The LHCb Software Stack: and contributing to LHCb software

Andy Morris

Aix Marseille Univ, CNRS/IN2P3, CPPM, Marseille, France

Andy.M@cern.ch – [27/Nov/24]

What is the stack and LHCb software?

- In general a software stack represents a self-contained (and sometimes interdependent) set of software creating a platform for software to run on

- In LHCb this generally refers to all of our centralised software e.g.
  - The framework almost everything is based on – Gaudi
  - The description of LHCb's geometry and conditions, event model, … – LHCb
  - The functors and general reconstruction algorithms – Rec
  - The HLT1 framework (separate to Gaudi) and its execution – Allen
  - Configuration for HLT2 and sprucing – Moore
  - The nTupling software – DaVinci
  - The simulation framework – Gauss

What is the stack and LHCb software?

- During your time at LHCb you will likely contribute to a software project
  - As examples, they roughly fall into these categories

RTA	DPA	Simulation
Moore Allen Rec LHCb	Moore DaVinci Rec LHCb	Gauss LHCb

- The method by which these contributions enter production are:
  1. Check out the software using git
  2. In a new branch, make the changes you'd like to see
  3. Make a merge request into the project you'd like to edit

When to use the stack

- It is possible to checkout and build the entire stack – this is often overkill!
  - The full stack, when built, represents $\mathcal{O}(10\mathrm{GB})$ of storage space to fit
  - Depending on the machine used, it can take hours to build (even on the fastest machines it will be 40+ mins)!
- However if you are making several interdependent changes to multiple projects, it can then be necessary to check out the whole thing
  - E.g. You're adding a new functor to Rec, and then using this in a trigger line in Moore – this might be a time to check out the full stack
  - E.g.2 You're writing software which is dependent on another person's merge request, which itself touches many things – another reason

How to check out LHCb software – lb-dev

- To check out a specific project with the lhcb-software we have lb-dev!
  - E.g. for moore: `lb-dev --platform x86_64_v3-el9-gcc13+detdesc-opt+g Moore/v55r16p4 --name MooreDev`

    - lb-dev = check out a project
    - --platform = Specifying the binary tag to be used (more on this later)
    - The project to be installed and its version
    - --name MooreDev = The folder name for the dev area

- For changes within a single project e.g. to cuts in a trigger line, this is likely enough!

How to check out LHCb software – The stack

- To check out the full stack instructions may be followed from here -
  - https://gitlab.cern.ch/rmatev/lb-stack-setup/-/blob/master/README.md

- Not all the projects need to be built locally
  - It is possible to specify `cvmfsProjects` in the configuration which will take versioned builds of those projects from cvmfs – you won't need to build them locally, but they can't be modified

Platforms and Binary tags

- When it comes to LHCb software – portability is very important!
  - Sometimes CPUs will be available with x86 architecture, sometimes ARM architecture
  - The compiler to be used (gcc, clang, …) can be specified
  - The operating system needs to be specified – usually el9 very occasionally centos7
  - Sometimes GPUs will be available, sometimes not
  - Sometimes it’s best to have a debug build, other times an optimised build
  - Sometimes we need to use LHCb’s geometry and conditions from detdesc, other times dd4hep
    - These are two different ways to input the conditions and geometry:
    - In Run 3, dd4hep is used for data (this is the default), detdesc is used for MC

Nightly builds

- The nightly builds give a centralised place where we build the most recent versions of the codebase every day

- This both acts as a test that the software is working and also provides a centralised place where people can fetch the most up-to-date builds of the software
  - --nightly <day of the week> is an option which may be given to lb-dev to fetch the nightly build of a software project instead of a tagged version (e.g. vXrYpZ)

Nightly builds

lhcb-2024-patches/199 (2 days ago) prev

Test slot with patches for 2024 production stack

build 100% failing: 9 errors tests 100% ✔ 13465 | ✘ 493

Compare with previous build | Compare two slots | Browse artifacts

Examples of binary tags →

Software projects ↓

Project	Version	x86_64_v2-el9-gcc13-opt		x86_64_v2-el9-gcc13-dbg		x86_64_v3-el9-gcc13-opt+g		x86_64_v2-el9-clang16-opt		x86_64_v2-el9-clang16+detdesc-opt		x86_64_v3-el9-gcc12+cuda12_1-opt+g		x86_64_v2-el9-gcc13+detdesc-opt		x86_64_v2-el9-gcc13+detdesc-dbg		x86_64_v3-el9-gcc13+detdesc-opt+g	
		build	tests	build	tests	build	tests	build	tests	build	tests	build	tests	build	tests	build	tests	build	tests
DBASE	None																		
PARAM	None																		
LCG	105a																		
Gaudi	v38r1p1	0 / 0	300 / 0	0 / 0	300 / 0	0 / 0	300 / 0	0 / 0	300 / 0	0 / 0	300 / 0	0 / 0	300 / 0	0 / 0	299 / 1	0 / 0	300 / 0	0 / 0	300 / 0
Detector	2024-patches	0 / 0	71 / 0	0 / 0	71 / 0	0 / 0	71 / 0	0 / 0	71 / 0	0 / 0	71 / 0	0 / 0	71 / 0	0 / 0	71 / 0	0 / 0	71 / 0	0 / 0	71 / 0
LHCb	2024-patches	0 / 0	263 / 0	0 / 0	262 / 0	0 / 0	263 / 0	0 / 0	262 / 0	0 / 0	293 / 0	0 / 0	263 / 0	0 / 0	294 / 0	0 / 0	293 / 0	0 / 0	294 / 0
Lbcom	2024-patches	0 / 0	1 / 0	0 / 0	1 / 0	0 / 0	1 / 0	0 / 0	1 / 0	0 / 0	1 / 0	0 / 0	1 / 0	0 / 0	1 / 0	0 / 0	1 / 0	0 / 0	1 / 0
Rec	2024-patches	0 / 0	48 / 0	0 / 0	48 / 0	0 / 0	48 / 0	0 / 0	48 / 0	0 / 0	45 / 0	0 / 0	47 / 1	0 / 0	45 / 0	0 / 0	45 / 0	0 / 0	45 / 0
Allen	2024-patches	0 / 0	17 / 0	0 / 0	17 / 0	0 / 0	16 / 1	0 / 0	17 / 0	0 / 0	15 / 0	0 / 0	16 / 1	0 / 0	15 / 0	0 / 0	15 / 0	0 / 0	15 / 0
Moore	2024-patches	0 / 0	212 / 14	0 / 0	210 / 14	0 / 0	212 / 14	0 / 0	206 / 14	0 / 0	176 / 25	0 / 0	0 / 0	0 / 0	181 / 26	0 / 0	180 / 25	0 / 0	182 / 25
DaVinci	2024-patches	0 / 0	60 / 13	0 / 0	60 / 13	0 / 0	59 / 14	0 / 0	60 / 13	0 / 0	57 / 18	0 / 0	58 / 15	0 / 0	57 / 18	0 / 0	57 / 18	0 / 0	57 / 18
LHCbIntegrationTests	2024-patches	0 / 0	19 / 3	0 / 0	19 / 3	0 / 0	19 / 3	0 / 0	19 / 3	0 / 0	43 / 15	0 / 0	19 / 3	0 / 0	43 / 15	0 / 0	43 / 15	0 / 0	44 / 14
Online	master	0 / 0	346 / 0	0 / 0	346 / 0	0 / 0	345 / 1	0 / 0	346 / 0	0 / 0	346 / 0	0 / 0	346 / 0	0 / 0	346 / 0	0 / 0	346 / 0	0 / 0	346 / 0
Alignment	2024-patches	0 / 0	13 / 0	0 / 0	10 / 1	0 / 0	5 / 0	0 / 0	13 / 0	0 / 0	10 / 0	0 / 0	5 / 0	0 / 0	10 / 0	0 / 0	8 / 0	0 / 0	3 / 0
MooreOnline	2024-patches	0 / 0	37 / 1	0 / 0	36 / 2	0 / 0	37 / 1	0 / 0	37 / 1	0 / 0	23 / 0	0 / 0	29 / 5	0 / 0	23 / 0	0 / 0	23 / 0	0 / 0	23 / 0
Panoptes	2024-patches	0 / 0	9 / 0	0 / 0	9 / 0	0 / 0	9 / 0	0 / 0	9 / 0	0 / 0	3 / 0	0 / 0	4 / 5	0 / 0	3 / 0	0 / 0	3 / 0	0 / 0	3 / 0
Boole	master	0 / 6	0 / 9	0 / 6	0 / 9	0 / 6	0 / 9	0 / 7	0 / 9	0 / 9	0 / 14	1 / 6	0 / 9	0 / 8	0 / 14	0 / 8	0 / 14	0 / 8	0 / 14

Testing and QM tests

- When contributing to LHCb software, especially when adding new functionality to the code, it's often the case that we want to write automated tests for this
  - This will ensure that future changes don't break this earlier functionality
  - All new functionality should in principle come with an accompanying test!

- The way this is done in LHCb's stack is with QM-tests – however this is being depreciated in favour of instead using pytest

- These tests can be run with `make test`
  - Arguments can be specified to run specific tests only, multiple tests at once…

MRs and centralised testing

- Once happy with your code changes, a merge request (MR) can be put together to request it be added to the production codebase

- An MR contains your intended changes to the code as well as a description you write of what it does
  - In this description it's good to include as much information as possible e.g. for changes to the trigger, tests on throughput and bandwidth should be included

- Let's take a look at an RTA MR together:
  - https://gitlab.cern.ch/lhcb/Moore/-/merge_requests/3793

MRs and centralised testing

- For RTA MRs, once it has been given to the shifter, they may request a ci-test
  - This will recreate the nightly tests but including your changes building the relevant parts of the stack and checking nothing has broken for any platform
  - This will also run all of the tests for projects with the MR's changes as well as downstream projects
- It's possible to include the changes from multiple MRs at once if you have several interdependent ones
- It's also possible to request additional 'PR' tests to run on the output of the ci-test, typically testing the bandwidth and/or throughput of your changes, comparing it to the nightlies – these are specified in the MR's labels
  - Let's look at this in the same MR as before

The plan for today

1. Using lb-dev we'll check out Moore's Hlt library:
   - `lb-dev --platform x86_64_v3-el9-gcc13+detdesc-opt+g Moore/v55r16p4 --name MooreDev`
   - `cd MooreDev`
   - `git lb-use Moore`
   - `git lb-checkout Moore/v55r16p4 Hlt`
   - `make install`

Explaining the individual parts

- The parts in ‘step 1’ all do slightly different things:
  - lb-dev is creating the local directory and setting up some initial files
  - lb-use is actually checking out the core parts repository from gitlab
    - It knows where to get this from due to the setup done by lb-dev
  - lb-checkout is then adding the specific additional packages needed on top (this time Hlt, the code which configures HLT1*, HLT2 and sprucing)

The plan for today

2. Modifying the snippet found here*:
   - Test the bandwidth of a trigger line (try finding the one from your own analysis) using the instructions in the doc string
   - Change the selection in your trigger line and run the snippet again – comparing the bandwidths
     - If you don't know which trigger line to use, try "Hlt2QEE_DiMuonNoIP_massRange3", this can be found in `qee/dimuon_no_ip.py`

   - The lines may be found in:
     - `Hlt/Hlt2Conf/python/Hlt2Conf/lines`

*https://gitlab.cern.ch/-/snippets/3342