RSE-ops: Fostering Culture for Collaboration in Research Software Engineering

Sochat, Vanessa Gamblin, Todd Add your name here!

Abstract

Research Software Engineering is becoming increasingly more complex in terms of technology and the need for communication between teams needing to work in both high performance computing (HPC) and cloud computing environments. This challenge would be well addressed by a movement to identify categories of functional need in this space, and best practices and tools used for each. In this paper we introduce "RSE-ops," or Research Software Engineering Operations, a movement to mirror DevOps (Atlassian, n.d.), best practices to bridge development and operations in typically cloud communities, and provide structure for better collaboration and navigation of the space.

Introduction

Successful development, deployment, and maintenance of research software is central to scientific discovery. In the last decade, the role of Research Software Engineer (RSE) ("A Not-so-Brief History of Research Software Engineers," n.d.) has risen to awareness, and fostered a community of combined researchers and software developers that focus almost exclusively on this task.

While some RSEs work on research software separate from its application, others are embedded in labs and responsible for data processing, analysis, and otherwise running tasks at scale to produce research outputs. These RSEs, whether they be staff at national labs, academic institutions, or private research institutes, historically have used some form of high performance computing (HPC) to achieve this scale (Wikipedia contributors 2021c; "A Brief History of High-Performance Computing (HPC) - XSEDE Home - XSEDE Wiki," n.d.).

This traditional practice has slowly been changing with the availability of cloud computing (Scality 2020). As the technological gap between HPC and cloud computing is closing (Guidi et al. 2020), and the cloud can equally meet the needs of research groups ("Challenging the Barriers to High Performance Computing in the Cloud - HPCwire" 2020), Research Software Engineers are pre-

sented with the task of working in both spaces. As they discover best practices and tools, there arises the need to write all of this knowledge down. Synthesizing what we know not only identifies what we know, but also what we don't know and where there are gaps that require attention or work. Arguably, a mature community should have awareness of:

- What are functional categories of need for the community?
- What are best practices?
- What tools are out there and recommended for each use case?

Further, there is separation between the developers of research software, and those that deploy it as a workflow or service. This problem isn't new, and in fact we can look to cloud computing for inspiration. Although cloud computing goes back to the 1960s (Foote 2017) and the term wasn't coined until 1996 (Scality 2020), what we are specifically interested in is DevOps – a movement that sought to bring together development of software and services ("Dev") with their deployment (operations, or "Ops") starting around 2007 (Atlassian, n.d.). Interesting, Research Software Engineering is going through the same challenges, and would benefit from the same kind of movement.

This white-paper introduces the concept of RSE-ops, or the intersection between Research Software Engineering and operations, which for research can mean running workflows or services. We present a first effort at defining relevant functional categories for the community, best practices, and the current landscape of potential areas of growth. We hope this structure can provide a basis for inspiring community and initiative around collaborative and meaningful work.

What is DevOps?

A definition of Research Software Engineering Operations (RSE-ops) can best be derived by first explaining the philosophy behind DevOps ("What Is Devops," n.d.) is a term that refers to best practices to bridge development and operations. It was coined in 2008 ("DevOps," n.d.a), and has grown out of a web services oriented model. The term has expanded to refer to other subsets of expertise in this area such as cloud security operations, which is called "DevSecOps," adding the "Sec" for "Security." In DevOps, best practices are generally defined around:

1. continuous integration: automated integration of code changes into a repository
2. continuous delivery: automated release of software in short cycles
3. monitoring and logging: tracking errors and system status in an automated way
4. communication and collaboration: interaction between team members and optimally working together

5. ”infrastructure as code”: provisioning resources through simple text files
6. using micro-services: single function modules that can be assembled into a more complex service

The above best practices are done for the purposes of speed and efficiency, reliability, scale, and collaboration. It has been shown that teams that adopt these practices can see improvements in productivity, efficiency, and quality across the board (“DevOps: The Shift That Changed the World of Development” 2020). It is a culture because along with these best practices, it also alludes to a way of thinking and working. Where there were barriers before between development and operations teams, DevOps brought them down. You can grow a community around these ideas, which is a powerful thing.

DevOps as the Driver of the Cloud

And surely the statistics are alarmingly good, as teams that practice DevOps outperform their peers in number and speed of deployments, recovery from downtime events, and employee ability to work on new things over tenuous maintenance (Webteam, n.d.). Recognizing these gains and providing structure for collaboration, training, and projects was arguably just one of the goals of the Cloud Native Computing Foundation (CNCF), which was founded in 2015 (Wikipedia contributors 2021b). Specifically, the primary stated reason for foundation of CNCF was to foster community and support around container technologies, which often are the core unit of automation and DevOps practices (nishanil, n.d.)). A new term, ”cloud-native” was coined with this title, which is heavily reliant on DevOps. DevOps practices are considered the fundamental base of taking on a cloud-native approach, and another term, ”Cloud Native DevOps” (“Cloud Native Devops,” n.d.) was even coined to specifically refer to the application of DevOps practices to the cloud. Since the two are so inexplicably intertwined, for the remainder of this paper, we will refer to them interchangeably (Choice 2020).

What is RSE-ops?

The role of Research Software Engineer (RSE) has been emerging in the last decade, and due to the hybrid nature of working environments across cloud, and HPC, DevOps practices are logically being adopted. So if DevOps is the intersection of ”Developer” and ”Operations,” then how does this concept map to this new space, where high performance computing, or more generally, Research Software Engineering is at the forefront?

Inspired by DevOps, we can define a similar term for the Research Software Engineering community to also inspire collaboration and champion best practices – RSE-ops. Research Software Engineers (RSEs) (“A Not-so-Brief History of Research Software Engineers,” n.d.) are those individuals that write code for scientific software, and more generally support researchers to use codes on high performance computing systems, cloud services, and lab computers. Akin

to traditional Software Engineers at major tech companies, they are responsible not just for software development, but also for deployment of analysis pipelines and general services. It can be noted that we are not calling the new term RseDevOps (dropping "Dev"), and this is done intentionally as the term "Research Software Engineering" encompasses this "Development" portion. RSE-ops, then, appropriately refers to best practices for ensuring the same reliability, scale, collaboration, and software engineering for research codes. We may not always be running a scaled web service, but we might be running scaled jobs on a manager, profiling performance, or testing software in development.

Thus, RSE-ops is the intersection of Research Software Engineering and Operations, and generally refers to best practices for development and operations of scientific software. Arguably, the RSE community has just as much to gain by building community and putting structure around these practices. It's important to note that while high performance computing (HPC) has traditionally been a large part of scientific computation, researchers have extended their tools to also use cloud services and other non-HPC tools, so HPC is only considered a subset of Research Software Engineering and thus RSE-ops. Many modern applications are web-based and extend beyond HPC, and so it is important to consider this set as part of the larger scientific or research software engineering universe. However, the dual need to run or deploy application across environments presents greater challenges for the community.

Comparison of RSE-ops vs. DevOps

The easiest way to start to map out the space of RSE-ops is to address a series of questions about people, goals, and practices, and make a direct comparison to DevOps. On a high level, RSE-ops has a stronger association with HPC, while DevOps has a stronger association with the cloud, but the lines are blurry. While early efforts of some of these clouds attempted to re-brand HPC ("Google HPC," n.d.), progress has been made to the point that the gap between cloud and HPC is narrowing, and HPC centers are able to take advantage of cloud technologies, and vice versa. There are still subtle differences, and ideally there could be convergence to empower researchers to use software across different platforms. For this reason, we think that making comparisons between the two can be helpful to understand what practices are well established for RSE-ops, and which require further development. Since there is a stronger association of HPC with RSE-ops, in the discussion below we will often be comparing HPC with cloud, however this does not say that there is always a strong dividing line between the two. We will proceed in the following sections to ask questions of each, speculate on best practices, and then summarize our findings in a table.

What are the goals of each?

Arguably, the goals of DevOps are to provide applications and services, typically web-based. The goals of RSE-ops are to sometimes provide services for scientific

applications, but more-so to provide infrastructure and means to run, develop, and distribute, scientific software. RSE-ops, then, is for research software and services, while DevOps is typically for more widely available, persistent services and corresponding software. This does not mean, however, that RSEs are never involved with DevOps, nor that industry Software Engineers are never working on research software.

Who is involved?

You will typically find individuals practicing RSE-ops at academic institutions, national labs, and some private industry, or anywhere that high performance computing is the primary means of compute. While some companies might also use high performance computing, typically we likely find that larger companies maintain their own internal container orchestration system (e.g., Google uses Borg ("Large-Scale Cluster Management at Google with Borg," n.d.), and smaller companies pay to use cloud services that offer a similar set of tooling. Likely this decision results from some cost-benefit analysis (Prabhakaran and J. 2018) that determines that one is more cost effective than the other. Whether we look at Google Cloud ("DevOps," n.d.b), Microsoft Azure ("DevOps," n.d.c) or Amazon Web Services ("What Is Devops," n.d.), all of these cloud environments have a primary focus on distributed, scaled, and "server-less" technologies. We might call this cloud computing.

When we look closely at individuals involved, it tends to be the case that institutions with HPC have a combination of Linux Administrators, Support Staff, Research Software Engineers, and Researchers. The Research Software Engineers in particular play an interesting role because they can sit on the administrative side (with Linux Administrators and Support Staff), on the user side (with Researchers) or somewhere in between. For this reason, they are essential staff for communication, or ensuring that the needs of the researchers are known by those that run the resources. For tech companies, it's likely the case that a DevOps team or team of Support Reliability Engineers (SREs) is tasked with managing software and services for the company. The SREs are primarily concerned with how things should be done, and developing monitoring and other support tools, while a DevOps teams is primarily concerned with doing it ("Google - Site Reliability Engineering," n.d.). The line gets blurry with respect to titles, because a company can have some flexibility with respect to naming these roles. However, it's common to see titles like Software Engineer, DevOps Engineer, SRE, or even Cloud Architect.

Accessibility for RSE-ops vs. DevOps

When we talk about accessibility in this space, we usually are referring to how a user gets access to software and services. RSE-ops that is focused around HPC usually means providing access to a research cluster, or typically hundreds to thousands of nodes that are connected on a network. These clusters are opti-

mized for command line usage, meaning that users log in via a terminal, and type commands in the prompt to run and monitor jobs, use software, or otherwise interact with the cluster. While some centers have figured out options for interactive notebooks that are accessible via a web interface (e.g., OnDemand from the Ohio Supercomputing Center ("OnDemand," n.d.) or JupyterHub ("Jupyter Hub," n.d.) for several national labs and academic institutions), typically, opening up resources to a web interface is not a well developed area. Some might argue it's an issue of not wanting to take risk of "opening up" clusters to a web interface, but realistically it could be called a skill gap. Web technologies were greatly developed and adopted in the more commercial world, and already overworked system admins don't have the bandwidth to learn this new skill to develop apps and what is required to secure them. As a buffer to this missing service, many centers take a hybrid approach, meaning they make cloud resources (from a vendor like Amazon Web Services (AWS) or Google Cloud or Azure) available separately to groups that have need for these interfaces. This is often called a hybrid cloud ("What Is Hybrid Cloud? - Benefits and Advantages of a Hybrid Cloud," n.d.). However, as many centers embrace container clusters like Kubernetes and OpenShift, there will soon be this kind of cluster alongside an HPC cluster to run web and storage services asked for by users. The early work to bring interfaces to high performance computing ("Web Portals for High-Performance Computing: A Survey," n.d.) will continue, and following this development will be exciting in the coming years.

DevOps, or providing software and services in the cloud, might have an accessibility advantage over RSE-ops in that the cloud is more likely to be accessible. Theoretically, both HPC clusters and cloud resources can be accessed anywhere with an internet connection, however large HPC centers are more likely to have more stringent access requirements (e.g., logging in with a token as opposed to OAuth2 in a browser), and are less likely to be accessible from the browser or even a mobile phone. As a buffer against this, many app teams go out of their way to provide web of container applications for distribution of their work.

Maintenance for RSE-ops vs. DevOps

RSE-ops on HPC means that a number of systems are running all the time. They require a team to monitor, maintain, and take care of them. Cloud development does not require the resource requester to think about maintenance of a system any longer than the resource is needed. The resources are truly "on demand." An instance, or small container cluster, might be requested when it's needed, and brought down and forgotten about when the work is done. This is a much easier interaction for the user, as they can request what they need when they need it. For a DevOps team, even if they are deploying container-based services, they don't need to worry about long term care or maintenance of servers.

From the user perspective (the researcher or end-user of a cloud service), both HPC and cloud users don't need to think about maintenance of the system. The resources are available and can be requested. Thus, the stark difference is

that organizations with HPC typically host them on-premise, meaning that the organization needs to pay for everything from people to maintain and operate to the energy and cooling of the systems (Carlyle, Harrell, and Smith 2010). On the other hand, although operational costs might be reduced using a cloud resource, that cost can explode quickly depending on the kind and scale of cloud resource that is needed (e.g., GPUs) (Li, Walls, and Guo 2020). There is much knowledge in the HPC community about hidden costs from the cloud, and doing comparisons between cloud and on-premise to find that on-premise can be half the costs (Morgan and Hemsoth 2021; "Magellan: A Cloud Computing Testbed," n.d.). One of the challenges is that the costs are constantly changing. Despite wanting to ultimately maximize profits, cloud providers typically provide cost calculators to help with this, provide free-tiers and "spot instances" at a lower cost, high use discounts, and do not charge for services (e.g., instances) that are turned off (Power and Weinman 2018).

While we cannot suggest universal best practices for maintenance, it is logical that each institution needing resources do a cost benefit analysis to compare all options that meet a set of community needs. It might be more logical for small organizations (that perhaps cannot afford hosting and maintaining their own) to use cloud resources, and for larger organizations that are constantly using resources to host their own to save money.

Scientific Software for RSE-ops vs. DevOps

Installation of scientific software on HPC is a non-trivial task. To maximize the efficiency of any piece of software and maximally utilize an underlying resource, it's typically ideal to install it natively on the system, as opposed to a containerized environment, which is arguably a foundation of modern DevOps ("Containers Intro," n.d.). However, studies are increasingly showing that containerization overhead is in fact low (Torrez, Randles, and Priedhorsky 2019), and so the main distinguishing difference between using containers ("DevOps") and native install (traditional HPC) is ultimately the portability of the software, and that HPC software must be built across a wide set of architectures. RSE-ops is an interesting combination of these two realities, as containers are widely used on HPC ("Singularity: Scientific Containers for Mobility of Compute," n.d.; "Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC," n.d.; "Podman - : A Tool for Managing OCI Containers and Pods" 2018) but the environments must also support having complex software stacks alongside one another without conflicting dependencies. This means that modules are common ("LMOD," n.d.; "Environment Modules," n.d.), and even modules paired with containers ("Singularity Registry HPC," n.d.) and users are encouraged to "BYOE" or "bring your own environment" through containerization and local install in the case that the software can be installed in user space. Otherwise, the software must be requested to be installed by the administrators of the cluster.

Thus, the strategy for deployment of scientific software usually comes under

the decision of the Linux administrators that manage the cluster. A piece of software or new technology can only be embraced when it is asked for heavily by users, and this ask is finally heard by the Linux administrators. While some large centers have separate teams for system admins and user support staff, smaller institutions tend to have a small group of people serving both roles. In practice, these over-burdened staff cannot devote extra time to learning new technologies and bringing more DevOps-like practices to their systems. Users of the systems, although they can control external services, don't have control to do so either. This arguably is why DevOps practices have been slower to become part of high performance computing culture. Given the complexity of using software on HPC and the many different strategies for enabling users to run it, some trade-off between portability and performance is required. Due to limited permissions, users are not always empowered to install and run services like they can in the cloud. Using cloud resources for development (and embracing DevOps) gives users a lot more freedom because they typically do not need to ask for permission. Arguably, there is a learning curve to this knowledge, but also arguably, there is a learning curve for most new things that have the potential to improve a daily workflow.

Best Practices for Scientific Software

We argue that it should not be entirely the burden of Linux administrators to learn and adopt new practices, but rather they should have the support of Research Software Engineers (RSEs). If Linux administrators have little bandwidth left for testing, bench-marking, or otherwise working on research software, RSEs can step in to both understand the workings of the underlying system, but also the needs of the researchers or user-base. This is a a compelling example that demonstrates the need for Research Software Engineers (RSEs), who can not only better bridge the gap between the administrators and users of a system, but also can focus on defining best practices in systems, behavior, and software for using it. Adding this layer of best practices for automated builds, testing, organization, and deployment of scientific software on HPC is what we would define as RSE-ops.

Given the current ecosystem where RSEops best practices are not known, we can make suggestions that best handle this dual need for modularity or containers and native installations. For installing scientific software, the package managers spack and software build and installation framework easybuild or module managers LMOD and environment modules ("LMOD," n.d.; "Environment Modules," n.d.) are most commonly used to install and manage scientific software. Container technologies are also used to allow researchers to use containerization ("Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC," n.d.; "Podman - : A Tool for Managing OCI Containers and Pods" 2018; "Singularity: Scientific Containers for Mobility of Compute," n.d.; "Shifter," n.d.). Many of these tools allow flexibility to transition between tools, such as spack producing files to build containers, and containers installing

software from spack ("Autamus," n.d.). Package managers are a strong collaborative framework because they provide a structured way for people to work together on software together.

Testing for RSE-ops vs. DevOps

There are arguably two kinds of testing - testing of systems, and testing of scientific software.

Testing Scientific Software

There is a huge divide between testing scientific software and using it on the shared resource. The reproducibility crisis that grew in the early 2010s ("Replication Crisis," n.d.) did advocate for more use of version control ("GitHub," n.d.; "GitLab," n.d.) for research software collaboration and development, and the ease of integration of continuous integration (CI) external services for automated testing and deployment ("Travis-CI," n.d.; "Circle-CI," n.d.; "Jenkins," n.d.) made it more common for researchers to test and deploy their scientific software. It became more common practice to create releases alongside code, distribute them via packages managers, and even to provide containers that could be pulled and used on a cluster. However, this test and deployment process was notably separate from the HPC resources – it simply was not possible to test codes on all of the possible semi-customized HPC environments on which they might be run.

Understandably there would be security issues connecting an external service to a shared resources, however this does not fully excuse the lack of innovation in this area. It should be easier for a researcher to test and even deploy their codes on or beside the resource that they ultimately will be used. While some centers do have testing clusters, it's not always easy to justify cycles on them to be used just for testing. When software testing isn't in the hands of the user, testing the software stack might fall in the hands of the system administrator or support staff of the HPC cluster, or even worse, nobody at all.

Ideally, best practices for testing will embrace the current approaches that developers are already using to test and deploy software, namely using services like GitHub or GitLab and then creating releases, contributing install recipes to package managers, and deploying containers for quick usability. Arguably, what HPC centers might be able to bring to testing is scale. As tests need to be run frequently and across many different architectures, the incentive structure is arguably not there to maintain such a testing framework. Testing tools such as Pavilion2 and ReFrame are intended for facility testing, but there is no convergence on any kind of standard. Perhaps there should be, and this is something that should be discussed by the HPC community. The need seems to be there, as academic centers and national labs are slowly adopting their own strategies for quickly being able to run tests from version control on a cluster resource, , and likely these technologies will need to be shared with other institutions

followed by convergence on a shared best practice.

Testing of Resources

Most centers do some kind of kernel and application bench-marking ("Measuring High-Performance Computing with Real Applications," n.d.), along with testing of resources. For example, system administrators might use regression testing, including testing file permissions and mounts, communication between nodes and compatibility with the resource. For these regression tests, tools like Pavilion, NHC and Interstate are popular.

Another continuous integration strategy is to focus on regression testing via tools like ReFrame. Tools of this type try to test HPC systems alongside software stacks directly on the resource, and typically separate from any web interface or alongside the code. ReFrame and Pavilion are designed to test the machine, and not the software. Ideally we would have a standard application testing tool. Likely better ability to do testing will come from improved integration of Continuous Integration (CI) services, discussed later in this paper.

Scaling for RSE-ops vs. DevOps

High performance computing was designed for scaling, but it must be done intentionally. By way of job managers ("SLURM: Simple Linux Utility for Resource Management," n.d.; "Flux Framework," n.d.), Message Passing Interface (MPI) programming ("MPI: A Message Passing Interface," n.d.), and extreme parallelism ("Inside HPC Future Technologies," n.d.), it's possible to scale a task on an HPC cluster. Libraries like MPI are tested and can be used by HPC experts to run parallel applications.

This is in contrast to some traditional DevOps tools that handle scaling or scaled testing for the user, and typically have a strategy of deploying many separate instances up or out instead of using a connected fabric (Reese, n.d.). A prime example is the container orchestration tool Kubernetes ("Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus," n.d.), which centers are starting to deploy alongside traditional resources, possibly allowing RSE-ops to better overlap with DevOps. Emerging tools like the Flux Framework ("Flux Framework," n.d.) are further closing the gap between traditional HPC managers and container orchestration tools. The work remaining to be done is bridging the gap between the developers of the tools and the user base. If it isn't infinitely easy for a user to launch a job, then arguably the frameworks and techniques are not successful. There is a huge opportunity for development of RSE-ops tools that can better teach and possibly automate this scaling. There is also opportunity for cloud or DevOps practices to learn from traditional HPC and think about providing similar options.

Software Distribution for RSE-ops vs. DevOps

The end-goal of an HPC environment is to make software available to researchers. This is abstractly similar to the "end goal" of a cloud environment, which is to make software available to users via services. For the cloud, this typically means interactive web interfaces or services that are accessed from other web or terminal clients via application programming interfaces (APIs). Since HPC environments are typically running jobs that rely on system software and some number of known aliases, the bulk majority of "software distribution" for HPC is probably command line interfaces. In this framework, the administrators must provide an easy way to manage a collection of software, and make it available to users, and app developers that maintain their own deployments have to develop within that. At best, you can create modular structure and exposure of different software via file-system organization (e.g., putting apps in '/project') and using environment modules ("LMOD," n.d.; "Environment Modules," n.d.; "Singularity Registry HPC," n.d.) (e.g., 'module load' only what you need).

Users are somewhat empowered to install their own software, given that 1. they have the file-system space to do so, and 2. installation does not require any escalated permissions on the system. While this could mean a "bring your own environment" scenario via containers pulled from external registries, more likely it means relying on package managers and module systems. Another solution to this problem could be from better HPC container run-times.

Ideally, every module and package would have a suite of tests to ensure its functionality, but in real world scenarios its likely that the software is tested elsewhere (e.g., a CI service associated with the source code) and distributed to the system for immediate usage.

Dependency Management for RSE-ops vs. DevOps

Dependency management refers to a set of practices used to manage software versions. It is arguably a subtopic of software distribution. Every piece of scientific software requires a specific set of versioned dependencies, and these dependencies must co-exist alongside one another on a shared resource. This also means that architectures must be maintained for possibly older dependencies that require them. Tools have grown out of this need that allow for flexible management of dependencies and software, including easybuild ("EasyBuild," n.d.), spack ("Spack," n.d.), and environment modules ("LMOD," n.d.; "Environment Modules," n.d.). Development of these projects has also led to a powerful model of development – bringing many people (administrators and users) together to collaborate on the software. It's not hard to read user surveys ("Spack User Survey," n.d.) and see that the open source model of development, when done right, is successful. People are using the software, excited about it, and working together to make it better. It also doesn't hurt that some of these projects have the backing of entire institutions and within- and inter-institutional funding

programs and resources.

These success stories give us a hint that working together on software, likely in an open source, collaboration fashion, is a good model for successful distribution and improvement of the software. This does not imply that collaboration is always, universally better ("The Mythical Man Month," n.d.), but the authors here believe that it's an honorable if not idealistic goal to strive for. The challenge here, of course, is the extra work that it takes to seek out, interact with, and inform contributors. Not everyone may agree that software can and should be open and collaborative, and it may even depend on the kind of software in question.

DevOps is different because you only install what you need, and when you need it. It's uncommon to require older versions of the same software to co-exist for a service. Is there any kind of mapping of this freedom to research software engineering? Yes, arguably containerization, primarily with container technologies suitable for an HPC environment ("Singularity: Scientific Containers for Mobility of Compute," n.d.; "Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC," n.d.; "Shifter," n.d.), has allowed for encapsulation of an entire operating system and software stack that can be used in a portable manner. However, as was noted in the description of testing, development and deployment of these containers is unlikely to be on the resource itself. Ultimately, if we can better use unprivileged container technologies, running builds on clusters could be possible.

There also, until recently ("Singularity Registry HPC," n.d.) has not been an easy way for a cluster user or Linux administrator to install, manage, and provide containers. External registries that might be used ("Singularity Registry," n.d.; "OCI Distribution Spec," n.d.) must exist alongside a resource and then require an individual to explicitly pull a container binary. While this is better than not having containers available at all, it creates redundancy of file-system space, and requires the individual to be a stickler about container versions pulled, good practices, and organization of said containers. This is not an easy thing to do, and thus it's hard to follow best practices ("Ten Simple Rules for Writing Dockerfiles for Reproducible Data Science," n.d.). Development of these technologies, however, hinted at the idea that with some innovation, it could be possible to empower users to better embrace more DevOps style practices of developing, testing, and deploying research software (RSEops). It also hinted at a more modular strategy for dependency management. Interestingly, containers are used heavily by both cloud and HPC developers, but developing them and ensuring security in their execution adds additional challenges.

Permissions for RSE-ops vs. DevOps

One of the biggest challenges of using an HPC system is likely permissions. Although users can pull from container registries or install software from repositories, they are not permitted to write to anywhere other than a standard

home or scratch space. While some users have project spaces that are shared by multiple users, they can be a pain to manage. This is in contrast to a cloud environment where it's fairly easy to spin up a new instance and be root to do whatever you please. This is unlikely to change, and will be a factor that needs to be worked around.

This isn't to say that root should be always required, or that allowing the user to have it is best practice. Container technologies are increasingly going "rootless," meaning they can operate fairly successfully in user space ("Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC," n.d.; "Podman - : A Tool for Managing OCI Containers and Pods" 2018; Priedhorsky et al. 2021). Arguably, if the HPC community had been more involved with the Open Container Initiative (OCI) earlier, "rootless" containers would have been a prominent point earlier and we'd be farther along now. There is clearly no "fix" to give a user extended permissions, but rather software that can empower them to deploy their user stacks without asking for permission. Along with rootless container technologies, package managers like pip or conda can make it easy to install software in user space.

Portability for RSE-ops vs DevOps

In DevOps workflows, portability is achieved by way of using entirely isolated container environments. Although there still may be host or kernel dependencies, it's fairly trivial on the cloud to select a different architecture for this. Running on HPC, however, is much more complex, as there is typically more of a seamless environment with the host, or if not, the need to share libraries between the container and host. Technologies such as MPI and infiniband, and more generally, complex hardware and different kinds of performance optimization setups mean that there is a trad-eoff between portability and performance (Younge 2019). Even if a container does run, it could be that there are huge losses in performance.

The current "best practices" for this issue are generally to try binding libraries from the host into the container, or to add needed libraries from the host to the "LD_LIBRARY_PATH" (Younge 2019). A better future, however, could go in one of two directions. A more hard-coded approach would be to define a standard set of metadata (e.g., labels) alongside the container, and have them checked with a hook ("Opencontainers/Runtime-Spec Hooks," n.d.) that would do a quick comparison of labels with what is available on the host. More ideally, there could be some kind of compatibility layer that is able to trace or discover libraries in the container, compare to what is available on the host, and do some kind of ABI compatibility solve to determine if the two are compatible. This work is currently underway with the BUILD SI project at Lawrence Livermore National Lab ("BUILD," n.d.), and discussion of such a compatibility layer has happened previously (Younge 2019).

Community standards for RSE-ops vs. DevOps

Cloud services provided by several different vendors have flourished for many reasons, one of which is directly related to the community effort established around open source and standards.

Specifically, the primary unit of operation, for DevOps, the container, has had extensive collaborative work in a community and governance structure called the Open Container Initiative (OCI) ("OpenContainers," n.d.). The history behind OCI reflects an organic, open source and community effort growing to meet the needs of a changing industry landscape. Specifically, Docker ("Docker," n.d.), by way of being the first prominent, commercial container technology and having developed its own container registry, Docker Hub ("Docker Hub," n.d.), was an early contributor to these standards. An early version of their RESTful API to interact with container manifests and layers to push or pull containers was adopted into the first distribution spec of OCI, along with an image, runtime, and digest specifications. A container registry or technology is considered OCI compliant if it meets the criteria of these standards. The OCI standards ensured that there was consistency between cloud providers, and that users could translate between them without issue. Representatives from the major industry leaders worked on the standards together, and eventually community members for traditionally non-OCI compliant containers like Singularity joined the effort.

While having common standards is considered good practice in the RSE-ops space, because the effort has been primarily industry driven, the needs of more academic or HPC container technologies are not well represented. Academic and high performance computing efforts are arguably siloed, meaning that there are many libraries developed over time that are rooted in a particular lab or institution, and (with a few exceptions) groups do not work together. RSE-ops can only be successful if this issue is addressed, meaning that groups from national labs, academic institutions, and other communities that rely on HPC figure out how to successfully work together, not just on software but on standards. A good first step would be joining the industry effort to work on OCI standards so that the needs of this different environment are represented in the community, and an equivalent step would be to better collaborate on software projects, training, and best practices. While collaboration does not always lead to the best outcome (Wikipedia contributors 2021a), arguably it is inherit in software development (Whitehead et al. 2010) and we have a lot to gain to work harder at doing it well.

This might be more challenging than it seems at face value, because not every organization has spare developers or staff to devote to this responsibility. Discussing and maintaining standards can also be tireless, hard work.

Continuous Integration for RSE-ops vs. DevOps

Continuous integration is an established practice of continually testing and building before deployment. It typically is focused around a version controlled

code-base (e.g., GitHub or GitLab ("GitHub," n.d.; "GitLab," n.d.)) and developers collaborate to review code, ensure that tests pass, and then merge into a main branch. The goals are generally to ensure that when a piece or software or service hits a production resource, there will be no bugs or errors. Artifacts can be built and deployed on an event such as a merge, or for a versioned release. The benefits of CI practices are obvious, allowing developers to more easily collaborate on a code-base, and interact with production artifacts on resources of interest. For DevOps this typically includes different container orchestration services, instances, or app deployments, and for RSE-ops this would likely mean a local server or HPC cluster. The version control system also typically provides an interface to make it easy to report and respond to bugs (typically called issues) along with making suggestions to improve code (called pull or merge requests) and tracking progress. This kind of workflow fits well into any DevOps environment where change is a constant.

In the context of HPC, the diversity of resources that codes need to run on proves to be a much larger challenge for CI. The most popular, public CI services are not designed to interact with a batch system. It typically isn't sufficient to just test on a standard Linux x64, Mac, and Windows architecture, but several architectures across different operating systems (primarily Linux) that reflect the variety of the cluster nodes. The first challenge this presents is that most standard, web-based CI services (e.g., GitHub actions runners, standard GitLab runners, TravisCI, Jenkins, and CircleCI) don't provide a rich variety of architectures or graphical processing units (GPU), or when they do, they don't provide much control of when you get them.

For this reason, for research software we typically see the developers still develop using these systems, but then are not able to test on all the systems that might be needed. The challenge is thus getting any kind of modern web-based interface linked up directly to a cluster to run tests. Work at national labs using GitLab (Mendoza, n.d.; "Welcome to the ECP CI Documentation — ECP Continuous Integration Documentation," n.d.), and specifically having a locally deployed Gitlab server with custom runners on various resources ("Administration — ECP Continuous Integration Documentation," n.d.) has been a way to unite these two worlds. Managing such an integration requires thinking about how to manage resources, user accounts, time allocations, and machine access. Another challenge is that there is is no standard workflow language for CI, and perhaps there should be.

Continuous Deployment for RSE-ops vs. DevOps

Continuous deployment is the step after continuous integration finishes, namely the deployment or archive of build artifacts for use in production. For DevOps, this typically means pushing containers to a registry where they can be pulled to some production or testing environment.

On HPC the software is optimally installed on bare metal instead of relying

on containers or some kind of container orchestrator. For HPC it's also up to workload managers ("SLURM: Simple Linux Utility for Resource Management," n.d.; "Flux Framework," n.d.) to run the software. These resources are also typically protected, and not easily accessible from a web interface.

For RSE-ops on HPC, a similar approach using containers could be adopted, however due to the private nature of software it could be that registries and services are internal. A self-deployed CI service (e.g., Jenkins or GitLab ("Jenkins," n.d.; "GitLab," n.d.)) that has access to cluster resources could either deploy an artifact or use SSH to execute a command to do similar. A container could be pushed to an open source or internal registry. One important question that results from this is how to control access and security. If a user builds a container in CI, how is it assessed or scanned to be safe to deploy? There is also one additional level of complexity with providing containers with different architectures to run optimally on the system. Finally, build caches (meaning binary packages) would be an equally good solution, or in fact a supplement. Build caches are ideal for reuse, and containers for reproducibility.

Monitoring for RSE-ops vs. DevOps

Monitoring for both RSE-ops and DevOps can generally be referred to as looking at metrics and logs to assess performance of systems and software. It can even be extended to talking about performance testing, provenance and reproducibility of scientific workflows, and data organization. In both spaces, administrators of resources typically choose strategies to monitor their systems. In cloud development, monitoring services might be more easily integrated into different services, and in HPC some more traditionally DevOps tools like Kubernetes, Grafana, and Prometheus are starting to be used ("Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus," n.d.). However, best practices for monitoring have not been established, nor have best practices for running workflows and storing provenance.

Security for RSE-ops vs. DevOps

Security is of utmost importance for HPC, and consequently the environment is much more restricted for the average user than if the same user was using the cloud. Specifically, instead of a single user having root, there are potentially thousands of users sharing the same systems, and thus the systems are structured to ensure scoped permissions. The user cannot write to anywhere in the system other than a home or work-space, and the user largely does not have control over networking, or further customizing a system provided resource. These points are related to the permissions section.

Within DevOps, a community has formed around best practices for security called "DevSecOps" ("What Is DevSecOps?" n.d.). Although there is more freedom in the cloud, multi-tenant projects still require that different teams

maintain unique namespaces, and that services are deployed to avoid any kind of security breach or intellectual property violation. Thus, DevSecOps (akin to DevOps) aims to automate having security checks for each step in the life-cycle of a cloud service. DevSecOps also is a community, and strives to increase general awareness about security best practices. This is a different approach than traditional security checks, which might have done an evaluation of a piece of software before deploying it to production, and likely this is still considered the best practice for HPC. However, ideally there could be an equivalent movement to focus and work on security best practices, culture, and community for RSE-ops.

For both these groups, containers present a unique challenge, as it seems almost impossible to keep track of every container that a user might bring to a cluster or cloud environment. At best there is security scanning in the registry (and the user of the container takes notice) and the container technology is designed in a way to not allow any escalation of the user to a root user (e.g., Singularity ("Singularity: Scientific Containers for Mobility of Compute," n.d.), or Podman ("Podman - : A Tool for Managing OCI Containers and Pods" 2018)), unlike cloud container run-times (e.g., Docker ("Docker," n.d.)).

Description	DevOps
Goals	Commercial software and services
People	Industry software engineers
Accessibility	Freedom to access from browsers, and anywhere with an internet connection
Maintenance	Request, use, and throw away when done
Staffing	No staffing required
Scientific Software	Software and services can be modular, and optimized for the application or servic
Scaling	Scaling is typically automated.
Software Distribution	Complete freedom to use any software distribution or package manager.
Permissions	Complete freedom
Accessibility	Browser and command line, even from mobile
Testing	Automated testing and deployment alongside and integrated with cloud resources
Dependency Management	Easy to use bleeding edge software, and install only what you need when you nee
Community Standards	Significant time and effort to establish standards for containers
Continuous Integration	Well established practices and integration of version control with build, test, depl
Continuous Deployment	Comes down to pushing containers to registries for production systems
Monitoring	Monitoring is well integrated into services
Security	DevSecOps is leading the way to make security an automated part of the develop

Why can't RSE-ops follow DevOps?

The biggest factor that originally seemed to separate RSE-ops with DevOps was HPC system complexity, and inability to change quickly. However, clouds now are also increasing in complexity, not just in services offered, but machines

(and general details) relevant to each service. A second factor is availability of resources. On HPC, "on-demand" usually means waiting in a queue for your turn. This means that software development on HPC is a slow task, and typically the software is not developed alongside the system, but perhaps only used or tested there after the fact.

For working in the cloud, we operate on the basis of being able to make services and systems modular. Any piece of software can be installed when needed on a cloud resource, whereas the process is more complicated for HPC, primarily needing to be done by someone with adequate permissions, and in a way that the software can exist alongside other software and be accessed in a particular way (loaded as a module, as a container, etc.). On the cloud a piece of software can be run exactly with a container base that is needed, and there is little or no concern for dependency with other bases. The two biggest hardware dependencies that containers have, arguably host MPI and GPU, are also hugely important factors when figuring out compatibility.

There is also little concern with using updated systems or software – unlike HPC that needs to always support legacy systems, deployment on the cloud is more "use and throw away" to easily support whatever system is desired. As long as services are written in portable languages and don't require high performance with respect to node configuration, they can be run anywhere, and people are agnostic to the details.

This allows developers to work on the bleeding edge and perhaps only be concerned with supported the latest top systems that might want to run the software (e.g., Mac, Linux, and Windows latest). Cloud DevOps also allows for scaling and deploying on demand resources with the only constraint being cost. For RSE-ops on HPC, the scaling is limited to the cluster, along with the number of people using it.

A major helper to this current landscape would be well-established tools to allow for ease of testing for an HPC user. Whether this means testing on a node allocated for testing on the cluster, or a separate build system that mimics the cluster environment, being able to trigger builds and tests alongside a code base with a CI pipeline would greatly improve the software development life-cycle on HPC. If such a system existed, and if it were easy for RSE software developers to develop, test, and deploy their own software, system administrators could better focus on improving or enhancing the systems instead of managing software for them. We could even imagine going a step further and having a more automated ability to use software for the average user. For example, if a workflow system is better integrated into the cluster, akin to how an RSE developer could push and trigger builds and tests, an HPC user could push a workflow to trigger running a test job, and upon success, if desired or needed, running it at scale. It would be amazing if we could eliminate the need to ssh in to a log in node, learn how to use a job manager, and otherwise interact with HPC. To the user, if the cluster could be exposed as a service with an easy interface to monitor jobs and job submission could be integrated with the user's workflow, the speed

of development and analysis would greatly increase.

Obviously, the greatest challenges to making the above a reality come down to security and centers being able to support more modern authentication systems. Exposing any cluster resource to be accessible via continuous integration or version control triggers requires connecting the resource to external services, which always is a risk, but there are well developed authentication frameworks for that.

Moving data from some external storage to a cluster resource to be analyzed is another challenge. A tiny step of progress is the ability to link a traditional HPC single sign on (SSO) scheme (e.g., LDAP or Kerberos) to a version control service (e.g., GitLab or GitHub) to allow for role based access, but it's not clear how often this is done or if it's a good direction.

Informing one another: metadata for containers

Containers are an essential tool for RSE-ops, as they are involved with continuous integration, software deployment and reproducibility, community standards, and testing. As an exercise to find opportunities for those in the DevOps and Rse-ops spaces to work together on tools or standards, we can look more closely at container technologies that are used in both spaces, and identify several use cases where this ample room for improvement and collaboration. A particular topic of interest for this exercise it looking at container metadata. There are currently no tools, standards, or best practices for definition and use of container metadata, and there are several use cases where it is badly needed. Ideally we could define a set of metadata that can solve use cases in both, and then tools and best practices around that. As was started with the containers working group here are several use cases that can be mapped to HPC or cloud with metadata that are needed for both. This kind of exercise can be extended to talk about more areas than just container technologies.

Use Case: Architecture and Host-aware Pulls

A user pulling a container from a registry to a host, regardless if we are on HPC or a cloud resource, should be able to get the best matching one. The container already exists, so we could imagine some hook before the pull looking at metadata on the host, and then sending it with a request for a specific container. This idea is similar to one discussed in (Younge 2019). In terms of metadata we would likely need:

- Image architecture (to match to host) or best available for host per arch-spec
- Hardware (and namely software compatibility with it)
- Memory / resource needs (perhaps we can be more specific here) (an example that requires "big" nodes)

- Software-level compatibility information (e.g., target CUDA driver version, MPI variant and version)
- ABI info

For the above, it could be that there is a simple metadata matching algorithm, or something as complex as an actual solve. For this to work, the process of building images for different architectures needs to be made easier. A cluster would need to be able to create a listing of architectures present, and then issue those as requests to a build service or registry.

This use case is different from using an HPC scheduler one because the pull is happening from a node where it is intended to be run. Although the scheduler could also be given a container unique resource identifier to pull and then match it accordingly, the scheduler likely is given an unknown (already pulled) container, and needs to match it to a node.

Use Case: Container Discoverability

A user might want to quickly find a container that most closely matches what they need. This likely would be some kind of database that indexes metadata about containers, and provides it in a searchable interface. In terms of metadata this use case would likely need:

- Software and libraries
- Domain-specific labels
- ABI compatibility about the container (to assess compatibility with the host)
- Image architecture

Use Case: Dynamic Builds

A user on a resource should be able to navigate to a web interface, select a set of software or features desired for a container, and then have it built for use. This use case is slightly different because we are not searching for existing containers, but building custom ones. In terms of metadata, this use case would require:

- Operating systems
- Compatible package managers / software

Optimization

Several user groups are interested in being able to collect metrics about software and services. If containers could provide some of these metrics, it would help their use case. In terms of metadata this use case would likely need:

- System and architecture details for where it was built

- System and architecture details for where it has been successfully tested/run

For a limited scope of containers, such as those designed completely from a package manager, some of this metadata could be included automatically. However, most containers "in the wild" won't have been developed from a single package manager, and there should still be a way to discover these metrics.

Use Case: Scheduler

A scheduler that is handed an unknown container should be able to quickly sniff it and be able to best match it to a node. Akin to Kubernetes, we would want to be able to match labels of nodes with labels of containers to be run there. For this use case, we would also need:

- Image architecture
- Hardware (e.g., ABI)
- Memory / resource needs

A Harder Challenge is Changing Culture

If DevOps is a culture and community that emphasizes collaboration between "operations" and "development," would RSE-ops be the same idea but focusing on the collaboration between HPC system administrators, and research software engineers? And instead of automating services, would we want to automate scientific workflows, only with small services when they are needed? Some might argue that it isn't about the tools per-say, but the practices and culture around them. So it simply is not enough to just create a new tool or state a practice without asking how it fits into current culture. From the other side of the coin, having an open mind is also important. If any member of an organization doesn't have a mindset that is open and amenable to change and innovation, it's going to be hard to practice. This might be especially challenging in the HPC/RSE community where stability and consistency (arguably the opposite of change) is already part of the culture. A strategy to go about inspiring change might be to start with a current practice, and ask the questions:

- "How do we automate this?"
- "How do teams X and Y work together, what are the common goals and units of operation?"

A lot of RSE-ops seems to come down to automation, which is why it's a good question to start with. Another good suggestion is to consider RSE-ops from the level of the system. Each of these components can be discussed in terms of who works on it, how that is collaborative, and how it could be more collaborative or automated.

- booting
- web servers and hosting static and dynamic sites
- process management
- ssh
- file systems and volumes
- system logging, monitoring, troubleshooting
- protocols like SSL, TLS, TCP, UDP, SFTP, etc.
- managing services (e.g., initd, systemd)
- load balancing
- breaking things and troubleshooting (this might be good for practice or learning to work together)

Another interesting question is how could this stack span HPC and cloud? How can we find some convergence? If we can create a map of cloud services, we could then try to map that to HPC. We ultimately want a layout of the land for what constitutes the "infrastructure" of HPC and what is missing or could be improved. Each node in this map would then be linked to documentation in a consistent, well-branded way.

Mapping the Space

During this exercise, we can see that there are two different communities that come together to form what we might call the HPC community, and you might guess this comes down to users (researchers, research software engineers) and admins (system administrators and also research software engineers). You'll notice that research software engineers can be part of both groups, which is why the profession has been helpful to shed light on the needs of the user base. The two groups include:

- users
- admins

And the two groups can view the same ecosystem in very different ways. The two groups can also have some overlap in shared value for software or services, however not everything has perfect overlap. In many cases, even the user community is broken into smaller user communities based on domain of science, or preferred tools. For example, a physics group at a national lab might use a lot of MPI or Fortran codes, while a neuroimaging lab's bread and butter is launching SLURM jobs with a container technology. This realization requires us to make a distinction between the two groups, and clarify that RSE-ops does not include domain or scientific libraries, but all of the testing, infrastructure, and support

tools around them. We are making an effort to map out this space, which can be viewed at https://vsoch.github.io/rse-ops/. We are also clearly noting that RSE-ops extends beyond HPC (but includes it).

Summary of Opportunities

The following areas of RSE-ops are not well developed, and we have opportunity to build or seek out tools to fill the space.

- **Web Applications**: Development framework for interactive (browser based) applications for HPC. This relies on being able to ensure secure access in a browser.
- **Automated Scaling**: Being able to scale a job, task, or service without thinking about it.
- **On Demand Resources**: An HPC user should theoretically be able to request a custom resource, and "throw it away" when done. For HPC an administrative team would still need to monitor and maintain the resources, but to the user there could be a more interactive, customized experience. E.g., "I want an instance with this amount of memory, storage, and with this software installed. And I want to customize and request it in a web interface." While traditional batch systems can support this kind of request, what they aren't designed for is on-demand computing. Batch systems expect jobs that take a long time, and they aren't responsive.
- **Continuous Integration**: Application developers on HPC should have easy access to run tests on clusters, which can be difficult to do. Some national labs have workflows that integrate HPC with CI (e.g., GitLab) and other academic institutions are nowhere near this kind of integration.
- **Dependency Management**: With spack, easybuild, and containers, we are doing fairly well. However, constant training and education about these projects is still a challenge. Along with dependency management we also need rich metadata that is paired with distributed binaries and containers.
- **Community Standards**: Have one or more representatives join the OCI community, along with other standards bodies that work on ideas relevant to our communities.
- **Continuous Integration**: Can there be a CI resource or service provided that can better mock HPC environments? Can there be a standard CI service?
- **Continuous Deployment**: Why can't traditional tools to produce software be hooked up with the CI service?

- **Monitoring**: better integration of traditional workflow/job management tools with monitoring, and establishing best practices.

References

"A Brief History of High-Performance Computing (HPC) - XSEDE Home - XSEDE Wiki." n.d. https://confluence.xsede.org/pages/viewpage.action?pageId=1677620.

"A Not-so-Brief History of Research Software Engineers." n.d. https://www.software.ac.uk/blog/2016-08-17-not-so-brief-history-research-software-engineers-0.

"Administration — ECP Continuous Integration Documentation." n.d. https://ecp-ci.gitlab.io/docs/admin.html.

Atlassian. n.d. "History of DevOps." https://www.atlassian.com/devops/what-is-devops/history-of-devops.

"Autamus." n.d. https://autamus.io/.

"BUILD." n.d. https://computing.llnl.gov/projects/build.

Carlyle, Adam G, Stephen L Harrell, and Preston M Smith. 2010. "Cost-Effective HPC: The Community or the Cloud?" In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 169–76.

"Challenging the Barriers to High Performance Computing in the Cloud - HPCwire." 2020. https://www.hpcwire.com/solution_content/aws/manufacturing-engineering-aws/challenging-the-barriers-to-high-performance-computing-in-the-cloud/.

"Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC." n.d. https://dl.acm.org/doi/10.1145/3126908.3126925.

Choice, Editor's. 2020. "How DevOps Is Integral to a Cloud-Native Strategy." https://www.information-age.com/how-devops-integral-cloud-native-strategy-123488706/.

"Circle-CI." n.d. https://circleci.com.

"Cloud Native Devops." n.d. https://www.oreilly.com/library/view/cloud-native-devops/9781492040750/ch01.html.

"Containers Intro." n.d. https://www.oreilly.com/library/view/cloud-native-devops/9781492040750/ch01.html#containers-intro.

"DevOps: The Shift That Changed the World of Development." 2020. https://www.narwalinc.com/blog/devops-the-shift-that-changed-the-world-of-development/.

"DevOps." n.d.b. https://cloud.google.com/devops.

———. n.d.c. https://azure.microsoft.com/en-us/services/devops/.

———. n.d.a. https://dl.acm.org/doi/10.1109/MS.2016.68.

"Docker Hub." n.d. https://hub.docker.com/.

"Docker." n.d. https://docker.io.

"EasyBuild." n.d. https://easybuild.io/.

"Environment Modules." n.d. https://modules.readthedocs.io/en/latest/.

"Flux Framework." n.d. https://flux-framework.org/.

Foote, Keith D. 2017. “A Brief History of Cloud Computing - DATAVERSITY.” https://www.dataversity.net/brief-history-cloud-computing/.

“GitHub.” n.d. https://github.com.

“GitLab.” n.d. https://gitlab.com.

“Google - Site Reliability Engineering.” n.d. https://sre.google/sre-book/introduction/.

“Google HPC.” n.d. https://cloud.google.com/solutions/hpc.

Guidi, Giulia, Marquita Ellis, Aydin Buluc, Katherine Yelick, and David Culler. 2020. “10 Years Later: Cloud Computing Is Closing the Performance Gap,” November. https://arxiv.org/abs/2011.00656.

“Inside HPC Future Technologies.” n.d. https://insidehpc.com/2017/03/future-technologies-rise-hpc/.

“Jenkins.” n.d. https://jenkins.io.

“Jupyter Hub.” n.d. https://jupyter.org/hub.

“Large-Scale Cluster Management at Google with Borg.” n.d. https://research.google/pubs/pub43438/.

Li, Shijian, Robert J Walls, and Tian Guo. 2020. “Characterizing and Modeling Distributed Training with Transient Cloud GPU Servers.” In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 943–53.

“LMOD.” n.d. https://lmod.readthedocs.io/en/latest/.

“Magellan: A Cloud Computing Testbed.” n.d. https://www.nersc.gov/research-and-development/archive/magellan/.

“Measuring High-Performance Computing with Real Applications.” n.d. https://engineering.purdue.edu/paramnt/publications/SBA+08.pdf.

Mendoza, Thomas. n.d. “LC User Meeting.” https://hpc.llnl.gov/sites/default/files/GitLab-Status-LCuserMtg-202012-Mendoza.pdf.

Morgan, Timothy Prickett, and Nicole Hemsoth. 2021. “The Many Other High Costs Cloud Users Pay.” https://www.nextplatform.com/2021/06/25/the-many-other-high-costs-cloud-users-pay/.

“MPI: A Message Passing Interface.” n.d. https://ieeexplore.ieee.org/document/1263546.

nishanil. n.d. “Containers as the Foundation for DevOps Collaboration.” https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-application-lifecycle/containers-foundation-for-devops-collaboration.

“OCI Distribution Spec.” n.d. https://github.com/opencontainers/distribution-spec.

“OnDemand.” n.d. https://www.osc.edu/resources/online_portals/ondemand.

“OpenContainers.” n.d. https://opencontainers.org.

“Opencontainers/Runtime-Spec Hooks.” n.d. Github.

“Podman - : A Tool for Managing OCI Containers and Pods.” 2018.

Power, Brad, and Joe Weinman. 2018. “Revenue Growth Is the Primary Benefit of the Cloud.” *IEEE Cloud Computing* 5 (4): 89–94.

Prabhakaran, Akhila, and Lakshmi J. 2018. “Cost-Benefit Analysis of Public Clouds for Offloading In-House HPC Jobs.” In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 57–64.

Priedhorsky, Reid, R Shane Canon, Timothy Randles, and Andrew J Younge. 2021. "Minimizing Privilege for Building HPC Containers," April. https://arxiv.org/abs/2104.07508.

Reese, George. n.d. *Cloud Application Architectures*. O'Reilly Media, Inc.

"Replication Crisis." n.d. https://en.wikipedia.org/wiki/Replication_crisis.

Scality. 2020. "The History of Cloud Computing - SOLVED." https://www.scality.com/solved/the-history-of-cloud-computing/.

"Shifter." n.d. https://github.com/NERSC/shifter.

"Singularity Registry HPC." n.d. https://singularity-hpc.readthedocs.io.

"Singularity Registry." n.d. https://joss.theoj.org/papers/10.21105/joss.00426.

"Singularity: Scientific Containers for Mobility of Compute." n.d. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0177459.

"SLURM: Simple Linux Utility for Resource Management." n.d. https://link.springer.com/chapter/10.1007/10968987_3.

"Spack User Survey." n.d. https://spack.io/spack-user-survey-2020/.

"Spack." n.d. https://spack.io/.

"Ten Simple Rules for Writing Dockerfiles for Reproducible Data Science." n.d. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008316.

"The Mythical Man Month." n.d.

Torrez, Alfred, Timothy Randles, and Reid Priedhorsky. 2019. "HPC Container Runtimes Have Minimal or No Performance Impact." In *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 37–42.

"Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus." n.d. https://ieeexplore.ieee.org/document/9060302.

"Travis-CI." n.d. https://travis-ci.org.

"Web Portals for High-Performance Computing: A Survey." n.d. https://dl.acm.org/doi/pdf/10.1145/3197385.

Webteam, Puppet. n.d. "2016 State of DevOps Report." https://puppet.com/resources/report/2016-state-devops-report/.

"Welcome to the ECP CI Documentation — ECP Continuous Integration Documentation." n.d. https://ecp-ci.gitlab.io/index.html.

"What Is Devops." n.d. https://aws.amazon.com/devops/what-is-devops/.

"What Is DevSecOps?" n.d. https://www.redhat.com/en/topics/devops/what-is-devsecops.

"What Is Hybrid Cloud? - Benefits and Advantages of a Hybrid Cloud." n.d. https://www.netapp.com/hybrid-cloud/what-is-hybrid-cloud/.

Whitehead, Jim, Ivan Mistrík, John Grundy, and André van der Hoek. 2010. "Collaborative Software Engineering: Concepts and Techniques." In *Collaborative Software Engineering*, edited by Ivan Mistrík, John Grundy, André Hoek, and Jim Whitehead, 1–30. Berlin, Heidelberg: Springer Berlin Heidelberg.

Wikipedia contributors. 2021a. "The Mythical Man-Month." https://en.wikipedia.org/w/index.php?title=The_Mythical_Man-Month&oldid=1025350

772.

———. 2021b. “Cloud Native Computing Foundation.” https://en.wikipedia.org/w/index.php?title=Cloud_Native_Computing_Foundation&oldid=1034940082.

———. 2021c. “History of Supercomputing.” https://en.wikipedia.org/w/index.php?title=History_of_supercomputing&oldid=1034423047.

Younge, Andrew J. 2019. “A Case for Portability and Reproducibility of HPC Containers.” SAND2019-14107C. Sandia National Lab. (SNL-NM), Albuquerque, NM (United States).