Overview Notes from Edward Smith:

Scope

Continuous integration for scientific software (i.e. high performance code with minimal unit testing and some acceleration: MPI, OpenMP, CUDA, etc) Deployment on HPC including:

1) Best ways to automate building within the module environment

2) HPC specific problems for testing, which includes:

a) Initialisation through job submission scripts (Hook to github like Travis CI or a GUI type interface like Jenkins)
b) Scaling (Unique HPC problem and efficiency bottlenecks should be spotted to prevent wasted resources)
c) Jobs tested over varying numbers of processor (bytewise comparison needed/possible?)
d) Frequency of tests (every github commit or user triggered?)
e) Location/queue for tests (Optimal use of resources is essential).

Aims/outcomes:

An identification of the HPC and CI community. Discussion of the best way to proceed by sharing lessons learnt.

This is split into

1) For RSEs, how to implement this in the best way

2) Getting users to start CI for their code on HPC (e.g. can we make it as simple as using a Travis style yml file)

A two page report to be shared on the SSI and RSE network on insights. Perhaps a move towards adoption of a similar approach across platforms and RSE groups.

Talking Points from Chris Richardson

- What are we aiming to achieve/measure?

- reliability of 'build'
- reliability of 'run'
- timing regression

- How to access resources

- who is willing to run these kind of jobs on their machine?

- accounting/payment
- job size/frequency
- repository type (which VCS system? public/private?)
- credential management (if needed)
- job scheduling (on commit, nightly, weekly etc.)

● Reporting

- Type of data to report (pass/fail, timings, ...)
- Format (xml, json, stdout, or in file etc.)
- Means of transmission from test system (http, email etc.)
- Reporting frontend (web, email, etc.)
- Database (to keep long term record over time)
- Normalising results (as machines change)

<u>Talking Points from Mark Woodbridge (RSE Team Lead @ IC)</u>

- I'd be interested to see a discussion about the requirements of an "ideal" HPC-compatible CI system - and how existing systems would need to be developed to meet these needs.
- Maybe aim for the output of the workshop to be a white paper that can be shared with the SSI etc.
- What do people like/dislike about existing solutions, both proprietary (e.g. Travis, Circle, Codeship) and open (Drone, GitLab, GoCD etc)?
- If a national VCS/CI platform existed what would it look like? I guess that some clear guidance on this would be hugely valuable to the STFC in the context of the SESC Build Service. They obviously already have the cluster environment, but could maybe do some magic to enable you to use your institution's resources too.
- Has anyone used the Build Service? What were their experiences?
- I'm a big advocate of simply(!) funding a UK research GitLab instance, with some effort dedicated to writing an "HPC" compatible Runners. I know this has been attempted before, but not in a very concerted manner (as far as I can see).
- GitLab's CI really is a model system (IMO): you just add a YAML file to the root of your repo and then you get automated builds on every check-in - which can be configured to run tests, auto deploy, send arbitrary notifications etc (as you'd expect). You don't need to configure any third-party system. It can build in a container environment, which means no restrictions on languages, libraries etc.
- In the simplest terms we need to consider not only how to build and fund this, but how to make it trivial to use. If it was as simple as picking a template YAML file, adding a new remote in Git and pushing using your institution's credentials then you could be using CI within 5 minutes.

Suggestion for Topics in Discussion Sessions

Please add here -- these will be used when we break into groups and then report back at the end.

1) **Best practice in deployment on HPC**, including use of containers, virtual environments, packing systems (e.g. conda) and automation of build on the HPC platform (i.e. running over a range of modules). Do we aim to build big codes each time (e.g. OpenFOAM takes 10 hours) or reuse until it changes?

Team I: Jeffrey Salmond, Krishna Kumar, Jan Hybs, Eduardo Ramos Fernandez, Edward Smith, Nicolas Gruel

- Tricky problem, no same queue system or module versions even within the UK!
- Ideally we'd remove the build time on head node, use only for testing and build off site and deploy when needed.
- Container with singularity seems easiest deployment at present.
    - no clear performance penalty in running.
    - MPI interface in container quite tricky.
    - Built separately on a dedicated machine.
    - Containers are massive, Gb need to be copied.
- Conda deployment is another way building for deployment, need virtual machine to get old libc. Same issue with containers.
- A movement like OpenHPC standardise the deployment of module is **very important** to improve ease of deployment on various platforms.
- Spack python tools to build range of combination of different packages. Can use compute nodes.
- ccache to optimise build process by storing only changes

2) **How to reach users outside of the RSE community?** How do we provide a tool so we are at the stage where a workshop for new users is possible? Is this a good idea to try or should this be a job for the growing RSE community?

Team: Sam Cox; Catherine Jones; David Pérez-Suárez, Dimitar Pashov

At UCL they are planning to **provide a simple guide** on how to add jobs to our jenkins infrastructure, mostly for code that needs license (e.g., matlab). Then the RSE team would have only to review that jenkins job builder config file.
People may also be afraid of creating tests, in case they get "different" results.
Training is an important aspect. **People don't know they don't know**. **Training** like software carpentry, but with more focus sessions about testing and CI. Leicester group is planning a set of training courses to include git and more advanced python.
More **HPC meetups should happen to share the pain**. Python **clubs** where they share experiences...

3) **Is it possible to automate this and provide a framework which is less hassle to learn than doing it yourself?** Where does this run and does this cost AHU resource? How to trigger the test, through version control hook or manual? How to optimise using pbs?

Chris R., Steven L., Jon W., Jamie R. and Alin E.

- Different types of CI:
    1. Build - may need VM image of HPC login node or direct login node
    2. unit test - may not even need HPC
    3. **Performance - needs to run on HPC resource**
    4. Deploy and packaging - not HPC
- Credential storage. API key? SSH key? Tied to specific researcher for security.
- Testing frequency:
    - on commit: build, unit test
    - daily/weekly etc.: performance
    - manual

  All these can be configured in the .yml config file or equivalent
- **No real technical barriers to doing any of this**
- Can we have an experimental project with Tier2 resource? E.g. Cirrus.
- Example that we can show to people how it works… make technical report on figshare?

4) **How do you unit test MPI code as there is a significant build-up needed before test (MPI_Init, etc)?** Is it only possible to run integration/top level tests on HPC and use dedicated CI platforms for unit tests? Should range of processor topologies be part of test? How do we test MPI specific problems such as deadlock, race condition, IO bound problems, scaling efficiency? How do we test big jobs?

5) **Setting up a national CI service**
    - a) What is the cost of running a national system?
        - i) Hardware - very bursty. May not need a large physical resource? Could potentially run 5 projects on one server (machine is idle 4/5ths of the time). National service may only need about 50 servers. Possibly build an elastic system that bursts to the cloud, and replaces cloud resources with dedicated servers as it grows.
        - ii) People - very busy. Set up of the actual CI service. Will be quicker bootstrapped from a community project. We know what it would look like. There is plenty of experience. Should not set up something completely new from scratch.
            - (1) Then lots of sysadmin / updating work. Keeping up to date with software, installing updates, managing the service.
            - (2) To manage resource, need to select a set range of operating systems and compilers. Don't try to do everything.
        - iii) Why not have travis provide everything?
            - (1) Lack of windows (at the moment)
            - (2) Limited (2 cores, 90 minutes)
            - (3) Can't do multinode
            - (4) Lack of custom images
            - (5) Doesn't work with private repositories
        - iv) Who is the user community of the service?
            - (1) Is this an HPC build service or an academic build service?
                - (a) Maybe have projects bid to have access to the national service. This way the service weeds out projects that could use travis for free.
                - (b) Maybe you automatically get a national CI service project created with a funded EPSRC grant. What about other research councils? Should this be UKRI funded?
                - (c) Alternativly, set up a central BuildBot master and have universities donate local resources to a shared community. Users are only able to run on the local resources to which they have access.
                - (d) Then have criteria to use national resource is that the project is multi-institution, or has broken past the limits available locally.