Creating Continuous Integration Infrastructure for Software Development on U.S. Department of Energy High-Performance Computing Systems

Ryan Adamson and Paul Bryant, *Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA*

Dave Montoya, *Trenza Inc., Los Almos, NM, 87544, USA*

Jeff Neel, *Argonne National Laboratory, Lemont, IL, 60439, USA*

Erik Palmer, *Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA*

Ray Powell, *Argonne National Laboratory, Lemont, IL, 60439, USA*

Ryan Prout, *Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA*

Peter Upton, *Argonne National Laboratory, Lemont, IL, 60439, USA*

The Exascale Computing Project software deployment effort developed and advanced DevOps capabilities. One goal was to enable robust continuous integration (CI) workflows that span the protected high-performance computing environments found within many of the U.S. Department of Energy's (DOE's) national laboratories. This article highlights several challenges encountered with enabling automation, such as charging models for CI jobs and meeting individualized security requirements that revolve around strongly associating running code with a human identity. It also describes how the Jacamar CI tool evolved to meet later requirements and became a key aspect of the solutions currently offered. Derived from this experience, we offer a conceptual framework for understanding current and future CI challenges at DOE facilities and offer suggestions for long-term solutions.

The Exascale Computing Project (ECP) software deployment at facilities focus area was created to ensure the successful integration of ECP-developed software within the facility-managed software stacks at U.S. Department of Energy (DOE) high-performance computing (HPC) facilities. Of these tools, few are more beneficial to the development cycle than continuous integration/continuous deployment (CI/CD) workflows that automate testing code changes as they are incorporated into the code base. CI/CD, however, presents unique organizational and technical challenges for the DOE's Advanced Scientific Computing Research High-End Computing (ASCR HEC) facilities: the Argonne Leadership Computing Facility, National Energy Research Scientific Computing Center, and Oak Ridge Leadership Computing Facility. In this article, we discuss how these challenges shaped CI infrastructure solutions and offer suggestions for continued advancement.

CI/CD is an established software best practice with well-documented benefits. In our context, CD is not a main focus, and, thus, we drop the CD term. We therefore use CI to refer to an automated pipeline that goes from a change in a code repository to an action—often code compilation and unit tests—taken on another machine. Unique challenges for CI in the ECP arise due to the differing nature of two computing environments: open access code development in public repositories in one

Digital Object Identifier 10.1109/MCSE.2024.3362586
Date of publication 7 February 2024; date of current version 5 July 2024.

environment and secure computing done on ASCR HEC machines in the other.

In the first, software development is done in public repositories that leverage generic and plentiful computing resources, such as personal laptops and CI pipelines, that run on public repository hosts, such as GitHub[a] or GitLab.[b] In the second environment, ASCR HEC systems lead the world in compute capabilities by offering next-generation hardware and configurations not found elsewhere. Developers using these machines are granted a finite number of compute hours through a competitive award process and, thus, employ it sparingly. At the same time, properly protecting ASCR resources requires a comprehensive security posture in which each user is vetted, with their access and usage monitored. Solving the CI problem, therefore, means seamlessly bridging the gap between these conflicting computing environment paradigms to amplify the benefits each offers to the progress of scientific software development.

CI WORKFLOWS FOR ECP SOFTWARE TEAMS

A typical CI workflow is described within text files (usually written in YAML format and stored in the source code repository) that the CI system uses to determine how to respond to specific events. A runner process will poll the code repository server for CI jobs. When the runner process finds a job, it runs the actions specified in the scripts in the code repository on its own systems. Runner processes may run on various systems, such as institutional servers; developer desktops; or, most commonly, in a cloud environment. CI jobs typically run tests for performance, regression, and/or compilation. Results are then shared back to the code repository by the runner process, where the user who made the change can easily monitor whether the code changes introduced any bugs or performance regressions. If monitoring exposes an error or regression is discovered, the developer can then roll back any changes to the source code and troubleshoot the issue until the appropriate code behavior is achieved.

This practice improves engineering communication and accountability within software projects. Many software teams develop robust CI workflows on dedicated resources,[1] but it is important to acknowledge that these workflows are still limited in terms of their direct interaction with provided ASCR HEC systems.

Unique characteristics of the CI workflow arise for ECP software teams for two reasons. The first is that it is common for ECP software teams to contribute to public code repositories during development since most codes are open source and widely used. In fact, contributors to ECP codes may have no affiliation with the DOE if the contributions are made to fix software bugs or create features desired by other organizations. The second characteristic is that code development occurs in an environment that differs from where performance is measured, such as when code is written on a laptop, and performance is measured on an HPC machine. Because each of the ASCR facilities has unique hardware, compilers, and operating systems, it is possible that code that compiles locally may not even compile on an ASCR system. CI workflows offer teams an automated way of determining the impact of code changes and design choices in the target environments. For this reason, CI workflows that originate from open access repositories and run CI workflows on ASCR machines carry the potential for high productivity value for ECP software teams.

[a]https://github.com
[b]https://gitlab.com

CI WORKFLOW VALUE PER CYCLE

When CI workflows are enabled at ASCR HEC facilities, they do not fit established charging models for system usage.[2] A CI runner process typically runs on a login node as a persistent, low-resource process polling the code repository server at regular intervals. Only when it finds a CI job does it initiate more computationally intensive work on the machine. However, this work is typically light because, for a CI workflow to provide beneficial feedback to developers, it must have a low turnaround time. Therefore, developers design these automated tests to be quick and minimal. One common use is a compilation test, which is not considered resource intensive under normal circumstances. This raises the question of whether facilities should charge for a CI pipeline that automates the process of signing on to a login node to do a code compile, when, if it is done manually by a user, it would be free of charge. The answer to this question is not simple.

CI workflows are fundamentally different than human-operated tasks and require an appropriate charging scheme. The value of CI to software development is derived from its low-effort, low-cost, and quick-turnaround characteristics. The difference in effort between human-operated tasks and automated CI workflows means that CI workflows have the potential to be employed at high frequencies. For example, it is easy to press a button on a Web interface 10 times to queue up 10 code compilations on a machine and report the results back when they are complete.

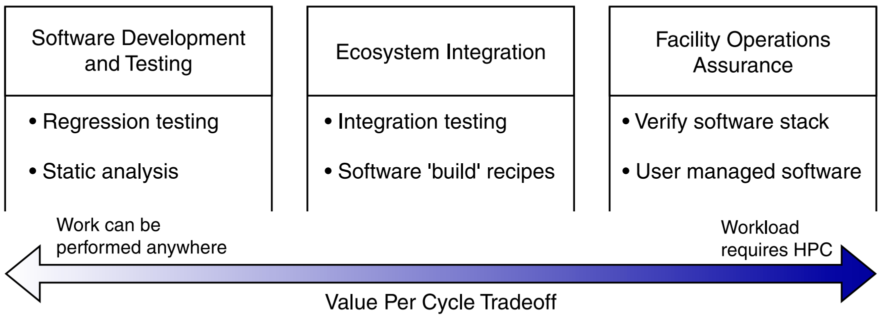

FIGURE 1. Different CI actions can generate added value when they are unique to a given HPC system. This tradeoff is important to understand, as we want to avoid using compute cycles on actions that can be achieved on alternative resources. CI: continuous integration; HPC: high-performance computing.

Compared to the effort of manually compiling a code 10 times, this is an enormous difference. Given the potential for increased use, facility staff could be tempted to charge higher compute-node rates for CI workflows. This is not necessary. Instead, software teams must be enabled to use CI workflows frequently to achieve benefit. The "cost" of CI jobs needs to remain low, and facilities should trust that the third characteristic of useful CI pipelines—a quick turnaround time—will help combat excessive use of resources.

In the current environment, CI pipelines at ASCR HEC facilities are underutilized. This is because node usage hours are distributed through a competitive process, and the cost of initiating CI workflows cannot match freely available cloud-based solutions, infringing on the low-cost requirement. Moreover, current security requirements increase the effort required to launch a CI pipeline, infringing on the low-effort requirement. With these two factors weighing against the benefits to development, the value per cycle of running CI at ASCR HEC facilities is less desirable than in other scenarios (illustrated in Figure 1). To encourage additional use of CI workflows, administrators at ASCR facilities will need to continue to minimize the cost and effort required of software development teams.

SECURITY ASSURANCE

Meeting the requirements for secure capabilities characterizes CI solutions at ASCR HEC facilities. The facilities are part of a larger national laboratory that is managed through management and operating contracts with the DOE. One of the main stipulations of these contracts is that federal regulations, such as ones laid out in the Federal Information Security Management Act, are followed for all laboratory systems that include the HPC resources that will run CI pipelines. In practice, each ASCR HEC facility maintains a customized set of security policies tailored to its supported use cases. Enabling any two facilities to meet these requirements with a single unifying set of policies thus becomes a highly complex endeavor. In addition, given the sensitive nature, any flexibility in security policy is exercised with the utmost care. Another major contention point is that the performance of early access systems is treated as a trade secret by vendors. The status of software that is passing or failing cannot show up in a publicly accessible way without violating many of the procurement contracts executed for HPC systems.

These policy differences mean that the most difficult technical challenges for CI at ASCR HEC facilities are not about the CI infrastructure or tests themselves but, instead, are adequately addressing the security needs for each site. This is especially true for processes that require elevated permissions, as a runner process shared among multiple users would. Moreover, CI presents a wealth of nontrivial security challenges.[3] For the purposes of this article, it is possible to state generally that the challenges having the most impact for CI workflows at ASCR facilities are the following:

› Strongly associating each running process with the human identity that intended it to launch.
› Ensuring that each user-initiated process cannot influence files or aspects of the system beyond their permission.

These general guidelines give a perspective for evaluating whether different configurations of CI infrastructure deployments satisfy necessary security assurances.

Associating Code With a User

Of the two challenges mentioned, the need to associate running code with a human identity is the primary challenge, and it revolves around allowing any level of automation, traditionally associated with CI/CD workflows, to occur on ASCR HEC resources. This is due to specific authentication requirements placed upon users as well as the need to create a strong association between cause and effect. The authenticator assurance level (AAL) is documented in the NIST *Digital Identity Guidelines*[5] as the "robustness of the authentication process itself, and the binding between an authenticator and a specific individual's identifier." In this context, we refer to this degree of verification as the "assurance level." This is an important concept when we think about the challenges associated with allowing CI/CD triggered from external sources, as we can easily envision workflows that could allow actions from a lower AAL source to trigger jobs on systems with potentially higher assurance requirements. Such a workflow would necessitate a more limited scope, thus potentially limiting the flexibility afforded to the users of the CI/CD system.

TOWARD CROSS-LABORATORY CI WORKFLOWS

During ECP, the CI team took up the task of establishing CI workflows for software development teams at the ASCR HEC facilities. Along the way, they met many challenges, and, in addressing each of them, the current CI solution took form. In this section, we take a holistic view of the development of CI infrastructure during ECP, detailing several important aspects, discussing motivations and status quo constraints, and explaining design decisions. While more could be said, what is described herein includes the major factors that shaped and will continue to shape CI infrastructure at the ASCR facilities for the foreseeable future.

Federated CI Model

At first, the plan for allowing projects to run CI jobs on ASCR HEC machines included cross-facility workflows managed from a central code repository—a federated model. Access to computing resources across the ASCR HEC facilities would be controlled by a single centralized mechanism. In this model, a single code repository from this centralized server would have the capability to launch CI jobs at any of the three ASCR facilities with simple user modifications to the associated job scripts (shown in Figure 2). This model would have required addressing a number of complicated policy and technical hurdles, such as tracking assurance levels—the degree of confidence in a user's identity, indexing identities across facilities, and implementing common resource access policies. In one instance of these challenges, it would be necessary to track the assurance level of a user at any point where one of their code modifications could trigger a CI job and match it to the appropriate permissions on the facility resource. For the federated CI model to be fully implemented, all of the associated challenges must be overcome. Instead, the team saw more gain in focusing on enabling CI individually within the scope of each facility's boundaries.

Site-Local CI Model

In the site-local CI model, each facility hosts its own code repository server. ECP software teams that were developing for all three ASCR facilities would then be able to pull their code into each site-local code repository, where corresponding CI pipelines could be launched at each respective site. The main advantage to this model is that it allows flexibility at the facility level to meet each individual site's security requirements. For this reason, it became the most widely accepted model for CI workflows at the facilities.

Shared HPC Integrated Runner

In the CI workflow, the runner process holds special importance to the security of the CI workflow. The

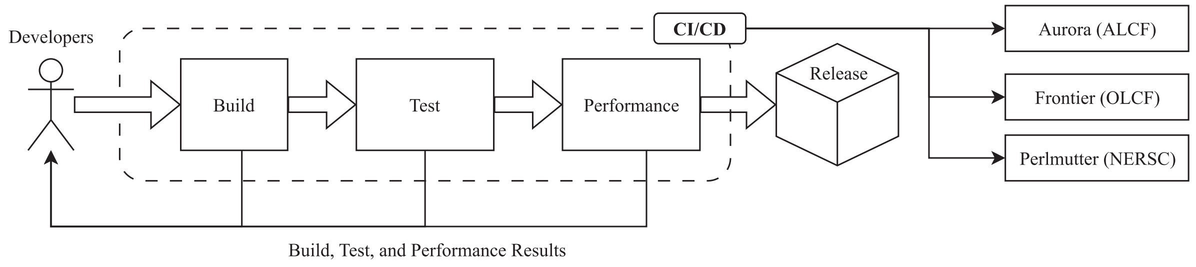

FIGURE 2. The primary goal for ECP CI was to empower developers with traditional CI/CD workflows that could be targeted at a range of a new systems with unique software environments, ensuring that builds are properly tested and obtaining the best performance possible. ALCF: Argonne Leadership Computing Facility; CD: continuous deployment; NERSC: National Energy Research Scientific Computing Center; OLCF: Oak Ridge Leadership Computing Facility.

runner process is installed and runs directly on the ASCR HEC machine login node, determining whether and how to execute the CI jobs it pulls down from the code repository. There are several possible arrangements for runner processes when enabling CI as a service. In the context of ECP, we can simplify the analysis into two representative configurations: a single staff-managed runner process, which we call a "shared runner," and a many-individual-runners configuration where each user or project runs their own runner process. Of these two approaches, the shared runner is preferable for several reasons:

- A single process lowers the maintenance burden for system administrators.
- It is easier to monitor.
- It minimizes the amount of time a runner process would be running while not doing actual compute work.
- A single runner process means there are fewer entry points into the system that a malicious user could compromise.
- Controls can be established to ensure that access to resources for CI processes can be fairly distributed.

While the many-individual-runners approach may have its place, the shared runner approach has been the main source of development for these reasons.

JACAMAR CI

To bridge the gap between CI services offered by GitLab and facility security requirements, the ECP CI team developed Jacamar CI.[4] Jacamar CI establishes an application layer that resides between the GitLab Runner[c] persistent service and the user's environment (shown in Figure 3). The primary goal of Jacamar CI is to establish a maintainable and configurable tool that is capable of extending the security model from the GitLab server to the host CI system, enforcing all required Unix file permissions, incorporating site-specific validation requirements, and providing flexibility in the directories used for job storage. Utilizing contextual information coupled with facility infrastructure, it is capable of mapping the user responsible for initiating the CI job to a local account; authorizing access to the runner; restricting permissions to match the privilege level of the user; and, finally, executing all runner-generated scripts as said user. This capability preserves the user identity from the server all the way through the completion of the CI pipeline, thereby ensuring that security requirements for CI workflows are met.

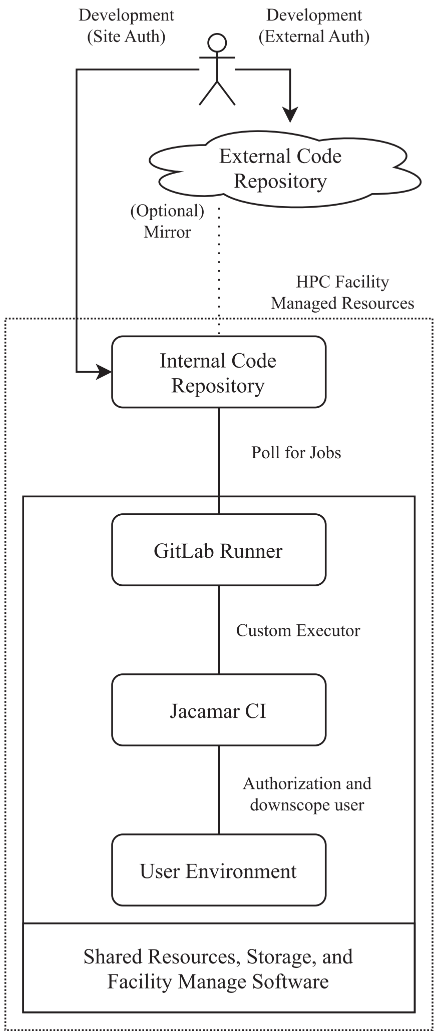

FIGURE 3. Depicts the most common deployment model for ECP CI that offers local resources to users through an internal repository that enforces the required authentication levels and user policies. ECP: Exascale Computing Project.

Before attempting to understand the Jacamar CI application and how it is used to approach the challenges of running CI at ASCR HEC facilities, it is valuable to closely examine the GitLab custom executor. The custom executor was created by GitLab as a way to meet the bespoke needs of ASCR HEC facilities and the niche requirements of other applications. Any GitLab runner has the configuration option to operate as the custom executor type. This mode provides administrators with a custom layer that can be used to insert desired controls between the GitLab runner and the target runtime environment.

[c]https://docs.gitlab.com/runner

Traditionally, a runner process would create and execute job scripts, such as the user's script as well as those responsible for running the related Git, artifact, and cleanup operations, targeting a defined executor (i.e., Shell with a traditional Bash shell or Kubernetes using a Pod). However, when the custom executor is used, these scripts are, instead, presented to the configured application along with the related job context, all of which can be used to determine whether and how the scripts will be run. The configured application can behave in any manner so long as it conforms to the GitLab-documented workflows and the expectations of the custom executor. To the user of a runner configured in this way, it is possible to continue to leverage standard GitLab CI/CD behaviors without knowledge of the existence of this custom layer or the controls it exerts on the job.

By leveraging the custom executor, Jacamar CI has been engineered as a completely separate application that an administrator of a runner can choose to use. Its goal is to exist between the runner and the HPC resource and enforce the following practices:

- Use of a JavaScript Object Notation Web token, generated and signed by the GitLab server, that authorizes each incoming request for a CI job and ensures that the user responsible for triggering the job has appropriate permissions to the underlying hardware.
- Setting the CI job permissions from the privileged runner process to match those of the authorized user.
- Execution of the runner-generated job scripts in a stable and repeatable manner using the underlying HPC scheduling system or simply the user's Bash shell. The goal in either case is to mirror the expectations one might have for running a script from a noninteractive terminal they can obtain logging in manually to the same resource.
- Ensuring that all job results are captured and communicated back to the user so they can be reviewed via a traditional Web interface.

Jacamar CI is, therefore, key to allowing direct access to managed HPC resources for testing and maximizes the reliability of the software and its ability to successfully leverage the hardware/software resources of the provided system.

A single authentication experience can be created when Jacamar CI is combined with an internally hosted GitLab code server, as shown in Figure 3. In the site-local model, each facility hosts its own code server and enforces its own authentication mechanisms to gate access. This greatly eases the process of supporting workflows that bridge the code server and machine because Jacamar CI relies on the server to enforce authentication. However, for ECP software development teams, this can be extremely limiting, as the majority of codes are managed externally in public spaces. The solution employed to date is to allow users to mirror limited aspects of their external repositories to the internal site-local servers, with targeted workflow restrictions. The exact scope of these restrictions and allowed CI workflows is established to meet the requirements for the environment. Some common restrictions include removing the ability to trigger pipelines based on actions occurring in the mirrored repository and restricting the lifespan of server-generated tokens. In all cases, restrictions increase user effort and can further diminish the "continuous" aspect of CI.

THREE POTENTIAL CI SCENARIOS

The difficulty of developing a CI solution for ASCR HEC facilities comes from balancing usability and adherence to security policy requirements for automatically triggering jobs on protected systems. Traditionally, security policies require a trusted user to initiate a job. Using the lens of trust between system providers and the developer community, thus, gives us a way to conceptualize several potential CI scenarios under three categories: external only, in which development is open access, and CI runs in the cloud; internal only, in which access to code and CI resources is restricted; and cross-boundary, where development is open access, but the resources for CI are restricted. Examining the broad characteristics of each mode brings into relief the challenges of providing CI workflows.

Scenario One: External Only

The simplest scenario is to place the source code and testing environment outside the facility. In this case, development is done on a public platform, like GitHub[d] or GitLab,[e] and computational resources are provided via cloud services in an open way that allows for little to no vetting of users. This type of openness provides many benefits to software teams.[6] Usually, security requirements for the cloud providers themselves necessitate running any CI pipeline within isolated containerized environments. This arrangement means that, in this scenario, the entire CI workflow in the

[d]https://github.com
[e]https://gitlab.com

development–test software cycle exists outside the security envelope of the facilities.

The challenge of an external-only model is that the combination of unique hardware and proprietary system software found at ASCR HEC facilities cannot be effectively replicated outside of a facility. The hardware, vendor software, and low-level systems are all highly customized, and the provided software environments are licensed by vendors. Although these challenges are not insurmountable, and even a partial replication of the ASCR HEC environment in containerized form could provide benefits, development in this direction requires technical support and a shift in the legal posture of vendors, which may or may not be financially viable.

Scenario Two: Internal Only

In the second scenario, code repositories that trigger CI pipelines exist within the same security envelope as the HPC resources. Only facility-vetted users are allowed access to the code repositories that trigger CI pipelines. In this case, code on the repository can be trusted to run at the HPC facility. This type of arrangement is most commonly seen in private industry and high-security computing environments. In addition, some software development teams may use the internal-only model for development but then mirror their source code to public repositories. Because the CI pipeline in this case still exists within the security envelope, we consider this development practice to be internal only and do not address it further.

The downside of the internal-only scenario is the negative impact it has on the productivity and goals of scientific software teams. In the case of ECP, teams need to obtain accounts at every ASCR HEC facility. Then, they need to manually ensure software was pushed to every single internal code repository using facility credentials. In a sense, this makes the “continuous” in CI a misnomer because many manual steps must be taken to completely test software changes, thereby removing the benefit to software development.

Scenario Three: Cross Boundary

The last scenario is a hybrid approach in which development is done in open access code repositories, and CI pipelines run on facility machines (illustrated in Figure 4). This approach holds the promise of combining all of the benefits of open development for the software team combined with access to the specialized hardware at ASCR HEC facilities. Unfortunately, triggering automated workflows from public code repositories means that, as the CI workflow crosses the security boundary, it moves from a less secure to a more secure environment. This conflict is central to the elevated difficulty of providing CI workflows to ECP software projects.

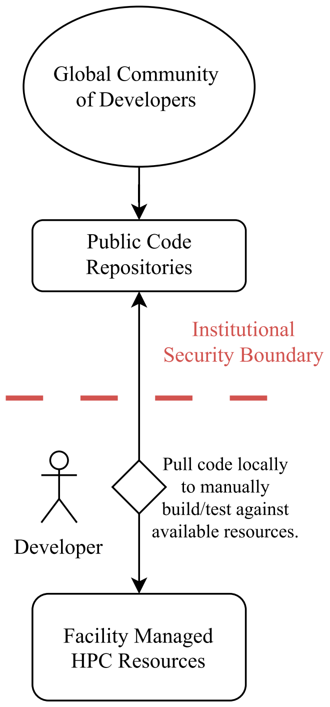
Global Community of Developers
Public Code Repositories
Institutional Security Boundary
Developer
Pull code locally to manually build/test against available resources.
Facility Managed HPC Resources

FIGURE 4. Considering the boundary between public repositories and protected HPC resources, ECP CI developed applications and workflows that ease the manually intensive nature of developing software for HPC resources in potentially restrictive environments.

Acknowledging these issues, the ECP took steps to advance the integration of automatic testing capabilities for software teams targeting the ASCR HEC facilities. Indeed, the promise of combining the benefits of a cross-boundary development–test model is too great to ignore. Although industry-standard CI practices, in which code is continually and seamlessly tested on ASCR machines, remain elusive, the ECP implementation has enabled capabilities that did not exist before. For example, teams can now integrate testing on some facility resources by triggering the CI pipeline manually or on a schedule and have the results reported back to the code repository server. These experiences have shown that, with the right CI infrastructure, the cross-boundary model yields worthwhile benefits.

FUTURE WORK

Promoting software sustainability and enabling good development practices for DOE software CI infrastructure at ASCR HEC requires continued technical and policy development. In this section, we discuss several suggestions for addressing outstanding challenges:

- Security constraints on ASCR HEC systems will continue to be at odds with low-effort CI workflows. Sharing resources puts a higher burden on CI workflows to meet security requirements. A solution to this problem is to provide dedicated CI hardware that can be supported in parallel to production resources. This would allow CI jobs to run in an environment that closely replicates the production system while remaining completely isolated. This isolation could allow less restrictive security policies.
- Developing and maintaining CI infrastructure will require continued support. As security threats and requirements continue to evolve, so must the tools we put in place to mitigate them. Jacamar CI is well positioned to be extremely effective in this regard because it is nimble enough to rapidly address both changes in code repository servers and facility security requirements. However, this will require continued code updates by skilled developers. Therefore, supporting this development is key to ensuring the success of future CI infrastructure at ASCR HEC facilities.
- Software automation should be recognized as a key part of the DOE science mission. This includes recognizing the need to allocate node hours on ASCR HEC machines for CI pipelines that power software development as well as understanding that developing CI pipelines is an essential aspect of developing quality software. Therefore, as software development becomes recognized as a need worthy of funding, so too should the work of writing code tests and automated pipelines.

CONCLUSION

During the ECP, the infrastructure to allow CI workflows at ASCR HEC facilities established a strong footing. However, a continued challenge to lower the effort required by software development teams to take advantage of these resources remains. As a result, many software projects are still cobbling together their own solutions for CI against ASCR HEC facility systems. In the abstract, multiple CI scenarios exist, but it is clear that open code development coupled with CI pipelines running on restricted-access ASCR HEC machines offers the greatest and most immediate potential for effective solutions. Through consensus building among facility staff, it was determined that federated CI—a one-size-fits-all model—would not be effective due to the complexities of both policy and technical solutions. Moreover, effective CI solutions will also require cultural shifts around the way resources are managed for automated CI processes.

By nature, the largest obstacle to enabling low-effort CI workflows is the inherent conflict with secure computing. Indeed, a naive implementation would contradict important security tenets that limit user permissions and associate user-initiated actions with their resulting effects. The success of Jacamar CI has proven that it is possible to develop effective solutions in this space by utilizing context from the server to identify, authorize, and ensure that permissions for the user responsible for the job are enforced. When combined with code repositories managed by the ASCR HEC facilities, it is possible to offer a seamless CI experience for software development teams.

Current CI solutions still do not offer the convenience available from common cloud-native services, such as https://github.com or https://gitlab.com. One compromise has been to mirror code from the public repositories of ECP software teams to the facility code servers where CI pipelines on the ASCR HEC machines can be run. This is done in exchange for increased security constraints, which hamper the convenience and, thus, the benefit to the software development teams that employ them. Looking toward the future, we see dedicated hardware for CI workflows with specialized security postures that could offer isolated cloud-like resources as a potential path forward. We also recognize the importance of the continued development of Jacamar CI to meet evolving security challenges. Ensuring effective CI is crucial to the software sustainability effort, and, as such, it needs to play a role in the allocation of node hours and the allocation of funding for the development of automated tests and pipelines.

ACKNOWLEDGMENTS

This research was supported by the ECP (17-SC-20-SC), a joint project of the U.S. DOE's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative. The ECP CI effort consisted of several years of work by many individuals spread out across the DOE as well as academic and industry partners. The authors would like to thank all those who participated and continue to participate in bringing this important capability to the DOE's most advanced machines.

REFERENCES

1. W. F. Godoy et al., “Software engineering to sustain a high-performance computing scientific application: QMCPACK,” 2017, *arXiv:2307.11502*.
2. “NERSC resource usage policies.” NERSC Documentation. Accessed: Jan 6, 2024. [Online]. Available: https://docs.nersc.gov/policies/resource-usage/
3. M. Shahin, M. Ali Babar, and L. Zhu, “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.
4. P. Bryant. “Jacamar CI (v0.16.0).” GitLab. Accessed: Jan 6, 2024. [Online]. Available: https://gitlab.com/ecp-ci/jacamar-ci
5. P. Grassi, M. Garcia, and J. Fenton, “Digital identity guidelines,” National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication 800-63-3, 2017.
6. O. Jokonya, “Investigating open source software benefits in public sector,” in *Proc. 48th Hawaii Int. Conf. Syst. Sci.*, Kauai, HI, USA, 2015, pp. 2242–2251, doi: 10.1109/HICSS.2015.268.

RYAN ADAMSON is the Security and Information Engineering Group leader at the Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA. His research interests include high-performance computing security, data science, and research software engineering, Adamson received an M.S. in computer science from the University of Tennessee. Contact him at rmadamson@gmail.com.

PAUL BRYANT is a software engineer for the Software Services Development Group, Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA. His research interests include internal applications and continuous integration (CI) technologies used on high-performance computing (HPC) systems. Bryant received an M.S. in computer science from Kent State University. Contact him at bryantpj@ornl.gov.

DAVE MONTOYA is the president of Trenza Inc., Los Alamos, NM, 87544, USA. His research interests including furthering CI capability with the addition of performance metrics to improve the development and deployment of applications in the HPC ecosystem. Montoya received an M.B.A. in information management systems from the University of Rochester. Contact him at dmont@trenzasynergy.com.

JEFF NEEL is a cybersecurity engineer at the Leadership Computing Facility, Argonne National Laboratory, Lemont, IL, 60439, USA. His research interests include ensuring that HPC systems are kept secure through security architectural reviews, compliance, and conducting security assessments. Neel received an M.S. in computer engineering and cybersecurity from Iowa State University. Contact him at jneel@anl.gov.

ERIK PALMER is a software integration engineer in the User Engagement Group at the National Energy Science Research Center, Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA. His research interests include consulting with users on HPC issues, building software, and coordinating user-facing CI efforts. Palmer received a Ph.D. in applied and computational mathematics from the University of South Carolina. Contact him at epalmer@lbl.gov.

RAY POWELL is a system integration administrator at the Argonne Leadership Computing Facility, Argonne National Laboratory, Lemont, IL, 60439, USA. His research interests include artificial intelligence and research software engineering. Powell received an M.S. in computer engineering from the Illinois Institute of Technology. Contact him at rpowell@anl.gov.

RYAN PROUT is a software engineering manager for the Software Services Development Group at the Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA. His research interests include furthering the HPC ecosystem through robust and distributed software systems. Prout received a B.S. in computer science from the University of Illinois-Springfield. Contact him at proutrc@ornl.gov.

PETER UPTON is a system integration administrator at the Argonne Leadership Computing Facility, Argonne National Laboratory, Lemont, IL, 60439, USA. His research interests include maintaining and supporting the GitLab instances used for CI by internal teams and facility users. Upton received a B.S. in computer science from the University of Witchita. Contact him at pupton@anl.gov.