2019-12-12: ecbuild

The topic of the day is ecbuild and related issues such as CMake and eckit. The agenda included the following questions:

- What version of ecbuild should I be using and why?
- How to fix any build problems you may be having
- What changes do we want to make to ecbuild and why?
- The status of the various branches in our JCSDA fork of ecbuild
- How best to coordinate with ecmwf so that we can make changes in our forks of ecbuild, eckit, and fckit to accomplish what we want to accomplish while not diverging so far from ecmwf that upstream merges become painful

Before beginning the topic discussion, Hailing reported on some new COSMIC 2 results that have been posted this week on theJCSDA web page. To see them, navigate to COSMIC-2 Prov. Release under the Products menu. At the time of writing, one needs a password to access this but we intend to make it public very soon.

Mark O then opened the ecbuild discussion addressing some issues we have had this week with the build of the Eigen3 library. Previously, the jedi build system was including this through exported cmake configuration of eckit. Mark O made this more robust by by explicitly including the eigen3 target in oops. However, after this was merged earlier this week, it was causing problems with several bundles (in particular soca, mpas, and lfric). However, these problems have been fixed by making the hints in the eigen3 find_package() command more explicit. If anyone continues to have problems with Eigen in the build process please let us know.

Mark O went on to explain that CMake has changed substantially in recent years. In particular, it is increasingly using the more general concept of targets to control the configuration instead of relying on internal and cached variables as it had previously. Targets have a syntax that is reminiscent of namespaces in C++, for example Eigen3::Eigen, and as the syntax suggests, are defined within scopes. Targets can be files, libraries, header-only include directories such as EIgen, or more general applications such as MPI. The new CMake proceeds in well-defined steps of defining targets, linking targets, and then exporting targets for use with other components of the build. The linking stage is generally done with the target_link_libraries() CMake command. Rahul asked if target_include_directories should be used instead for header-only targets such as Eigen. Mark O responded that that is not necessary: though target_include_directories() could in principle be used, target_link_libraries() is generic enough to handle header-only libraries and is the direction that they are moving in for future CMake development. Mark O did add, however, that target_include_directories() is still useful at times to provide backward compatibility with earlier versions of CMake.

Yannick then asked if it would be worthwhile to update the findEigen cmake module. Mark O said there is no need - the way it is done now in oops is to use the NO_MODULE option in find_package - since Eigen has it's own cmake config modules one just needs to set environment variables properly to locate it, which is done in jedi-stack (e.g. EIGEN_ROOT or EIGEN_PATH to point to the install directory and/or Eigen_DIR to point to the cmake config module). Mark O added that this is a simpler case than something like LAPACK, which can vary in how it is installed, for example, using MKL in place of lapack or installing with or without cmake modules. Yannick then asked if findEigen should be removed but it was agreed that it provides backward compatibility that could be useful.

Mark M then raised two questions (1) is there anything now in the jcsda fork that requires us to use it - will JEDI still build with the ecmwf eckit releases?, and (2) will we require a more recent version of CMake to build JEDI?

The answer to the first question is a bit complicated. The short answer is that you could build JEDI on many systems with recent releases of ecbuild from ecmwf. However, there are some situations where the jcsda fork is required. For example, if you are working with intel 17-18 compilers on a system with an old version of binutils (as is the case for ubuntu 16.04, Discover, and S4), then you'll currently need to use the jcsda::feature/old_linker branch of ecbuild (we're working on a more robust longer-term fix). Or, if you are using static libraries for netcdf (such as the baselibs builds on Discover), then you'll need the jcsda/release-stable branch. In short, if you are using a branch of ecbuild that is problematic then let us know and we can help you fix it.

Furthermore, we are now in the process (led by Maryam) of storing at least some test files to AWS S3 instead of git LSF. We already merged some changes into ecmwf's upstream branch to accommodate this but to achieve the full functionality currently requires the feature/test_lists or feature/old-linker branches of jcsda's fork. We plan to issue a PR to ecmwf to enable this functionality but it is not currently there.

In response to Mark M's second question, it is recommended that you use the most current version of CMake that you have access to. We often use 3.13-3.14. Mark O recommended upgrading the minimum required version to 3.10. Mark M added that CMake is straightforward to build from source. So, if you are working on a machine with an old version of CMake, you can install a newer version using, for example, the build scripts in jedi-stack.

Given the current maze of ecbuild branches for different scenarios, the discussion then turned to how best we should make changes to our fork while still keeping pace with ECMWF. If we diverge too much, future merges could become painful.

Rahul recommended that we have a branch in our fork that tracks ecmwf's develop version. After further discussion, we agreed to call this the develop-ecmwf branch of our fork *(after the meeting Mark M created this branch in the jcsda fork of ecbuild)*. Then, if we have a modification that we intend to push upstream, we can create a feature branch that branches off of develop-ecmwf. After it passes our internal code reviews, we can then merge it into our own develop branch but we should not delete the feature branch until we issue the upstream PR. Then we can periodically merge our ecmwf-develop branch into develop to keep up to date.

Jim R then mentioned that he is having problems with FindNETCDF on his laptop and asked if this is fixed in develop. The answer is no. We do have a fix for this in our release-stable branch but while we (Rahul and Mark O) were implementing this, ecmwf made some beneficial upgrades to their FindNETCDF module so when we tried to merge them there were conflicts. Ideally, we would like to implement the best of both worlds, merging the modern target-based CMake approach of ecmwf with the increased functionality of release-stable that works with static libraries. But, this will take some work and it's not likely to happen this month.

Yannick then opened the floor to other build-related issues. Mark M announced that ODC and Odyssey have been added to the containers, though it's not fully tested. Odyssey in particular was giving some problems with the intel containers with regard to the python configuration. This led to a more general discussion on how we should handle python dependencies in JEDI. For example, as discussed in our Nov 7, 2019 focused discussion, when installing python packages like Odyssey on HPC systems is a challenge because we do not have root privileges. So, we can only install in user space but then a single installation will not be able to serve all JEDI users as is the case with the environment modules. As discussed at that previous meeting, it was agreed that the best solution would be to encapsulate all the JEDI python functionality in a virtual environment that each user can install by running pip or an alternative setup script. Again, this will require some effort so it will not happen immediately.

Since odc is now available in the containers and we're moving toward installing it on other systems (it is also available on S4 after loading the jedi/intel17-impi module), can we eliminate the ODB API? Steve V mentioned that ODB has some functionality, in particular filtering tools, that ODC does not have yet, so it was agreed that we should keep our ODB functionality for now.

Yannnick then closed the meeting with a reminder to please give us feedback on what topics you would like to see discussed at these meetings. You can provide feedback by either emailing any member of the core team (Mark M can serve as the main point of contact) or by editing the ZenHub board directly. Yannick added that next week's meeting will be a round-table update (despite what it says on the Calendar invitation) and then the following two meetings are cancelled because of the holidays. After the holidays, we will re-establish the normal alternating meeting schedule.