Continuous Integration for HPC Facilities

Requirements Document
Exascale Computing Project ST/Facilities Technical Working Group on CI

I. Motivation

Web-hosted tools for Continuous Integration (CI) are ubiquitous in modern software development, and they allow tight integration between source code repositories and associated testing, code quality checks, and code review. Developers can use continuous integration to ensure that their code is tested in many different software environments with many different configurations and options. Cloud services like Travis CI, GitLab CI, Circle CI, AppVeyor, and Bitbucket pipelines are available at little cost or for free, and they allow public or private cloud-hosted repositories to be linked with on-demand continuous integration systems. With these systems, Linux VMs and/or containers are spawned on demand in the cloud, sometimes in combinatorial configurations to ensure test coverage across a matrix of environments of interest. CI tools are also available for use behind the firewall. Tools like Jenkins and Bamboo and enterprise versions of GitLab, GitHub, and Bitbucket can launch *runner* processes (or *agents)* to run CI tasks on local resources in response to repository events.

HPC developers need to deploy similar services but they have more stringent testing requirements. HPC developers need to run on large-scale supercomputers and exotic testbed machines, and the hardware and software environments on these machines are often *unique*. Leadership-class machines serve thousands of users, and each center sets up their machines differently. It is therefore important that CI jobs run *on the real hardware* at each site, *in that site's environment* to test for hard-to-find bugs related to the site's deployment environment. However, most HPC sites also have strict security requirements. Most use two-factor authentication, which makes it hard to launch automated tasks on an HPC machine from outside its facility. Further, HPC users typically build, install, and run software *as users* on the host machine, and HPC users typically *do not* have root access to cluster nodes. CI jobs therefore need to be run as specific users to comply with each center's security model.

Existing CI systems assume that developers can either set up their own hardware, or that their production environment can be duplicated in the cloud (e.g., in a VM or container). Neither of these applies for most HPC systems: supercomputers are often unique resources, and HPC applications do not typically run in VMs. While containers are rapidly becoming more popular at HPC centers, many applications still need to deploy on bare metal, and containers built on HPC machines may still not be portable outside their host facility. HPC centers therefore do not typically provide CI as a general service for their users, and HPC testing cannot be automated easily. The cost of this lack of automation is tremendous; each new machine and each new environment requires additional manual engineering effort to test and deploy software, and bugs on DOE production systems are only caught at the last minute or, worse, in production runs. Future HPC environments will *increase* in diversity, and without automated testing and continuous integration, the HPC software ecosystem will cease to be sustainable.

II. Request for Proposals

The U.S. Exascale Computing Project's (ECP's) mission is to develop a comprehensive software stack for exascale machines. The stack includes HPC applications, libraries, tools, and system software. To test and deploy this stack, a comprehensive, secure, easy-to-use CI solution that can overcome the above limitations is needed for deployment at large-scale HPC facilities.

ECP requests development proposals for CI systems that can satisfy the requirements described in Section III of this document. We do not anticipate that vendors will need to implement a new CI system from scratch, rather that vendors will be able to develop enhancements to existing CI systems either as plug-ins or open source contributions to existing CI systems, or external services that interoperate with existing CI systems. Responsive proposals will deliver a final product that will enable HPC facilities and the ECP project as a whole to deploy CI systems that meet the requirements in Section III.

III. Requirements

See the **Intro to CI for HPC Centers** presentation for background.

These are requirements for a CI system that can be deployed securely for use by HPC facilities. Generally speaking, *most* requirements for HPC centers are met by existing CI systems, but HPC centers need new security features to serve thousands of users with with unique hardware and security requirements. Responsive vendors will likely modify an *existing*, *widely used* CI system rather than inventing a completely new system. We request the following capabilities:

1. **Permission requirements for all runners**
    a. CI jobs can run automatically on a schedule (like cron) or in response to actions on source repositories, such as pushes, merges, configuration changes, etc.
    b. Runners run CI jobs only in response to actions taken by authorized users. In the web UI, users can be authorized to use particular runners. See below sections on Web UI for requirements on authentication/authorization.
2. **Setuid runners** (Traditional CI launch/spawn)
    a. "Setuid" CI job runners can be launched as secure, persistent services on bare-metal HPC login or bare-metal HPC compute nodes.
    b. Setuid runners ensure 1(a-b) by receiving jobs from a trusted server, forking, and calling setuid to execute each CI job as a specific HPC user.
    c. Jobs launched as specific HPC users can run parallel batch jobs as those users normally would on the HPC system.
3. **Batch runners** (*i.e., CI job is spawned as batch job)*
    a. The CI system can spawn job runners on demand, using the batch system at each HPC site.
    b. Batch runners can run anywhere the batch system can launch jobs (compute nodes, potentially login nodes, etc.)
    c. CI system ensures 1(a-b) by running as a trusted service and submitting runner jobs to the batch system as different HPC users. Jobs run as those users.
    d. Batch runners can be launched in parallel (multi-node) allocations if the user's job requires MPI or other distributed parallel testing.

    e. Batch runners should not depend on any specific batch system, as HPC centers use many different batch systems. i.e., runners should be easily extended to work with new batch systems through a public, well documented API.

4. **Intranet Web UI**
    a. The CI web UI can be hosted behind an HPC facility's firewall.
    b. There will be a mechanism in the web UI for project administrators to specify the user for runners launched on behalf of a project.
    c. There will be a mechanism in the web UI for users to authorize projects to run as them. Ideally, project administrators can request that their jobs run as a particular user, and the requested user will be prompted to authenticate to confirm the authorization.
    d. Authorizations can be granted with a user-specified expiration time.
    e. Administrators can opt to *require* authorizations to expire, and can set a maximum expiration window.
    f. The Web UI should support authentication of users via pre-existing credentials databases such as LDAP and Atlassian Crowd.
    g. Permission actions in the web UI (e.g., adding/removing access to specific runners) can also be executed via a web API (e.g., REST, json, etc.), so that facilities can script the solution to conform to site-specific requirements and account processes.
5. **Externally hosted Web UI**
    a. The CI web UI can be hosted externally to the HPC facility, e.g., in Amazon Web Services (AWS).
    b. Accounts at HPC centers (i.e., runner users) can be associated with accounts in the web UI, either:
        i. On the server side, e.g., web UI user can authenticate using credentials for an HPC facility to associate their HPC account with the CI account; or
        ii. On the facility side, e.g., facilities maintain mappings from web users to local users that CI runners must enforce.
    c. HPC facility staff should be able to approve account associations, e.g., through the external web UI (as in 5.b.i) or locally (as in 5.b.ii).
    d. An external CI instance can securely run jobs on behalf of users at centers using their associated facility accounts.
6. **Permissions in the Web UI**
    a. CI configuration in the web UI (e.g. projects, build plans, repository credentials, etc.) should be isolated in the UI and **not** world-readable by default.
    b. Users should be able to grant and revoke read/write access to their projects to other users in the web UI.
7. **Auditing**
    a. HPC centers will be able to audit when runners were launched on particular hosts, and as what users.
    b. Web UI administrators will be able to see who logged into the web UI and when.
    c. Web UI administrators will be able to see audit logs of when and how users change permissions on their repositories in the Web UI.

IV. Use Cases

This section provides detailed use cases that illustrate the need for the requirements in Section III. The use cases are provided by DOE HPC facilities. Facilities were asked to provide the current state of their

CI implementation (if any), issues with their current setup, and a wish list for what they would *like* an HPC CI system to do. Use cases here should not be interpreted as hard requirements; they are provided to help vendors understand the requirements in the context of real HPC centers.

Lawrence Livermore National Laboratory (LLNL)

Current state
Livermore Computing (LC) currently runs Atlassian Bamboo on a central server inside their HPC center. All LC users can log into this server with their LC credentials. Within the application, user CI configuration is isolated at the build plan level, and users can grant/revoke permissions to their build plans. Each build plan is associated with one or more dedicated runners, which run as HPC users on login nodes. Any user can deploy a dedicated runner for themselves using a custom web portlet developed by LC. The portlet launches runners as users on login nodes, but administrators must ensure that each runner is dedicated to build plans owned by its same user. We must thus *manually* ensure that runners are secure, and users cannot use the CI system to run as other users or to access their data.

Issues
LC has centralized the CI server, but the administrative overhead of managing the agent-to-plan mapping in Bamboo is growing. We do not believe it is scalable in the long run. In our current configuration, each runner is usable only by a small number of build plans, whereas a system with setuid or batch runners would allow much more scheduling flexibility of plans among runners.

Wish List
We need a CI system with either batch or setuid runners in order to reduce manual configuration overhead and increase utilization. Batch integration is particularly desirable as it would allow us to control the amount of compute resource dedicated to CI in the same way we already manage HPC jobs. Either type of runner would allow LC to deploy runners once and eliminate per-user/per-runner deployment overhead.

Oak Ridge National Laboratory (ORNL)

Current state
The Oak Ridge Leadership Computing Facility (OLCF) and Oak Ridge National Lab (ORNL) each host Gitlab services accessible to our users through their OLCF and ORNL credentials respectively. These Gitlab instances are configured to support Gitlab CI. The CI runner instances must currently be started by individual users and staff members directly in specific-runner-mode on OLCF resource login nodes. Use of CI the runner instances is restricted to members of the projects to which it is associated. It is currently the responsibility of the user who registers a runner to configure the it such that it cannot be used by other unauthorized projects.

Issues
Jobs from all associated projects are run in shell-executor mode as the user who started the runner process. Restarting runners after system reboots is the responsibility of the user. The jobs performed are entirely in control of the project members through configuration stored in the associated repositories. However, CI actions are only feasible when the code is hosted on the OLCF or ORNL Gitlab servers.

Wish list

The OLCF is interested in a CI system where runners can be started by a setuid daemon process as specific authorized users on resource login nodes or to the batch queue to allow the OLCF to maintain resource utilization balance. The runners should be able to accept and run jobs only from specific authorized projects that may be hosted on either our internal VCS services or external services. Runners should only be employed by CI tasks triggered by a specific OLCF user by default. Any changes to the permissions that would allow jobs to be launched by persons other than an authorized OLCF user must require an authorized OLCF user to delegate that permission after validating their OLCF credentials. The ability to spawn delegated runners should require regular re-authorization by the user under whom the jobs will run.

National Energy Research Scientific Computing Center (NERSC)

Current state
NERSC has a jenkins installation maintained for one group, the JGI (Joint Genome Institute). Other communities have expressed interest in CI, but aren't as advanced yet. In addition to the local Jenkins install, NERSC is using the cloud-hosted gitlab.com site and deploying gitlab runners internally on SPIN (SPIN is the NERSC-internal cloud infrastructure).

Issues
The JGI Jenkins installation is viewed as flaky and expert-intensive. JGI is investigating GitLab because of its ease of use and runner deployment options.
Gitlab runners on SPIN currently run internally but can't access global filesystems or to access batch queues, due to security concerns. Specifically, Gitlab-triggered builds can't trigger builds for specific NERSC identities, so security can't be guaranteed end-to-end from gitlab.com to NERSC. Further, storing credentials is difficult and insecure. The biggest security risk faced with shared runners like this is that they frequently point at very open GitHub/GitLab projects, and anyone with commit access on the remote site can commit a new YAML file, or simply commit to the project, and essentially run things on the supercomputers. The problem of running unsanitized code isn't really unique to CI; users pull in other people's OSS all the time, but unsecure CI exposes a more obvious attack vector. There is a need to educate users on how best to lock down public repositories.

Wish list
In addition to the SPIN runners, NERSC would like to have gitlab runners on the login nodes of their clusters, with a full Cray environment. That environment is not easy to replicate in a container, and there is little desire to try to do so.

NERSC is interested in using containers for CI builds, particularly to monitor for malicious activity.

Argonne National Laboratory (ANL)

Current state
At the time of the working group, ANL was working on deploying a Jenkins-based solution. The idea was to use 2 dedicated nodes with GPFS and lustre mounts for the build agents, and to run a Jenkins master in a VMware environment. To ensure security on runners, ANL would either deploy a runner per project (each running as a dedicated project user) or would develop a Jenkins plugin to fork and setuid to dedicated project users. A plugin would have provided isolation and flexibility, reducing the number of agents that would need to be running at any one time.

Since the working group concluded, ANL has scrapped their Jenkins plans and decided to go with GitLab.

ANL is still working on setting policies about how users can use CI. These will likely be fleshed out once 'friendly users' are on the system, e.g.:

- How often can they build?
- Minimum number of builds per day?
- How running test jobs affect their allocations?
- Should all project users be able to kick-off a build or test suite?

Issues
Cryptocard authentication, ability of runners to setuid to particular users.

Wish list
As mentioned above, setuid runners are the major wish list item.

Los Alamos National Laboratory (LANL)

Current state
LANL currently deploys CI in two environments:

1. Next-generation Computing (NGC)
2. Production codes

The NGC environment uses a GItLab instance with a special cluster where containers can be used. On this cluster, the teams run open and ASC code tests at scales of ~100 MPI ranks in docker and on the bare metal. The production environments rely on Jenkins; one Jenkins instance per team, and these CI servers can run tests on production systems.

Issues
From experience: Jenkins is an order of magnitude more complicated than GitLab for a team to set up; GitLab enables more users to get CI. With GitLab, multi-OS testing with containers is easy and PR integration with the web UI is easier than with Jenkins.

Wish list
Setuid runners are the major request for the GitLab instance, so that users could potentially share an internal GitLab instance on production machines. It is unclear how soon production teams would move away from Jenkins, as each team customizes Jenkins for their project, and the teams already know Jenkins (despite the complexity).

Sandia National Laboratories (SNL)

Current state
A few Jenkins servers are deployed at Sandia which run as entity account users. Credentials are configured for each account to connect through ssh to the HPC cluster machine login nodes. The entity accounts are then allowed to submit batch jobs on the clusters. One of the Jenkins servers is locally modified to run the slave process with setuid to an entity account attached to the given Jenkins Project.

Issues
The local modification increases the cost of managing and maintaining the Jenkins installation. Furthermore, a side effect is that the machine resources cannot be load balanced across Jenkins Projects (machines must be partitioned). For the stock Jenkins server that runs as a single "jenkins" entity account, all jobs are submitted to the HPCs as that entity account. This means no per-project accounting can be done for utilization of the HPC machines.

Wish list
A couple of additional impediments that have not been mentioned yet are: (1) requirement for very fast/efficient CI build environments. In order for CI to be effective it must have fast turnaround time. This means the build environment for applications to be rebuilt for CI has to be very efficient and fast. In has been our experience that the number and configuration of HPC login nodes is often inadequate to support this. Supported cross-compile environments would also be helpful here. Fast filesystems for builds is also necessary. And, (2) Depending upon the maturity level of the application, the application's test suite may be quite large which can stress the HPC systems scheduler and resource manager. These layers need to better support high throughput computing (HTC) for CI.

Kitware

Current state
Kitware does not deploy CI, but works with HPC facilities on ECP projects that need it.

Issues
Kitware echoes NERSC's security concern about anyone who can make a merge request having permission to run jobs on the machine, and about the dangers of shared build runners. For example, NERSC has in the past configured buildbot to allow particular users to trigger tests by commenting on a remote site.
To support all of its projects, Kitware needs the ability to run CI jobs at completely different site from where code is hosted. A central server with runners at different ECP sites could accomplish this. An example is the ADIOS ECP code, which is hosted on GitHub, but CI for this project needs to run at the facilities in order for it to be robust and well tested for users.

Wish list
Kitware and its projects need to be able to trigger builds at *multiple* DOE sites from a single repository/project. A central server with runners at the various ECP sites would satisfy Kitware's use case.

HDF5 - The HDF Group (THG)

Current state
The HDF Group (THG) uses Buildbot to run HDF5 CI testing. Results of the testing are published on CDash and email notifications are sent to the HDF5 developers.. There are no automatic testing on HPC systems. All components needed to test on HPC systems are in place, but submission and results retrieval will depend on the system.

One of the possible solutions will be running Buildbot "slave" on the HPC system. The slave will be in communication with the Buildbot Master that runs at The HDF Group dedicated hardware and execute

commands to run HDF5 testing configurations. This may include scripts unique to HPC systems for running HDF5 regression test suite.

The results of tests could be submitted to the CDash that is maintained by The HDF Group for external test submission and will be open to the interested parties.

Issues/Assumptions

1. Buildbot software is Python based and can be run on the HPC system
2. Established connection between the HPC system and The HDF Group testing servers
3. Dedicated developers to address code issues identified by testing
4. Dedicated developers to maintain Buildbot setup at the HPC system and at The HDF Group.

Wish list

1. Access to >8K cores to run parallel HDF5 tests on large-scale systems
2. Access to different file systems (e.g., GPFS, Lustre, SSD)
3. Access to different compilers and versions of MPI I/O libraries
4. Support of using Containers for CI builds, (similar to NERSC's interest in using Containers)

V. Implementation Platform

GitLab

We do not strictly require a solution proposed for this RFP to use any particular CI technology, but we have a strong preference for modifications to GitLab, based on responses from members of the CI working group. GitLab is currently the most popular *behind-the-firewall* CI solution, and based on many labs' experiences, it is also much more feature-rich and easier to use than competitors. We also have a preference for GitLab because a) it is open source and HPC facilities will be able to customize any contributions made to it; and b) its runners are implemented in Go, a systems language that lends itself to the fork/setuid behavior likely needed to satisfy some of the requirements in Section III. Labs have attempted to implement this themselves with Java-based CI tools, with limited success.

GitHub Integration

Despite GitLab's popularity for use on intranets, the majority of the ECP software stack is open source and hosted on GitHub.com, and we do not expect this to change in the near-term. We would therefore like the proposed CI solution to interoperate with GitHub, regardless of the platform it is implemented on. For example, if pull requests are posted to a project's main GitHub repository, or if commits are added to a repo on GitHub, the CI solution should be able to run CI jobs for these contributions at ECP facilities and update the status of the PR on GitHub. This should be possible without significant effort on the part of the project team. Currently, GitLab users can achieve this with a few configuration steps, but simpler, more tightly integrated solutions are under discussion (see https://gitlab.com/gitlab-org/gitlab-ce/issues/32052).

Detailed Feedback

Detailed feedback from WG members on the implementation platform:

1. LLNL leans towards GitLab, *even though* it will be painful to switch from Bamboo/Bitbucket. GitLab makes it easy for new teams to get started with CI, and it is open source, so the facility can easily make customizations. Further, GitLab's runners are written in Go, which lends itself to integration with security in the underlying OS much better than Java-based solutions.
2. ORNL leans towards GitLab. They have tried jenkins and other solutions; setup was far easier with GitLab, and ORNL's current CI offering uses GitLab.
3. NERSC leans towards GitLab, as they are already using gitlab.com. NERSC says that requiring implementations for more than one major CI solution will take more time -- if other solutions are to be considered, prototyping with gitlab first will likely result in a usable product sooner.
4. ANL sees nothing wrong with GitLab but will look at this exercise as a proof-of-concept before it goes all in. ANL has started using GitLab internally since this working group ended.
5. LANL leans towards GitLab long-term, and they prefer it over implementing a first proof-of-concept in Jenkins (similar to ANL and NERSC). LANL wants to see a product that is open and maintained by a company. LANL also notes that there is a GItHub plugin that allows GitHub CI to run on GitLab runners, so there is a possibility that this will also enable testing for GitHub-based projects. While LANL does lean towards GitLab, it does have a large investment in Jenkins for production teams, which may continue to use Jenkins for some time before eventually migrating. Jenkins is more "universal" and general in many ways but complexity is difficult.
6. SNL has a substantial investment in Jenkins and utilizes both GitHub and GitLab. Either implementation would require some migration at SNL. We would like to see this effort produce a robust solution to help justify that migration.
7. Kitware has a concern that GitLab CI is closely tied to the GitLab server, and integrating cross-site runners may be difficult. Absent this issue, i.e., if on-site runners at different DOE sites can be made to work with a shared GitLab instance that tests at ANL, ORNL, and NERSC, then Kitware also leans towards GitLab over other solutions.

VI. Open Source

It is *strongly* preferred that any proposed solution be open source, and that it can be contributed back to the CI tool on which it is based. The DOE laboratories have a long history of deploying open source software for HPC, and it has many advantages for HPC centers, including the ability to customize heavily for each site. There may be some components, such as plugins for enterprise versions of, e.g., GitHub or GitLab, that cannot be OSS, but we prefer that the bulk of proposed solutions to be OSS, released under a permissive license.