Tools and testing: validation, continuous integration (CI)

abstract: long form

Continuous integration (CI) and continuous delivery (CD) is hard. When we do this at scale across the many phases of a "large" project, let's say the delivery of a supercomputer with novel hardware, innovative software, and varied application expectations, how do we do this well from project inception to release, including all those things which we refer to as "in progress" or "in development", e.g, software and the many incarnations of hardware along the roadmap to the "real" thing including subsequent maintenance, software and hardware updates? Challenges include creating per-component micro test cases to explore depth and overall coverage; involving all participants who contribute to the software and hardware deliverables; including scenario-based testing which aligns to customer needs/expectations into the validation/regression testing workflow and framework; and maintaining a working workflow while the runtime software stack (including OS and kernel updates) and the underlying hardware platform changes hundreds, if not thousands, of times over the course of the entire project.

I approach this from an industry perspective and acknowledge this is challenging. I want to understand how these challenges have been approached previously and the gaps which remain, with output of wg to be key points for addressing CI/CD gaps in at scale projects.

abstract: broken down

- Main concept: Continuous integration (CI) and continuous delivery (CD) is hard.
- Question: How to excel at scale across the many phases of a "large" project, when both hardware and software are the expected delivered package?
  - "in progress" or "in development" software stack
  - many incarnations of hardware along the roadmap to the "real" thing
  - Post delivery subsequent maintenance to accommodate software and hardware updates
- Challenges:
  - creating per-component micro test cases to explore depth and overall coverage
  - involving all participants who contribute to the software and hardware deliverables
  - including scenario-based testing which aligns to customer needs/expectations into the validation/regression testing workflow and framework;
  - maintaining a working workflow while the runtime software stack (including OS and kernel updates) and the underlying hardware platform changes hundreds, if not thousands, of times over the course of the entire project.
- WG goal:
  - Leverage expertise of attendees to note BKMs for how such challenges are/have been approached.
  - List gaps which remain for achieving CI/CD excellence for at scale projects.

Challenges for CI/CD at Scale (1/2)

- **Access to HPC resources for build/test**
  - Many HPC projects rely on open-source libraries developed externally (e.g., on GitHub)
  - Most HPC resources are only for trusted users with accounts
  - Testing in **relevant environments** (at large human scale) is impossible b/c can’t run tests on arbitrary PRs
    - Early access machines often NDA, so yet another security constraint
    - Duplicated effort to run tests in cloud and on NDA machines
  - Access to facility software stack
    - Releases break builds – need to keep up with latest facility stack
- **Export controlled codes have their own challenges**
  - Layered testing + CI/CD
  - Internal codes Building on CI for OSS product
- **Testing in general**
  - Validating output
    - Validating performance data (and numerical data) involves tolerances
    - Code teams use baselines and compare to baseline
    - Often human required to determine whether a delta from a baseline is acceptable
  - Microbenchmarks can be effective for measuring performance
  - What’s the ratio of dev engineers to test engineers
    - *Moving towards “there are no test engineers” – devs write their own tests*

Challenges for CI/CD at Scale (2/2)

- **Testing at scale**
    - You don’t have infinite cycles
    - Choosing what tests to run at scale
    - No widely used/robust MPI testing framework
    - Every resource manager is different and requires special arguments
        - Different sites have internal consistency, but no community
- **Availability of containers**
    - GPU, MPI in containers is still hard to set up
    - NVIDIA-docker simplifies but can’t run docker on HPC
- **Having to build from source all the time**
    - Not that many HPC binary distributions (or they just came out last month)

Best Practices for CI/CD at Scale

- **Writing unit and integration tests is critical**
    - Can’t scale project to more contributions or concurrent development

- **Automation**

- **Early access machines can help get running**
    - Porting to early access hardware helps move software stack in advance of production machine
    - Often several generations of pre-release hardware machines
    - Open challenge: how to do CI?

- **Building communities around tools can help make them best practices**
  - Jacamar CI becoming widely used by many centers
    - Solves security issues

Current Gaps in CI/CD at Scale

- HPC lacks a **community parallel testing framework**

- **Skill gap** for modern tools
  - Python/automation
  - Gitlab
  - Kubernetes/openshift/containers/etc.

- HPC centers don’t (yet) understand **degree of code reliance on open source**
  - Need for public CI/CD machines and/or more job isolation (so we can run untrusted workloads)

- Need for **cloudy workflows** alongside HPC applications
  - Persistent services, AI jobs with GPUs, databases

Improving CI/CD at Scale

- **Grow HPC CI/CD communities**
  - HPC CI/CD practices
  - HPC software products
    - leverage support and contribution from vendors
    - More people involved == more potential customers
    - More potential customers == more vendor willingness to contribute/support
  - Parallel community test harness
- **Common CI/CD infrastructure may accelerate/compliment vendor adherence to SOWs**
  - HPC CI/CD
  - Cloud nodes which truly mirror HPC machines ?
  - Containers which truly mirror HPC environments ?
  - Cloud versions of resource managers
- **Increase automation in HPC centers**
  - Internal REST APIs
  - ↑ cloud/HPC convergence to enable automation
  - Goal for user to be able to to provision services much like they do in cloud
    - Avoid requiring persistent service for provisioning which is not good fit for batch-driven HPC sites
- **Ease point of entry for users on the path to more automation**
  - More cloud convergence for provisioning services