Open Source Best Practices: From Continuous Integration to Static Linters
Date: September 19, 2018
Presented by: Daniel Smith and Ben Pritchard (Molecular Sciences Software Institute)

Q: Can you put the links to the slides and recordings in this document?

A: Yes. The slides have been posted to the event page located at: https://www.exascaleproject.org/event/ci2sl/. We will also post the video and the Q&A transcript to that page when they are ready.

Q: I've been using CPPUTest. Should I switch?

A: Depends on your use case. CPPUTest is great for C programs and C++ programs that do not use C++11. Catch2 makes testing with C++11 and later quite easy, so projects with modern C++ might be better with Catch2. Always evaluate your options and the cost of switching.

Q: Do you know of any cloud-based CI services that include GPUs?

A: No free services that we are aware of. There are services out there but they are hundreds of dollars per month, this is mostly due to the fact that individual GPUs are generally more expensive than a given CPU and services like Travis are able to put 10-20 users per CPU so the overall cost of CPU CI is an order of magnitude cheaper. We often find heavy GPU users use local CI services like Jenkins or TeamCity to run CI on a box that they purchase and place in their local lab.

Q. The biggest problem I find with C++ based project development is getting the dependencies right - i mean installing the packages. Conda for python works pretty well but my C++ work lags due to it. Do you have any suggestions for C++ package management ?

A: There tends not to be good package management for C++, unfortunately. Many codes are moving towards CMake package management, but that is also far from

perfect. CMake also has superbuilds, which can be used to build the dependencies on-demand, although it can be complex to orchestrate correctly.

There are also options like spack (https://spack.io), which is more customizable but requires you to compile everything.

Another option is to look into services like conda-forge (https://conda-forge.org) to maintain cross-platform libraries.

Q: What about GitLab CI? ECP seems to be pushing that. This is that solution that ECP is funding to run at the different HPC centers!

A: GitLab CI can function both like Travis CI and as a local service on your own resources which I believe makes GitLab CI appealing to ECP. This is similar to others such as Azure Pipelines, Jenkins, TeamCity, etc. We do not have much experience with this service unfortunately and cannot comment more.

Q: Can you differentiate system-level (shell command) testing frameworks like CTest vs. unit testing frameworks like Google Test and Catch2 where you run many unit tests in an individual executable?

A:Testing on the shell-command level (CTest) is language agnostic, although you generally have to write your own self-contained test programs - it is basically a test runner, but with some very nice features.

Unit testing frameworks like Google Test, Boost Unit Test, or Catch2 test can be a bit intrusive, but can help test deeply buried pieces of code that are not exposed to an API. They are necessarily language specific.

Both types of frameworks are complementary - I tend to think of system-level being much better for smoke testing and sometimes regression tests. But you often need something language-specific for unit tests, particularly for C/C++.

Q. I seem to recall OakRidge hosting CI capabilities through GitLab on Titan, is that still available? Does anyone know how to get started?

A: OLCF is working towards being able to operate CI services while maintaining the security posture of the facility. So far, activity on this front has been in the nature of prototyping and there are no officially supported CI capabilities available to OLCF users as of yet. (For those interested in joining a pilot project once OLCF is ready, please contact Ashley Barker, ashley@ornl.gov)

Q: Does anyone know of any good static analysis/linters for Fortran?

A: Looking around we found I-CodeCNES (https://github.com/lequal/i-CodeCNES), but do not have experience with it. A public list of static linters for many languages can be found here (https://github.com/mre/awesome-static-analysis).

Q: What about false failures in static analyzers? Are many of those not bogus? That seems to be a big disadvantage of using a tool like this.

A: Static linters rarely have false positives, but some static linters can go overboard on warnings (similar to compiler warnings). We do like LGTM, since we have yet to find a false positive and catches many routine errors that should in fact be fixed. In addition, you can tell the static linter through code comments to ignore a given false positives in the future.