  - Sources stored in EOS independent of changes in code
  - Specific branches per release
  - Handling of the “internal” packages dependencies

- Scalability
  - Build software structure based on specific cmake modules handling dependencies
  - Modular approach for the implementation of new packages and versions

- Relocation
  - Ability to install packages in different $PREFIXes

- Automation
Technical aspects of the project
LCGCMAKE: the core

Responsible of the individual packages build recipes and the default packaging

- Build and testing infrastructure CMAKE based
  - Build specifications included in platform-independent list files
  - Modular and scalable system

- Based on “ExternalProject” CMAKE module implementation enabling:
  - Builds from external software sources

```c
# ------ EXAMPLE OF a PACKAGE BUILD
LCGPackage_Add(
  mpfi
  URL ${GenURL}/mpfi-
  ${mpfi_native_version}.tar.gz
  CONFIGURE_COMMAND ./configure --
  prefix=<INSTALL_DIR> --with-
  gmp=${gmp_home} --with-
  mpfr=${mpfr_home}
  BUILD_COMMAND make
  INSTALL_COMMAND make install
  BUILD_IN_SOURCE 1
  DEPENDS gmp mpfr)
```
The system elements

1) Sources download

2) Build orders per package Machinery

Files including declaration of packages to build and the versions

 Packages creation in tar files

 Dependencies text (.txt)

Separated build execution for every platform

CDASH publication

JENKINS

http:// (*)

EOS (**)

GITLAB repo

LCGCMAKE

Toolset.cmake
Lifecycle of a new package

1. Local tests in build nodes
2. Experimental slots
3. Incremental nightly builds
4. Releases LCG_XX and LCG_XXpython3 (Full build)
The packaging approach

- Default packaging: `.tgz files` created after the individual build of any package → Stored in EOS
  - Same structure for nightlies and releases
  - Specific dependencies `.txt` file available
  - Copy to EOS based on an incremental approach handled by the inclusion of a `HASH` parameter for each package (determined by the version-revision-dependencies)

- Releases only:
  - `RPMS` provided also in the case of releases by experiment demand → Stored in EOS
  - Flexible Provision of `docker containers` → Stored in EOS
    - Total or partial number of packages managed
Binaries distribution workflow

1st Job
- Build of the binaries in VM/dockers
- Creation of .tgz
- Copy to EOS

2nd Job
- Stratum0 are build nodes in Jenkins
- Download of .tgz from EOS
- Expansion and reallocation in CVMFS
  - Different mount point for releases/nightlies
  - Job granularity: Platform
Automation infrastructure

- Around 500 CPUs distributed among ubuntu/mac/slc6/centos6/fedora systems
- Docker containers (Centos7 VM) available for SLC6/Centos7/Ubuntu16/Ubuntu18/Fedora
- https://epsft-jenkins.cern.ch
Build results dashboard

http://cdash.cern.ch → accessible outside CERN
Publication of the releases

http://lcginfo.cern.ch/ → accessible outside CERN

## LCG Software Elements

<table>
<thead>
<tr>
<th>Tool</th>
<th>LCG Configuration 93</th>
<th>LCG Configuration 92</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boost</td>
<td>1.66.0</td>
<td>1.64.0</td>
</tr>
<tr>
<td>CLHEP</td>
<td>2.4.0.1</td>
<td>2.3.4.4</td>
</tr>
<tr>
<td>delphes</td>
<td>3.4.2pre12</td>
<td>3.4.0</td>
</tr>
</tbody>
</table>
Summary and Future

- The EP-SFT group has enabled a robust and scalable structure to provide hundreds of binaries to the experiments and users
  - Stable working bleeding edge available on a daily basis
  - Biggest effort in the last years: Ensure a high level of automation
  - Structure ready to escalate to a large range of compilers and OS
  - Support to three packaging structures: tar files, RPMS and containers

- Covering the needs of:
  - ATLAS, LHCb, SWAN, FCC, Spark, Hadoop, BE dept. and a large range of individual users

- Future:
  - Increase the level of validation
  - Provide pre-releases
  - Structure embedded on the HSF project as part of the future build and packaging