Self-Scaling Clusters and Reproducible Containers to Enable Scientific Computing

Peter Z. Vaillancourt*, J. Eric Coulter†, Richard Knepper‡, Brandon Barker§
*Center for Advanced Computing
Cornell University, Ithaca, New York, USA
Email: pzv2@cornell.edu
†Cyberinfrastructure Integration Research Center
Indiana University, Bloomington, IN, USA
Email: jecoulte@iu.edu
‡Center for Advanced Computing
Cornell University, Ithaca, New York, USA
Email: rich.knepper@cornell.edu
§Center for Advanced Computing
Cornell University, Ithaca, New York, USA
Email: brandon.barker@cornell.edu

***Abstract*—Container technologies such as Docker have become a crucial component of many software industry practices especially those pertaining to reproducibility and portability. The containerization philosophy has influenced the scientific computing community, which has begun to adopt – and even develop – container technologies (such as Singularity). Leveraging containers for scientific software often poses challenges distinct from those encountered in industry, and requires different methodologies. This is especially true for HPC. With an increasing number of options for HPC in the cloud (including NSF-funded cloud projects), there is strong motivation to seek solutions that provide flexibility to develop and deploy scientific software on a variety of computational infrastructures in a portable and reproducible way. The Cyberinfrastructure Resource Integration team in the XSEDE project has developed a simple tool which provides HPC infrastructure in the cloud that scales with user demand. We now present one possible solution which uses the Nix package manager in an MPI-capable Docker container that is converted to Singularity. It provides consistent installations, dependencies, and environments in each image that are reproducible and portable across scientific computing infrastructures. We demonstrate the utility of these containers with cluster benchmark runs in a self-scaling virtual cluster using the Slurm scheduler deployed in the Jetstream and Aristotle Red Cloud OpenStack clouds. We conclude that this technique is useful as a template for scientific software application containers to be used in the XSEDE compute environment, other Singularity HPC environments, and cloud computing environments.**

***Index Terms*—Containers, Docker, Singularity, Slurm, Cluster, Cloud, Scientific Computing, HPC, MPI, Cyberinfrastructure**

I. Introduction

The Cyberinfrastructure Resource Integration (CRI) group of Extreme Science and Engineering Development (XSEDE) [1] provides software and services to bring best practices from the National Supercomputing Centers to university campuses interested in implementing their own computational infrastructure. By providing software and documentation for campus adopters, CRI has been able to assist in the creation of cyberinfrastructure at a number of campuses, enabling them to provide computational resources to their faculty members [2]. As more organizations utilize cloud resources in order to provide flexible infrastructure for research purposes, CRI has begun providing tools to harness cloud infrastructure, working together with the members of the Aristotle Cloud Federation Project [3]. Red Cloud, the Cornell component of the Aristotle Cloud Federation, provides an OpenStack cloud services much like the Jetstream [4] resource available through XSEDE, that is available both to Cornell faculty as well as to Aristotle members.

CRI focuses its activities on knitting together cyberinfrastructure resources at multiple institutions. While the XSEDE Federation presents the resources of sundry service providers at different levels of interoperability [5], there are many campus cyberinfrastructure installations that are focused on meeting local needs. Expertise in cluster administration is a scarce resource, and not all institutions that elect to provide a centralized computing cluster are immediately able to leverage technical and policy knowledge to implement and maintain it. The CRI team assists these campuses in set-up, maintenance, and policy generation. If the institution has interest, CRI assists with integration into the larger XSEDE framework. CRI has worked with 6 different institutions in 2019 and 2020 to provide support for cyberinfrastructure installations, and has directly assisted in bringing over 1 PetaFLOP of computing capacity online, in addition to advising on other cyberinfrastructure activities.

Previously, CRI introduced the virtual cluster toolkit [6], [7], which allows for the creation of HPC-style infrastructure on cloud resources. It was specifically created for the Jetstream [4] research cloud, but is adaptable to other cloud resources, as seen in our deployment to Red Cloud in this paper.

CRI has also taken efforts to expand software offerings which take advantage of the benefits of cloud computing

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

models: reproducible application containers and self-scaling clusters built on cloud infrastructure. Reproducible containers take advantage of container runtime environments which allow applications to run in lightweight virtual spaces that make use of the underlying operating system, while providing reliable installation of a set of software and dependencies at a given version. Self-scaling clusters rely on common cluster system images and the Slurm [8] scheduling tool in order to create and destroy additional virtual machines as jobs are placed into the Slurm queue. Reproducible containers provide reliable, portable applications with the same constituent software at every instantiation, while self-scaling clusters provide efficient use of computational infrastructure without incurring more usage than the applications require.

II. Compute Environment Choices

A. XSEDE Compute Environment

Researchers at U.S. universities and federally-funded research and development centers (FFRDC's) consistently report a lack of sufficient computational capability available to them [9], [10]. The XSEDE project provides a socio-technical system which facilitates access to resources provided through the XSEDE federation, but despite the considerable resources available, not all requests can be accommodated. XSEDE allocation proposals outstrip allocable resources by a factor of three. By passing on best practices and interoperability to other institutions, the XSEDE project can foster the extension of computational capacity beyond what is available through the project alone, and in addition, can make it easier for researchers to move between systems. By providing common filesystem layouts, software, and scheduling systems, adopters of XSEDE software capabilities can make it easier for researchers to work on XSEDE systems as well as other systems which have adopted similar affordances, making it easier to move between resources and facilitating scaling up or down in compute capability as necessary.

However, not all XSEDE resources and not all campus resources can be exactly the same. Local context dictates particular choices which affect individual system implementations as well as which software is made readily available. Analytical workflows cannot be instantaneously replicated on different systems, and barriers to computational transparency between systems will always be a factor. For XSEDE systems, not only are there dedicated, system-specific allocations which determine access to resources through the XSEDE access system, but system differences and separation of file systems mean that researchers must make changes to their workflows in order to move to different systems within the XSEDE framework. Researchers making use of the national infrastructure at a variety of sites, facing switching costs to move between systems, require a common set of tools which facilitate their use of computational resources.

1) XCBC: In an effort to help solve the problems facing campus resource providers in giving their researchers access to quality High Performance Computing (HPC) environments, the XSEDE CRI team developed a toolkit called the XSEDE-Compatible Basic Cluster (XCBC) to easily build a local HPC system conforming to the basic environment available on most XSEDE resources, modulo differences in specialized hardware or site-specific software. This project evolved [11] from being based on the Rocks [12] cluster management software to the OpenHPC [13] project with Ansible [14]. The CRI team has performed a dozen in-person site visits to help implement HPC systems using this toolkit, in addition to dozens more remote consultations. Discussions with sites implementing the toolkit [15]–[17] have shown that the local XSEDE-like environment does in fact help researchers onboard to XSEDE more easily, in addition to providing an environment with low wait times compared to the full queues on larger systems, where researchers may wait for days before a particular job runs.

2) Singularity in HPC: The problem remains, however, that individual researchers may not have the software they need available on either a national or a local campus system, and getting new custom software installed is non-trivial for the administrators of those systems. In many cases, this can be addressed through the use of containers, which have sparked a revolution in horizontal scaling of software in the enterprise world. In addition, containers provide the opportunity to utilize differing versions of software than those installed, or to package private or custom software for development purposes.

There are significant challenges to using containers effectively on HPC systems, however, due to security concerns, complexity of installations for scientific software, and lack of widespread adoption as a distribution method for scientific software. The popularity and prevelance of Docker [18] outside of the HPC community notwithstanding, security concerns around the fact that a user running docker has root access to a system prevent its use on HPC systems without modifications to mitigate risk. Newer container runtime environments have been developed with an eye towards solving these security concerns and even designed specifically for HPC systems, such as CharlieCloud [19] and Singularity [20], [21].

Singularity has become the de-facto preference for containers in many HPC environments, including the XSEDE compute environment. Singularity is available on all XSEDE allocated resources, and included in the XCBC toolkit as part of OpenHPC, thus becoming available on many campus systems. Other HPC systems using versions of OpenHPC, even outside of the XCBC toolkit, can also trivially provide access to Singularity to their users. This prevalence within HPC makes Singularity an attractive option for the distribution of scientific software and dependencies, which is further helped by Singularity support for the conversion of Docker container images to Singularity [22]. So long as best practices for conversion to Singularity are followed (or checked in the case of pre-existing images), the same Docker container that was developed for deployment to a cloud environment can be converted to Singularity for effective use on an HPC resource. An essential component of both Docker and Singularity containers is that the common means for instantiation relies on a pull

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

method, which generally grabs the most recent version of the container image. This results in the situation that execution of the "same" container at different times could use different versions of the included software, potentially affecting the results of computation. This complicates the replication and reproducibility of calculations later on, and necessitates a replication-friendly container architecture.

B. The Need for Cloud-Integrated Computational Software

In many cases, researchers do not have access to local HPC hardware on which to test and profile their software, before moving to large-scale XSEDE environments such as Comet or Stampede2. In order to provide an HPC testbed environment to such researchers, the XCRI team has developed a self-scaling virtual cluster [23] for use on Openstack clouds, such as Jetstream [4]. This toolkit provides researchers with a basic HPC system, based on the OpenHPC [13] project using the same software as those powering the systems available within XSEDE. This allows for rapid prototyping, scaling tests, and scientific software development within an XSEDE-like environment, without the associated "cost" encountered on large systems of long wait times in filled queues, environmental roadblocks, having to ask overwhelmed sysadmins to help with your software install, and navigating policies that favor large highly-tuned jobs over small productivity-oriented jobs.

Nonetheless, over the course of providing virtual HPC environments for dozens of projects [23], it has become quite clear that the administrative overhead of maintaining scientific software does not scale well within a small team, *nor* does it effectively empower users to easily transfer software and workflows into XSEDE without assistance. While the premise of familiarizing users with an HPC environment holds in situations where researchers are new to HPC, things fall apart in the case of established research teams who want to take customized software from a cloud environment with root access into a heavily managed multi-user system. Thus, we have begun transitioning from custom software installations on a per-cluster basis to helping users containerize their software for use in multiple XSEDE environments. By providing container templates based on best practices for the container runtimes available on national resources, we are able to further erode the barriers to computing that prevent researchers from making maximally effective use of their time and allocations. To further reduce barriers to scientific computing, we provide Docker container templates that can run in a variety of cloud environments – while still maintaining best practices for conversion to Singularity – to enable consistency of software and environment across different cyberinfrastructures. Thus, researchers can employ these container templates for greater access to computing through whatever means (i.e. cloud or HPC resources) are available to them.

III. Reproducible Application Containers

When developing containers for scientific software applications, it quickly becomes aparent that subtle version changes, differing compilation or configuration steps, or any number of other nuances of the build and installation process can cause unexpected behavior or even failure to build. Though version pinning is the best practice for software installation inside a container in order to achieve consistency, it is not common practice. Additionally, scientific software in some domains – especially legacy codes used in tandem with newer packages – often has an extensive list of dependencies, making version pinning a tedious task for researchers managing their own development code, environment, and dependencies. Hence, even many currently available container specifications fail to build when attempted a short time period after they were written. In spite of this discrepancy, if the container images are publicly available, they can continue to be used for much longer than the build specification so long as no changes are needed in version or other configurations. On the other hand, scientific software development often involves an iterative process, where compilation steps and configuration choices change over time. If other users or even the original user return to the container build specification to make changes, they can be thrust back into the role of a system administrator tracking down the source of discrepancies in software installation and setup as compared to previous instantiations of the container image. Moreover, the conversion of a container between the Docker and Singularity container runtime environments does not guarantee consistency, or reproducibility, of software installations or configurations even when best practices are followed.

Reproducibility and associated terms like replicability and reliability represent considerable issues for researchers in demonstrating that their research is supported by data that is correctly analyzed, with results that are descriptive of the subject of inquiry, with consistent support for the inferences that those results yield. These components have been described as "methods reproducibility", "results reproducibility", and "inferential reproducibility" [24]. Funding agencies such as the NSF and NIH state that the ability to reproduce research is the basis of research findings that are "believable and informative" [25], [26]. With researchers who want to provide a means for reproducing their analyses facing a considerable number of variables, the current state of scientific software in containers represents an issue in "methods reproducibility", which we believe can be addressed by providing a framework that delivers software components of the same version from the original analysis at every build and instantiation time, providing transparency into the components used, and facilitating the simple reproduction of those software components by later scientists even after many years. In order to support this framework, we leverage the Nix package manager and expression language.

A. NIX

Nix is a package manager and associated language (Nix Expression Language [27]) for specifying package builds and dependencies [28], [29]. It has two major sister projects, the nixpkgs package collection that contains many thousands of package definitions in a single GitHub repository [30], and

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

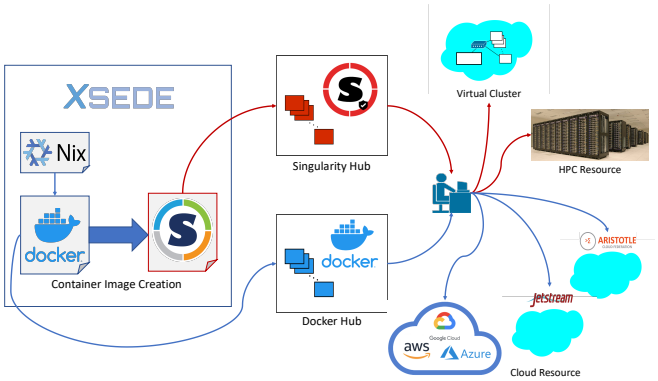

Fig. 1. A depiction of the process from container template development by the XSEDE XCRI team through deployment of Singularity or Docker by the user in various environments.

the Linux distribution NixOS [31], which uses the former projects on top of the Linux kernel and systemd to realize the vision of an entire Operarting System (OS) environment that is easily reproducible. This reproducibilty is achieved as part of the Nix-language architecture and by pinning the particular version (or versions) of nixpkgs being used. The Aristotle Science Team has taken advantage of this feature of Nix within Docker containers for cloud deployments of scientific applications [32]–[35], building upon previous work which demonstrated it's use in scientific computing and HPC [36], [37], and here we apply the usage and extend it to Singularity containers.

Although most are familiar with the idea of pinning versions in various settings to obtain reproducibility, the Nix language takes this a step further. Package specifications – called Nix *derivations* – are just a kind of Nix function written in the Nix Expression Language. The function takes inputs such as other prerequisite packages (i.e. compilers), build configurations for the same package, possibly that package's dependencies, and any source or binary files needed. It is important to note that this sort of specification is always more precise than just specifying a package's version. The entire installation process can be customized within a Nix expression, if desired, included everything from simple directory and file management to complicated or intricate steps for compilation or configuration. Even beyond this, the Nix language is a pure functional language, meaning that one will always get the same output for a given input. The inputs are hashed, whether they other nix expressions, binary or source blobs downloaded over the internet, or configuration files that are read in by the Nix derivation. If a hash changes in the build, a new package is created with the new hash recorded. For instance, by changing a build setting, one would wind up with two installations of the same package indexed by the hash of the derivation output. Applications requiring the package as a dependency will automatically get the correct package loaded in the environment depending on what configuration settings they specify for the package in their derivations.

While using Nix in containers is not the most widepsread use of the Nix package manager, it can be used in Windows Subsystem for Linux (WSL), Mac OS X, and practically any Linux distribution, including in Linux containers. By using a container that includes the Nix package manager, we can create distribution-independent and largely environment-independent solutions that are portable across all environments supporting Nix (whether inside a container or not). One caveat to this that we have encountered is with MPI applications, where the implementation and version of MPI within the container must match that of MPI on the host (within a range of major release version). This is because for distributed MPI jobs, there are components of MPI that must run at a privileged level on the host, e.g. when dealing with network fabric. To migitate this challenge, our container templates will be flexible with respect to MPI versions and implementations where possible.

B. Container Template Library

As described above, the CRI team has begun to curate a Container Template Library (CTL) to share with the wider scientific computing community. The goal of the project is to create easy-to-use, and reproducible templates for containerizing scientific software using Docker or Singularity with Nix. A "template" consists of an open source GitHub repository with the Dockerfile (build specification for Docker) and any associated scripts (such as Nix expressions) used to build the container, the Docker image hosted on Docker Hub [38], an open source GitHub repository with the Singularity definition file (build specification for Singularity) and any example Slurm scripts, and a Singularity image hosted on SingularityHub [39]. The benefit of providing *templates* for containers, as opposed to just the container images, lies in empowering the user to be able to modify the build to suit their needs, knowing that it will be reproducible and result in a consistent environment.

Fig. 1 demonstrates the entire process for a single container template. The CRI team begins by writing any needed Nix expressions (where standard Nix packages cannot be used) and providing a Dockerfile that properly builds and installs the environment, packages, dependencies, etc. These are used to produce a public Docker image that is hosted on Docker Hub, which the user can then pull to deploy on any cloud resource they have access to. Alternatively, the user can return to the Dockerfile or Nix expression to make modifications or add their own software, and then build their own image. The CRI team also creates a Singularity definition file which exercises the built-in Singularity methods to convert a Docker image to a Singularity image, and adds tests and scripts to run the container. The definition file is then added to the SingularityHub collection and built to create a public image which the user can pull to either a virtual cluster in the cloud or any HPC resource they have access to. Current available containers and images include:

- Docker base container with Nix [40], [41]
- Singularity base container with Nix [42], [43]
- Docker container with Nix and OpenMPI [44], [45]
- Singularity container with Nix and OpenMPI [46], [47]

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

In this paper, we have chosen the Singularity container with Nix and OpenMPI to deploy and run on a virtual cluster.

IV. Virtual Clusters

As discussed in Section II-B, the XSEDE-like Virtual Cluster (VC) has been developed to provide researchers with an environment for testing and scaling scientific workflows. The basic environment provides the following features, which are often the hardest for a researcher to navigate upon gaining access to a new system:

- **Slurm** - The scheduler and resource manager used to split the work of multiple users across multiple nodes
- **Lmod** - The environment module system in use across XSEDE resources, allowing for easy setting of environment variables related to specific software packages [48]
- **Singularity** - The container runtime environment available across XSEDE HPC resources, designed specifically to ease use of MPI codes and integrate with Slurm.
- **Shared storage** - every HPC system has different requirements for shared storage and scratch space; the VC uses a simple system of three shared spaces (user home directories, a working directory space, and a shared space for installed software).

These virtual HPC environments have also proven useful in educating users and potential sysadmins about what it takes to run an HPC system. They have been used in a series of tutorials [6], [49], including in-person workshops at George Mason University and Clark Atlanta University.

A. Virtual Cluster Architecture

The VC infrastructure is primarily centered on the headnode, which also provides users login access (and an optional graphical environment). In a sense, the VC consists solely of the headnode - all work is done on ephemeral instances, created when a user submits a job to Slurm, and destroyed when there is no more work in the queue. As we show in Fig. 2, from a user perspective, it is possible to deploy HPC-style infrastructure across clouds, and run jobs using reproducible containerized software. The VC manages compute (worker) instances without user intervention, allowing for efficient usage of resources. The container images are easily made available to other researchers via container registries, allowing for easier collaboration and greater transparency of published work.

V. Deployment Process

The current VC can be deployed via two commands after cloning the GitHub repository [7] hosting the code. The first creates the headnode which will: spawn worker nodes via Slurm, contain the user environment and software (container images), and provide the filesystem shared to worker nodes. The second command installs all of the required software on that node.

For production useage, edits must be made to two files to reflect the size of jobs expected on the VC, to increase the size and number of worker nodes, or create larger shared

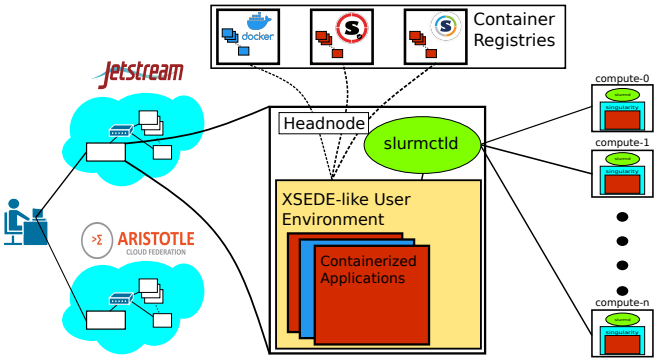

Fig. 2. Overview of the VC architecture from the user perspective.

storage volumes. Otherwise, the addition of scientific software can be done simply by pulling down a Singularity image, and jobs can be run without further intervention. A more detailed discussion of the configuration of any individual VC is provided in [23]. For clarity in the following discussion in Section V-A, it is worth mentioning that the software on the cluster is installed from repositories provided by the OpenHPC [13] project, including Singularity version 3.4.1 and OpenMPI version 3.1.0 at the time of writing.

1) Mutation for Different Clouds: In a similar vein to the issues discussed in Section II-A, there still remain barriers to implementing the same infrastructure on different research cloud providers *even* in the case where the underlying software is the same. For the purposes of this paper, we created example clusters on Jetstream and Red Cloud, to demonstrate reproducibility of the entire stack and workflow on different cloud environments, which still requires modification of the scripts used to create the VC infrastructure. This problem is also commonly encountered on public clouds as well, given the rapid pace of changes to APIs and charging mechanisms. The differences in cloud providers using OpenStack primarily stem from the vast array of underlying services that make up a cloud system, which naturally leads to different network configurations, instance flavors, etc. across providers.

Specifically, in order to re-create the VC architecture on Red Cloud rather than Jetstream, there were three changes to be made:

1) **Change base image name for instances**: Red Cloud uses a simpler image naming scheme than the more detailed convention used on Jetstream, where there is a larger user base and set of images.
2) **Change instance flavor names**: Red Cloud uses AWS-style names (c1.m8, c2.m16, etc.) for instance "flavors" as opposed to Jetstream's OpenStack-default convention (m1.tiny, m1.quad, etc.).
3) **Update network creation scripts**: Red Cloud requires explicitly setting the internal DHCP servers when creating a private network, where Jetstream sets this by default, which can lead to confusion when changing the network configuration of extant instances.

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

System	1 Node GFLOPS	Rmax	4 Node GFLOPS	Rmax
Jetstream	146	160	500	640
Red Cloud	105	140	412	540

TABLE I
RESULTS FROM BEST OF THREE RUNS OF HPL IN SINGULARITY CONTAINERS ON TWO OPENSTACK CLOUDS.

For the user deploying a VC in Red Cloud, these changes amount to small edits in four files used to control the VC configuration, which will be simplified in future releases.

A. MPI Application Run

Purely as an example application, we have chosen the ever-popular HPL benchmark [50], the measure of choice for the Top500 list, in which HPC systems are ranked by Floating-point Operations Per Second (FLOPs). This software choice is not intended to be used as a thorough analysis of the performance of the cluster, but rather as a proof-of-concept example to demonstrate that MPI applications can run on the cluster. The HPL software is included in the Singularity container with Nix and OpenMPI discussed in Section III-B. Tuning the HPL benchmark for maximum performance is a highly involved process for large systems [51]. In light of this, we do not present these results as true measures of the performance of the VC, but as simple representation of the fact that our container builds allow us to run MPI codes across multiple systems with very little effort. The main effort in running these jobs, in fact, was in creation of the input file for the HPL run, using the tool provided by Advanced Clustering [52], which nicely codifies the advice provided by the creators of HPL [53]. It is worth noting again that the version of OpenMPI provided by nixpkgs is 4.0.1, which "just worked" in conjunction with the host OpenMPI at version 3.1.0. These jobs were run using the "hybrid model" [54] of MPI execution in which the host version of MPI is used to execute the singularity command. For the full Slurm script, see [46], but the main command looks like the following:

```
mpirun -np $NUMPROCS singularity \
       exec hpl.sif xhpl ./HPL.dat
```

This does require that the MPI version internal to the container is compatible with that of the host, but as demonstrated here, that covers a wide range of versions, leading to maximum flexibility. Of course, we have also designed our Dockerfiles to be flexible with respect to the internal version and implementation of MPI used.

As shown in Table I, the results generally indicate performance in the range of $73 - 78\%$ of the theoretical maximum performance of the system, which in the authors' experience running naive builds of HPL on bare metal is reasonable. It is fairly likely that with concerted effort, near bare-metal performance could be extracted, based on other work comparing performance losses due to both containerization and virtualization [55]–[59]. For example, during the initial testing of the Jetstream platform, virtualization induced only a 3% hit in HPL performance [4]. This is not to say that it is without dangers, as virtualization can have differing impacts depending on the actual workload [60]. Since we are not carefully benchmarking the performance, we simply interpret these results as a functional deployment for MPI applications.

VI. PRACTICAL OUTCOMES

Practically, this work demonstrates a method for creation of infrastructure-independent (within the bounds of container runtime environment and MPI implementations and versions), reproducible containers for scientific software, with or without the requirements of running across multiple nodes. While we focus predominantly on software requiring MPI, this is only to illustrate the use of the Nix plus Docker/Singularity stack for the more difficult case where multiple nodes are required. Scientific software that would not previously be, can now become portable and reproducible with the flexibility develop and deploy on a personal computer, cloud computing environment, or an HPC resource. Furthermore, the user can enjoy consistency of environment across deployments, and even customize the environment to serve their research projects. We have additionally demonstrated a simple, flexible HPC-style infrastructure which is deployable across multiple cloud providers at nearly the push of a button, which can provide an invaluable testing or bursting environment for researchers lacking in local resources. Additionally, this infrastructure is already battle-tested behind the scenes of numerous Science Gateways [23], [61]–[63].

VII. CONCLUSION

In terms of future work, a few clear points for improvement are methods to allow expansion of the VC infrastructure to public clouds, possibly through the use of Terraform [64]. While some providers offer services that putatively offer HPC-style computation, the cost in terms of knowledge-gathering is often quite high, and is not even by default elastic, exposing the user to the potential for massive costs if worker nodes are not fully deleted subsequent to jobs finishing. In the same vein, a few small tweaks to the current VC configuration methods should allow users to specify the size of the desired cluster either through editing a single file, or providing a flag at build-time.

For the CTL, the next step for this team is to provide a larger set of stable templates for reproducible containers, based on Nix and Docker and converted to Singularity. We plan to also include in-depth documentation for the customization of the environment, adding scientific applications to the containers, and more example Slurm scripts. The ultimate goal of the project is a library containing a variety of scientific applications that have already been built within this framework, which the community can employ to enable the rapid deployment of their codes. This library of containers will be tested across the nationally available cyberinfrastructure for usability, and will also be leveraged by Science Gateways projects, in particular those powered by Apache Airavata, in order to expand the computational resources available through easy-to-use web interfaces, much as the VC has done for expanding the use of the Jetstream resource [23].

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

REFERENCES

[1] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson *et al.*, "Xsede: accelerating scientific discovery," *Computing in science & engineering*, vol. 16, no. 5, pp. 62–74, 2014.

[2] E. Coulter, J. Fischer, B. Hallock, R. Knepper, and C. Stewart, "Implementation of Simple XSEDE-Like Clusters: Science Enabled and Lessons Learned," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, ser. XSEDE16. New York, NY, USA: ACM, 2016, pp. 10:1–10:8. [Online]. Available: http://doi.acm.org/10.1145/2949550.2949570

[3] R. Knepper, S. Mehringer, A. Brazier, B. Barker, and R. Reynolds, "Red Cloud and Aristotle: campus clouds and federations," in *Proceedings of the Humans in the Loop: Enabling and Facilitating Research on Cloud Computing*, 2019, pp. 1–6.

[4] C. A. Stewart, T. M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, S. Tuecke, G. Turner, M. Vaughn, and N. I. Gaffney, "Jetstream: A Self-provisioned, Scalable Science and Engineering Cloud Environment," in *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, ser. XSEDE '15. New York, NY, USA: ACM, 2015, pp. 29:1–29:8. [Online]. Available: http://doi.acm.org/10.1145/2792745.2792774

[5] X. Project, "Service providers - xsede," 2020. [Online]. Available: https://www.xsede.org/ecosystem/service-providers

[6] E. Coulter, J. Fischer, R. Knepper, and R. Reynolds, "Programmable Cyberinfrastructure: Introduction to Building Clusters in the Cloud," 2017.

[7] J. E. Coulter. (2020) Xcri jetstream cluster. XSEDE. [Online]. Available: https://github.com/XSEDE/CRI_Jetstream_Cluster

[8] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[9] C. A. Stewart, D. S. Katz, D. L. Hart, D. Lantrip, D. S. McCaulay, and R. L. Moore, "Technical Report: Survey of cyberinfrastructure needs and interests of NSF-funded principal investigators," XROADS, Tech. Rep., Jan 2011. [Online]. Available: https://scholarworks.iu.edu/dspace/handle/2022/9917

[10] (2012) NSF Advisory Committee for Cyberinfrastructure Task Force on Campus Bridging. Final Report, 2011. [Online]. Available: https://www.nsf.gov/cise/oac/taskforces/TaskForceReport_CampusBridging.pdf

[11] J. Fischer, R. Knepper, M. Standish, C. A. Stewart, R. Alvord, D. Lifka, B. Hallock, and V. Hazlewood, "Methods For Creating XSEDE Compatible Clusters," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, ser. XSEDE '14. New York, NY, USA: ACM, 2014, pp. 74:1–74:5. [Online]. Available: http://doi.acm.org/10.1145/2616498.2616578

[12] "Rocks Cluster Software," http://www.rocksclusters.org/, 2016, [Online; accessed 26-April-2016; University of California San Diego].

[13] K. W. Schulz, C. R. Baird, D. Brayford, Y. Georgiou, G. M. Kurtzer, D. Simmel, T. Sterling, N. Sundararajan, and E. Van Hensbergen, "Cluster Computing with OpenHPC," http://hdl.handle.net/2022/21082, 2016.

[14] R. H. Inc. (2020) Ansible configuration management software. Red Hat, Inc. [Online]. Available: https://www.ansible.com/

[15] C. Langin, "From BigDog to BigDawg: Transitioning an HPC Cluster for Sustainability," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3332188

[16] J. Wells and J. E. Coulter, "Research Computing at a Business University," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3333161

[17] J. E. Coulter, R. Knepper, and R. Reynolds, "Balancing local and national cyberinfrastructure," New York, NY, USA, 2019. [Online]. Available: https://pearc19.conference-program.com/presentation/?id=pan110&sess=sess137

[18] Docker Inc. (2020) Docker. Docker Inc. [Online]. Available: https://www.docker.com

[19] R. Priedhorsky and T. Randles, "Charliecloud: Unprivileged containers for user-defined software stacks in hpc," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126925

[20] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, vol. 12, no. 5, pp. 1–20, 05 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0177459

[21] Sylabs Inc. (2019) Singularity. Sylabs Inc. [Online]. Available: https://sylabs.io

[22] S. Inc. (2019) Singularity Support for Docker. Sylabs Inc. [Online]. Available: https://sylabs.io/guides/3.4/user-guide/singularity_and_docker.html

[23] J. E. Coulter, E. Abeysinghe, S. Pamidighantam, and M. Pierce, "Virtual Clusters in the Jetstream Cloud: A Story of Elasticized HPC," in *Proceedings of the Humans in the Loop: Enabling and Facilitating Research on Cloud Computing*, ser. HARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3355738.3355752

[24] S. N. Goodman, D. Fanelli, and J. P. Ioannidis, "What does research reproducibility mean?" *Science translational medicine*, vol. 8, no. 341, pp. 341ps12–341ps12, 2016.

[25] K. Bollen, J. T. Cacioppo, R. Kaplan, J. Krosnick, and J. Olds, "Social, behavioral, and economic sciences perspectives on robust and reliable science," 2015.

[26] F. S. Collins and L. A. Tabak, "Policy: Nih plans to enhance reproducibility," *Nature*, vol. 505, no. 7485, pp. 612–613, 2014.

[27] N. Foundation. (2020) Nix Expression Language. NixOS Foundation. [Online]. Available: https://nixos.wiki/wiki/Nix_Expression_Language

[28] E. Dolstra, "The purely functional software deployment model," Ph.D. dissertation, Utrecht University, 2006.

[29] N. Foundation. (2020) Nix Package Manager. NixOS Foundation. [Online]. Available: https://nixos.org/nix/

[30] NixOS. (2020) Nixpkgs. [Online]. Available: https://github.com/NixOS/nixpkgs

[31] N. Foundation. (2020) NixOS Manual. NixOS Foundation. [Online]. Available: https://nixos.org/nixos/manual/

[32] P. Vaillancourt, B. Wineholt, B. Barker, P. Deliyannis, J. Zheng, A. Suresh, A. Brazier, R. Knepper, and R. Wolski, "Reproducible and Portable Workflows for Scientific Computing and HPC in the Cloud," 2020.

[33] Brandon Barker and Peter Vaillancourt. (2019) NixTemplates. Aristotle Cloud Federation. [Online]. Available: https://github.com/federatedcloud/NixTemplates

[34] Julianne Quinn and Peter Vaillancourt. (2019) Lake_Problem_DPS. Aristotle Cloud Federation. [Online]. Available: https://github.com/federatedcloud/Lake_Problem_DPS

[35] Bernardo Trindade, Brandon Barker, Peter Vaillancourt, and Jackie Zheng. (2019) WaterPaths. Aristotle Cloud Federation. [Online]. Available: https://github.com/federatedcloud/WaterPaths

[36] B. Bzeznik, O. Henriot, V. Reis, O. Richard, and L. Tavard, "Nix As HPC Package Management System," in *Proceedings of the Fourth International Workshop on HPC User Support Tools*, ser. HUST'17. New York, NY, USA: ACM, 2017, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org/10.1145/3152493.3152556

[37] A. Devresse, F. Delalondre, and F. Schürmann, "Nix Based Fully Automated Workflows and Ecosystem to Guarantee Scientific Result Reproducibility across Software Environments and Systems," in *Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, ser. SE-HPCCSE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 25–31. [Online]. Available: https://doi.org/10.1145/2830168.2830172

[38] D. Inc. (2020) Docker Hub: Docker Container Registry. Docker Inc. [Online]. Available: https://hub.docker.com/

[39] S. University and SingularityLLC. (2019) Singularity Hub: Singularity Container Registry. Stanford University and SingularityLLC. [Online]. Available: https://singularity-hub.org/

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

[40] Peter Z. Vaillancourt, NixOS. (2020) Dockerfiles to package Nix in a minimal Docker container. XSEDE. [Online]. Available: https://github.com/XSEDE/docker-centos-nix-base
[41] Peter Z. Vaillancourt. (2020) centos-nix-base Docker Image. XSEDE. [Online]. Available: https://hub.docker.com/r/xsede/centos-nix-base
[42] ——. (2020) Singularity base container with Nix. XSEDE. [Online]. Available: https://github.com/XSEDE/singularity-nix-base
[43] ——. (2020) Singularity base image with Nix Collection. XSEDE. [Online]. Available: https://singularity-hub.org/collections/4462
[44] Peter Z. Vaillancourt, NixOS. (2020) Docker container with Nix and OpenMPI. XSEDE. [Online]. Available: https://github.com/XSEDE/docker-centos-nix-openmpi
[45] Peter Z. Vaillancourt. (2020) centos-nix-openmpi Docker Image. XSEDE. [Online]. Available: https://hub.docker.com/r/xsede/centos-nix-openmpi
[46] Peter Z. Vaillancourt, J. Eric Coulter. (2020) Singularity container with Nix and OpenMPI. XSEDE. [Online]. Available: https://github.com/XSEDE/singularity-nix-openmpi
[47] Peter Z. Vaillancourt. (2020) Singularity image with Nix and OpenMPI Collection. XSEDE. [Online]. Available: https://singularity-hub.org/collections/4465
[48] J. Layton, "Lmod–alternative environment modules," *Admin HPC, http://www. admin-magazine. com/HPC/Articles/Lmod-Alternative-Environment-Modules*, 2015.
[49] E. Coulter, R. Knepper, and J. Fischer, "Programmable Education Infrastructure: Cloud resources as HPC Education Environments," *JOCSE*, vol. 1, p. 107, Jan 2019. [Online]. Available: https://doi.org/10.22369/issn.2153-4136/10/1/18
[50] A. Petitet, R. Whaley, J. Dongarra, and A. Cleary, "Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers." [Online]. Available: https://www.netlib.org/benchmark/hpl/
[51] D. J. J., L. Piotr, and P. Antoine, "The linpack benchmark: Past, present, and future," 2001. [Online]. Available: http://www.netlib.org/utk/people/JackDongarra/PAPERS/hpl.pdf
[52] A. C. Technologies, "How do i tune my hpl.dat file?" [Online; accessed 25-Jun-2020; Advanced Clustering Technologies]. [Online]. Available: https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/
[53] D. Jack and G. Eric, "The netlib repository," [Online; accessed 25-Jun-2020;]. [Online]. Available: https://netlib.org
[54] Sylabs, "Sylabs singularity 3.4 documentation," [Online; accessed 25-Jun-2020; Sylabs.io]. [Online]. Available: https://sylabs.io/guides/3.4/user-guide
[55] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 342–346.
[56] P. Saha, A. Beltre, P. Uminski, and M. Govindaraju, "Evaluation of Docker Containers for Scientific Workloads in the Cloud," in *Proceedings of the Practice and Experience on Advanced Research Computing*, ser. PEARC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219104.3229280
[57] A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell, "A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 74–81.
[58] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.
[59] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 233–240.
[60] T. Huber, L. Junji, S. Chandrasekaran, and R. Henschel, "Impact of Virtualization and Containers on Application Performance and Energy Consumption," July 2018. [Online]. Available: https://scholarworks.iu.edu/dspace/handle/2022/22285
[61] M. Babbar-Sebens, S. Rivera, E. Abeysinghe, S. Marru, M. Pierce, E. Coulter, M. Farahani, D. Wannipurage, and M. Christie, "Interactwel science gateway for adaptation planning in food-energy-water sectors of local communities," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3333253
[62] E. L. Morgan, E. Abeysinghe, S. Pamidighantam, E. Coulter, S. Marru, and M. Pierce, "The distant reader: Tool for reading," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3333260
[63] S. P. Eric Coulter, Richard Knepper, "Using the Jetstream Research Cloud to provide Science Gateway resources," *CCGrid*, 2017.
[64] H. Inc. (2020) Terraform. HashiCorp, Inc. [Online]. Available: https://www.terraform.io/

978-1-7281-9219-2/20/$31.00 ©2020 IEEE